University of California
Santa Barbara

Memory-Hard Functions: When Theory Meets Practice

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Binyi Chen

Committee in charge:

Professor Stefano Tessaro, Co-Chair
Professor Huijia Lin, Co-Chair
Professor Subhash Suri

June 2019

The Dissertation of Binyi Chen is approved.

Professor Subhash Suri

Professor Huijia Lin, Committee Co-Chair

Professor Stefano Tessaro, Committee Co-Chair

June 2019

Memory-Hard Functions: When Theory Meets Practice

To my parents and Doris, for their support and love.

Acknowledgements

Many thanks to everyone who provided me continuous support during my invaluable PhD journey.

First and foremost, I would like to express my sincere gratitude to my advisors Stefano Tessaro and Huijia (Rachel) Lin. I barely knew anything about cryptography when I applied for PhD. Rachel made me intrigued with the magic of Zero Knowledge proofs and guided me to the world of cryptography. Rachel is extremely kind, caring and supportive. She is willing to spend tons of hours helping me with research, writing, presentation, and has provided me invaluable life guidance when I was unclear about the future. The same holds true for Stefano, who introduced me to the field of memory-hard functions. I would have never been able to finish most of my research work without his great guidance. He is the one who taught me how to be rigorous, how to think everything in a systematic and logical way, how to always strike for perfection. I thank Stefano and Rachel for their patience, motivation and immense knowledge. This thesis would have never been possible without their continuous help, and I will be forever grateful to them.

I would like to thank Prof. Subhash Suri for being in my thesis committee as well. Subhash is a great teacher and researcher, I have learned so much from his algorithm class on how to ask good questions and how to ponder research problems. I also thank him for his insightful comments and encouragement during my MAE, proposal, and PhD defense.

I want to thank Prof. Leo Reyzin, Prof. Krzysztof pietrzak, and Dr. Joël Alwen for their patient guidance and insightful comments during my preparation of Eurocrypt'17 best paper talk. I especially thank Leo for his immense help during the entire process. He is extremely creative, kind, charming and inspiring, and I am always grateful for his generous encouragement.

I would also like to thank VISA research crypto group that I was fortunate to be a part of during Summer'18. I am grateful to my mentors: Dr. Pratyay Mukherjee and Dr. Yilei Chen. I thank Pratyay for introducing me to the field of non-malleable code, thank Yilei for his continuous inspirations on my research and unconditional help during the summer. I will never forget the World Cup matches that we three have watched together. Besides, I want to thank Payman, Atul, Shashank, and all of the other researchers for their great help during my internship.

My PhD life would have been boring and dull without the company of my lab-mates. I especially thank Pratik and Ben for their help during my last four years. They are extremely kind, industrious and caring. They are great leaders when organizing events (e.g. reading groups) for the lab; they are inspiring co-workers when discussing research; they are considerate and caring friends when we eat, chat, and play together. I sincerely wish them the brightest future. Besides, I would like to thank the other members of our lab as well, Priyanka, Wei, Xihu, Ji, and Ashrujit, they are all very nice and helpful, and all of them are stars that enlighten my PhD journey.

I would like to thank my friends as well, including Yixin, Peihan, Yujun, Da, Tianxing, Zhixiao, Chengyu, Shiliang, for their unconditional help and encouragement, and for the precious time we have ever had together. Besides, I am grateful to my friends Kelu, Qiming, Sidi and Lin, for their care and help when I was visiting UW during my last year of PhD.

I am extremely grateful for my love, Doris, coming to my life, making the final year of my PhD more meaningful. Last but not least, I am eternally grateful to my parents, Zhaohui Long and Jianwen Chen, for their continuous and unconditional love, support, and patience. They made me who I am today.

Curriculum Vitæ
Binyi Chen

Education

2019	Ph.D. in Computer Science, University of California, Santa Barbara.
2014	B.S. in Computer Science, Shanghai Jiao Tong University.

Publications

B. Chen, S. Tessaro, *"Memory-Hard Functions from Cryptographic Primitives"*, CRYPTO 2019

B. Chen, Y. Chen, K. Hostáková, P. Mukherjee *"Continuous Non-Malleable Codes from Stronger Proofs-of-Space"*, CRYPTO 2019

J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, S. Tessaro, *"Scrypt is Maximally Memory-Hard"*, **(BEST PAPER AWARD)** EUROCRYPT 2017

J. Alwen, B. Chen, C. Kamath, V. Kolmogorov, K. Pietrzak, S. Tessaro, *"On the Complexity of Scrypt and Proofs of Space in the Parallel Random Oracle Model"*, EUROCRYPT 2016

B. Chen, H. Lin, S. Tessaro, *"Oblivious Parallel RAM: Improved Efficiency and Generic Constructions"*, TCC 2016-A

B. Chen, T. Qin, T.Y. Liu, *"Mechanism Design for Daily Deals"*, AAMAS 2015

H. Wang, B. Chen, W.J. Li, *"Collaborative Topic Regression with Social Regularization for Tag Recommendation"*, IJCAI 2013

B. Chen, H. Lin, *"Correlation Intractable Functions for Samplable Relations and Their Applications"*, Manuscript

Awards

Best Paper Award, EUROCRYPT Conference	2017
Outstanding Publication Award, UCSB CS Department	2017-2018
Microsoft Academic Scholarship, Microsoft Research Asia	2013
XD.com Scholarship, XD.com	2013
Pan Wen-Yuan Foundation Scholarship,	2011
Academic Excellence Scholarship (First-Class), SJTU	2011
Gold Medalist, 2nd place, ACM-ICPC (Jakarta)	2010
Gold Medalist, 4th place, ACM-ICPC (Chengdu)	2010

Abstract

Memory-Hard Functions: When Theory Meets Practice

by

Binyi Chen

Memory-hard functions (MHFs) is a class of hash functions whose fast evaluation requires the heavy use of memory, and an evaluation that spends less memory has to incur a much larger time penalty. Memory-hardness is particularly useful in the setting of password hashing and cryptocurrencies, as memory cost is platform-independent and efficient special-purpose hardware for brute-forcing attacks becomes much harder to be built. Since its first proposal by Colin Percival in 2009, many memory-hard hash heuristics were proposed, and the notion/design of MHFs has received a considerable amount of theoretical scrutiny as well. However, a large gap still exists when theory meets practice. On the one hand, most of the practical schemes are only heuristics without formal analysis, and attacks do exist for some of them; on the other hand, theoretical analyses are usually based on unrealistic assumptions: they consider MHFs as modes of operation of some underlying hash function $\mathcal{H}$, modeled as a *random oracle*. Unfortunately, in practice, this is never the case as $\mathcal{H}$ is usually a heuristic design built from simpler primitives.

This dissertation makes progress in addressing both of the problems. Our main contributions are threefold. First, we prove that a widely-used MHF candidate, called `scrypt`, is *provably* and *optimally* memory-hard, thus shedding light on the confidence of its wide application. Second, we model simple cryptographic tools (e.g. AES) as the underlying ideal primitives and present a *generic* and *provably-secure* MHF construction from hard-to-pebble graphs. The resulting scheme significantly decreases the efficiency gap between legitimate users and ASICs-equipped attackers. Finally, given the practice demands for $\mathcal{H}$ to have large outputs (to increase memory hardness without changing the description size of MHFs), we go back to the framework of constructing MHFs from $\mathcal{H}$ (with large output). Different from previous work, we take finer-granularity of the hash function $\mathcal{H}$ into account, and provide the first provably secure design of $\mathcal{H}$ from simpler primitives (e.g. fixed-key AES).

Contents

Curriculum Vitae **vii**

Abstract **viii**

1 Introduction **1**
1.1 Password Hashing: Hardware Attacks 1
1.2 Memory-Hard Functions 3
1.3 Our Contributions 5
1.4 Thesis Organization 7
1.5 Associated Publications of the Thesis 9

2 Preliminaries **10**
2.1 Basic Notation 10
2.2 Memory-Hard Functions in the Ideal Primitive Model 11
2.3 Compression Arguments 15
2.4 Graphs and Pebbling Models 17
2.5 Graph-based iMHFs from Labeling Functions 21
2.6 Related Work 23

3 Scrypt is Maximally Memory Hard **26**
3.1 Introduction 27
3.2 Results Overview 29
3.3 Optimal Cumulative Complexity in the Pebbling Game 31
3.4 Single-Challenge Time Lower Bound in the PROM 38
3.5 Main Result: Optimal Memory Hardness of `scrypt` in the PROM 46
3.6 Conclusions and Open Problems 57

4 Data Independent MHFs from Simple Primitives **59**
4.1 Introduction 60
4.2 Small-Block Labeling Functions 62
4.3 Small-Block Labeling Functions: Pebbling Reducibility 64
4.4 iMHFs from Small-Block Labeling Functions 78

4.5 Instantiation 79
4.6 Conclusions and Open Problems 80

5 Data Independent MHFs from Wide-Block Labeling Functions 81
5.1 Introduction 81
5.2 Wide-Block Labeling Functions 82
5.3 Wide-Block Labeling Functions: Pebbling Reducibility 88
5.4 iMHFs from Wide-Block Labeling Functions 96
5.5 Instantiation 100
5.6 Conclusions and Open Problems 100

A Missing Proofs for the Analysis of `scrypt` 102
A.1 Proof of Claim 3.8 102
A.2 The Uniformity of Oracles $\{\mathsf{ro}_k\}$ 103
A.3 Probability Analysis 103

B Missing Proofs in Section 4.3 112
B.1 Proof of Lemma 4.1 112
B.2 Proof of Lemma 4.2 114
B.3 Proof of Claim 4.2 117

C Instantiations of Graph-based iMHFs 119

Bibliography 122

Chapter 1

Introduction

Outline of the Chapter. In this chapter, we introduce the background of memory-hard functions and state our main results. In section 1.1, we take password hashing as an example for motivating the use of memory-hard functions, that is, to harden the special-purpose hardware attacks towards inverting hashes. Then in Section 1.2, we briefly introduce the definition and the state-of-art schemes of MHFs. After introducing the gap between theory and practice, we present our main contributions in Section 1.3. The chapter ends with a thesis organization and associated publications in Section 1.4 and Section 1.5.

1.1 Password Hashing: Hardware Attacks

Password is one of the most important tools for authenticating online users because of its simplicity and easy deployment. However, as the explosive development of ecommerce/online industry, the password information of users becomes a lucrative target for attackers, and password-file breaches of giant organizations/companies [1, 2, 3, 4, 5] is no longer rare news in recent years, making the protection of clients' online secret an urgent task.

To protect passwords, an authentication scheme called password hashing is widely used: instead of storing clients' passwords in the clear, the server stores the *hashes* of passwords. The authentication is done by comparing the input hash with that in the server storage. The hashing scheme significantly increases the cost of cracking passwords: Even if a database was breached and the hash-file was stolen, to obtain passwords, the attacker still needs to invert the hashes which was usually considered to be a hard task. For example, the naïve idea of brute-forcing all possible inputs consumes huge amounts of resources as long as the password input has large entropy.

Unfortunately, human beings are the weakest links of internet security and many users tend to select low entropy passwords. This opens the door for attackers to mounting offline dictionary attacks, where a table (that stores commonly used passwords and their hashes) was precomputed, and after stealing the hash file, the attacker can obtain the passwords by looking up the dictionary table. A countermeasure against offline dictionary attacks is to use public salt to re-randomize hashes so that different users will have different hashes even if they share the same password. Therefore, the attacking process becomes much harder as the dictionary has to be computed on the fly after seeing the salt. Nevertheless, even with salt, *weak* passwords are still susceptible to *brute-force attacks* when the hashes *are easy to compute.*

Towards the end of mitigating offline/brute-force attacks, hash iteration schemes (e.g., BCRYPT [6], PBKDF2 [7]) were introduced to make each hash evaluation more expensive, thus reducing the number of passwords guesses an attacker can attempt. The idea of iteration schemes is to take a basic hash function (e.g. SHA-1/2/3) and iterate it for t times, where t is a tunable parameter that can be increased as hardware chips become faster. However, such iteration technique suffers from two issues. First, by increasing the number of iterations, the computational cost for authenticating a *legitimate* user is increased as well. Second, and more seriously, the fast development of *special-purpose hardware* (e.g., GPU, FPGA, and ASICs) leads to a huge efficiency gap between legitimate users and attackers. In particular, an attacker equipped with ASICs can evaluate the hash function on large amounts of inputs with negligible energy cost. For example, as noted by [8]: "the Antminer S9, an ASIC Bitcoin [9] miner, can compute SHA256 hashes at a rate of 13.6 trillion hashes per second using just 1274 Joules of energy per second (Watts). In contrast, the energy cost for evaluating the same amounts of hashes on a general-purpose CPU is around six orders of magnitude higher." Therefore, confronting the challenge that special-purpose hardware has been rapidly developed (mostly because of the popularity of mining cryptocurrencies), a new hash design that minimizes the efficiency advantage of ASICs (over CPUs) is desired.

1.2 Memory-Hard Functions

With the goal of eliminating special-purpose hardware attacks, researchers observed that i) memory access latency is similar across different platforms, and ii) chips (e.g. energy-efficient DRAM chips) with large memory are relatively expensive in production as that will require considerable amounts of silicon area. Moreover, the increased chip area also makes the cost of accessing memory much higher.[1]

Therefore, a promising direction against ASIC-attacks is to design *memory-hard functions* (MHFs). More specifically, MHFs is a type of hash functions whose *efficient* evaluation requires the use of considerable amounts of memory. The memory-hardness property makes it much harder to launch efficient password recovery attacks as the special-purpose hardware is required to have *huge* amounts of chip space/RAM memory to speed-up the computation (e.g., by exploiting parallelization/pipelining) on inputs a lot of guessing

[1]For example, let m^2 be the area of a chip square, the memory access cost will be roughly linear to m.

passwords.

There has been a lot of interests in designing/analyzing memory-hard functions (cf. e.g. [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]). The Password Hashing Competition announced in 2013 has indicated memory hardness as a de-facto requirement, and there are many hash candidates [21, 22, 23, 24, 25, 26, 27, 28, 29] that are meant to be memory hard. However, most of the practical schemes are only heuristics without formal analysis, and many attacks have been found [12, 30, 31, 16, 32]. Provably secure memory-hard functions have been designed as well [10, 11, 13, 14, 15, 16, 17, 19, 20]. Unfortunately, for most of the theoretically sound constructions, there exists a big *concrete* gap between the achieved memory hardness and the default sequential evaluation cost, making the theoretical results less useful in practice. Worse still, the provably secure constructions are usually based on unrealistic assumptions that make the analysis less convincing in the real world scenario. For example, the elegant work by Alwen and Serbinenko [10] showed a generic approach for constructing memory-hard functions from hard-to-pebble graphs. However, their construction makes use of a random oracle which is usually instantiated with an ad-hoc construction based on some simpler objects (e.g., the hash function underlying Scrypt resembles a permutation-based stream-cipher design). Thus we have no guarantee on its security when the adversary can exploit the internal structure of the hash function.

Given issues arised on MHFs when theory meets practice, we therefore ask the following question.

Can we design a memory-hard function that i) satisfies strong (if not optimal) memory-hard hardness, and ii) enjoys a provable security guarantee in a realistic computational model (e.g. based on a primitive that is highly efficient on general-purpose CPUs)?

We provide affirmative answers to above questions. Moreover, a few of our constructions satisfy extra desired properties for password hashing, e.g., side-channel resilience and small description complexity. We will give more details of our contributions in the next section.

1.3 Our Contributions

Our main contributions are threefold. First, we prove that a widely-used MHF candidate, called `scrypt`, is *provably* and *optimally* memory-hard, thus substantiating its usage. Second, given the observation that AES operations are already fast on general-purpose CPUs (e.g., AES-NI instruction set is embedded in many modern chips), we present a generic MHF construction from hard-to-pebble graphs and AES. The resulting scheme significantly decreases the efficiency gap between legitimate users and ASICs-equipped attackers. Third, we propose a hash primitive that has large output space called *wide-block labeling functions*. The primitive can be built from AES and enables us to increase the memory hardness of MHFs *without changing the size of the function*.

Scrypt is Optimally Memory Hard. Scrypt [33] is a popular MHF candidate proposed by Colin Percival in 2009. It has gained a lot of research/industry interests after the proposal: It is used as a proof-of-work scheme within many cryptocurrencies (most notably Litecoin [34], but also Tenebrix or Dogecoin), was published by IETF as a key-derivation function (RFC 7914) [35], and has inspired the design of one of the Password-hashing Competition's [36] winners, Argon2d [37].

We show that Scrypt is *optimally* memory hard in a computational model proposed by [10], called the parallel random oracle model (PROM). The result is also the *first* unconditional memory hardness lower bound for Scrypt in the PROM. To appreciate the novelty of our results, along the road, we also obtain an *optimal* pebbling complexity lower bound for a combinatorial game that is tightly related to the evaluation of Scrypt.

iMHFs from Simple Primitives. Data independent memory-hard function (iMHFs) is a class of MHFs whose evaluation pattern is orthogonal to the data input. It satisfies a property called side-channel resilience which is desirable for password hashing where the data input is usually sensitive information (i.e. passwords). Alwen and Serbinenko [10] proposed an elegant framework for constructing iMHFs from a DAG $\mathbb{G}$ and a hash function $\mathcal{H}$ modeled as a random oracle. However, the underlying hash $\mathcal{H}$ is usually instantiated with an ad-hoc construction based on simpler building blocks (e.g. stream ciphers or block ciphers). This leads to two issues: First, in practice, the hash function $\mathcal{H}$ is far from an ideal random oracle, and highly efficient ASICs that exploits the internal structure of $\mathcal{H}$ do exist; second, since the evaluation of $\mathcal{H}$ is slower on CPUs, the entire computation of MHFs on general-purpose CPUs becomes less efficient as well, which slows down the authentication speed.

To remedy the issues, we observe that AES operations are particularly fast on general-purpose CPUs (e.g., AES-NI instruction set is widely embedded in CPU chips), and an ASIC (for AES) that achieves high efficiency advantage over CPUs is much harder to be built. From this observation, we propose a generic framework for constructing iMHFs from (fixed-key) AES modeled as a random permutation. For completeness, we also provide constructions based on other simple cryptographic primitives (e.g., compression functions and keyed block ciphers). Towards the goal, we provide an efficient instantiation of $\mathcal{H}$ that invokes the fast primitive only once. We will adapt previous lemmas based on ex-post-facto arguments (dating back to [38]) to reduce the security of iMHFs to the pebbling complexity of the underlying DAG.

Bootstrapping Memory Hardness from Wide-Block Labeling Functions. The construction above achieves high CMC (e.g. $N^2L/\log N = 2^{60}$) in practice if we choose a big graph parameter N. However, the program size (that is superlinear to N) will increase as well. This posts obstacles for embedding the program into mobile devices/hardware.

A solution adopted by practitioners is to use a tailored-made hash function and expand the block size from $L = 128$ to a larger value W (e.g., $W = 8192$), and hence the CMC lower bound is now $N^2W/\log N$. However, the previous analyses consider the underlying hash function $\mathcal{H}$ as a random oracle where an adversary cannot exploit the inner structure. We open the box of $\mathcal{H}$ and explore a step further. Specifically, we provide a generic construction of the hash function $\mathcal{H}$ (with W-bit output) using a simple primitive that has a shorter output. We call it as a *wide-block labeling function.* For the graph-based iMHFs scheme proposed by [10], we replace the underlying hash function $\mathcal{H}$ with our construction. As long as the graph $\mathbb{G}$ satisfies a widely-used property called depth-robustness, the resulting scheme is memory-hard even if the adversary can exploit the structure of $\mathcal{H}$ and make oracle queries to the underlying simpler primitive.

1.4 Thesis Organization

The remainder of the thesis is organized as follows:

Chapter 2 starts with the basic notation we will use in the following chapters. We then formally define memory-hard functions in a computational model called *parallel ideal-primitive model* where an adversary can adaptively query an ideal-oracle on many inputs at a single step. This is followed by defining several combinatorial games on directed acyclic graphs called *pebbling games*, which have tight relation to the evaluation of MHFs. We then review a framework for constructing iMHFs from graphs and hash primitives. The chapter ends with some related work of memory-hard functions.

In **Chapter 3** we show that a popular MHF candidate – Scrypt [33] – is optimally memory-hard. We start off with a description of the Scrypt scheme and explain its applications. We then provide an overview of our contributions: as a warm-up, we state an optimal lower bound on the complexity of a pebbling game that is tightly related to the evaluation of Scrypt; this is followed by our major contribution, an unconditional optimal lower bound on Scrypt's memory hardness in the parallel random oracle model. To highlight our technical contribution, we also provide a general single-challenge time lower bound in the parallel random oracle model. After the overview, we provide with formal treatments of our main results. Finally, we end the chapter with some open problems.

Chapter 4 presents a class of *data-independent* memory-hard functions (iMHFs) built from fast symmetric cryptographic primitives (e.g. fixed-key blockciphers). The construction follows the framework by Alwen and Serbinenko [10], which builds MHFs from graphs and hash primitives. We start by defining and constructing a hash primitive called *small-block labeling functions*. After proving that the labeling function satisfies a property called pebbling reducibility, we show how to construct iMHFs from small-block labeling functions.

Finally in **Chapter 5** we construct *succinct* iMHFs from a primitive called *wide-block labeling functions*. We start by motivating the use of *wide-block labeling functions*, then we give a formal definition and a construction from simple cryptographic primitives used in Chapter 4. After showing the pebbling reducibility of our construction, we finish by giving a generic framework for constructing iMHFs from wide-block labeling functions.

1.5 Associated Publications of the Thesis

The contents of Chapter 3 is the result of a paper co-authored with Joël Alwen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. This paper merged the work of Leonid Reyzin, Joël Alwen, and Krzysztof Pietrzak, and the author's work with Stefano Tessaro. Thus the paper was jointly written by all of the authors. The joint work [15] has won the *Best Paper Award* in the proceedings of Eurocrypt 2017. Some of the contents of Chapter 3 are taken from this joint write up (with permission of co-authors).

Chapters 4 and 5 are based on a joint work [39] with Stefano Tessaro, which was published in the proceedings of Crypto 2019.

The copyright for the publications above is held by © IACR 2017, 2019.

Chapter 2

Preliminaries

Outline of the Chapter. In this chapter, we introduce the basic definitions and notation used in the thesis for constructing memory hard functions (MHFs). We start with basic notational conventions in Section 2.1. Then in Section 2.2 we describe the definition of memory hard functions in a computational model called *the parallel ideal primitive model* [10, 39], and in Section 2.3 we review the classical compression arguments. Next in Section 2.4, after introducing necessary graph notation, we describe a combinatorial game called *pebbling game* [40, 41], which is tightly related to the computation of MHFs. The rest of the chapter is then focused on the background of graph-based MHFs, whereas we describe a general framework for constructing MHFs from graphs and hash functions. Our treatment is based on [10], with notational changes borrowed from [39]. Finally, we discuss some related work of memory-hard functions in Section 2.6.

2.1 Basic Notation

Following convention, we use $\mathbb{N}$ and $\mathbb{R}$ to denote the set of natural numbers and real number. For any $n \in \mathbb{N}$, $[n]$ is used to denote the set $\{1, \ldots, n\}$. For any $a, b \in \mathbb{R}$,

$(a, b]$ is used to denote the set of real numbers larger than a but no larger than b. For binary strings x and y, $|x|$ is the length of x and we use $(x||y)$ or (x, y) to denote the concatenation of x and y. For a set $\mathbb{X}$, $|\mathbb{X}|$ is the number of elements in $\mathbb{X}$, and $x \overset{\$}{\leftarrow} \mathbb{X}$ is the process of assigning x as a uniformly chosen element from $\mathbb{X}$. For a distribution $\mathcal{D}$, we use $x \leftarrow \mathcal{D}$ to denote the sampling of x from distribution $\mathcal{D}$. By $\log(\cdot)$ we always refer to binary logarithm.

2.2 Memory-Hard Functions in the Ideal Primitive Model

2.2.1 Ideal Primitives

In practice, MHFs are built from symmetric cryptographic primitives, e.g., block ciphers, compression functions, or variable-length hash functions. In this thesis, we model these symmetric cryptographic primitives as ideal and analyze the security of MHFs in the ideal model. Fix $L = 2^\ell$ and $W = 2^w$ (where $W \gg L$), we consider four types of ideal primitives: (In the following context, we will omit L and W in the ideal-primitive notation if there is no ambiguity.)

1. Ideal compression function: We use $\mathbb{CF}$ to denote the set of functions[1] with domain $\{0,1\}^L \cup \{0,1\}^{2L}$ and image $\{0,1\}^L$.

2. Ideal cipher: We use $\mathbb{IC}$ to denote the set of keyed permutations with domain $\mathcal{K} \times \{0,1\}^L$ and image $\{0,1\}^L$. For simplicity, the key space is set as $\mathcal{K} := \{\perp\} \cup \{0,1\}^L$

[1]Though most compression functions do not allow L-bit inputs by design, we could however easily take L-bit inputs by reserving one input bit of the compression function to implement domain separation, and then padding short inputs.

in the following context[2].

3. Random permutation: We use $\mathbb{RP}$ to denote the set of permutations with input/output space $\{0,1\}^L$.

4. Random oracle: We use $\mathbb{RO}$ to denote the set of functions with input/output space $\{0,1\}^W$.

2.2.2 Parallel Ideal Primitive Model

Alwen and Serbinenko [10] introduced the *parallel Random Oracle Model* (pROM) to model the computation of memory hard functions. We generalize the pROM into a new computational model called the *parallel Ideal Primitive Model* [39] where the random oracle is replaced with any one out of four ideal primitives above. We note that the model is the same as pROM when the ideal primitive is a random oracle.

Let $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}/\mathbb{RO}$ be a type of ideal primitive set. For an oracle-aided algorithm A, input x and random coins r, the execution $\mathsf{A}(x;r)$ works as follows. First, a function ip is uniformly chosen from the set $\mathbb{IP}$. The oracle-aided algorithm A can make oracle query to ip as follows: If $\mathsf{ip} = \mathsf{ro}$ is randomly sampled from the set $\mathbb{RO}$, the algorithm can make queries with form $(\text{“}\mathbb{RO}\text{”}, x)$ and receive value $\mathsf{ro}(x)$. If $\mathsf{ip} = \mathsf{cf}$ is a randomly sampled compression function, the algorithm can make queries with form $(\text{“}\mathbb{CF}\text{”}, x)$ and receive value $\mathsf{cf}(x)$. If $\mathsf{ip} = \mathsf{ic}$ is a randomly sampled ideal cipher, the algorithm can make *forward* queries with form $(\text{“}\mathbb{IC}\text{”}, +, k, x)$ and receive value $\mathsf{ic}(k,x)$, or make *inverse* queries with form $(\text{“}\mathbb{IC}\text{”}, -, k, y)$ and receive value $\mathsf{ic}^{-1}(k,y)$. Similarly, if $\mathsf{ip} = \mathsf{rp}$ is a randomly sampled permutation, the algorithm can make *forward* queries with form $(\text{“}\mathbb{RP}\text{”}, +, x)$ and receive value $\mathsf{rp}(x)$, or make *inverse* queries with form $(\text{“}\mathbb{RP}\text{”}, -, y)$

[2] $\perp$ is a designated key separate from the L-bit strings, which is necessary to implement variable input length.

and receive value $\mathsf{rp}^{-1}(y)$.

Denote as $\sigma_0 = (x, \emptyset)$ the initial input state. For each round $i \in \mathbb{N}$, $\mathsf{A}(x; r)$ takes input state σ_{i-1}, performs *unbounded* computation, and generates an output state $\bar{\sigma}_i = (\delta_i, \mathbf{q}_i, \mathbf{out}_i)$, where δ_i is a binary string, $\mathbf{q}_i$ is a vector of queries to the ideal primitive ip, and $\mathbf{out}_i$ is a vector of output labels. $\sigma_i = (\delta_i, \mathbf{ans}(\mathbf{q}_i))$ is defined as the input state for round $i+1$, where $\mathbf{ans}(\mathbf{q}_i)$ is the vector of ideal primitive answers to $\mathbf{q}_i$. The execution terminates after round $T \in \mathbb{N}$ if $|\mathbf{q}_T| = 0$. We use $\mathsf{A}^{\mathsf{ip}}(x; r)$ to indicate both the execution output (i.e., the concatenation of output labels) and the execution *trace* (i.e., all of the input and output states $(\sigma_0, \bar{\sigma}_1, \sigma_1, \dots)$). We call A a *sequential* algorithm if $|\mathbf{q}_i| = 1$ for every $1 \le i < T$, otherwise A is a *parallel* algorithm.

Complexity Measures. We measure the complexity of computation using ST-complexity and cumulative memory complexity (CMC) [10, 14]. For a trace $\mathsf{A}^{\mathsf{ip}}(x; r)$ with respect to input x, randomness r and ideal primitive ip, the time complexity $\mathsf{Tm}(\mathsf{A}^{\mathsf{ip}}(x; r))$ is the number of rounds ran by A and the space complexity $\mathsf{Spc}(\mathsf{A}^{\mathsf{ip}}(x; r))$ is the bit-size of the maximal input state. We define ST-complexity and cumulative memory complexity as follows.

Definition 2.1 (Complexity Measures) *Given a trace $\mathsf{A}^{\mathsf{ip}}(x; r)$, the ST-complexity of $\mathsf{A}^{\mathsf{ip}}(x; r)$ is*

$$\mathsf{ST}(\mathsf{A}^{\mathsf{ip}}(x; r)) := \mathsf{Spc}(\mathsf{A}^{\mathsf{ip}}(x; r)) \cdot \mathsf{Tm}(\mathsf{A}^{\mathsf{ip}}(x; r))\,;$$

the cumulative memory complexity of $\mathsf{A}^{\mathsf{ip}}(x; r)$ is

$$\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(x; r)) := \sum_{i=0}^{\mathsf{Tm}(\mathsf{A}^{\mathsf{ip}}(x;r))} |\sigma_i|\,,$$

where σ_i is the input state of round i. Note that $\mathsf{ST}(\mathsf{A}^{\mathsf{ip}}(x; r)) \ge \mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(x; r))$.

For any $\epsilon \in (0,1]$, $\mathsf{q} \in \mathbb{N}$, *and any functions family* $\mathcal{F} = \{ f^{\mathsf{ip}} : \mathcal{X} \to \mathcal{Y} \}_{\mathsf{ip} \in \mathbb{IP}}$, *we define the* (ϵ, q)*-cumulative memory complexity of* $\mathcal{F}$ *to be*

$$\mathsf{CMC}_{\epsilon,\mathsf{q}}(\mathcal{F}) := \min_{x \in \mathcal{X},\, \mathsf{A} \in \mathcal{A}^{\parallel}_{x,\epsilon,\mathsf{q}}} \mathbb{E}\left[\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(x;r))\right] ,$$

where the expectation is taken over the uniform choices of ip *and* r. $\mathcal{A}^{\parallel}_{x,\epsilon,\mathsf{q}}$ *is the set of parallel algorithms that make at most* q *queries and outputs* $f^{\mathsf{ip}}(x)$ *with probability at least* ϵ *(over the uniform choices of* ip *and* r*). We will sometimes omit the parameter* q *when there is no ambiguity.*

For any $T \in \mathbb{N}$, *we define the* T*-ST complexity of* $\mathcal{F}$ *to be*

$$\mathsf{ST}(\mathcal{F}, T) := \min_{\mathsf{A} \in \mathcal{A}_T} \left(\max_{x \in \mathcal{X},\, \mathsf{ip} \in \mathbb{IP}} \mathsf{ST}(\mathsf{A}^{\mathsf{ip}}(x)) \right) ,$$

where $\mathcal{A}_T$ *is the set of* deterministic *and* sequential *algorithms that takes at most* T *steps*[3] *and outputs* $f^{\mathsf{ip}}(x)$ *for any* $x \in \mathcal{X}$ *and* $\mathsf{ip} \in \mathbb{IP}$.

ST-complexity works for measuring *sequential efficiency*, it is an upper bound on the *sequential* complexity of $\mathcal{F}$; CMC works for measuring *security*, it is a *lower bound* on the computational complexity of any *parallel* algorithm that evaluates $\mathcal{F}$.

Assumption 2.1 *In Chapter 4 and Chapter 5, we will implicitly assume an upper bound on the number of queries being made when* q *is not explicitly stated, that is,* $\sum_{i \geq 1} |\mathbf{q}_i| + |\mathbf{out}_i| \leq \mathsf{q}$. *We will also omit* q *in notation for simplicity.*

2.2.3 Memory-Hard Functions

We define memory hard functions in the parallel ideal primitive model. Intuitively, there exists a relatively efficient *sequential* algorithm that computes the MHFs, and any

[3]Since the algorithm is sequential, it makes at most T queries as well.

parallel algorithm that evaluates the functions incurs high CMC cost.

Definition 2.2 (Memory Hard Functions) *For an ideal primitive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}/\mathbb{RO}$, a family of functions $\mathcal{F} = \{ f^{\mathsf{ip}} : \mathcal{X} \to \mathcal{Y} \}_{\mathsf{ip}\in\mathbb{IP}}$ is $(\mathsf{C}^{\parallel}_{\mathcal{F}}, \Delta_{\mathcal{F}}, T_{\mathcal{F}})$-memory hard if and only if the following properties hold.*

Memory-hardness: *For any $\epsilon \in (0,1]$, we have $\mathsf{CMC}_{\epsilon}(\mathcal{F}) \geq \mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon)$.*

Efficiency-gap: *For any $\epsilon \in (0,1]$, we have*

$$\frac{\epsilon \cdot \mathsf{ST}(\mathcal{F}, T_{\mathcal{F}})}{\mathsf{CMC}_{\epsilon}(\mathcal{F})} \leq \Delta_{\mathcal{F}}(\epsilon)\,.$$

We stress that the parameter $T_{\mathcal{F}}$ measures the sequential time cost *for evaluating the MHFs.*

2.3 Compression Arguments

In this section, we review the classical result from [42, 43, 44] that random strings are not compressible. After that, we extend the compression argument into the permutation setting, that is, random permutations/ideal ciphers cannot be compressed either.

Lemma 2.1 ([42, 43, 44]) *Fix an algorithm A, let $\mathcal{B}$ be a sequence of random bits. A on input a hint $h \in \mathcal{H}$ adaptively queries specific bits of $\mathcal{B}$ and outputs p indices of $\mathcal{B}$ that were not queried before, along with guesses for each of the bits. The probability (over the choice of $\mathcal{B}$ and randomness of A) that there exists an $h \in \mathcal{H}$ where $\mathsf{A}(h)$ guesses all bits correctly is at most $|\mathcal{H}|/2^p$.*

Lemma 2.2 *Fix $L \in \mathbb{N}$ and an algorithm A that can make no more than $\mathsf{q} = 2^{L-2}$ oracle queries. Let ic be an ideal cipher uniformly chosen from the set $\mathbb{IC}$ with domain $\mathcal{K}\times\{0,1\}^L$ and image $\{0,1\}^L$.*[4] *A on input a hint $h \in \mathcal{H}$ adaptively makes forward/inverse queries*

[4]A random permutation can be viewed as an ideal cipher with a fixed key.

to ic*, and outputs* $p \leq 2^{L-2}$ *ideal primitive entries (as well as guesses for each of the entry values) that were not queried before. The probability (over the choice of* ic *and randomness of* A*) that there exists an* $h \in \mathcal{H}$ *where* $\mathsf{A}(h)$ *guesses all permutation entries correctly is at most* $|\mathcal{H}|/2^{p(L-1)}$*.*

Proof: Without loss of generality we assume that A is deterministic. For every $h \in \mathcal{H}$, let good_h denote as the set of ideal ciphers ic where $\mathsf{A}^{\mathsf{ic}}(h)$ predicts all permutation entries correctly. Hence the probability we want to bound is

$$\frac{\left|\bigcup_{h\in\mathcal{H}} \mathsf{good}_h\right|}{|\mathbb{IC}|}.$$

Since $|\bigcup_{h\in\mathcal{H}} \mathsf{good}_h| \leq \sum_{h\in\mathcal{H}} |\mathsf{good}_h|$, it is sufficient to bound $|\mathsf{good}_h|/|\mathbb{IC}|$ for every $h \in \mathcal{H}$. Equivalently, it is sufficient to prove that for every $h \in \mathcal{H}$, the probability (over the *uniform* choice of ic) that $\mathsf{A}(h)$ guesses all permutation entries correctly is at most $1/2^{p(L-1)}$.

We consider a mental experiment where we simulate $\mathsf{A}(h)$ while lazily sampling the entry values of ic. Upon receiving a forward call with input (k, x) where $\mathsf{ic}(k, x)$ is undetermined yet, we uniformly sample $\mathsf{ic}(k, x) := y \in \{0,1\}^L$ from the image values that have not been used before in the permutation $\mathsf{ic}(k, \cdot)$; upon receiving an inverse call with input (k, y) where $\mathsf{ic}^{-1}(k, y)$ is undetermined yet, we uniformly sample $\mathsf{ic}^{-1}(k, y) := x \in \{0,1\}^L$ from the pre-image values that have not been used before in the permutation $\mathsf{ic}(k, \cdot)$. At the end of the simulation, A outputs p tuples $\{(k_i, x_i, y_i)\}_{i\in[p]}$, where for every $i \in [p]$, $\mathsf{ic}(k_i, x_i)$ and $\mathsf{ic}^{-1}(k_i, y_i)$ were not queried before and thus are un-determined. Finally, we sample the entry values of $\mathsf{ic}(k_i, x_i)$ for every $i \in [p]$, and sample the rest entries of $\mathsf{ic}(\cdot, \cdot)$. It is easy to see that ic is uniformly chosen from the set $\mathbb{IC}$. Moreover, for every $i \in [p]$, when sampling $\mathsf{ic}(k_i, x_i)$, the number of used image values in $\mathsf{ic}(k_i, \cdot)$ is at most $\mathsf{q} + p \leq 2 \cdot 2^{L-2} = 2^{L-1}$ and there are still at least $2^L - 2^{L-1} = 2^{L-1}$ values to choose. Therefore, the probability that all the guesses $\{y_i\}_{i\in[p]}$ are correct is at most $(\frac{1}{2^{L-1}})^p = \frac{1}{2^{p(L-1)}}$. ■

2.4 Graphs and Pebbling Models

In this section, we review necessary graph notation, and describe two combinatorial games called *black pebbling game* [40, 41] and *challenge pebbling game*, which are tightly related to the computation of MHFs. Most of the notational treatments are based on [11, 14, 39], and some passages were taken verbatim from [39].

Graph Notations. Denote as $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ a directed acyclic graph (DAG) with $N = 2^n$ vertices where we implicitly assume $\mathbb{V} = \{1, \ldots, N\}$). For each vertex $v \in \mathbb{V}$, $\mathsf{pred}(v) := \{u : (u, v) \in \mathbb{E}\}$ are the predecessor nodes of v, $\mathsf{succ}(v) := \{w : (v, w) \in \mathbb{E}\}$ is the set of v's successors. we denote as $\mathsf{src}(\mathbb{G}) \subseteq \mathbb{V}$ the set of source vertices, that is, the vertices which have no predecessors; and denote as $\mathsf{sink}(\mathbb{G}) \subseteq \mathbb{V}$ the set of sink vertices, that is, the vertices which have no successors. The indegree of v is defined as $\mathsf{indeg}(v) := |\mathsf{pred}(v)|$ and the indegree of $\mathbb{G}$ is defined as $\mathsf{indeg}(\mathbb{G}) := \max_{v\in\mathbb{V}} \mathsf{indeg}(v)$. For a directed acyclic path P, the length of P is the number of nodes it traverses. $\mathsf{depth}(\mathbb{G})$ is the length of the longest path in $\mathbb{G}$. For a nodes set $S \subseteq \mathbb{V}$, $\mathbb{G} - S$ is the DAG obtained from $\mathbb{G}$ by removing S and incident edges.

Black Pebbling Game. Consider a pebbling game played on a DAG $(\mathbb{G} = \mathbb{V}, \mathbb{E})$ [40, 41, 10]. At each step, the player can put pebbles on a subset of vertices according to some rules, and the goal is to put pebbles on sink vertices finally. We denote by a *parallel* (black) pebbling on $\mathbb{G}$ as a sequence of pebbling configurations $\mathsf{P} = (\mathsf{P}_0, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ where $\mathsf{P}_0 = \emptyset$ and $\mathsf{P}_i \subseteq \mathbb{V}$ $(1 \le i \le t_{\mathsf{peb}})$. We define two properties for P.

- *Legality:* We say P is legal if it satisfies follows: A pebble can be put on a node $v \in \mathbb{V}$ only if v is a *source* node or v's predecessors were all pebbled at the end of the previous step, that is, for any $i \in [t_{\mathsf{peb}}]$ and any $v \in \mathsf{P}_i \setminus \mathsf{P}_{i-1}$, it holds that $\mathsf{pred}(v) \subseteq \mathsf{P}_{i-1}$.[5]

- *Successfulness:* We say P is successful if it satisfies follows: Every sink node has been pebbled at least once, that is, for any $v \in \mathsf{sink}(\mathbb{G})$, there exists $i \in [t_{\mathsf{peb}}]$ such that $v \in \mathsf{P}_i$.

We say P is a *sequential* (black) pebbling if it further satisfies that $|\mathsf{P}_i \setminus \mathsf{P}_{i-1}| = 1$ for every $i \in [t_{\mathsf{peb}}]$.

Challenge Pebbling Game. Let $(\mathbb{G} = \mathbb{V}, \mathbb{E})$ be a single source/sink DAG and $\mathcal{D}$ be an efficiently samplable distribution over vertex set $\mathbb{V}$. For $m \in \mathbb{N}$, consider an m-round randomized pebbling game. Initially, the pebbling configuration P_0 contains only a pebble on the source vertex. At the beginning of each challenge phase i $(1 \leq i \leq m)$, a challenge node $c_i \leftarrow \mathcal{D}$ is sampled. The player at each step can put pebbles on a subset of vertices according to the same *legality* rule as in the black pebbling game, and the goal of the player is to put a pebble on c_i, which triggers the beginning of challenge phase $i+1$ *at the next step*. The game finishes after the last (i.e., the m-th) challenge node was pebbled. We similarly define parallel (and sequential) pebbling strategies as in black pebbling games. However, for simplicity of calculation in Section 3.3, we further assume that the pebbling configurations of pebbling strategies are *never empty*, that is, for any particular pebbling sequence $\mathsf{P} = (\mathsf{P}_0, \ldots, \mathsf{P}_t)$, we have $|\mathsf{P}_i| \geq 1$ for every $1 \leq i \leq t$.

Note that the pebbling strategies (i.e., the player) are adaptive on the choices of challenge nodes. For a pebbling strategy P, and a challenge sequence $\mathbf{c} = (c_1, \ldots, c_m)$,

[5] $\mathsf{pred}(v) = \emptyset$ for $v \in \mathsf{src}(\mathbb{G})$.

we use $\mathsf{P}(\mathbf{c})$ to denote the particular sequence of pebbling configurations given $\mathbf{c}$. (We assume that the pebbling sequence $\mathsf{P}(\mathbf{c})$ is deterministic for a fixed challenge sequence $\mathbf{c}$.) We emphasize that the decision of the strategy P (i.e. the choices of putting/removing pebbles) in round i ($1 \leq i \leq m$) is independent of the choices of challenges $c_{i+1}, \ldots, c_m$.

Complexity Measures. To measure the cost of pebbling, we define ST-complexity and cumulative complexity for black/challenge pebbling games.

Definition 2.3 (Complexity Measures for Black Pebbling Game [10]) *For a pebbling strategy $\mathsf{P} = (\mathsf{P}_0 = \emptyset, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$, we define the cumulative complexity (and ST-complexity) of P to be*

$$\mathsf{cc}(\mathsf{P}) := \sum_{i=0}^{t_{\mathsf{peb}}} |\mathsf{P}_i|\,, \qquad \mathsf{st}(\mathsf{P}) := t_{\mathsf{peb}} \cdot \max_{i \in [t_{\mathsf{peb}}]} (|\mathsf{P}_i|)\,.$$

For a DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, let $\mathcal{P}^{\parallel}(\mathbb{G})$ be the set of parallel pebblings of $\mathbb{G}$ that are legal and successful; for any $t \in \mathbb{N}$, let $\mathcal{P}_t(\mathbb{G})$ be the set of sequential pebblings of $\mathbb{G}$ that are legal and successful and takes at most t steps. We define the cumulative complexity (and ST-complexity) of $\mathbb{G}$ as

$$\mathsf{cc}(\mathbb{G}) := \min_{\mathsf{P} \in \mathcal{P}^{\parallel}(\mathbb{G})} \mathsf{cc}(\mathsf{P})\,, \qquad \mathsf{st}(\mathbb{G}, t) := \min_{\mathsf{P} \in \mathcal{P}_t(\mathbb{G})} \mathsf{st}(\mathsf{P})\,.$$

Definition 2.4 (Complexity Measures for Challenge Pebbling Game) *For any $m \in \mathbb{N}$, DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, and distribution $\mathcal{D}$ over $\mathbb{V}$, let $\mathcal{P}^{\parallel}(m, \mathbb{G}, \mathcal{D})$ be the set of parallel challenge pebblings w.r.t. m, $\mathbb{G}$ and $\mathcal{D}$. (In the following context, we implicitly assume that the challenge pebblings are legal and successful.) For any $t \in \mathbb{N}$, let $\mathcal{P}_t(m, \mathbb{G}, \mathcal{D})$ be the set of sequential challenge pebbling strategies that takes at most t steps for any choice of the challenge sequence. We define the cumulative complexity (and ST-complexity) of* *the $(m, \mathbb{G}, \mathcal{D})$-challenge pebbling game as*

$$\mathsf{cc}(m, \mathbb{G}, \mathcal{D}) := \min_{\mathsf{P} \in \mathcal{P}^{\parallel}(m,\mathbb{G},\mathcal{D})} \left(\mathop{\mathbb{E}}_{\mathbf{c} \leftarrow \mathcal{D}^m} [\mathsf{cc}(\mathsf{P}(\mathbf{c}))] \right) ,$$

and

$$\mathsf{st}(m, \mathbb{G}, \mathcal{D}, t) := \min_{\mathsf{P} \in \mathcal{P}_t(m,\mathbb{G},\mathcal{D})} \left(\mathop{\mathbb{E}}_{\mathbf{c} \leftarrow \mathcal{D}^m} [\mathsf{st}(\mathsf{P}(\mathbf{c}))] \right) ,$$

where $\mathsf{P}(\mathbf{c})$ is the particular pebbling configuration sequence given the challenge nodes vector $\mathbf{c}$. We emphasize again that the strategy P knows the m challenges phase by phase, instead of knowing all of them in a batch at the beginning (i.e., the prefix of $\mathsf{P}(\mathbf{c})$ up to round i is independent of the choices of $c_{i+1}, \ldots, c_m$).

Similar as in Section 2.2, we stress that ST-complexity measures *sequential efficiency*, while cumulative complexity measures the *memory hardness* against *parallel* algorithms.

Depth Robustness. We review two useful graph-theoretic property called *depth-robustness* and *source-to-sink depth-robustness.*

Definition 2.5 (Depth-Robustness [14]) *A DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is (e, d)-depth-robust if and only if for any $S \subseteq \mathbb{V}$ such that $|S| \le e$, it holds that $\mathsf{depth}(\mathbb{G} - S) \ge d$.*

Definition 2.6 (Source-to-Sink-Depth-Robustness [39]) *A DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is (e, d)-source-to-sink-depth-robust if and only if for any $S \subseteq \mathbb{V}$ with at most e elements, $\mathbb{G} - S$ has a path $(v_1, \ldots, v_t)$ such that $t \ge d$, $v_1 \in \mathsf{src}(\mathbb{G})$, and $v_t \in \mathsf{sink}(\mathbb{G})$.*

There is a tight relation between depth-robustness and cumulative complexity.

Lemma 2.3 ([14]) *For any (e, d)-depth-robust DAG $\mathbb{G}$, we have $\mathsf{cc}(\mathbb{G}) \ge e \cdot d$.*

2.5 Graph-based iMHFs from Labeling Functions

In this section, we review a framework for constructing data-independent MHFs (iMHFs) from graphs and labeling functions [10]. By data-independence we mean that the memory access pattern of default evaluation algorithm is independent of the input. The framework is widely used in Chapter 4 and Chapter 5. Most of the notational treatments were taken verbatim from [39].

Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a DAG with $N = 2^n$ vertices. Without loss of generality we assume that the vertex set $\mathbb{V}$ is sorted according to topological order, and for simplicity we index $\mathbb{V}$ as $\{1, \dots, N\}$, where $\mathsf{src}(\mathbb{G}) = \{1, \dots, n_s\}$ is the set of source vertices and $\mathsf{sink}(\mathbb{G}) = \{N - n_t + 1, \dots, N\}$ is the set of sink vertices. Let

$$\mathcal{H} = \mathcal{H}_{\delta,w} = \{\, \mathsf{lab}_{\gamma}^{\mathsf{ip}} : \{0,1\}^{\gamma W} \to \{0,1\}^{W} \,\}_{\gamma \in [\delta], \mathsf{ip} \in \mathbb{IP}}$$

be a family of labeling functions built upon ideal primitive $\mathbb{IP}$, where $W = 2^w$ is the output length and δ is the maximal ratio between the lengths of input and output. The family of *graph-based iMHFs* $\mathcal{F}_{\mathbb{G},\mathcal{H}} = \{\mathsf{F}_{\mathbb{G}}^{\mathsf{ip}}\mathcal{H} : \{0,1\}^{n_s W} \to \{0,1\}^{n_t W}\}_{\mathsf{ip} \in \mathbb{IP}}$ is defined as follows. Fix any $\mathsf{ip} \in \mathbb{IP}$ and input $\mathbf{x} = (x_1, \dots, x_{n_s}) \in \{0,1\}^{n_s W}$. Let $\ell_i := \mathsf{lab}_1^{\mathsf{ip}}(x_i)$ be the label of vertex i ($1 \le i \le n_s$). For each $v \in [N]$, the label of v is recursively defined as

$$\ell_v := \mathsf{lab}_{\gamma}^{\mathsf{ip}}(\ell_{u_1}, \dots, \ell_{u_\gamma}) \,,$$

where $(u_1, \dots, u_\gamma)$ is the list of v's predecessor nodes. The function output is $(\ell_{N-n_t+1} \| \dots \| \ell_N)$, where ℓ_i ($N - n_t < i \le N$) is the label of sink vertex i.

Input Constraint. For the general case where $n_s = |\mathsf{src}(\mathbb{G})| > 1$, we implicitly constrain the blocks of the input vector $\mathbf{x} = (x_1, \dots, x_{n_s})$ to be non-overlapping (and we call it a *non-colliding* input vector), that is, $x_i \neq x_j$ for any $i, j \in \mathbb{N}$ where $i \neq j$. This constraint is necessary for preventing the adversary from easily saving memory. For example, if all of the blocks are identical, the adversary can compress the input by storing only a single block. The constraint is also reasonable as we can re-randomize the original input using a random oracle $\mathsf{RO} : \{0,1\}^{n_s W} \to \{0,1\}^{n_s W}$, and the output blocks will be distinct with overwhelming probability as long as $n_s \ll 2^{W/2}$.

Graph Constraints. The graphs we are concerned within the thesis satisfy certain properties (which already hold for most of the practical graph constructions). In particular, each 2-indegree DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ considered in Chapter 4 is *predecessors-distinct*, that is, for any two different vertices $u, v \in \mathbb{V} \backslash \mathsf{src}(\mathbb{G})$, we have $\mathsf{pred}(u) \neq \mathsf{pred}(v)$. Looking ahead, this constraint is used to prevent non-source nodes label collisions. On the other hand, each δ-indegree DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ considered in Chapter 5 satisfies *first-predecessor-distinctness*, that is, there exists a way of choosing the *first* predecessor $\mathsf{fpre}(v) \in \mathsf{pred}(v)$ for every non-source vertex $v \in \mathbb{V}$, such that for any two different vertices $u, v \in \mathbb{V} \backslash \mathsf{src}(\mathbb{G})$, we have $\mathsf{fpre}(u) \neq \mathsf{fpre}(v)$. Looking ahead, this constraint is used to guarantee that the 2-indegree bootstrapped graph $\mathsf{Ext}_{\delta,W}(\mathbb{G})$ built upon $\mathbb{G}$ is *predecessors-distinct.* We stress that practical DAG constructions usually contain a subpath that traverses all of the vertices, and thus are both first-predecessor-distinct and predecessors-distinct. More specifically, first-predecessor-distinctness holds as each non-source node v can pick her previous node in the subpath (that traverses all of the vertices) as their *first predecessor*; predecessors-distinctness holds as otherwise a cycle would exist.

2.6 Related Work

In this section, we give a discussion for the related work of memory-hard functions. First, we discuss the applications of memory hard functions; next, we discuss the line of theoretical/practical attacks towards memory hard functions candidates. Finally, we review different notions of crypto-primitives and cost metrics that are highly related to the memory hard functions we present in this thesis.

Applications. One of the most prominent applications of memory hard functions is password hashing [45]. In this scenario, instead of storing user passwords in the clear, the server stores hash digests of passwords, with the hope that it is hard to invert the hash back to passwords even if attackers steal the hash file. Morris and Thompson [46] presented the notion of *salt* and there are many designs of password hashing schemes afterward, including a DES-based design called crypt, and a design (by Poul-Henning Kamp) based on md5 hash called md5crypt. However, the rapid improvement of hardware drive down the cost of dictionary attacks towards crypt/md5crypt significantly, rendering old designs obsolete. To address the issue of ever-increasing hardware speeds, a scheme with a tunable hardness parameter called Bcrypt [6] was introduced. Intuitively, it iterates a hash function a certain number of times, which can increased accordingly as hardware speed improves. The framework also leads to the standard of PBKDF2. However, as recent development of special-purpose hardware (e.g. Appilication Specific Integrated Circuits (ASICs)), the advantage of iterated hashing scheme was downgraded as ASICs can exploit parallelism, pipelining and amortization to efficiently evaluate huge amounts of hash instances. To increase the cost of special-purpose passward cracking hardware, researchers observed that memory cost is relatively platform independent, and introduced many memory hard functions candidates [29, 13, 21, 28, 27, 26] which enforce the use of huge amounts of memory when evaluating the function. Besides password hashing, memory hard functions also show wide applications in proofs-of-work [9], proofs-of-space [47, 48], combatting spam emails [49, 50], resisting DDoS attacks, etc.

Attacks. For an ideal memory hard function with sequential time cost n, the optimal CMC cost lower bound would be $\Theta(n^2)$. However, for many practical memory hard functions heuristics, there exist both theoretical/practical strategies that evaluates the functions with far less cost. For example, Alwen and Blocki [12] presented a generic parallel attack strategy that evaluates any graph-based iMHFs with complexity $O(n^2/\log n)$. Attacks towards practical MHFs candidates were introduced as well [12, 30, 31], including Catena [21], Balloon hashing [13], and the winner of Password Hashing Competition (PHC) – Argon2i [28]. Blocki and Zhou [16] improved the $O(n^{1.8})$ attack towards argon2i [30] and presented an algorithm with complexity $O(n^{1.768})$. Recently, Blocki et al. [32] showed a simple sequential attack towards DRSample [17], and Alwen et al. [31] presented attacks on five data-independent memory hard functions (iMHFs) submitted to PHC, including Rig.v2 [25], TwoCats [24] and Gambit [23] (with attack complexity $O(n^{1.75})$), Pomelo [22] (with attack complexity $O(n^{1.83})$) and Lyra2 [27] (with attack complexity $O(n^{1.67})$).

Cost Metrics. There are various notions of egalitarian cost metrics with different pros and cons. The CMC metric we consider in this thesis was proposed by [10] who take into account the possibility of amortization and parallelism attacks. However, the measure is not perfectly precise for measuring egalitarian hardware cost as it allows space-time trade-off. In practice, this makes a huge difference as memory cost *grows superlinearly as opposed to scales linearly.* Ren and Devadas [51] proposed a notion called consistent memory hardness, which requires any *sequential* evaluation to uses space S' for at least T' steps, otherwise the algorithm must have run for a long time. Alwen, Blocki, and Pietrzak [19] strengthened consistent memory hardness into the *parallel* setting and introduced a new notion called Sustained Memory Complexity. Intuitively, it requires any *parallel* evaluation to uses space S' for at least T' steps.

Besides memory cost, other notions of egalitarion cost (e.g., bandwidth, cache misses, and energy cost) were also considered. Abadi et al.[52] observed that large number of cache misses can slow down the computation and proposed the notion of memory bound functions that incur many expensive cache misses. However, their construction requires a large random string as input and is thus only of theoretic interests. Recently, Ren and Devadas [53] refined the notion of memory bound functions to *bandwidth hardness.* In contrast to memory bound functions which only considers cache misses, bandwidth hardness also models the cost of *computation* as non-free, which leads to more practical constructions.

Chapter 3

Scrypt is Maximally Memory Hard

Outline of the Chapter. In this chapter, we show that a popular MHF candidate – Scrypt [33] – is optimally memory hard in the parallel random oracle model. We start with an introduction in Section 3.1, explaining the design and background of Scrypt. In Section 3.2, we highlight our main technical contribution, that is, the optimal memory hardness proof of Scrypt in the PROM. Next, as a warmup, we introduce (in Section 3.3) a multi-challenges pebbling game that is tightly related to the evaluation of Scrypt, and provide an optimal proof for its CC lower bound. In Section 3.4, before showing the full analysis in the parallel random oracle model (PROM), we provide a general single-challenge time lower bound for better understanding. Finally, we present the full memory hardness proof of Scrypt in Section 3.5.

The contents of this chapter is the result of a paper co-authored with Joël Alwen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. The paper merged the results of Leonid Reyzin, Joël Alwen, and Krzysztof Pietrzak, and the author's work with Stefano Tessaro. Thus the paper was jointly written by all of the authors. Some of the contents of Chapter 3 are taken from this joint write up (with permission of co-authors). The full version of this work is available on [15, 54].

3.1 Introduction

In this section, we roughly review the design and background of Scrypt. We start by describing a core component of Scrypt called ROMix function, and then show several applications of Scrypt. At the end of the section, we explain at a high level the memory hardness intuition/proof challenges of Scrypt, and survey the previous work.

The Scrypt MHF. We study a component of Scrypt function called ROMix (as defined in [33]), which is the core that makes Scrypt memory hard. For simplicity of notation, we will call ROMix as $\texttt{scrypt}^{\mathsf{ro}}$ function throughout the thesis. Let $\mathsf{ro} : \{0,1\}^W \to \{0,1\}^W$ denote as a hash primitive, $\texttt{scrypt}^{\mathsf{ro}}$ on input $x \in \{0,1\}^W$ and parameter $N \in \mathbb{N}$ computes values $X_0, X_1, \ldots, X_{N-1}, S_0, \ldots, S_N$ and outputs S_N, where

- $X_0 = x$ and for $i = 1, \ldots, N-1 \ : \ X_i = \mathsf{ro}(X_{i-1})$
- $S_0 = \mathsf{ro}(X_{N-1})$ and for $i = 1, \ldots, n \ : \ S_i = \mathsf{ro}(S_{i-1} \oplus X_{S_{i-1} \bmod N})$

For simplicity of explanation, we also define intermediate variables $T_0, \ldots, T_N$ with $T_0 = X_{N-1}$ and $T_i = S_{i-1} \oplus X_{S_{i-1} \bmod N}$ for $1 \le i \le N$, so that $S_i = \mathsf{ro}(T_i)$. For simplicity, we assume that the input string is always with length of W-bit. For the more general case where x has arbitrary length, we can apply a hash function on x and let the hash output be the input of $\texttt{scrypt}$.

The Naïve Algorithms. We illustrate two simple *sequential* algorithms for evaluating $\texttt{scrypt}$. The first algorithm is fast but memory-intensive: In the first phase, the algorithm on input $X_0 = x$ evaluates $X_i = \mathsf{ro}(X_{i-1})$ $(1 \le i \le N)$ iteratively and stores all of the N values in memory. During the second phase, the algorithm computes S_i values $(1 \le i \le N)$ step by step: It first extracts $X_{S_{i-1} \bmod N}$ from the memory and computes $T_i = S_{i-1} \oplus X_{S_{i-1} \bmod N}$, then it evaluates S_i by applying the hash function ro on T_i.

After computing S_i, it releases memory and forgets the value of S_{i-1}. The space-time complexity of the algorithm is $2N \times NW = 2N^2W$ and the CMC is $1.5N^2W$.

The second algorithm is memory-less but takes much more time. For every i ($1 \leq i \leq N$), the algorithm recomputes the value $X_{S_{i-1} \bmod N}$ from the input, and evaluates $S_i = \mathsf{ro}(S_{i-1} \oplus X_{S_{i-1} \bmod N})$. The algorithm uses approximately W bits of memory; the expected time complexity is $N^2/2$. Thus the space-time complexity (as well as the CMC) is $N^2W/2$.

Applications. One of the main reasons that we study Scrypt in this chapter is its widespread popularity: it is used in several proofs-of-work schemes for cryptocurrencies (most notably Litecoin [34], but also Tenebrix or Dogecoin), was published by IETF as RFC 7914 [35], and has inspired the design of one of the Password-hashing Competition's [36] winners, Argon2d [37].

Memory Hardness: Intuition and Proof Challenges. Intuitively, Scrypt is memory-hard because of the following. Let $X_0, \ldots, X_{N-1}$ be the X-labels computed in the first phase. We can alternatively consider them as nodes labels for an N-vertices line graph. In the second phase, to compute S_{i+1}, an algorithm needs to extract the label $X_{S_i \bmod N}$, where ($S_i \bmod N$) is a pseudorandom index that is unpredictable until S_i is computed. Suppose the algorithm stores p (out of N) nodes labels before knowing S_i, in expectation, it will take at least $N/(2p)$ steps to compute a random node label from a node label that has been stored. This leads to a total memory×time cost $p \cdot N/(2p) = N/2$. Since there are N challenged value S_i to be computed, the CMC of the strategy will have CMC of $W \cdot N \cdot N/2 = \frac{1}{2}N^2W$.

Transforming the above intuition into a formal proof, however, is much harder. There are several main challenges. First, the memory state of an algorithm is not necessarily a collection of nodes labels, but can be arbitrary information. Surprisingly, approaches that decreases CMC by storing information other than just labels have been constructed in [11]. Second, the memory consumption of an algorithm can vary over time. In particular, an algorithm is not required to keep all p labels after knowing the index $S_i \bmod N$, but might decrease CMC by deleting labels and recomputing afterward. In fact, [10] showed that if one is given the indices $S_i \bmod N$ in advance, an algorithm exists which evaluates $\texttt{scrypt}^{\mathsf{ro}}$ with CMC only $O(W \cdot N^{1.5})$. Third, in the second phase of Scrypt, the indices ($S_i \bmod N$) are from the hash function ro, instead of uniformly and independently generated. It is harder to argue that an algorithm cannot obtain more information of future challenges by querying the hash function.

Previous Work. Percival's original paper [33] proposed an analysis of `scrypt` targeting the space-time complexity instead of CMC. As pointed out by [15], however, the analysis is incorrect. Alwen, Chen et al. [11] initiated the study of proving CMC lower bound of `scrypt`. They lower bound the CMC of `scrypt` by $\Omega(W \cdot N^2/\log^2 N)$, but in a restricted computational model.

3.2 Results Overview

In this section, we highlight our contributions for proving memory hardness of Scrypt. Our main result [15] is the *first* unconditional lower bound on CMC for Scrypt in the parallel random oracle model. The $\Omega(W \cdot N^2)$ bound we obtain is *optimal* in the parallel random oracle model as well. To appreciate the novelty of our results, along the road, we also obtain a CC lower bound for a challenge pebbling game that is tightly related to the evaluation of Scrypt.

Optimal Lower Bound on the Pebbling Complexity. In Section 3.3, we prove an optimal $\Omega(N^2)$ lower bound on the parallel cumulative pebbling complexity for a challenge game which abstracts the evaluation of `scrypt`: we consider a line graph with N vertices. An adversary's goal is to pebble N uniformly chosen challenge nodes on this graph, where the ith challenge is only revealed once the node of challenge $i-1$ is pebbled. In [15, 54] we introduce a new technique for analyzing cumulative complexity: Let t_i be the time spent for answering the ith challenge. we lower bound the cumulative pebbling cost of the $(i-1)$*-th challenge phase* as a function of t_i and t_{i-1}, and prove that the sum of the costs over N challenge phases is $\Omega(N^2)$. Similar as in [11], our proof relies on a generalization of the fact that given a configuration with p pebbles, and a random challenge, an adversary needs at least (approximately) $t = n/p$ steps to pebble the challenge (with probability at least $\frac{1}{2}$).

Optimal Memory Hardness of Scrypt. For the setting of *graph-based data independent MHFs* [10], there is an elegant ex-post-facto argument that transforms a lower bound on the cumulative complexity for the parallel pebbling game into a lower bound on CMC. However, `scrypt` is a *data-dependent* MHF and there is no result showing that the pebbling lower bound for the challenge pebbling game can lead to a lower bound on CMC for general adversaries.

Surprisingly, by extending our proof for the challenge pebbling game into the parallel random oracle model, we prove an *optimal* memory hardness of `scrypt` function in Section 3.5. More specifically, we have the following theorem.

Theorem 3.1 (Memory-hardness of `Scrypt`, main theorem) *For any $X \in \{0,1\}^W$ and $N \geq 2$, if $A^{\mathsf{ro}}(X,n)$ outputs $S_n = \mathtt{scrypt}^{\mathsf{ro}}(X,n)$ with probability χ, where the probability is taken over the choice of the random oracle* ro*, then with probability (over the* *choice of* ro*) at least* $\chi - .08N^6 \cdot 2^{-W} - 2^{-N/20}$,

$$\mathsf{CMC}(A^{\mathsf{ro}}(X)) > \frac{1}{25} \cdot N^2 \cdot (W - 4\log N) \ .$$

As mentioned before in memory-hardness proof challenges, we solve three major issues that i) memory state can be arbitrary; ii) memory consumption can vary; and iii) challenge indices are from the hash primitive instead of being independent and uniformly random.

3.3 Optimal Cumulative Complexity in the Pebbling Game

In this section, we prove a lower bound on the cumulative complexity of a challenge pebbling game (see definition in Section 2.4) that is tightly related to the evaluation of Scrypt. The pebbling game is played on a line graph with N vertices and takes m challenge phases. In Section 3.3.1, we demonstrate a lower bound on the time-space tradeoff for answering a single challenge. In Section 3.3.2, we exploit the single-round time-space tradeoff and obtain a $\Omega(mN)$ CC lower bound for the pebbling game with m challenges. The techniques in this section will be useful in the full proof of our main result in Section 3.5.

The (m, N)**-Challenge Pebbling Game.** Let $\mathbb{G}$ be a line graph with N vertices, indexed by[1] $\{0, \ldots, N-1\}$. We consider a challenge pebbling game on $\mathbb{G}$. Recall that in the *challenge* pebbling game (Section 2.4), initially at time $s_1 = 0$, the player receives a challenge c_1 sampled *uniformly* from the set $\{0, \ldots, N-1\}$. After answering this challenge

[1]We slightly abuse the notation and use $i \in \{0, \ldots, N-1\}$ to indicate the i-th vertex itself.

at the earliest step $s_2 > s_1$ (where the pebbling configuration P_{s_2-1} contains the vertex c_1). The player will receive the next *independent* and *uniformly random* challenge c_2, and so on, for m challenges, until challenge c_m is answered at time s_{m+1}. Our goal is to provide a lower bound on the (expected) cumulative complexity $\mathsf{cc}(\mathsf{P}(\mathbf{c})) = \sum_{t=0}^{s_{m+1}} |\mathsf{P}_t|$ where $\mathbf{c} = (c_1, \ldots, c_m)$. Also note that in the challenge pebbling game, we assume that $|\mathsf{P}_t| \geq 1$ for every step t.

3.3.1 Time-Space Tradeoff for a Single Challenge

Given an initial pebbling configuration P_0, we say that a player answers a challenge c ($0 \leq c < N$) in t steps if $t > 0$ is the earliest step such that $c \in \mathsf{P}_{t-1}$. It is easy to see that t is at least one plus the distance between c and the closest pebble (in P_0) preceding it. For any vertex v in P_0, there are at most $N/(2|\mathsf{P}_0|)$ challenges within distance $N/(2|\mathsf{P}_0|) - 1$ to v. Hence we have the following fact.

Fact 3.1

$$\Pr_c \left[t > \frac{N}{2|\mathsf{P}_0|} \right] \geq \frac{1}{2} \,.$$

3.3.2 CC of the (m, N)-Challenge Pebbling Game

In this section we show that the cumulative complexity for playing the (m, N)-challenge pebbling game is $\Omega(mN)$ with high probability. Note that Fact 3.1 only addresses the number of pebbles used *right before* a challenge is revealed. However, the player can adaptively vary the number of pebbles used throughout the game, and there is no trivial space lower bound guarantee for other steps. Fortunately, we will still be able to show the following.

Theorem 3.2 (Cumulative pebbling complexity of challenge pebbling game) *Fix any challenge pebbling strategy, the cumulative pebbling complexity of the* (m, N)*-challenge* *pebbling game is with high probability $\Omega(mN)$. More precisely, suppose the strategy has at least one pebble at each step. Then for any $\epsilon > 0$, with probability at least $1 - e^{-2\epsilon^2 m}$ over the choice of the m challenges $\mathbf{c} = (c_1, \ldots, c_m)$,*

$$\mathsf{cc}(\mathsf{P}(\mathbf{c})) \geq 1 + \frac{N}{2} \cdot m \cdot \left(\frac{1}{2} - \epsilon\right) \cdot \ln 2\,,$$

where $\mathsf{P}(\mathbf{c})$ is the pebbing sequence of the strategy given the challenges $\mathbf{c}$.[2]

Alternatively speaking, denote as $\mathbb{G}_N$ the line graph with N vertices and $\mathcal{U}_N$ the uniform distribution over $\{0, \ldots, N-1\}$. For any $\epsilon > 0$, we have

$$\mathsf{cc}(m, \mathbb{G}_N, \mathcal{U}_N) \geq \left(1 - e^{-2\epsilon^2 m}\right) \cdot \left(1 + \frac{N}{2} \cdot m \cdot \left(\frac{1}{2} - \epsilon\right) \cdot \ln 2\right)\,,$$

where $\mathsf{cc}(m, \mathbb{G}_N, \mathcal{U}_N)$ is the expectation of cumulative complexity. (See Definition 2.4.)

Proof: Recall time starts at 0, we denote as $|\mathsf{P}_t|$ the number of pebbles in pebbling configuration P_t at time t. Let s_i be the time moment when the ith ($1 \leq i \leq m$) challenge is issued, and t_i is defined as the amount of time needed to answer the ith challenge. More precisely, we have $s_1 = 0$, $s_{i+1} = s_i + t_i$; and we let $s_{m+1} = s_m + t_m$. We use $\mathsf{cc}(t_1, t_2)$ to denote $\sum_{t=t_1}^{t_2} |\mathsf{P}_t|$.

Proof Intuition. Note that Fact 3.1 only enables us to argue that the number of pebbles (*immediately before* the ith challenge) is inversely proportional to t_i. However, since a player can adaptively vary the number of pebbles used throughout the game, we have no guarantee for the following steps in challenge phase i. Fortunately, we overcome the difficulty by considering the number of pebbles at the step not only immediately before the challenge, but also j steps earlier (for $j = 1, 2, \ldots$).

[2]The corresponding theorem in full version [15, 54] is more general in that it replaces $\ln 2$ with a function that depends on the number of pebbles in the initial/minimal pebbling configuration. However, we stick with the simpler version of the theorem in this thesis for ease of explanation.

Warm-up: Assuming Deterministic Time-space Tradeoff. For better understanding, we first assume a stronger time-space tradeoff (than in Fact 3.1). That is, to answer the next challenge in time t, for the step right before the challenge is revealed, the size of the memory state should be at least $N/(2t)$. Moreover, we can apply this assumption to *every step before* the challenge is issued, that is, suppose s is the step when the challenge is issued, for every $j \geq 0$, the number of pebbles at time $s - j$ is at least $N'/(j + t)$ (where $N' = N/2$). (This holds because the challenge was answered in $j + t$ steps starting from time $s - j$, and even if the challenge were issued earlier at time $s - j$, $N'/(j + t)$ pebbles is still needed by time-space tradeoff assumption.)

Given the assumption above, for each challenge $i \geq 2$, the cumulative complexity during round $i - 1$ is at least

$$\begin{aligned}\mathsf{cc}(s_{i-1}+1, s_i) &\geq \sum_{j=0}^{t_{i-1}-1} |\mathsf{P}_{s_i-j}| \geq N' \left(\frac{1}{t_i} + \frac{1}{t_i+1} + \cdots + \frac{1}{t_i + t_{i-1} - 1}\right) \\ &\geq N' \int_{t_i}^{t_{i-1}+t_i} \frac{dx}{x} = N'(\ln(t_{i-1}+t_i) - \ln t_i)\,.\end{aligned}$$

Hence by adding up the bound for each round i $(2 \leq i \leq m)$, we obtain the cumulative complexity

$$\mathsf{cc}(1, s_{m+1}) \geq N' \sum_{i=2}^{m} (\ln(t_{i-1}+t_i) - \ln t_i)\,.$$

The term reaches the minimum when all t_i are equal (as we will show below), which becomes $N'(m-1) \cdot \ln 2$.

Back to Fact 3.1. The actual proof is harder as the Fact 3.1 is probabilistic instead of deterministic. Moreover, the time-space tradeoff does not provide a lower bound on the number of pebbles in terms of running time, but the opposite direction (we cannot talk probabilistically of the number of pebbles, which is already determined before revealing the challenge). We address the issue by checking the pebbling size for all steps before s_i, and find the one that gives us the best lower bound on t_i.

Hard Challenges. For each time moment $t \leq s_i$, we can apply Fact 3.1 to the pebbling size $|\mathsf{P}_t|$ (as the ith challenge is revealed after determining the pebbling configuration P_t). Hence with probability at least 1/2 over the choice of challenge i, we have $t_i + (s_i - t) > N'/|\mathsf{P}_t|$, that is, we can have a lower bound on t_i, which is $t_i > N'/|\mathsf{P}_t| - (s_i - t)$. We define r_i to be the moment that leads to the best lower bound on t_i:

$$r_i = \operatorname*{argmax}_{0 \leq t \leq s_i} \left(\frac{N'}{|\mathsf{P}_t|} - (s_i - t) \right) .$$

We call that the ith challenge is "hard" if $t_i > N'/|\mathsf{P}_{r_i}| - (s_i - r_i)$. We claim that if challenge i is hard, then for every step before the challenge, we can obtain a lower bound on the number of pebbles just as in the the warm-up proof.

Claim 3.1 *If challenge i is hard, then for any j, $0 \leq j \leq s_i$, $|\mathsf{P}_{s_i - j}| > N'/(t_i + j)$.*

Proof: For any j, let $t = s_i - j$. We have $N'/|\mathsf{P}_{s_i-j}| - j = N'/|\mathsf{P}_t| - (s_i - t) \leq N'/|\mathsf{P}_{r_i}| - (s_i - r_i)$ by the choice of r_i. This value is less than t_i by definition of a hard challenge. Hence $N'/|\mathsf{P}_{s_i-j}| - j < t_i$ and the claim holds. ■

Number of Hard Challenges. Next we claim that with high probability, the number of hard challenges is high.

Claim 3.2 *For any $\epsilon > 0$, with probability at least $1 - e^{-2\epsilon^2 m}$ (oveer the choices of challenges), the number of hard challenges is at least $H \geq m(1/2 - \epsilon)$.*

Proof: Intuitively, the idea is to apply Hoeffding's inequality [55] as challenges are independent. However, the hardness of challenges is not independent. Fortunately, we can argue that it can only be "worse than independent". More precisely, for any fixed choices of the first $i-1$ challenges $c_1, \ldots, c_{i-1}$, we can run the adversary up to time s_i. At this moment, r_i is well defined and we can apply Fact 3.1 to r_i to obtain that $\Pr[c_i \text{ is hard } | \, c_1, \ldots, c_{i-1}] \geq 1/2$. This inequality allows us to apply a generalized version of Hoeffding's inequality stated in Claim 3.3 (setting $V_i = 1$ if c_i is hard and $V_i = 0$ otherwise) to get the desired result. ■

Claim 3.3 (Generalized Hoeffding's inequality) *If $V_1, V_2, \ldots, V_m$ are binary random variables such that for any i ($0 \leq i < m$) and any values of $v_1, v_2, \ldots, v_i$, $\Pr[V_{i+1} = 1 \,|\, V_1 = v_1, \ldots, V_i = v_i] \geq \rho$, then for any $\epsilon > 0$, with probability at least $1 - e^{-2\epsilon^2 m}$, $\sum_{i=1}^{m} V_i \geq m(\rho - \epsilon)$.*

Proof: For $0 \leq i < m$, define the binary random variable F_{i+1} as follows: for any fixing of $v_1, \ldots, v_i$ such that $\Pr[V_1 = v_1, \ldots, V_i = v_i] > 0$, let $F_{i+1} = 1$ with probability $\rho / \Pr[V_{i+1} = 1 \,|\, V_1 = v_1, \ldots, V_i = v_i]$ and 0 otherwise, independently of $V_{i+1}, \ldots, V_m$. Let $W_{i+1} = V_{i+1} \cdot F_{i+1}$. Note that $\Pr[W_{i+1} = 1] = \rho$ regardless of the values of $V_1, \ldots, V_i$, and thus W_{i+1} is independent of $V_1, \ldots, V_i$. Since $F_1, \ldots, F_i$ are correlated only with $V_1, \ldots, V_i$, we have that W_{i+1} is independent of $(V_1, \ldots, V_i, F_1, \ldots, F_i)$, and thus independent of $W_1, \ldots, W_i$. Therefore, $W_1, \ldots, W_m$ are mutually independent (this standard fact can be shown by induction on the number of variables), and thus $\sum_{i=1}^{m} V_i \geq \sum_{i=1}^{m} W_i \geq m(\rho - \epsilon)$ with probability at least $1 - e^{-2\epsilon^2 m}$ by Hoeffding's inequality. ■

Lower Bound on the Sum of $|\mathsf{P}_i|$. Assume H challenges are hard (and thus satisfy Claim 3.1), next we show a lower bound on the sum of $|\mathsf{P}_i|$.

Claim 3.4 *Fix real value N' and integers $t_1, \ldots, t_m$. Define $s_1 = 0$, and $s_i = s_{i-1} + t_{i-1}$ for $i = 2, \ldots, m+1$. Denote as $|\mathsf{P}_0| = 1, \ldots, |\mathsf{P}_m|$ a sequence of real values such that*

$|\mathsf{P}_t| \geq 1$ *for every* $t \geq 1$*. Suppose that there exist at least* H *distinct indices* i $(1 \leq i \leq m)$ *(called "hard indices") such that for any* $0 \leq j \leq s_i$*,* $|\mathsf{P}_{s_i - j}| \geq N'/(t_i + j)$*. Then we have*

$$\sum_{i=1}^{s_{m+1}} |\mathsf{P}_i| \geq N' \cdot H \cdot \ln 2 \,.$$

Proof: Let $i_1 < i_2 < \cdots < i_H$ be the hard indices. Recall the notation $\mathsf{cc}(i,j) = \sum_{t=i}^{j} |\mathsf{P}_t|$. Then for $k \geq 2$,

$$\begin{aligned}\mathsf{cc}(s_{i_{k-1}}+1, s_{i_k}) \geq \mathsf{cc}(s_{i_k} - t_{i_{k-1}} + 1, s_{i_k}) &= \sum_{j=0}^{t_{i_{k-1}}-1} |\mathsf{P}_{s_{i_k}-j}| \\ &\geq \sum_{j=0}^{t_{i_{k-1}}-1} \frac{N'}{t_{i_k}+j} \geq N' \cdot (\ln(t_{i_{k-1}} + t_{i_k}) - \ln t_{i_k}) \,,\end{aligned}$$

Similar as in the warm-up proof, we try to provide a lower bound on $\mathsf{cc}(1, m+1)$ by adding up the pebbling lower bound above for each k and find the minimum over all choices of t_{i_k}. However, the term will decrease if t_{i_1} decreases and t_{i_H} increases, for which we have no bounds. We address the issue by providing tailored lower bound for special cases of $k = 2$ and $k = H + 1$.

For $k = 2$, by noticing that $s_{i_2} \geq t_{i_1} + s_{i_1} \geq N'$ (where the second step follows by Claim 3.1 with $j = s_{i_1}$), we can bound $\mathsf{cc}(1, s_{i_2})$ as follows.

$$\begin{aligned}\mathsf{cc}(1, s_{i_2}) &\geq \sum_{j=0}^{s_{i_2}-1} |\mathsf{P}_{s_{i_2}-j}| \geq N' \sum_{j=0}^{s_{i_2}-1} \frac{1}{t_{i_2}+j} \geq N' \int_{t_{i_2}}^{s_{i_2}+t_{i_2}} \frac{dx}{x} \\ &= N'(\ln(s_{i_2} + t_{i_2}) - \ln t_{i_2}) \geq N' \cdot (\ln(N' + t_{i_2}) - \ln t_{i_2}) \,.\end{aligned}$$

For $k = H + 1$,

$$\begin{aligned}\mathsf{cc}(s_{i_H}+1, s_{H+1}) \geq t_{i_H} \geq N' \cdot \left(\frac{1}{N'} t_{i_H}\right) \geq N' \cdot \left(\frac{1}{N'} + \cdots + \frac{1}{N' + t_{i_H} - 1}\right) \\ \geq N' \cdot \left(\ln(t_{i_h} + N') - \ln N'\right).\end{aligned}$$

Adding these up, we get

$$\begin{aligned}\mathsf{cc}(0, s_{i+1}) &= |\mathsf{P}_0| + \mathsf{cc}(1, s_{i_2}) + \mathsf{cc}(s_{i_2}+1, s_{i_3}) + \cdots + \mathsf{cc}(s_{i_{H-1}}+1, s_{i_H}) + \mathsf{cc}(s_{i_H}+1, s_{i_{H+1}}) \\ &\geq |\mathsf{P}_0| + c \cdot \sum_{i=1}^{H} \left(\ln(x_i + x_{i+1}) - \ln x_{i+1}\right),\end{aligned}$$

where $x_1 = N'$, $x_2 = t_{i_2}$, $x_3 = t_{i_3}$, $\ldots$, $x_H = t_{i_H}$, and $x_{H+1} = N'$. For each i, the first derivative with respective to x_i is $\frac{1}{x_i+x_{i-1}} + \frac{1}{x_i+x_{i+1}} - \frac{1}{x_i}$. Assumming all the x_is are positive, the function reaches its minimum when each x_i $(2 \leq i \leq H)$ is equal to $\sqrt{x_{i-1}x_{i+1}}$, or equivalently $x_i = N'$. This setting of x_i thus gives us the desired result. ■

Plugging in $m \cdot (1/2 - \epsilon)$ for H, the Theorem 3.2 holds true. ■

3.4 Single-Challenge Time Lower Bound in the PROM

Before proving our main result, to highlight one of our technical contributions, in this section, we extend the Fact 3.1 to the parallel random oracle model and provide a more general single-challenge time lower bound.

Notation. Fix parameters N and W, input $X \in \{0,1\}^W$. For a function $\mathsf{ro} : \{0,1\}^W \to \{0,1\}^W$, define $X_i = \mathsf{ro}^i(X)$. Let A be any oracle algorithm (in the parallel random oracle model as defined in Section 2.2) that on any input and oracle makes at most $\mathsf{q} - 1$ total queries to its oracle. Suppose $A^{\mathsf{ro}}(X, j)$ starts on input state σ_0 with the goal of eventually querying X_j to ro. Let t_j be the earliest round in which $A^{\mathsf{ro}}(X, j)$ queries X_j to ro (with $t_j = \infty$ if never).

We show that A (with an M-bit initial state) cannot do much better than the following pebbling strategy: Initially placing $p \approx M/W$ equidistant pebbles. Given a random challenge on the line graph, the strategy pebbles the challenge from the closest pebble preceding it. This leads to the following theorem.

Theorem 3.3 (Single-Challenge Time Lower Bound) *Assume the adversary A makes no more than $\mathsf{q} - 1$ RO queries. Fix any memory size M and let $p = \lceil (M+1)/(W - 2\log N - \log \mathsf{q}) + 1 \rceil$. With probability at least $1 - \epsilon_{\mathsf{bad}} = 1 - (\mathsf{q}+1)N^2 2^{-W}$ (over the uniform choice of random oracles and A's internal coins) the following holds: For every input state σ_0 of length at most M bits, we have*

$$\Pr_{j \leftarrow \{0,\dots,N-1\}} \left[t_j > \frac{N}{2p} \right] \geq \frac{1}{2}\,, \tag{3.1}$$

where the probability is taken over the challenge j.

First, without loss of generality, we can fix A to be deterministic, as otherwise we can fix the coins r to be the optimal randomness that minimizes the probability (over ro) such that the above property (i.e. Inequality 3.1) holds.

Before the formal proof, similar as in [10], we define a useful notion called *ex-post-facto initial pebbling.*

Ex-post-facto initial pebbling. Fix input X, random oracle $\mathsf{ro} \in \mathbb{RO}$ and initial state σ_0. For every challenge $j \in \{0, \dots, N-1\}$, we consider the first t_j rounds of the execution $A(X, j)$: At round k ($1 \leq k \leq t_j$), A takes as an input state containing oracle answers $\mathsf{ro}(\mathbf{q}_{k-1})$ (except for $k = 1$, when A reads σ_0). Then A performs arbitrary computation, and generates an output state containing oracle queries $\mathbf{q}_k$. We denote as trace $A(X, j)$ the sequence of input/output states in the execution.

Consider running $A(X)$ on every challenge j $(0 \leq j \leq N-1)$ in parallel, one round at a time. For every node index $i \in \{0, \ldots, N-1\}$, if the *first* appearance of X_i is in some round of trace $A(X, j)$ $(0 \leq j \leq N-1)$, we define $\mathsf{bestchal}_i = j$ as the best challenge of i, and we define the best position β_i as follows: if the first appearance of X_i is a query to ro in round k, then we assign $\beta_i = k$ (meaning that X_i is revealed at round k without recomputing first). If the first appearance of X_i is an oracle answer from ro to query X_{i-1} made at round k, then we assign $\beta_i = k + 1/2$ (meaning that X_i is revealed at round k by recomputing). In all other cases, we let $\beta_i = \infty$. We define *ex-post-facto initial pebbling* $B \subseteq \{0, \ldots, N-1\}$ as the set of indices i where β_i is a finite integer, that means, when running $A(X)$ on every challenge j $(0 \leq j \leq N-1)$ in parallel, X_i is used without recomputing first. In the following proof, we will show that the size of B cannot be much larger than the size of the initial state σ_0.

Proof: [of Theorem 3.3] Fix input $X = X_0$ and (deterministic) algorithm A, the proof consists of the following steps.

Step 1: Time lower bound on challenge-answering. First we show that given the ex-post-facto initial pebbling B, for any challenge j, the number of rounds needed to answer the challenge is at least the distance from j to the closest pebble in B preceding it.

Claim 3.5 *Fix input X, algorithm A, initial state σ_0 and random oracle $\mathsf{ro} \in \mathbb{RO}$. Let B denote as the ex-post-facto initial pebbling. For every j, $0 \leq j < N$, let t_j be the earliest round in $A(X, j)$ where the challenge j is answered. We have $t_j \geq 1 + j - k$, where $k = max\{a \leq j \mid a \in B\}$.*

Proof: For those j in the set B, the claims holds because $t_j \geq 1$ as we start from round 1. Thus we only consider j not in B in the following context. We note that it is sufficient to show that $\lceil \beta_j - \beta_k \rceil \geq j - k$, where β_j is the first step that X_j appears in some trace when running $A(X)$ on all possible challenges in parallel. This is because $\lceil \beta_j \rceil \geq j - k + 1$ as $\beta_k \geq 1$. Hence we have $t_j \geq \lceil \beta_j \rceil \geq j - k + 1$, which finishes the proof.

Next we show that $\lceil \beta_j - \beta_k \rceil \geq j - k$. This holds true because $\beta_{k+1} \geq \beta_k + 1/2$ and $\beta_\ell \geq \beta_{\ell-1} + 1$ for every ℓ between $k + 2$ and j. The first argument holds because X_k is the query input when X_{k+1} is revealed as an RO answer in $A(X, \mathsf{bestchal}_j)$; the second argument holds because $X_{\ell-1}$ is the query input when X_ℓ is revealed as an RO answer in $A(X, \mathsf{bestchal}_j)$ (thus $\beta_\ell > \beta_{\ell-1}$), and because neither of β_ℓ or $\beta_{\ell-1}$ are integers by definition of k. ■

Step 2: Label collisions. Next, we show that the labels $\{X_i\}_{i\in\{0,\ldots,N-1\}}$ are distinct with high probability. This property is useful for identifying index i from a label X_i. More importantly, looking ahead, it guarantees the existence of a predictor that (on input a small hint including the initial state σ_0) can extract (approximately) $|B|$ random oracle entries, where B is the ex-post-facto initial pebbling.

Claim 3.6 *Fix input $X = X_0$. With probability at least $1 - \epsilon_{\mathsf{coll}} = 1 - N^2 2^{-W-1}$ (over the uniform choice of RO), the labels $\{X_i\}_{i\in\{0,\ldots,N-1\}}$ are distinct.*

Proof: Fix input $X = X_0$, let E_{coll} be the event where there is a X-label collision. We show a predictor P that when E_{coll} happens, there exists a small hint $h = (u, v)$ (where the size of the hint space is at most $N^2/2$), such that P on input h predicts a random oracle entry. Thus by Lemma 2.1, we have $\Pr[E_{\mathsf{coll}}] \leq N^2 2^{-W-1}$ and the claim holds.

The hint. If E_{coll} happens, the hint is (u, v) $(0 \leq u < v < N)$ where $X_u = X_v$ and v is the minimal index i such that X_i collides with some labels in $\{X_0, \ldots, X_{i-1}\}$.

The predictor P. P^{ro} (with hardwired X_0) parses the input into (u, v), computes $b = \mathsf{ro}^u(X_0)$ and $a = \mathsf{ro}^{v-1}(X_0)$, then P predicts (a, b) which satisfies $b = \mathsf{ro}(a)$. Moreover, P nevery query $\mathsf{ro}(a)$: if $v = 1$, then P predicts $\mathsf{ro}(X_0) = X_0$ without querying $\mathsf{ro}(X_0)$; if $v > 1$, P nevery query $\mathsf{ro}(a)$ because otherwise $v - 1$ would be the minimal index i such that X_i collides with some labels in $\{X_0, \ldots, X_{i-1}\}$. ■

Step 3: Pebbling reduction. Next we build the connection between the initial state size and the size of the ex-post-facto initial pebbling. In particular, we show that with high probability, the size of pebbling $|B|$ is at most $\approx M/W$, where M is the length of the initial input state and W is the length of labels.

Lemma 3.1 *Fix input X and algorithm A (that is deterministic and makes at most $\mathsf{q}-1$ RO queries). Fix any memory size M and let $p = \lceil (M + 1)/(W - 2\log N - \log \mathsf{q}) + 1 \rceil$. Define event $\mathrm{E}^M_{\mathsf{pred}}$ where the following conditions* all *hold:*

1. *The labels $\{X_0, \ldots, X_{N-1}\}$ are distinct.*

2. *There exists an input state σ_0 of length at most M bits, where the ex-post-facto initial pebbling B satisfies that $|B| > p$.*

We show that

$$\Pr\left[\mathrm{E}^M_{\mathsf{pred}}\right] \leq \epsilon_{\mathsf{pred}} = \mathsf{q}N^2 2^{-W} ,$$

where the probability is taken over the uniform choice of the random oracle.

Proof: We will show a predictor P (that has oracle access to random oracle ro), such that if $\mathrm{E}^M_{\mathsf{pred}}$ happens, denote as $|B| > p$ the size of the ex-post-facto initial pebbling B and let $B' = B \setminus \{0\}$, there will be a hint h with no more than $M + |B'| \cdot (2\log N + \log \mathsf{q}) < (|B'| - 1) \cdot W + 2\log N + \log \mathsf{q}$ bits, where $P(h)$ predicts $|B'|$ random oracle entries correctly. Thus by the compression argument that random oracles cannot be compressed (i.e., Lemma 2.1), $\mathrm{E}^{M}_{\mathsf{pred}}$ happens with probability no more than ϵ_{pred} and the lemma holds. Next we describe the hint h and the predictor P. (Without loss of generality we assume $|B'| - 1 = p$, otherwise we can renew B' as the first p elements of the original B', and the same argument below still holds.)

The hint. If $\mathrm{E}^{M}_{\mathsf{pred}}$ happens, the hint is the input state σ_0 and the following helper information. For each $i \in B' = B \setminus \{0\}$, we add into hint the index i, the best challenge $\mathsf{bestchal}_i$ and the $\log \mathsf{q}$-bit index q_i that identifies the first RO query in $A(X, \mathsf{bestchal}_i)$ where the query input is X_i. Note that the extra hint is with length at most $|B'| \cdot (2\log N + \log \mathsf{q})$ bits where $|B'| \geq p$.

The predictor. The predictor P hardwires the algorithm A, input $X = X_0$, and has oracle access to the random oracle. Given an input h, the predictor P parses h into a state σ_0, and a list of tuples $\{(i, \mathsf{bestchal}_i, q_i)\}$ where i is in the set B'. Then P runs $A(X)$ on every challenge in parallel, one round at a time, and attempts to predict $(X_{i-1}, X_i = \mathsf{ro}(X_{i-1}))$ for every $i \in B'$ without querying $\mathsf{ro}(X_{i-1})$. Since X-labels are distinct, the predictor attempts to predict exactly $|B'|$ random oracle entries.

To achieve the goal, P builds a table that keeps track of the X-labels X_i for every $i \geq 0$ (initializing $X_0 = X$). After each round k, P will obtain all the queries A makes for all the challenges in round k. Then P will update the table and answer the RO queries as input states for round $k+1$ by performing the following steps:

1. *Extracting labels:* For every index $i \in B'$, given the hint $(i, \mathsf{bestchal}_i, q_i)$, if the RO query corresponding to index q_i is in round k, then P updates X_i from the query input. (Note that queries and their positions in the table can easily be recognized

from the hint.)

2. *Answering RO queries for $i \in B'$:* For each RO query, if there exists $i \in B'$ such that X_i and X_{i-1} were updated, and X_{i-1} matches the query input, P will answer with value X_i *without querying the random oracle.*

3. *Answering other RO queries:* P sends remaining queries to ro, return the answers to A, and update any new entries in the table that can be updated in (i.e., for every query that matches X_{i-1} in the table for some updated position $i-1$, update position i with the answer to that query).

Once every X_i for $i \in B'$ is in the table, P queries ro to update the missing entries in the table, and outputs the prediction that $\mathsf{ro}(X_{i-1}) = X_i$ for $i \in B'$.

Correctness of the predictor. We first prove that P simulates ro correctly. This is achieved by showing that the table contains correct labels. We argue it by induction on the time order of updating. Initially, only the correct input X_0 is updated in the table. Assume all the labels in the table are correct up to now. A new label X_i enters the table either because it is in the set B' (and thus correct by the hint) or is obtained as an answer from ro to the query that P identified as X_{i-1} using the table (which is correct by inductive hypothesis). Note that this also implies that P will not output an incorrect prediction.

Next we show that P will never query to ro the value X_{i-1} for any $i \in B'$. We first show that for every node index $i \in B'$, the label X_i will be updated to the table at the *beginning* of round β_i; and for every $i \in \{0, \dots, N-1\} \setminus B'$, X_i will be updated to the table at the *end* of round $\lfloor \beta_i \rfloor$. We prove this by induction on i. The base case where $i = 0$ holds true as X_0 will be updated initially. If i is in B', this is also true because β_i is the *first step* where the label X_i appears in some trace, and P will update X_i at this step using the hint. For $i > 0$ that is not in B', on the one hand, X_i cannot be updated earlier than round $\lfloor \beta_i \rfloor$ as there was no query input/output that matches X_i; moreover, $\beta_{i-1} < \beta_i$ (because X_{i-1} appears as a query at round $\lfloor \beta_i \rfloor$). Hence by inductive hypothesis, before the end of round $\lfloor \beta_i \rfloor$, the table will already contain X_{i-1}, and thus position i will get updated when X_{i-1} gets queried to ro.

Therefore, for every $i \in B'$, when A queries X_{i-1} to ro in a round $k \geq \beta_i$, before sending the query to the random oracle, we argue that X_i and X_{i-1} were already updated and P can answer the query without asking the random oracle. In particular, the label X_i was already updated at the beginning of round $\beta_i \leq k$; if $i-1 \in B'$, X_{i-1} was updated at the beginning of round $\beta_{i-1} \leq k$ (as X_{i-1} is being queried in round k and thus $\beta_{i-1} \leq k$); if $i-1 \notin B'$, X_{i-1} was updated before the end of round $\lfloor \beta_{i-1} \rfloor < k$ (as β_{i-1} is not an integer and X_{i-1} is being queried in round k, and thus $\beta_{i-1} < k$). ■

Remark 3.1 *To simplify the explanation of the full proof in Section 3.5, we will denote* $\mathsf{predictable}$ *as the set of random oracles where there exists a hint h with no more than $(p-1) \cdot W + 2\log N + \log \mathsf{q}$ bits, such that $P(h)$ predicts p random oracle entries correctly. With the same argument above, we know that*

$$\Pr_{\mathsf{ro} \xleftarrow{\$} \mathbb{RO}} [\mathsf{ro} \in \mathsf{predictable}] \leq \epsilon_{\mathsf{pred}} .$$

Putting all things together. In summary, for every input state σ_0 of length at most M bits, by Lemma 3.1 and Claim 3.6, with probability at least $1 - (\mathsf{q}+1)N^2 2^{-W}$, the X-labels are distinct and the size of the ex-post-facto initial pebbling B is no more than p. Then by Claim 3.5, we have $\Pr[t_j > \frac{N}{2|B|}] \geq \frac{1}{2}$ and the theorem holds.

■

Remark 3.2 *To simplify the explanation of the full proof in Section 3.5, we will denote* $\mathsf{bad}_{\mathsf{ro}}$ *as the set* $\mathsf{predictable}$ *plus the random oracles where the X-labels collide. With the* *same argument above, we know that*

$$\Pr_{\mathsf{ro} \xleftarrow{\$} \mathbb{RO}} [\mathsf{ro} \in \mathsf{bad}_{\mathsf{ro}}] \leq \epsilon_{\mathsf{bad}} .$$

3.5 Main Result: Optimal Memory Hardness of `scrypt` in the PROM

In this section, by extending the techniques from previous two sections, we prove the optimal memory hardness of `scrypt` in the parallel random oracle model.

Theorem 3.4 *Fix positive integers $N \geq 2$ and W, and $X \in \{0,1\}^W$. Let A be any oracle algorithm (in the parallel random oracle model as defined in Section 2.2) with input X. Assume $A^{\mathsf{ro}}(X)$ outputs $S_N^{\mathsf{ro}} = \mathtt{scrypt}^{\mathsf{ro}}(X)$ correctly with probability χ, where the probability is taken over the uniform choice of the random oracle ro.*[3] *Then for any $\epsilon > 0$ and $\mathsf{q} \geq 2$, with probability (over the uniform choice ro) at least $\chi - 2\mathsf{q}N^4 \cdot 2^{-W} - e^{-2\epsilon^2 N}$ one of the following two statements holds: either $A^{\mathsf{ro}}(X)$ makes more than q queries (and thus $\mathsf{CMC}(A^{\mathsf{ro}_N}) > \mathsf{q}W$ by definition) or*

$$\mathsf{CMC}(A^{\mathsf{ro}}(X)) \geq \frac{\ln 2}{6} \cdot \left(\frac{1}{2} - \epsilon\right) \cdot N^2 \cdot (W - 2\log N - \log \mathsf{q} - 1) \ .$$

The rest of this section is devoted to the proof of this theorem.

3.5.1 Proof Outline

Before the formal proof, we highlight some challenges for extending Theorem 3.3 and Theorem 3.2 into the setting of computing `scrypt` in the parallel random oracle model.

[3]We assume that the adversary A is deterministic without loss of generality.

Technical obstacle: challenges are from the oracle. In the proof of Theorem 3.3, we are dealing with a *fixed* oracle ro and a *uniformly random* challenge. Moreover, the proof essentially relies on the ability to run every challenge for a given oracle, and the fact that each challenge is independent and uniformly random until being issued after answering the previous challenge. However, in the actual computation of `scrypt`, those pseudorandom challenge indices $\{S_{i-1} \bmod N\}$ were from the *fixed* random oracle. Even if we consider the oracle being lazily sampled, the challenges obtained from the random oracle ro are not independent once we condition that ro falls outside the bad set $\mathsf{bad}_{\mathsf{ro}}$ (which is needed in Theorem 3.3.)

Working with multiple random oracles. We solve the above issues by working with multiple carefully chosen random oracles. Fix input X, let A^{ro} denote as an algorithm running with oracle ro, we denote as $X_i^{\mathsf{ro}} = \mathsf{ro}^{(i)}(X)$ $(0 \leq i < N)$; $T_0^{\mathsf{ro}} = X_{N-1}^{\mathsf{ro}}$, $S_0^{\mathsf{ro}} = \mathsf{ro}(T_0)$, and for $i = 1, \ldots, N$, $T_i^{\mathsf{ro}} = S_{i-1}^{\mathsf{ro}} \oplus X^{\mathsf{ro}}_{S^{\mathsf{ro}}_{i-1} \bmod N}$ and $S_i^{\mathsf{ro}} = \mathsf{ro}(T_i^{\mathsf{ro}})$. (For notational simplicity, we will omit the superscript when the notation is clear in the context.)

Let changeModn(S, i) be the function that keeps the quotient $\lfloor S/N \rfloor$ but changes the remainder of S modulo N to i. We consider the following process of sampling random oracles. A uniformly random oracle ro_0 is chosen initially. Then uniformly random challenges $c_1, \ldots, c_N \in \{0, \ldots, N-1\}$ were sampled. We denote as ro_1 the oracle that is equal to ro_0 at every point, except that $\mathsf{ro}_1(T_0^{\mathsf{ro}_0}) = \text{changeModn}(S_0^{\mathsf{ro}_0}, c_1)$. Similarly, we let ro_2 be identical to ro_1 except that $\mathsf{ro}_2(T_1^{\mathsf{ro}_1}) = \text{changeModn}(S_1^{\mathsf{ro}_1}, c_2)$, and so on, until we sampled the last oracle ro_N. This process of sampling random oracles enables us to explicity embeds uniformly random challenges. Moreover, the distributions of the random oracles are close to uniform. By the same argument as in Theorem 3.3, unless some "bad" random oracles have been chosen, we can show that with high probability, it takes a long time to answer each challenge. Since the (embedded) challenges are independently sampled, we can bound the cumulative complexity as in Theorem 3.2.

Bounding bad behaviors. It remains to define "bad" choices of random oracles and bound their probabilities without affecting the independence of the (embedded) challenges. More details will be given in the following full proof.

3.5.2 The Formal Proof

Recall that we assume that the adversary A is deterministic without loss of generality. In particular, the randomness of the execution is only over the random oracle A is given access to.

Sampling Random Oracles. Following the proof outline above, we make a precise definition of ro_k. Before illustrating the sampling process of random oracles, we introduce the following definitions. Let changeModn(S, i) be a function that keeps the quotient $\lfloor S/N \rfloor$ but changes the remainder of S modulo N to i if possible: it views S as an integer in $[0, 2^W - 1]$, computes $S' = \lfloor S/N \rfloor \cdot N + i$, and outputs S' if $S' < 2^W$, and S otherwise.

Definition 3.1 *Define* $\mathsf{roundingProblem}_k$ *as the set of all random oracles* ro *such that the value of at least one of* $S_0^{\mathsf{ro}}, \dots, S_k^{\mathsf{ro}}$ *is greater than* $\lfloor 2^W/N \rfloor \cdot N - 1$ *(i.e., those for which* changeModn *does not work on some* S *value up to* S_k*).*

Definition 3.2 *Define* $\mathsf{colliding}_k^*$ *as the set of all* ro *where there is at least one collision among the values* $\{X_0, X_1^{\mathsf{ro}}, X_2^{\mathsf{ro}}, \dots, X_{N-2}^{\mathsf{ro}}, T_0^{\mathsf{ro}}, T_1^{\mathsf{ro}}, \dots, T_k^{\mathsf{ro}}\}$. *Let*

$$\mathsf{colliding}_k = \mathsf{roundingProblem}_{\min(k,N-1)} \cup \mathsf{colliding}_k^* \, .$$

Let ro_0 be a uniformly chosen random oracle. We define the sampling process of random oracles as follows.

Definition 3.3 *For every k $(0 \leq k < N)$, let $\mathsf{ro}_{k+1} = \mathsf{ro}_k$ if $\mathsf{ro}_k \in \mathsf{colliding}_k$; else, choose c_{k+1} uniformly at random between 0 and $N-1$, let $\mathsf{ro}_{k+1}(T_k^{\mathsf{ro}_k}) = \text{changeModn}(S_k^{\mathsf{ro}_k}, c_{k+1})$, and let $\mathsf{ro}_{k+1}(x) = \mathsf{ro}_k(x)$ for every $x \neq T_k^{\mathsf{ro}_k}$. (Recall that ro_0 is chosen uniformly.)*

Remark 3.3 *Note that this particular way of choosing ro_{k+1} is designed to ensure that it is uniform, as we will argue in Claim A.1. Besides, another reason that we define $\mathsf{colliding}_k$ for every k (instead of only $\mathsf{colliding}_0$) is to enable embedding (independent) challenges without changing the values of previous labels (i.e., X_i, T_i, and S_i).*

Extending Theorem 3.3: Single Challenge Space-time Tradeoff. In the argument of Theorem 3.3 (including the definition of ex-post-facto initial pebbling, as well as the proof of Lemma 3.1), the predictor simulates A with different challenges to A. Here, the predictor will run A with different oracles. Specifically, for every k $(1 \leq k \leq N)$ and every oracle $\mathsf{ro}_{k-1} \notin \mathsf{colliding}_{k-1}$, we consider the N oracles $\mathsf{ro}_{k,j}$ for each $0 \leq j < N$, defined to be the same as ro_{k-1}, except $\mathsf{ro}_{k,j}(T_{k-1}^{\mathsf{ro}_{k-1}}) = \text{changeModn}(S_{k-1}^{\mathsf{ro}_{k-1}}, j)$ (instead of $S_{k-1}^{\mathsf{ro}_{k-1}}$).

Since $\mathsf{ro}_{k-1} \notin \mathsf{colliding}_{k-1}$, $T_{k-1}^{\mathsf{ro}_{k-1}}$ is distinct to every element in $\{X_i^{\mathsf{ro}_{k-1}}\}_{i \in \{0,\ldots,N-1\}}$ and $\{T_i^{\mathsf{ro}_{k-1}}\}_{i \in \{0,\ldots,k-1\}}$. Therefore (since ro_{k-1} and $\mathsf{ro}_{k,j}$ differ only at the point $T_{k-1}^{\mathsf{ro}_{k-1}}$), we have $X_i^{\mathsf{ro}_{k-1}} = X_i^{\mathsf{ro}_{k,j}}$ for every $0 \leq i \leq n-1$ and $T_i^{\mathsf{ro}_{k-1}} = T_i^{\mathsf{ro}_{k,j}}$ for every $0 \leq i \leq k-1$. In particular, the execution of A with oracle $\mathsf{ro}_{k,j}$ (for any j) will all be identical to the execution of $A^{\mathsf{ro}_{k-1}}$ up to the step when the query T_{k-1} is first made. For notational simplicity, we omit the superscript on T_{k-1} for the remainder of this argument.

We observe that at the moment when T_{k-1} is queried, we can have a predictor which substitute different answers to this query and run A on these different answers in parallel, hence obtain a similar compression argument as in Lemma 3.1, and finally obtain a time/space tradeoff lower bound as in Theorem 3.3. In fact, we will prove a stronger result for any step *before* T_{k-1} is queried.

Next, for oracle $\mathsf{ro}_{k-1} \notin \mathsf{colliding}_{k-1}$, and any round r before T_{k-1} is queried, we will specify the ex-post-facto initial pebbling, the hint given to the predictor, as well as the predictor description.

Ex-post-facto initial pebbling. Consider the execution $\mathsf{A}^{\mathsf{ro}_{k-1}}$. Let $s_k > 0$ be the round in which T_{k-1} is first queried, that is, T_{k-1} appears in $\mathbf{q}_{s_k}$ as a query input. For any integer $r \leq s_k$, denote as $\bar{\sigma}_r$ the output state of $A^{\mathsf{ro}_{k-1}}$ from round r. From $\bar{\sigma}_r$, we consider n different subsequent execution. In particular, for each j ($0 \leq j < N$), we replace ro_{k-1} with $\mathsf{ro}_{k,j}$, and let $r + t_j > s_k$ be the first step such that the query $T_k^{\mathsf{ro}_{k,j}}$ is contained in $\mathbf{q}_{r+t_j}$. (Note that if the query is later than the q-th query in the execution starting from $\bar{\sigma}_r$, we will define $t_j = \infty$.) We can thus define β_i, $\mathsf{bestchal}_i$, ex-post-facto initial pebbling B, and the set $B' = B \setminus \{0\}$ the same way as in Theorem 3.3. The *slight differences* are as follows: First, we will count the number of rounds after round r (instead of from 0). Second, we will substitute "challenge j" with oracle $\mathsf{ro}_{k,j}$ and "query X_j" with "query X_j or $T_k^{\mathsf{ro}_{k,j}}$". Finally, we will stop the execution of $A^{\mathsf{ro}_{k,j}}(\bar{\sigma}_r)$ after q number of queries.

Time lower bound on challenge-answering. Similar as in Claim 3.5, we show that given ex-post-facto initial pebbling B, for any challenge j, the number of rounds needed to answer the challenge is at least the distance from j to the closest pebble in B preceding it.

Claim 3.7 *Fix any $\mathsf{ro}_{k-1} \notin \mathsf{colliding}_{k-1}$, and any round r before T_{k-1} is queried. Let B denote as the ex-post-facto initial pebbling. For every j, $0 \leq j < N$, let $t_j > 0$ be the smallest value such that such $r + t_j > s_k$ and the query $T_k^{\mathsf{ro}_{k,j}}$ is contained in $\mathbf{q}_{r+t_j}$ (if ever before query number $\mathsf{q}+1$). We have $t_j \geq 1 + j - k$, where $k = max\{a \leq j \mid a \in B\}$.*

Proof: The proof is the same as in Claim 3.5. ■

<u>*The hint.*</u> Fix any $\mathsf{ro}_{k-1} \notin \mathsf{colliding}_{k-1}$ and any round $r \le s_k$, consider the execution of $A^{\mathsf{ro}_{k-1}}$ starting from round r, which determines β_i, $\mathsf{bestchal}_i$, and the ex-post-facto initial pebbling B mentioned before. We index all the oracle queries that A makes across all rounds sequentially, and only consider the first q oracle queries that A makes, where q is a tunable parameter. We construct the hint as follows.

- The input state σ_r of $A^{\mathsf{ro}_{k-1}}$ for round $r+1$.
- The indices for all vertices in $B' = B \setminus \{0\}$, where B is the newly defined ex-post-facto initial pebbling.
- For every $i \in B'$, we also store the previously defined β_i and the best challenge $\mathsf{bestchal}_i$.
- The index (between 1 and q) for the first query where T_{k-1} appears in the input (across the execution starting from round r). This index is used for knowing when to reply with $S_{k-1}^{\mathsf{ro}_{k,j}} = \text{changeModn}(S_{k-1}^{\mathsf{ro}_{k-1}}, j)$ instead of $S_{k-1}^{\mathsf{ro}_{k-1}}$ itself.
- If $s_k = r$, we need to slightly modify the input state σ_r. In particular, we need to replace the label $S_{k-1}^{\mathsf{ro}_{k-1}}$ in σ_{s_k} with T_{k-1}.
- Finally, if $\mathsf{bestchal}_j = j$ for some $j \in B'$, we need one additional bit of hint, indicating whether X_j is first queried by itself or as part of the query $T_k^{\mathsf{ro}_{k,j}} = S_{k-1}^{\mathsf{ro}_{k,j}} \oplus X_j$.

We thus need to give (in addition to σ_r) $\log \mathsf{q} + |B'|(1 + 2\log N + \log \mathsf{q})$ bits of hint to P.

<u>*The predictor.*</u> We now show that, similarly to Lemma 3.1, given oracle access to ro_{k-1}, we can design a predictor algorithm P that predicts $X_i^{\mathsf{ro}_{k-1}}$ for every i in B' without querying ro_{k-1} at input $X_{i-1}^{\mathsf{ro}_{k-1}}$. The difference is that instead of running $A^{\mathsf{ro}_{k-1}}$ on σ_0

and giving A different challenges j, P will run A with initial input state σ_r (which is given in the hint), and simulate different oracles $\mathsf{ro}_{k,j}$ (which differ from ro_{k-1} on only the input T_{k-1}).

One of the further tricks is that the predictor needs to recognize T_{k-1} everytime to ensure subsequent queries to T_{k-1} are answered consistently. If $s_k > r$, this is easy as P (given the hint that indicates the first time when T_{k-1} appears in a query), will see the query T_{k-1} itself, and thus be able to answer subsequent queries to T_{k-1}. However, if $s_k = r$, then we need to give T_{k-1} (which is included into the hint) to P. Note that we do so without increasing the length of the hint, as we have replaced $S_{k-1}^{\mathsf{ro}_{k-1}}$ in σ_r (which can be obtained by querying $\mathsf{ro}_{k-1}(T_{k-1})$) with T_{k-1}.

Finally, there is one more small trick: if $\mathsf{bestchal}_j = j$ for some $j \in B'$, then in order to correctly predict X_j itself, P will need to know (from the hint) whether X_j is first queried by itself, or as part of the query $T_k^{\mathsf{ro}_{k,j}} = S_{k-1}^{\mathsf{ro}_{k,j}} \oplus X_j$. In the latter case, P will xor the query input with $S_{k-1}^{\mathsf{ro}_{k,j}}$ (which has been computed when answering the query T_{k-1}) to obtain X_j.

Correctness of the predictor. With similar argument as in Lemma 3.1, P is guaranteed to be correct as long as $\mathsf{ro}_{k-1} \notin \mathsf{colliding}_{k-1}$.

Suppose σ_r has m_r bits. We modify Lemma 3.1 as follows. We replace p with a function p_r of the memory size m_r, defined as

$$p_r = \lceil (m_r + 1 + \log \mathsf{q})/(W - 2\log N - \log \mathsf{q} - 1) + 1 \rceil \tag{3.2}$$

We now define $\mathsf{predictable}$ according to our new definition of P, p_r, and hint length.

Definition 3.4 *Define* $\mathsf{predictable}$ *as the set of all random oracles* $\mathsf{ro} \notin \mathsf{colliding}_0$ *for which there exists an input state* σ_r *of size* m_r *(such that* $1 \le p_r \le N-1$*) and a hint of length* $\log \mathsf{q} + p_r(1 + 2\log N + \log \mathsf{q})$*, given which* P *(described above) can correctly output*

p_r distinct X-labels without querying them.

Finally, we replace $\mathsf{bad}_{\mathsf{ro}}$ (in Theorem 3.3) with $\mathsf{bad}_{\mathsf{ro}}^{k-1} = \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$. As long as $\mathsf{ro}_{k-1} \notin \mathsf{bad}_{\mathsf{ro}}^{k-1} = \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$, with easy argument just as in Theorem 3.3, we are guaranteed that $\Pr_j[t_j > N/(2p_r)] \geq 1/2$, like in Theorem 3.3.

The arguments above lead to the following lemma (analogous to Theorem 3.3).

Lemma 3.2 *Fix any k ($1 \leq k \leq N$). Assume $\mathsf{ro}_{k-1} \notin \mathsf{bad}_{\mathsf{ro}}^{k-1} = \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$ and consider execution $A^{\mathsf{ro}_{k-1}}$. Define s_k as the first round such that $T_{k-1}^{\mathsf{ro}_{k-1}} \in \mathbf{q}_{s_k}$. For any $r < s_k$, let m_r be the bit-length of the input state σ_r of $A^{\mathsf{ro}_{k-1}}$ in round $r+1$. Let $t_{k,j,r} > 0$ be such that the first time $T_k^{\mathsf{ro}_{k,j}}$ is queried by $A^{\mathsf{ro}_{k,j}}$ after round s_k is in round $r + t_{k,j,r}$ ($t_{k,j,r} = \infty$ if no such query exists among the first q queries after round s_k). We say j is "hard" for time r if $t_{k,j,r} > N/(2p_r)$, where $p_r = \lceil (m_r + 1 + \log \mathsf{q})/(W - 2\log N - \log \mathsf{q} - 1) + 1 \rceil$. We have*

$$\Pr_j\left[j \text{ is hard for time } r\right] \geq \frac{1}{2}.$$

Extending Theorem 3.2: Lower Bound on the Sum of p_r. For now, we have proved Lemma 3.2, and build the relation between state size m_r and pebbling configuration size p_r. Next, we use the techniques in Section 3.3.2 to lower bound the sum of p_r, and thus obtain a lower bound on the sum of m_r, that is, the CMC lower bound.

Hardness of challenge c_k. From Lemma 3.2, in order to obtain a space lower bound for each step (as in Claim 3.1), we need to similarly define hardness for challenges.

Consider the execution of A^{ro_k}. Denote as $t_k > 0$ the value such that round $s_k + t_k$ is the first round where $T_k^{\mathsf{ro}_k}$ is queried (we define $t_k = \infty$ if such round does not exist among the first q queries). Define $r_k = \operatorname{argmax}_{0 \leq r \leq s_k}(N/(2p_r) - (s_k - r))$, where m_r is the size of the state σ_r at the end of round r, and p_r is the function of m_r defined by

Equation 3.2. We say challenge c_k is "hard" if it satisfies follows.

Definition 3.5 *A challenge c_k is hard if for $r_k = \operatorname{argmax}_{0 \leq r \leq s_k}(N/(2p_r) - (s_k - r))$ we have $t_{k,c_k,r_k} > N/(2p_r)$, where $s_k, t_{k,j,r}$ and p_r are as defined in Lemma 3.2.*

The multiple challenges argument. Next we will argue that the actual execution of A on oracle ro_N is identical to that with the oracle ro_k $(0 \leq k < N)$ until the moment when $T_k^{\mathsf{ro}_k}$ is queried.

Definition 3.6 *$\mathsf{wrongOrder}_k$ consists of all ro for which there exist i_1 and i_2 such that $0 \leq i_1 < i_2 \leq k$ and, in the run of A^{ro}, query $T_{i_2}^{\mathsf{ro}}$ occurs, while query $T_{i_1}^{\mathsf{ro}}$ does not occur before query $T_{i_2}^{\mathsf{ro}}$ occurs.*

We show the following claim.

Claim 3.8 *If for every j $(0 \leq j \leq N)$, $\mathsf{ro}_j \notin \mathsf{colliding}_j \cup \mathsf{wrongOrder}_j$, then for every k and $i \leq k$, $T_i^{\mathsf{ro}_N} = T_i^{\mathsf{ro}_k}$, and the execution of A^{ro_N} is identical to the execution of A^{ro_k} until the query T_k is first made, which (for $1 \leq k \leq N$) happens later than the moment when query $T_{k-1}^{\mathsf{ro}_N} = T_{k-1}^{\mathsf{ro}_k}$ is first made.*

Proof: The proof is deferred to Appendix A.1. ■

From Claim 3.8 and by definition of $\mathsf{wrongOrder}_k$, we obtain the following analogue of Claim 3.1, that is, if a challenge is hard, then the pebbling configurations before the challenge has size inversely proportional to the time for answering the challenge. (The proof is the same as in Claim 3.1.)

Claim 3.9 *Given adversary A, assume for every k $(0 \leq k \leq N)$, $\mathsf{ro}_k \notin \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$. Let $N' = N/2$. If challenge i is hard (i.e., $t_i + (s_i - r_i) > N'/p_{r_i}$), then during the execution of A^{ro_N}, for any $0 \leq j \leq s_i$, we have $p_{s_i - j} \geq N'/(t_i + j)$.*

Next, we show that if $\{T_0, T_1, \ldots, T_N\}$ are queried in the correct order and the number of hard challenges is large, the sum of p_r will be large as well.

Definition 3.7 *Let E_{hard} be the event that there are at least $H \geq N(\frac{1}{2} - \epsilon)$ hard challenges (as defined in Definition 3.5). Let E_{corr} be the event that $\mathsf{ro}_k \notin \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$ (see Definitions 3.6) for every k, and A^{ro_N} queries $T_N^{\mathsf{ro}_N}$. Let E_q be the event that A^{ro_N} makes no more than q total queries.*

Claim 3.10 *If $E_{\mathsf{hard}} \cap E_{\mathsf{corr}} \cap E_q$, then*

$$\sum_{r=1}^{s_{N+1}} p_r \geq \ln 2 \cdot \left(\frac{1}{2} - \epsilon\right) \cdot \frac{1}{2} \cdot N^2 .$$

Proof: Since E_{corr} holds, $T_0, \ldots, T_N$ are queried in the correct order. Since E_q holds, all these queries happen no later than the q-th query, hence Claim 3.9 applies and each t_k is finite. Moreover, by definition of p_r in Equation 3.2, $p_r \geq 1$ and $p_0 = 1$. Therefore, we can apply Claim 3.4 to the execution of A^{ro_N}, and finish the proof. ■

Converting from $\sum p_r$ to CMC. Now we need to convert from $\sum p_r$ to $\sum m_r$.

Claim 3.11 *For every $r > 0$,*

$$m_r \geq p_r \cdot (W - 2\log N - \log \mathsf{q} - 1)/3 .$$

Proof: By definition of p_r, we have that

$$p_r = \left\lceil \frac{m_r + 1 + \log \mathsf{q}}{W - 2\log N - \log \mathsf{q} - 1} + 1 \right\rceil \leq \frac{m_r + 1 + \log \mathsf{q}}{W - 2\log N - \log \mathsf{q} - 1} + 2,$$

because the ceiling adds at most 1. Therefore,

$$(p_r - 2) \cdot (W - 2\log N - \log \mathsf{q} - 1) \leq m_r + 1 + \log \mathsf{q},$$

(because we can assume $(W - 2\log N - \log \mathsf{q} - 1) > 0$ — otherwise, Theorem 3.4 is trivially true) and thus

$$\begin{aligned} m_r &\geq (p_r - 2) \cdot (W - 2\log N - \log \mathsf{q} - 1) - \log \mathsf{q} - 1 && (3.3)\\ &= p_r \cdot (W - 2\log N - \log \mathsf{q} - 1) - 2 \cdot (W - 2\log N - \log \mathsf{q} - 1) - \log \mathsf{q} - 1 \\ &= p_r \cdot (W - 2\log N - \log \mathsf{q} - 1) - 2 \cdot (W - 2\log N - 0.5\log \mathsf{q} - 0.5). && (3.4) \end{aligned}$$

Since $m_r \geq W$ (see our complexity measure definition in Section 2.2), $m_r \geq W - 2\log N - 0.5\log \mathsf{q} - 0.5$ and therefore we can increase the left-hand side by $2 \cdot m_r$ and the right-hand side by $2 \cdot (W - 2\log N - 0.5\log \mathsf{q} - 0.5)$ and the inequality still holds; and therefore $3m_r \geq p_r \cdot (W - 2\log N - \log \mathsf{q} - 1)$. ■

Lemma 3.3 *Assuming $E_{\mathsf{hard}} \cap E_{\mathsf{corr}}$ (see Definition 3.7), for any integer q, either A^{ro_N} makes more than q queries (and thus $\mathsf{CMC}(A^{\mathsf{ro}_N}) > \mathsf{q}W$ by definition) or*

$$\mathsf{CMC}(A^{\mathsf{ro}_N}(X)) \geq \frac{\ln 2}{6} \cdot \left(\frac{1}{2} - \epsilon\right) \cdot N^2 \cdot (W - 2\log N - \log \mathsf{q} - 1)\ .$$

Proof: We observe that if A^{ro_N} makes no more than q queries, then $E_{\mathsf{hard}} \cap E_{\mathsf{corr}} \cap E_q$ hold, and we can combine Claims 3.10 and 3.11 to get

$$\begin{aligned} \mathsf{CMC}(A^{\mathsf{ro}_N}(X)) = \sum_{r=1}^{s_{N+1}} m_r &\geq \frac{1}{3} \cdot \sum_{r=1}^{s_{N+1}} p_r \cdot (W - 2\log N - \log \mathsf{q} - 1) \\ &\geq \frac{\ln 2}{3} \cdot \left(\frac{1}{2} - \epsilon\right) \cdot \frac{1}{2} \cdot N^2 \cdot (W - 2\log N - \log \mathsf{q} - 1)\ . \end{aligned}$$

This concludes the proof of Lemma 3.3. ■

All that remains is to show a lower bound on the probability of $(E_{\mathsf{hard}} \cap E_{\mathsf{corr}} \cap E_q) \cup \overline{E_q}$ (over the uniform choice of ro_N), and to argue that ro_N is indeed uniform. We defer the proof of oracle uniformity and the probability analysis to Appendix A.2 and

Appendix A.3.

3.6 Conclusions and Open Problems

In this chapter, we have provided an *optimal* CMC lower bound for the single evaluation of Scrypt in the parallel random oracle model. There are still several promising research directions worth exploring.

1. Our work shows a CMC lower bound for a single evaluation. Intuitively it extends to the setting where the adversary can evaluate the function on multiple inputs, and the average CMC lower bound seems still hold. However, it is harder to provide a formal proof as we do not know whether an adversary can cleverly choose inputs so that different instances share similar computation patterns which helps to amortize the computation. Finding such a proof would be valuable as in practice the brute force attackers do evaluate the hash functions on huge amounts of inputs.

2. Our memory hardness proof is with respect to Cumulative Memory Complexity (CMC) which measures the hardware costs of attackers. However, as pointed out by [19] and [53], CMC is not always an accurate metric as it allows for time-memory trade-off and the memory cost (i.e. storage and access cost) is not linear to the memory size; moreover, it ignores the energy cost of adversaries which can be significant in some scenarios. Given this, many new cost metrics have been introduced. For example, sustatined memory complexity [19] was proposed to enforce the consistent use of large memory; bandwidth hardness [53] was proposed to measure energy cost. It would be nice if we can have similar proofs for the above new metrics.

3. Our lower bound still suffers from a constant factor loss. Since the hardness pa-

rameter N is relatively small in practice, optimizing the constant factor loss is meaningful to obtain a better concrete security.

4. Our result relies on the assumption that the underlying hash function $\mathcal{H}$ is modeled as a random oracle. However, in practice, given the demands for $\mathcal{H}$ to have large outputs (to increase memory hardness without changing the description size of MHFs), the hash function is usually instantiated with an ad-hoc design built upon simple primitives, and attacks which exploit the design of $\mathcal{H}$ might be possible. Therefore a promising future direction would be to provide a proof in a more fine-grained model, where we take the structure of $\mathcal{H}$ into consideration.

Chapter 4

Data Independent MHFs from Simple Primitives

Outline of the Chapter. In this chapter, we construct a class of *data independent* memory-hard functions that are resilient to side-channel attacks. We highlight that our scheme is highly efficient on general-purpose CPUs, and thus more ASIC-resistant than previous provably secure constructions. The scheme is built from a primitive called *small-block labeling functions* that can be instantiated from *fast* symmetric schemes (e.g. fixed-key AES). (Note that this notion is introduced for modularity reason – we could define our designs directly as depending on a primitive.) After introducing the background and our contributions in Section 4.1, we define and construct small-block labeling functions in Section 4.2. Followed by the construction, in Section 4.3, we show that the scheme satisfies an essential property for building iMHFs – pebbling reducibility. Finally in Section 4.4, we show a generic approach for constructing iMHFs from small-block labeling functions, whereas the MHFs are efficiently computable on general-purpose CPUs. The full version of this work is available on [39].

4.1 Introduction

Alwen and Serbinenko [10] proposed an elegant framework for constructing *graph-based MHFs* from a directed acyclic graph (DAG) and a hash function modeled as a random oracle. The construction is generic and enjoys a favorable property called *side-channel resilience* — namely, an adversary cannot gain information of the sensitive input by monitoring the function execution. However, in practice, the hash function $\mathcal{H}$ is usually an ad-hoc construction based on simpler building blocks (e.g., Scrypts resembles a permutation-based stream-cipher design), and highly efficient ASICs for $\mathcal{H}$ do exist. A possible approach for slowing-down adversary and hardening ASICs design is to use higher memory hardness parameter (e.g., use big graphs), however, this solution is unsatisfiable as it further slows down the authentication speed. In summary, the heuristic design of $\mathcal{H}$ renders the memory hardness proof of [10] less useful in the real world setting.

Our goal is thus to design a graph-based MHF scheme that is ASIC-resistant in practice. For example, it is desirable that the underlying hash function is relatively efficient on general-purpose CPUs, so that ASICs that achieve significant efficiency advantage (over CPUs) cannot be easily built. Moreover, another advantage of fast hashing on CPUs is that it enables fast authentication for legitimate users (in the password hashing setting) and efficient proof verifications (in the proof-of-work setting), without sacrificing too much of memory hardness (i.e. there is no need for the graph size N to be too small).[1]

We observe that AES operations are particularly fast on general-purpose CPUs, as AES-NI instruction set is widely embedded in CPU chips. Inspired by the observation, we propose a generic framework for constructing graph-based MHFs from (fixed-key) AES modeled as a random permutation. Note that for completeness, we also provide constructions based on other simple cryptographic primitives (e.g., compression functions

[1]For example, in Litecoin, to achieve fast verification, the hardness parameter is chosen as a relatively small value, which makes ASICs easier to be built.

and keyed block ciphers).

We remark that the underlying hash function $\mathcal{H}$ we use in this chapter has small output space (i.e. $\{0,1\}^{128}$). If we want to use a *wide-block labeling function* as in practice (to increasing memory hardness without changing function description size), we have to study MHFs at a finer level of granularity that considers the inner structure of $\mathcal{H}$, and we would like to understand how such an $\mathcal{H}$ is meant to be built in a sound way. This is exactly the main contribution of Chapter 5.

Our Contributions. We initiates the study of provably-secure MHFs built from basic symmetric primitives, which we model as ideal - we consider block ciphers, permutations and compression functions. We build graph-based MHFs based on them in a model where the primitives can be queried by the adversary on many inputs in parallel. Towards the goal, we provide one-call efficient instantiations of the hash function $\mathcal{H}$ from the basic primitives above. We will adapt previous lemmas based on ex-post-facto arguments (dating back to [38]) to reduce the security of graph-based MHFs to the pebbling complexity of the underlying DAG.

Before going to the detail, we provide some intuition for the small-block labeling function.

Small-Block Labeling Function: Constructions and Intuition. The small-block labeling functions $\mathcal{H}_{\mathsf{fix}}$ takes an input[2] $x \in \{0,1\}^L \cup \{0,1\}^{2L}$ and outputs an L-bit label. For a compression function cf, the resulting output is $\mathsf{cf}(x)$; for an ideal cipher ic, we split the input into a key part $k \in \{0,1\}^L \cup \{\perp\}$ (where $\perp$ is a designated key separate from the L-bit strings, which as with compression functions, will be necessary to implement

[2]We assume the compression function allows both L- and $2L$-bit inputs, though most compression functions do not allow this by design. This could however be easily achieved by reserving one bit of the input to implement domain separation, and then padding short inputs.

variable input length) and an input part $x \in \{0,1\}^L$, the resulting output is $\mathsf{ic}(k,x) \oplus x$; for a random permutation rp, we denote as x^* the exlusive-or sum of L-bit input blocks and the output is $\mathsf{rp}(x^*) \oplus x^*$.

For any graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, the memory hardness of the graph function $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ (See Section 2.5) is argued similarly as in previous work [10]. The high level idea is to transform the execution of any algorithm A that computes the MHF into an *ex-post-facto* pebbling for the graph $\mathbb{G}$, and argue that the cumulative memory complexity of A is proportional to the cumulative complexity of the pebbling. Here we generalize the technique of [10] – which relies on a compression argument – so that it works even if:

1. The ideal-primitive input contains no explicit information of the node v. (This was not the case in prior work.)
2. The adversary can make *inverse* queries to the ideal primitive (as we also consider block ciphers now).
3. The input length of the primitive is fixed, and usually shorter than the actual input length of the labeling function.

4.2 Small-Block Labeling Functions

In this section, we define and construct small-block labeling functions.

Definition 4.1 (Small-Block Labeling Functions) *For an ideal primitive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ with block length $L = 2^\ell$, we say*

$$\mathcal{H}_{\mathsf{fix}} = \{\, \mathsf{flab}^{\mathsf{ip}} : \{\, \{0,1\}^L \cup \{0,1\}^{2L} \,\} \to \{0,1\}^L \,\}_{\mathsf{ip} \in \mathbb{IP}}$$

is a family of β-small-block labeling functions if it has the following property.

$\beta(\cdot,\cdot)$**-pebbling reducibility:** *For any $\epsilon \in (0,1]$ and 2-indegree (predecessors-distinct) DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$,*[3] *let $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ be the graph function built upon $\mathbb{G}$ and $\mathcal{H}_{\mathsf{fix}}$. We have*

$$\mathsf{CMC}_{\epsilon}(\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}) \geq \beta(\epsilon, \log|\mathbb{V}|) \cdot \mathsf{cc}(\mathbb{G}),$$

where $\mathsf{cc}(\mathbb{G})$ is the cumulative complexity of $\mathbb{G}$ (Definition 2.3).

Construction. Next we show how to construct small-block labeling functions from ideal primitives. Our major contribution is the construction from random permutations, which can instantiated from fixed-key AES. For completeness, we also present the constructions from ideal ciphers and compression functions. We fix the input domain to be $\{0,1\}^L \cup \{0,1\}^{2L}$ and the output space to be $\{0,1\}^L$, and denote as $\mathbb{RP}$, $\mathbb{IC}$, $\mathbb{CF}$ the random permutations, ideal ciphers, and compression functions, respectively.

1. Given any $\mathsf{rp} \in \mathbb{RP}$, we define the labeling function $\mathsf{flab}^{\mathsf{rp}}(\cdot)$ as follows: For any input $x \in \{0,1\}^L$, the output is $\mathsf{flab}^{\mathsf{rp}}(x) := \mathsf{rp}(x) \oplus x$; for any input $(x_1, x_2) \in \{0,1\}^{2L}$, denote as $x^* := x_1 \oplus x_2 \in \{0,1\}^L$, the output is $\mathsf{flab}^{\mathsf{rp}}(x_1, x_2) := \mathsf{rp}(x^*) \oplus x^*$.

2. Given any $\mathsf{ic} \in \mathbb{IC}$, we define the labeling function $\mathsf{flab}^{\mathsf{ic}}(\cdot)$ as follows: For any input $x \in \{0,1\}^L$, the output is $\mathsf{flab}^{\mathsf{ic}}(x) := \mathsf{ic}(\perp, x) \oplus x$; for any input $(k, x) \in \{0,1\}^{2L}$, the output is $\mathsf{flab}^{\mathsf{ic}}(k, x) := \mathsf{ic}(k, x) \oplus x$.

3. Given any $\mathsf{cf} \in \mathbb{CF}$, we define the labeling function $\mathsf{flab}^{\mathsf{cf}}(\cdot)$ as follows: For any input $x \in \{0,1\}^L \cup \{0,1\}^{2L}$, the output is $\mathsf{flab}^{\mathsf{cf}}(x) := \mathsf{cf}(x)$.

Note that all of the above constructions are highly efficient as they call the ideal-primitive only once. Next we show that the constructions are pebbling reducible.

[3] $\mathbb{G}$ can have multiple source/sink nodes.

4.3 Small-Block Labeling Functions: Pebbling Reducibility

In this section, we show that the labeling functions constructed in Section 4.2 satisfy pebbling reducibility.

Theorem 4.1 *Assume an adversary can make no more than* q_1 *oracle calls and* q_2 *output calls such that* $\mathsf{q}_1 + \mathsf{q}_2 = \mathsf{q} = 2^{L/4}$. $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{cf}}\}_{\mathsf{cf} \in \mathbb{CF}}$ *built upon compression function* $\mathbb{CF}$ *is* $\beta(\cdot,\cdot)$*-pebbling reducible, where for all* $\epsilon \geq 3 \cdot 2^{-L/8}$ *and* $N \leq 2^{L/8}$*, it holds that* $\beta(\epsilon, \log N) \geq \frac{\epsilon L}{8}$.

Theorem 4.2 *Assume an adversary can make no more than* q_1 *oracle calls and* q_2 *output calls such that* $\mathsf{q}_1 + \mathsf{q}_2 = \mathsf{q} = 2^{L/4}$. $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ic}}\}_{\mathsf{ic} \in \mathbb{IC}}$ *built upon ideal cipher* $\mathbb{IC}$ *is* $\beta(\cdot,\cdot)$*-pebbling reducible, where for all* $\epsilon \geq 3 \cdot 2^{-L/8}$ *and* $N \leq 2^{L/8}$*, it holds that* $\beta(\epsilon, \log N) \geq \frac{\epsilon L}{8}$.

Theorem 4.3 *Assume an adversary can make no more than* q_1 *oracle calls and* q_2 *output calls such that* $\mathsf{q}_1 + \mathsf{q}_2 = \mathsf{q} = 2^{L/8}$. $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{rp}}\}_{\mathsf{rp} \in \mathbb{RP}}$ *built upon random permutation* $\mathbb{RP}$ *is* $\beta(\cdot,\cdot)$*-pebbling reducible, where for all* $\epsilon \geq 3 \cdot 2^{-L/10}$ *and* $N \leq 2^{L/10}$*, it holds that* $\beta(\epsilon, \log N) \geq \frac{\epsilon L}{40}$.

Remark 4.1 (Instantiation from fixed-key AES) *From a practical perspective, our main result is the construction from random permutations. This can be instantiated from fixed-key AES, thus eliminating the expensive re-keying costs when evaluating AES. On the other hand, we think that the construction from ideal ciphers is still of theoretical interest as it leads to a better provable bound.*

Remark 4.2 *The proof for compression functions addresses the fact that prior work included the node identity into the hash-function input, thus effectively requiring* $2L + \log N$*-bit inputs, and we get away with* $2L$.

Next we present the proofs for Theorem 4.1, Theorem 4.2 and Theorem 4.3. First, we introduce some notation for graph labeling, then we highlight the proof techniques and introduce the notion of ex-post-facto pebbling in the ideal primitive model. Finally, we provide the formal proof.

Graph Label Notations. Fix ideal primitive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$,[4] input vector $\mathbf{x}$ and any graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. For a primitive $\mathsf{ip} \in \mathbb{IP}$ and any node $v \in \mathbb{V}$, we denote as ℓ_v the graph label of v. If v is a source, $\mathsf{prelab}(v)$ is the corresponding input label x_v. If v is a non-source node, we define $\mathsf{prelab}(v)$ based on the type of the ideal primitive:

- If $\mathbb{IP} = \mathbb{RP}$, we define $\mathsf{prelab}(v)$ as the exclusive-or sum of v's parents' ℓ-labels.
- If $\mathbb{IP} = \mathbb{IC}/\mathbb{CF}$, we define $\mathsf{prelab}(v)$ as the concatenation of v's parents' ℓ-labels.

Similarly, for every node $v \in \mathbb{V}$, we define $\mathsf{aftlab}(v)$ based on the type of the ideal primitive:

- If $\mathbb{IP} = \mathbb{RP}/\mathbb{CF}$, we define $\mathsf{aftlab}(v) = \mathsf{ip}(\mathsf{prelab}(v))$.
- If $\mathbb{IP} = \mathbb{IC}$ and v has only one parent, we define $\mathsf{aftlab}(v) = \mathsf{ip}(\perp, \mathsf{prelab}(v))$. Otherwise if v has two parents, denote as $\mathsf{prelab}(v) = (y_1, y_2)$ (where $y_1, y_2 \in \{0,1\}^L$ are ℓ-labels of v's parents), we define $\mathsf{aftlab}(v)$ as $(y_1, \mathsf{ip}(y_1, y_2))$.

In the following context, we abuse the notation a bit in that if $\mathsf{prelab}(v)$ (or $\mathsf{aftlab}(v)$) is an L-bit string and ip is an ideal cipher, we use $\mathsf{ip}(\mathsf{prelab}(v))$ (or $\mathsf{ip}^{-1}(\mathsf{aftlab}(v))$) to denote $\mathsf{ip}(\perp, \mathsf{prelab}(v))$ (or $\mathsf{ip}^{-1}(\perp, \mathsf{aftlab}(v))$). Moreover, for an ideal cipher query with input $x_c = (\perp, x)$, we say $x_c = \mathsf{prelab}(v)$ (or $x_c = \mathsf{aftlab}(v)$) if and only if $x = \mathsf{prelab}(v)$ (or $x = \mathsf{aftlab}(v)$), respectively.

[4]$\mathbb{CF}$ denotes the compression function, $\mathbb{IC}$ denotes the ideal cipher, and $\mathbb{RP}$ denotes the random permutation.

We remark that $\mathsf{prelab}(v)$ (and $\mathsf{aftlab}(v)$) are more than just single labels, they are used to identify the node from a query input to the ideal primitive.

Proof Highlight. Similar as in [10], the proof idea is to transform any algorithm execution A into an *ex-post-facto* pebbling, and argue that the cumulative memory complexity of A is proportional to the cumulative complexity of the pebbling. This is proved by mapping each node $v \in \mathbb{V}$ to an ideal-primitive entry $(\mathsf{prelab}(v), \mathsf{ip}(\mathsf{prelab}(v)))$, and argue that for each round $i \in \mathbb{N}$, the input state σ_i should have large size as it is an encoding for many ideal-primitive entries. In particular, for every node v in the ith pebbling configuration, $\mathsf{ip}(\mathsf{prelab}(v))$ (and ℓ_v) can be decoded from an oracle-call input in the partial execution $\mathsf{A}(\sigma_i)$. Here we generalize the technique of [10] so that it works even if:

1. The ideal-primitive input $\mathsf{prelab}(v)$ contains no explicit information of the node index v. (Note that in previous work [10], $\mathsf{prelab}(v)$ has v as a prefix.)
2. The adversary can make *inverse* queries to the ideal primitive.
3. The input length of the primitive is much shorter than the actual input length of the labeling function.

Ex-post-facto Pebbling. Similar as in [10], we define a notion called *ex-post-facto pebbling in the ideal primitive model.* Fix ideal primitive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$, input vector $\mathbf{x}$, DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. For any $\mathsf{ip} \in \mathbb{IP}$, randomness r, and the execution trace $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ (that runs for $t_{\mathsf{peb}} + 1$ rounds), we turn the trace into an *ex-post-facto pebbling*

$$\mathcal{P}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)) = (\mathsf{P}_0 = \emptyset, \ldots, \mathsf{P}_{t_{\mathsf{peb}}}) .$$

For each oracle call/query that asks for the ideal-primitive value on input x_c, we say that the call is a *correct* call for a node $v \in \mathbb{V}$ if and only if x_c matches $\mathsf{prelab}(v)$ and the call is forward, or x_c matches $\mathsf{aftlab}(v)$ and the call is an inverse call. We define the ex-post-facto pebbling configurations in reverse order. For i from t_{peb} to 1, denote as σ_i the input state of round $i+1$ and $\mathsf{A}(\sigma_i)$ the partial execution of $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ after round i, the pebbling configuration P_i is defined as follows. (In the following context, by round γ, we always mean the γ-th round in the execution $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$.)

1. *Critical Calls:* We sort the output/ideal primitive calls of $\mathsf{A}(\sigma_i)$ in chronological order[5] and determine whether they are *critical* calls.

    - An output call (in round $\gamma > i$) with label (v, ℓ_v) is a *critical* call for $v \in \mathbb{V}$ if and only if v is a sink and in the trace $\mathsf{A}(\sigma_i)$, no correct call for v appeared before round γ.
    - An ideal-primitive call (in round $\gamma > i$) is a *critical* call for a node $u \in \mathbb{V}$ if and only if the following conditions both hold: i) the ideal-primitive call is a correct call for a successor node $v \in \mathsf{succ}(u)$ and in the trace $\mathsf{A}(\sigma_i)$, no correct call for u appeared before round γ; ii) v is in P_γ.

2. *Pebbling Configuration:* A node $v \in \mathbb{V}$ is included into the pebbling configuration P_i if and only if *both* of the following conditions hold:

    - There is at least one critical call for v in the trace $\mathsf{A}(\sigma_i)$.
    - There is at least one correct call for v between round 1 and round i (inclusively).[6]

In a critical call, the algorithm provides the information of a graph label without recomputing, hence the call is useful in extracting ideal-primitive entries. On a side note, by definition of critical call and ex-post-facto pebbling, for any round i, we might possibly

[5]We assume an implicit order for the calls in the same round.

[6]Note that the existence of a correct call for v in round i does not imply $v \in \mathsf{P}_i$, because v may not have a critical call in the future.

put a node u into P_i only if u is a sink or one of its successor $v \in \mathsf{succ}(u)$ is in P_γ for some $\gamma > i$. Looking ahead, this property significantly simplifies the proof of pebbling reduction in our new computational model.

Proof: [of Theorem 4.1, Theorem 4.2 and Theorem 4.3] Fix ideal primitive type $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$, (non-colliding) input vector $\mathbf{x}$, adversary A, and any graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, the proof consists of the following four steps.

Label Collisions. First, we show that with probability at least $1 - \epsilon_{\mathsf{coll}}(\mathbb{IP})$ (over the uniform choice of the ideal primitive), the pre-labels $\{\mathsf{prelab}(v)\}_{v\in\mathbb{V}}$ are all distinct.[7]

Lemma 4.1 *Fix ideal primive $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ (with block length L), predecessors-distinct DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ (with n_s source nodes), non-colliding input vector $\mathbf{x} \in \{0,1\}^{n_s L}$,[8] and $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ip}}\}_{\mathsf{ip}\in\mathbb{IP}}$ constructed in Section 4.2. With probability at least $1 - \epsilon_{\mathsf{coll}}(\mathbb{IP})$ (over the uniformly random choice of ip), the graph labeling satisfies that the pre-labels $\{\mathsf{prelab}(v)\}_{v\in\mathbb{V}}$ are all distinct. Here $\epsilon_{\mathsf{coll}}(\mathbb{CF}) = |\mathbb{V}|^2/2^{L+1}$ and $\epsilon_{\mathsf{coll}}(\mathbb{IC}) = \epsilon_{\mathsf{coll}}(\mathbb{RP}) = |\mathbb{V}|^2/2^L$.*

Proof: The proof is deferred to Appendix B.1. ■

The property in Lemma 4.1 is useful in determining the node index v when one sees an ideal-primitive query related to v. Moreover, it guarantees that each node v maps to a *unique* ideal-primitive input entry $\mathsf{prelab}(v)$.

Pebbling Legality. Next, we show that with high probability (over the uniform choices of the ideal primitive ip and random coins r), the *ex-post-facto* pebbling $\mathcal{P}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r))$

[7]If $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$, this also implies that $\{\mathsf{aftlab}(v)\}_{v\in\mathbb{V}}$ are distinct.

[8]By non-colliding, we mean $\mathbf{x} = (x_1, \ldots, x_{n_s})$ where $x_i \neq x_j$ for every $i \neq j$.

$= (\mathsf{P}_0 = \emptyset, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ for $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ is *legal*, and thus

$$\sum_{i=0}^{t_{\mathsf{peb}}} |\mathsf{P}_i| \geq \mathsf{cc}(\mathbb{G})$$

as long as the pebbling is *successful*.

Before presenting the lemma, we prove a claim that will be useful in many places.

Claim 4.1 *Fix any execution* $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ *(with input states* $\sigma_0, \sigma_1, \ldots$*) whose* ex-post-facto pebbling *is* $\mathcal{P}(\mathsf{A}) = (\mathsf{P}_0, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$*. For any* $i \in [t_{\mathsf{peb}}]$ *and any vertex* $v \in \mathsf{P}_i \setminus \mathsf{P}_{i-1}$*, it holds that there is a correct call for* v *in round* i*.*

Proof: Since $v \in \mathsf{P}_i$, there is a correct call for v between round 1 and round i, and there is a critical call for v in $\mathsf{A}(\sigma_i)$. Suppose for contradiction that there is no correct call for v in round i, then there is a correct call for v between round 1 and round $i-1$, and there is a critical call for v in $\mathsf{A}(\sigma_{i-1})$, hence $v \in \mathsf{P}_{i-1}$, contradiction. ■

Lemma 4.2 *Fix* $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ *(with block length* L*), predecessors-distinct DAG* $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ *(with* n_s *source nodes), non-colliding input vector* $\mathbf{x} \in \{0,1\}^{n_s L}$*, algorithm* A*, and* $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ip}}\}_{\mathsf{ip} \in \mathbb{IP}}$ *constructed in Section 4.2. With probability at least* $1 - \epsilon_{\mathsf{coll}}(\mathbb{IP}) - \epsilon_{\mathsf{legal}}(\mathbb{IP})$ *(over the uniformly random choices of* ip *and* A*'s internal coins* r*), the pre-labels are distinct and the ex-post-facto pebbling for* $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ *is legal. Here* $\epsilon_{\mathsf{coll}}(\mathbb{IP})$ *is the same as in Lemma 4.1 and* $\epsilon_{\mathsf{legal}}(\mathbb{CF}) = \mathsf{q} \cdot |\mathbb{V}|/2^{L-1}$*,* $\epsilon_{\mathsf{legal}}(\mathbb{IC}) = \epsilon_{\mathsf{legal}}(\mathbb{RP}) = \mathsf{q} \cdot |\mathbb{V}|/2^{L-2}$*, where* q *is an upper bound on the number of calls made by* A*.*

Proof: The proof is deferred to Appendix B.2. ■

Pebbling Reduction. Next, we build the connection between the state size and the size of the pebbling configuration. In particular, we show that with high probability, the input state size $|\sigma_i|$ is proportional to $|\mathsf{P}_i|$ for all $i \in \mathbb{N}$.

Lemma 4.3 *Fix $L = 2^\ell$, predecessors-distinct DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ (with n_s source nodes), non-colliding input vector $\mathbf{x} \in \{0,1\}^{n_s L}$, algorithm A (that makes at most $\mathsf{q}-1$ calls), and $\mathcal{H}_{\mathsf{fix}} = \{\mathsf{flab}^{\mathsf{ip}}\}_{\mathsf{ip} \in \mathbb{IP}}$ constructed in Section 4.2. Set values $\beta_{\mathbb{CF}} := \lfloor L - 2\log \mathsf{q} - \log|\mathbb{V}| - \log 3 \rfloor$, $\beta_{\mathbb{IC}} := \lfloor L - 1 - 2\log\mathsf{q} - \log|\mathbb{V}| - \log 3\rfloor$, and $\beta_{\mathbb{RP}} := \lfloor \frac{L}{2} - 1 - 2\log\mathsf{q} - \log|\mathbb{V}| - \log 3\rfloor$. For any $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/\mathbb{RP}$ and $\lambda \in \mathbb{N}$, define $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ as the event where the following three conditions* all *hold:*

1. *The pre-labels are distinct from each other.*
2. *The ex-post-facto pebbling $(\mathsf{P}_0, \mathsf{P}_1, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ for $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ is legal.*
3. *There exists $i \in \mathbb{N}$ such that $|\sigma_i| < |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$, where σ_i denotes the input state for round $i+1$ and P_i denotes the pebbling configuration in round i.*

It holds that $\Pr\left[\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}\right] \le 2^{-\lambda}$ for all $\lambda \in \mathbb{N}$, where the probability is taken over the choice of $\mathsf{ip} \xleftarrow{\$} \mathbb{IP}$ and random coins of A.

Proof:

Without loss of generality we fix r to be the optimal random coins of A that maximizes $\Pr[\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}]$. We will show a predictor P (that hardwires r and has oracle access to ip), such that if $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens (which implies $|\sigma_i| < |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$ for some $i \in \mathbb{N}$), there will be a hint h with no more than $|\mathsf{P}_i| \cdot L - \lambda$ (and $|\mathsf{P}_i| \cdot (L-1) - \lambda$ when $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$) bits, where $P(h)$ can predict $|\mathsf{P}_i|$ ideal primitive entries correctly. Thus by the compression arguments that ideal primitives cannot be compressed (i.e., Lemma 2.1 and Lemma 2.2), $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens with probability no more than $2^{-\lambda}$ and the lemma holds. Next we describe the hint h and the predictor P.

<u>*The hint.*</u> For any choice of $\mathsf{ip} \in \mathbb{IP}$, if event $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens, there exists a round $i \in \mathbb{N}$ such that $|\sigma_i| < |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$, where σ_i is the input state of round $i+1$ and $\mathsf{P}_i = (v_1, v_2, \ldots, v_{|\mathsf{P}_i|})$ is the ex-post-facto pebbling configuration. The hint consists of the state σ_i and the

following helper information. (In the following context, if not describe explicitly, by critical call, we always mean a critical call in the trace $\mathsf{A}(\sigma_i)$.)

- A sequence $Q_i = (\mathsf{id}_1, \mathsf{id}_2, \ldots, \mathsf{id}_{|\mathsf{P}_i|}) \in [\mathsf{q}-1]^{|\mathsf{P}_i|}$, where id_j $(1 \leq j \leq |\mathsf{P}_i|)$ is the index of the *first* critical call for $v_j \in \mathsf{P}_i$ in the trace $\mathsf{A}(\sigma_i)$. (Recall that we sort the calls in chronological order, and assume an implicit order for the calls in the same round.)

- A nodes sequence $W_i = (w_1, w_2, \ldots, w_{|\mathsf{P}_i|})$, where $w_j = v_j$ $(1 \leq j \leq |\mathsf{P}_i|)$ if the id_j-th call is an output call; otherwise, if the id_j-th call is a correct call for some successor of v_j, then w_j is assigned as the corresponding successor node.

- A sequence $B_i = (b_1, b_2, \ldots, b_{|\mathsf{P}_i|})$, where $b_j \in \{0, 1, 2\}$ is used to indicate the relation between w_j and v_j. In particular, $w_j = v_j$ if $b_j = 0$, otherwise v_j is the b_j-th predecessor of w_j.

- A sequence $C_i = (\mathsf{cid}_1, \mathsf{cid}_2, \ldots, \mathsf{cid}_{|\mathsf{P}_i|})$, where $\mathsf{cid}_j = 0$ $(1 \leq j \leq |\mathsf{P}_i|)$ if there is no correct call for $v_j \in \mathsf{P}_i$ in the trace $\mathsf{A}(\sigma_i)$, otherwise cid_j is the query index of the *first* correct call for v_j.

- If $\mathbb{IP} = \mathbb{RP}$, the hint includes an extra sequence $H_i = (h_1, h_2, \ldots, h_{|\mathsf{P}_i|})$, where h_j $(1 \leq j \leq |\mathsf{P}_i|)$ is the label ℓ_{v_j} if there exists some $k > j$ such that $\mathsf{id}_j = \mathsf{id}_k$ (i.e., another node $v_k \in \mathsf{P}_i$ has the same *first* critical call), otherwise h_j is set as empty[9]. We see that there are at most $\lfloor |\mathsf{P}_i|/2 \rfloor$ non-empty values in the sequence, as any ideal-primitive call can be a critical call for at most two vertices.

Note that we can easily recover the configuration P_i from the hint W_i and B_i. The size of the hint is no more than $\mathsf{len}_{\mathbb{IP}} := |\mathsf{P}_i| \cdot L - \lambda$ (and $\mathsf{len}_{\mathbb{IP}} := |\mathsf{P}_i| \cdot (L-1) - \lambda$ when $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$) bits given the setting of $\beta_{\mathbb{IP}}$.

[9]Note that we don't need an indicator (e.g., $h_j = \perp$) to tell if h_j is empty or not, as we can know it from the sequence Q_i. This enables us to have a shorter H_i.

The predictor P. Given any input the predictor P parses the input into σ_i, Q_i, W_i, B_i, C_i and H_i as mentioned before[10], and recovers the pebbling configuration P_i. Then P runs the partial execution $\mathsf{A}(\sigma_i)$ and attempts to predict $(\mathsf{prelab}(v), \mathsf{ip}(\mathsf{prelab}(v)))$ for every $v \in \mathsf{P}_i$ *without querying* $\mathsf{ip}(\mathsf{prelab}(v))$. In the following context, if not describe explicitly, by critical call, we always mean a critical call in the trace $\mathsf{A}(\sigma_i)$.

When simulating $\mathsf{A}(\sigma_i)$, the predictor uses the following approach to determine if an ideal-primitive call is a correct call for a node v: The predictor keeps track of the labels $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v for every $v \in \mathbb{V}$. Moreover, after knowing the labels of v's *predecessors*, the predictor updates $\mathsf{prelab}(v)$ accordingly. Given an ideal-primitive call from A, P determines call correctness by checking the following cases *sequentially*:

- If P knows from hint Q_i that the call is the *first* critical call for some node $v \in \mathsf{P}_i$, then the predictor knows that it is also a correct call for some node w, where w can be extracted from the hint W_i.

- If the call is the first correct call[11] for some node $v \in \mathsf{P}_i$, then P will know it from the hint C_i.

- The call is a *forward* call. Then the predictor checks if there exists a node $v \in \mathbb{V}$ where $\mathsf{prelab}(v)$ was updated and $\mathsf{prelab}(v)$ matches the call input x_c. If so, P asserts that it is a correct call for v.

- $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$ and the call is an *inverse call.* Then the predictor first checks if there is a node $v \in \mathbb{V}$ where $\mathsf{aftlab}(v)$ was updated and $\mathsf{aftlab}(v)$ matches the call input x_c. If no such v exists, P queries the oracle, and checks if the answer is consistent with some updated $\mathsf{prelab}(v)$ $(v \in \mathbb{V})$.

[10]We assume that the encoding of the hint is unambiguous.
[11]Recall that there is an implicit chronological order for the calls.

- If one of the above checks succeed, then after recognizing the correct call for v, P updates $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v accordingly.

Claim 4.2 *Fix execution $\mathsf{A}^{\mathsf{ip}}(\mathbf{x};r)$ and round i, suppose the pre-labels are distinct, and the ex-post-facto pebbling $(\mathsf{P}_0,\ldots,\mathsf{P}_{t_{\mathsf{peb}}})$ is legal. For any round $\gamma > i$, assume the predictor successfully extracted ℓ-labels for all (first) critical calls (in $\mathsf{A}(\sigma_i)$) between round i and round γ. Then for any vertex $v \in \mathsf{P}_i \cup \cdots \cup \mathsf{P}_\gamma$, and any correct call for v in round γ (denote the call as $C(v)$), the predictor will correctly recognize the call $C(v)$ when simulating $\mathsf{A}(\sigma_i)$.*

Proof: The proof is deferred to Appendix B.3. ■

Next we show how to simulate $\mathsf{A}(\sigma_i)$ and predict ideal-primitive entries.

The predictor simulates $\mathsf{A}(\sigma_i)$ (which corresponds to the partial execution of A after round i) and keeps track of the labels $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v for every $v \in \mathbb{V}$. For each round $\gamma > i$, after receiving the calls from A, the predictor P does follows *sequentially.*

1. *Handling critical calls:* P first enumerates node $v_j \in \mathsf{P}_i$ according to *reverse topological order*[12] and checks the following: If the id_j-th call (i.e. v_j's *first* critical call) is in round γ and ℓ_{v_j} is unknown yet, the predictor uses the hint to extract the label ℓ_{v_j}. The extraction from a critical output call is trivial, thus we assume that the call is an ideal-primitive call. From the hint, the predictor knows that it is a correct ideal primitive call for a node $w_j \in \mathsf{succ}(v_j)$ where w_j can be extracted from the hint W_i. (Note that $w_j \in \mathsf{P}_\gamma$ by definition of critical calls.) The predictor first extracts $\mathsf{prelab}(w_j)$ from the call input/output:

    - If the call is forward, $\mathsf{prelab}(w_j)$ can be identified from the call input.

[12] $v_{|\mathsf{P}_i|}$ is picked first, then $v_{|\mathsf{P}_i|-1}$,..., and finally v_1.

- If $w_j \in \mathsf{P}_i$ and the call is an inverse call, since P chooses nodes in P_i according to reverse topological order, and in $\mathsf{A}(\sigma_i)$ the first critical call for w_j appears no later than any correct call for w_j, P must have already extracted ℓ_{w_j}, and thus the predictor can extract $\mathsf{prelab}(w_j)$ from ℓ_{w_j} and the call input without querying the oracle.
- If $w_j \notin \mathsf{P}_i$ and the call is an inverse call, P can query the oracle and extract the information of $\mathsf{prelab}(w_j)$ from the oracle answer.

Given w_j and $\mathsf{prelab}(w_j)$, if $\mathbb{IP} = \mathbb{CF}$ or $\mathbb{IP} = \mathbb{IC}$, P can directly extract the label ℓ_{v_j} from $\mathsf{prelab}(w_j)$ and b_j; if $\mathbb{IP} = \mathbb{RP}$ and v_j is the only predecessor of w_j, P can extract $\ell_{v_j} = \mathsf{prelab}(w_j)$; if $\mathbb{IP} = \mathbb{RP}$ and w_j has another predecessor u, we argue that ℓ_u was already known and thus the predictor can obtain the label $\ell_{v_j} = \mathsf{prelab}(w_j) \oplus \ell_u$.

- If $u \notin \mathsf{P}_i$, since the ex-post-facto pebbling is legal and $w_j \in \mathsf{P}_\gamma$[13], there exists a round γ' ($i < \gamma' < r$) such that $u \in \mathsf{P}_{\gamma'} \setminus \mathsf{P}_{\gamma'-1}$. By Claim 4.1, there is a correct call $C(u)$ for u in round γ'. Then by Claim 4.2, P will recognize the call $C(u)$ and update the label ℓ_u.
- If u is in P_i but the first critical call for u is before round γ (but after round i), then ℓ_u was already known before round γ.
- If u equals some node $v_k \in \mathsf{P}_i$ such that v_k and v_j have the same *first* critical call, since ℓ_{v_j} was unknown, it must be the case that $k < j$ and $h_k = \ell_{v_k}$, hence the predictors knew ℓ_u initially from the hint H_i.

2. *Handling correct calls for* P_i*:* For each node $v_j \in \mathsf{P}_i$ and each correct ideal-primitive call for v_j (note that by Claim 4.2, P correctly recognizes the call, as the ℓ-labels

[13]Here is the place where the second condition of the critical call definition becomes useful.

of (first) critical calls upto round γ were correctly extracted), since the predictor already knew ℓ_{v_j} after handling the first critical call for v_j,[14] she can answer the call *without* quering the ideal primitive:

- If $\mathbb{IP} = \mathbb{CF}$, then ℓ_{v_j} is the query answer.
- If $\mathbb{IP} = \mathbb{IC}$ and the call input has the value (k, x) where $k \in \{0,1\}^L \cup \{\perp\}$ and $x \in \{0,1\}^L$, the answer is $\ell_{v_j} \oplus x$ because $\ell_{v_j} = x \oplus \mathsf{ip}(k, x)$ for a forward call and $\ell_{v_j} = x \oplus \mathsf{ip}^{-1}(k, x)$ for an inverse call.
- If $\mathbb{IP} = \mathbb{RP}$ and the call input has the value x, the answer is $\ell_{v_j} \oplus x$ because $\ell_{v_j} = x \oplus \mathsf{ip}(x)$ for a forward call and $\ell_{v_j} = x \oplus \mathsf{ip}^{-1}(x)$ for an inverse call.

For each round $\gamma > i$, after checking correct/critical calls for all nodes in P_i, the predictor answers the other unanswered calls by making queries to the ideal primitive. Note that in round γ, for every node $v \in \mathsf{P}_i \cup \cdots \cup \mathsf{P}_\gamma$, if there is a correct ideal-primitive call for v, since P already extracted ℓ-labels for all (first) critical calls upto round γ, by Claim 4.2, P will recognize the call, get the call answer, then update the labels $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$, ℓ_v and the pre-labels of v's successors.

After executing $\mathsf{A}(\sigma_i)$, the predictor will compute $\mathsf{prelab}(v)$ for every $v \in \mathbb{V}$ according to topological order, and predict $\mathsf{ip}(\mathsf{prelab}(v))$ for every $v \in \mathsf{P}_i$. In particular, if $\mathbb{IP} = \mathbb{CF}$, $\mathsf{ip}(\mathsf{prelab}(v)) = \ell_v$; if $\mathbb{IP} = \mathbb{IC}$, let x be the last L-bit string of $\mathsf{prelab}(v)$, then $\mathsf{ip}(\mathsf{prelab}(v)) = x \oplus \ell_v$; if $\mathbb{IP} = \mathbb{RP}$, then $\mathsf{ip}(\mathsf{prelab}(v)) = \mathsf{prelab}(v) \oplus \ell_v$. Note that if $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens and the input is the hint h mentioned above, the predictor does not need to query $\mathsf{ip}(\mathsf{prelab}(v))$ for any $v \in \mathsf{P}_i$ as the answer can be computed from $\mathsf{prelab}(v)$ and the extracted label ℓ_v.

Correctness of the predictor. If $\mathrm{E}^{\lambda,\mathbb{IP}}_{\mathsf{pred}}$ happens and P's input is the hint mentioned above, the predictor will correctly predict $(\mathsf{prelab}(v), \mathsf{ip}(\mathsf{prelab}(v)))$ for every $v \in \mathsf{P}_i$ without

[14]Recall that in $\mathsf{A}(\sigma_i)$, there was no correct call for v before the round of the first critical call for v.

querying $\mathsf{ip}(\mathsf{prelab}(v))$. Recall that $\{\mathsf{prelab}(v)\}$ are distinct so that P also predicts $|\mathsf{P}_i|$ ideal-primitive entries.

First, we note that the labels being updated (including $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v for $v \in \mathbb{V}$) are correct by induction on the time order of updating. Initially, only the pre-labels of source vertices were updated which are correct. Assume all the labels being updated are correct up to now. A new label $\mathsf{prelab}(v)$ (or $\mathsf{aftlab}(v)$) will be updated because one of the following possibilities:

1. P recognizes the *first* correct call for $v \in \mathsf{P}_i$ from the hint C_i (and thus correct by the hint).

2. P recognizes a correct call for v by finding out that the call input/output matches the previously updated $\mathsf{aftlab}(v)$ (or $\mathsf{prelab}(v)$) which is correct by inductive hypothesis.

3. P computes the label according to topological order (at the end).

4. $\mathsf{prelab}(v)$ is updated because the ℓ-labels of v's predecessors were all updated previously (which are correct by inductive hypothesis).

Similarly, a new label ℓ_v will be updated either because the possibility 2 as above, or because P extracts ℓ_v from the *first* critical call for v (and thus correct by the hint). Note that the argumet above also implies that P will not output an incorrect prediction.

It remains to prove that P will never query $\mathsf{ip}(\mathsf{prelab}(v))$ for any $v \in \mathsf{P}_i$. First, when simulating $\mathsf{A}(\sigma_{\sigma_i})$, P will recognize the first correct call of v from the hint C_i and answer the call using the extracted label ℓ_v. Then $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v will all be updated. For the following correct calls, since $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v have been updated, P will recognize and answer the call without querying ip. Lastly, when computing $\mathsf{prelab}(v)$ for

$v \in \mathbb{V}$ according to topological order, P will not query $\mathsf{ip}(\mathsf{prelab}(v))$ for any $v \in \mathsf{P}_i$ as the answer will be computed from $\mathsf{prelab}(v)$ and the extracted label ℓ_v.

In summary, with probability at least $\Pr[\mathrm{E}_{\mathsf{pred}}^{\lambda,\mathbb{IP}}]$, there exists a short hint h where $P(h)$ correctly guesses $|\mathsf{P}_i|$ ideal-primitive entries, thus by Lemma 2.1 and Lemma 2.2, we have $\Pr[\mathrm{E}_{\mathsf{pred}}^{\lambda,\mathbb{IP}}] \leq 2^{-\lambda}$ and the lemma holds.

■

Putting All Things Together. For an execution $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$, we say $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$ is correct if the algorithm generates the correct graph function output at the end; we say $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$ is lucky if it is correct but there is a vertex $v \in \mathsf{sink}$ where A did not make any correct call for v before outputting the label ℓ_v. Note that if $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}, r)$ is correct but not lucky, the ex-post-facto pebbling will be successful. Moreover, with similar compression argument as in Lemma 4.1 and Lemma 4.2, the probability (over the uniform choice of ip and A's internal coins) that A is lucky is no more than $\epsilon_{\mathsf{luck}}(\mathbb{IP})$, where $\epsilon_{\mathsf{luck}}(\mathbb{CF}) = |\mathbb{V}|/2^L$ and $\epsilon_{\mathsf{luck}}(\mathbb{IC}) = \epsilon_{\mathsf{luck}}(\mathbb{RP}) = |\mathbb{V}|/2^{L-1}$.

In summary, for any algorithm A that correctly computes the graph function with probability $\epsilon_{\mathsf{A}} > 2 \cdot (\epsilon_{\mathsf{coll}}(\mathbb{IP}) + \epsilon_{\mathsf{legal}}(\mathbb{IP}) + \epsilon_{\mathsf{luck}}(\mathbb{IP}))$, we set $\lambda \in \mathbb{N}$ as the minimal integer such that $\epsilon(\lambda) = \epsilon_{\mathsf{coll}}(\mathbb{IP}) + \epsilon_{\mathsf{legal}}(\mathbb{IP}) + \epsilon_{\mathsf{luck}}(\mathbb{IP}) + 2^{-\lambda} \leq \epsilon_{\mathsf{A}}/2$. Then the following conditions hold with probability more than $\epsilon_{\mathsf{A}} - \epsilon(\lambda) \geq \epsilon_{\mathsf{A}}/2$:

1. The pre-labels are distinct from each other.

2. The ex-post-facto pebbling is legal and successful, hence

$$\sum_{i=1}^{t_{\mathsf{peb}}} |\mathsf{P}_i| \geq \mathsf{cc}(\mathbb{G})\,.$$

3. For every $i \in [t_{\mathsf{peb}}]$, it holds that $|\sigma_i| \geq |\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda$. Here $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ terminates

at round $t_{\mathsf{peb}} + 1$, σ_i is the input state for round $i + 1$, and P_i is the pebbling configuration in round i.

Thus we have

$$\begin{aligned}\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)) &\geq \sum_{i=1}^{t_{\mathsf{peb}}} |\sigma_i| \geq \sum_{i=1}^{t_{\mathsf{peb}}} (|\mathsf{P}_i| \cdot \beta_{\mathbb{IP}} - \lambda) \\ &\geq \left(\sum_{i=1}^{t_{\mathsf{peb}}} |\mathsf{P}_i| \right) \cdot \beta_{\mathbb{IP}} - t_{\mathsf{peb}} \cdot \lambda \\ &\geq \mathsf{cc}(\mathbb{G}) \cdot \beta_{\mathbb{IP}} - t_{\mathsf{peb}} \cdot \lambda \geq \mathsf{cc}(\mathbb{G}) \cdot \beta_{\mathbb{IP},\lambda} \,,\end{aligned}$$

where $\beta_{\mathbb{IP},\lambda} = \beta_{\mathbb{IP}} - \lambda$ as $\mathsf{cc}(\mathbb{G}) \geq t_{\mathsf{peb}}$.

Therefore we have

$$\mathbb{E}\left[\mathsf{CMC}(\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r))\right] \geq \frac{\epsilon_{\mathsf{A}}}{2} \cdot \beta_{\mathbb{IP},\lambda} \cdot \mathsf{cc}(\mathbb{G}) \,.$$

By plugging in the corresponding $\epsilon_{\mathsf{coll}}(\mathbb{IP})$, $\epsilon_{\mathsf{legal}}(\mathbb{IP})$, $\epsilon_{\mathsf{luck}}(\mathbb{IP})$ and $\beta_{\mathbb{IP}}$ for the ideal primitive $\mathbb{IP}$, we can find the optimal parameter $\lambda_{\mathbb{IP}}$, and compute

$$\beta(\epsilon_{\mathsf{A}}, \log |\mathbb{V}|) = \frac{\epsilon_{\mathsf{A}}}{2} \cdot (\beta_{\mathbb{IP}} - \lambda_{\mathbb{IP}}) \ ,$$

which leads to Theorem 4.1, Theorem 4.2 and Theorem 4.3. ■

4.4 iMHFs from Small-Block Labeling Functions

In this section, from any graph, we construct graph-based iMHFs from the small-block labeling functions built in Section 4.2.

Proposition 4.1 *Fix $L = 2^\ell$ and let $\mathcal{H}_{\mathsf{fix}}$ be the β-small-block labeling function built in Section 4.2. For any 2-indegree (predecessors-distinct) DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with $N =$*

2^n vertices and single source/sink, the graph labeling functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ is $(\mathsf{C}^{\parallel}_{\mathcal{F}}, \Delta_{\mathcal{F}}, N)$-memory hard, where for all $\epsilon \in [3 \cdot 2^{-L/10}, 1]$, it holds that

$$\mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon) \geq \Omega\left(\epsilon \cdot \mathsf{cc}(\mathbb{G}) \cdot L\right), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O\left(\frac{\mathsf{st}(\mathbb{G}, N)}{\mathsf{cc}(\mathbb{G})}\right).$$

Proof: The lower bound on $\mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon)$ is derived from Theorem 4.1, Theorem 4.2 and Theorem 4.3. The upper bound on $\Delta_{\mathcal{F}}(\epsilon)$ is obtained by showing a sequential algorithm that evaluates $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ using t steps and $s \cdot L$-bits of memory where $s \cdot t = \mathsf{st}(\mathbb{G}, N)$. The algorithm evaluates $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ according to a sequential pebbling strategy P (of $\mathbb{G}$) that has ST-complexity $\mathsf{st}(\mathbb{G}, N)$: Whenever P stores pebbles on a set $\mathsf{P}_i \subseteq \mathbb{V}$, the algorithm stores the graph labels for P_i (which takes no more than $s \cdot L$ bits of memory); whenever P puts a pebble on a vertex $v \in \mathbb{V}$, the algorithm invokes $\mathcal{H}_{\mathsf{fix}}$ and computes the graph label for vertex v. ■

The graph $\mathbb{G}$ in [14] has pebbling complexities $\mathsf{cc}(\mathbb{G}) = \Omega(N^2/\log N)$ and $\mathsf{st}(\mathbb{G}, N) = O(N^2/\log N)$, thus we obtain the following corollary.

Corollary 4.1 *Fix $L = 2^\ell$ and let $\mathcal{H}_{\mathsf{fix}}$ be the β-small-block labeling function built in Section 4.2. Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be the 2-indegree (predecessors-distinct) DAG in [14] (with $N = 2^n$ vertices). The graph labeling functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\mathsf{fix}}}$ is $(\mathsf{C}^{\parallel}_{\mathcal{F}}, \Delta_{\mathcal{F}}, N)$-memory hard, where for all $\epsilon \in [3 \cdot 2^{-L/10}, 1]$, it holds that*

$$\mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon) \geq \Omega\left(\frac{\epsilon \cdot N^2 \cdot L}{\log N}\right), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O(1).$$

4.5 Instantiation

We provide an instantiation of iMHFs from small-block labeling functions in Appendix C.

4.6 Conclusions and Open Problems

In this chapter, we initiate the study of developing data-independent MHFs from fast symmetric cryptographic primitives (e.g. AES). There are many open problems for future directions of research. For example, as mentioned in Chapter 3, can we provide memory hardness proof with respect to new metrics? (E.g., sustatined memory complexity [19] and bandwidth hardness [53].) Moreover, our proof framework only holds for graph-based iMHFs, and finding a similar framework for data-dependent functions (e.g. Scrypt) would be an interesting problem.

Chapter 5

Data Independent MHFs from Wide-Block Labeling Functions

Outline of the Chapter. In this chapter, we construct a family of graph-based iMHFs based on a primitive called *wide-block labeling functions*. The scheme enables us to obtain stronger memory hardness without increasing the description complexity of MHFs. We start by motivating the use of *wide-block labeling functions* in Section 5.1. Then in Section 5.2, we define and construct a family of wide-block labeling functions from small-block labeling functions. Next in Section 5.3, we prove that the construction satisfies pebbling reducibility with respect to depth-robust graphs. Finally in Section 5.4, we construct iMHFs from wide-block labeling functions. The full version of this work is available on [39].

5.1 Introduction

The MHFs construction in Chapter 4 already gives us iMHFs from fast primitives (e.g., AES), where the CMC lower bound is $\Omega(N^2L/\log N)$ for an N-vertices graph and

$L = 128$. Ideally, we target a CMC which is as high as possible, while keeping the evaluation of the function within a feasible margin for the legitimate users. An option is to use a bigger graph with high CC and small-block labeling functions. However, this can lead to large description size. A way out here is to choose a graph family that has succinct description. Unfortunately, as far as we know, practical hard-to-pebble graphs are randomly sampled and do not have a succinct description of the actual graph, only of the sampling process. To reduce description complexity of MHFs, a better option (adopted by practitioners) is to keep the same graph, but operate on larger blocks of size $W \gg L$, to ensure the time-memory product is now $N^2W/\log N$. To do this, we will provide a generic construction of a hash function with W-bit output using an underlying primitive with a shorter output. We refer to it as a wide-block labeling function. (Our design will have the added benefit of allowing for a variable W.) The resulting graph-based MHF scheme is memory hard as long as the graph $\mathbb{G}$ is sufficiently depth-robust.

We stress that all practical constructions implicitly design wide-block labeling functions, for which existing analyses provide no formal security guarantees, as they abstract them away as random oracles, which they are not. While we failed to provide either proofs or attacks on practical designs, initiating the study of provably secure constructions in the more realistic primitive-based setting is an important step.

5.2 Wide-Block Labeling Functions

In this section, we start by defining wide-block labeling functions. Then we present a generic approach that constructs wide-block labeling functions from small-block labeling functions.

Definition 5.1 (Wide-Block Labeling Functions) *For any ideal primitive* $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}/$

$\mathbb{RP}$, $\delta \in \mathbb{N}$ and $W = 2^w$, we say

$$\mathcal{H}_{\delta,w} = \{\, \mathsf{vlab}^{\mathsf{ip}}_{\gamma,w} : \{0,1\}^{\gamma W} \to \{0,1\}^{W} \,\}_{\mathsf{ip}\in\mathbb{IP},1\le\gamma\le\delta}$$

is a family of $\beta_{\delta,w}$-wide-block labeling functions if it satisfies the following property.

$\beta_{\delta,w}$-pebbling reducibility w.r.t. depth-robust graphs: *For any $\epsilon \in (0,1]$ and any δ-indegree (e,d)-depth robust (first-predecessor-distinct) DAG $\mathbb{G} = (\mathbb{V},\mathbb{E})$*[1]*, the graph functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ satisfies*

$$\mathsf{CMC}_\epsilon(\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}) \ge e \cdot (d-1) \cdot \beta_{\delta,w}(\epsilon, \log|\mathbb{V}|)\,,$$

where $\mathsf{CMC}_\epsilon(\cdot)$ is ϵ-cumulative-memory-complexity (Definition 2.1).

5.2.1 Construction of Wide-Block Labeling Functions

Next we show how to construct wide-block labeling functions from small-block labeling functions. The construction is the composition of two graph functions MIX and SSDR, which can be built from any small-block labeling functions.

Remark 5.1 *There are tailored-made variable-length hash functions available in the real-world, for example within Scrypt [33, 56] and Argon2 [37]. However, even by modeling the underlying block/stream cipher as an ideal primitive, we do not know how to prove the pebbling-reducibility of the hash functions. Hence we seek another construction of labeling functions.*

In the following context, we fix the indegree parameter $\delta \in \mathbb{N}$, ideal primitive length $L = 2^\ell$, output length $W = 2^w$ of the wide-block labeling functions, and denote as

[1]$\mathbb{G}$ has a single source/sink vertex.

$K = 2^k := W/L$ the ratio between W and L. We will omit these variables in notation when it is clear in the context.

We show how to construct the family of labeling functions $\mathcal{H}_{\delta,w}$. For any $1 \le \gamma \le \delta$, and any ideal primitive $\mathsf{ip} \in \mathbb{IP}$, we define the labeling function $\mathsf{vlab}^{\mathsf{ip}}_{\gamma,w} : \{0,1\}^{\gamma W} \to \{0,1\}^{W}$ as the composition of two functions, namely, $\mathsf{mix}^{\mathsf{ip}}_{\gamma} : \{0,1\}^{\gamma W} \to \{0,1\}^{W}$ and $\mathsf{ssdr}^{\mathsf{ip}}_{\delta} : \{0,1\}^{W} \to \{0,1\}^{W}$. More precisely, for an input vector $\mathbf{x} \in \{0,1\}^{\gamma W}$, we define the W-bit function output as

$$\mathsf{vlab}^{\mathsf{ip}}_{\gamma,w}(\mathbf{x}) := \mathsf{ssdr}^{\mathsf{ip}}_{\delta}(\mathsf{mix}^{\mathsf{ip}}_{\gamma}(\mathbf{x}))\,.$$

Next, we specify the functions $\mathsf{mix}^{\mathsf{ip}}_{\gamma}$ and $\mathsf{ssdr}^{\mathsf{ip}}_{\delta}$.

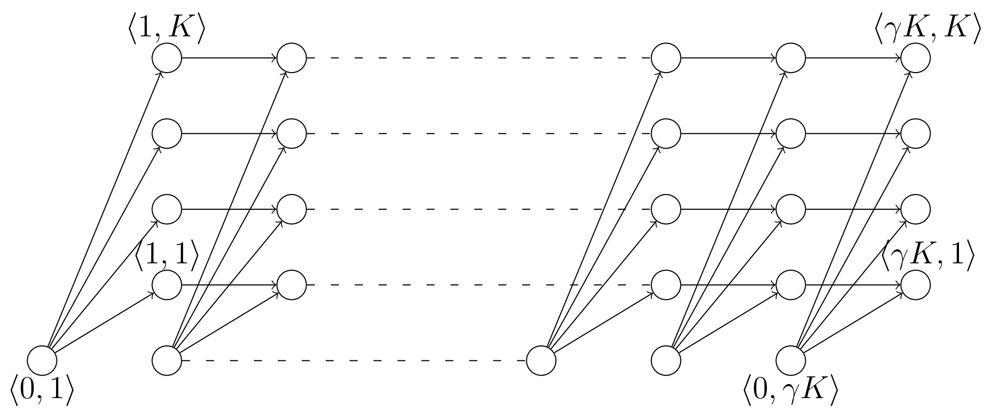
Component: MIX Functions. Denote as $\mathsf{flab}^{\mathsf{ip}} : \{0,1\}^{L} \cup \{0,1\}^{2L} \to \{0,1\}^{L}$ a small-block labeling function (Definition 4.1), and let $K := W/L$ be the ratio between W and L. We define

$$\mathsf{mix}^{\mathsf{ip}}_{\gamma} := \mathsf{F}^{\mathsf{ip}}_{\mathbb{G}^{\gamma,K}_{\mathsf{mix}}} : \{0,1\}^{\gamma W} \to \{0,1\}^{W=KL}$$

as the graph function (Section 2.5) built upon a DAG $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}$ and the labeling function $\mathsf{flab}^{\mathsf{ip}}$. (Note that we can use $\mathsf{flab}^{\mathsf{ip}}$ as the labeling function since the maximal indegree of $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}$ is 2.) The graph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}} = (\mathbb{V}^{\gamma,K}_{\mathsf{mix}}, \mathbb{E}^{\gamma,K}_{\mathsf{mix}})$ is defined as follows.

Nodes set: The set $\mathbb{V}^{\gamma,K}_{\mathsf{mix}}$ has γK source nodes (which represent the γK input blocks), and we use $\langle 0, j\rangle$ $(1 \le j \le \gamma K)$ to denote the jth source node. Besides, there are γK columns each with K nodes. We use $\langle i, j\rangle$ $(1 \le i \le \gamma K,\ 1 \le j \le K)$ to denote the node at the ith column and jth row. The K nodes at the last column are the sink nodes (which represent the K output blocks).

Edges set: The set $\mathbb{E}^{\gamma,K}_{\mathsf{mix}}$ has $\gamma K^2 + \gamma(K-1)K + K - 1$ edges. Each source node



$\langle 1, K\rangle$
$\langle \gamma K, K\rangle$
$\langle 1, 1\rangle$
$\langle \gamma K, 1\rangle$
$\langle 0, 1\rangle$
$\langle 0, \gamma K\rangle$

Figure 5.1: The graph $\mathbb{G}_{\mathsf{mix}}^{\gamma,K}$ for $K = 4$. We omitted the edges from $\langle 0, j\rangle$ to $\langle 1, j\rangle$ $(2 \le j \le K)$ for clarity of the figure.

$\langle 0, i\rangle$ $(1 \le i \le \gamma K)$ has K outgoing edges to the K nodes of column i, namely, $\{\langle i, j\rangle\}_{j\in[K]}$. For each column i $(1 \le i < \gamma K)$ and each row j $(1 \le j \le K)$, the node $\langle i, j\rangle$ has an outgoing edge to $\langle i + 1, j\rangle$ at the next column. Finally, each source node $\langle 0, j\rangle$ $(2 \le j \le K)$ has an outgoing edge to $\langle 1, j\rangle$. (The last $K - 1$ edges make sure that the K nodes at column 1 have distinct sets of predecessors (Section 2.5).)

Component: SSDR Functions. Denote as $\mathsf{flab}^{\mathsf{ip}} : \{0,1\}^L \cup \{0,1\}^{2L} \to \{0,1\}^L$ a small-block labeling function (Definition 4.1), and let $K := W/L$ be the ratio between W and L. Fix $\delta \in \mathbb{N}$, we define

$$\mathsf{ssdr}_\delta^{\mathsf{ip}} := \mathsf{F}^{\mathsf{ip}}_{\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}} : \{0,1\}^W \to \{0,1\}^W$$

as the graph function built upon the labeling function $\mathsf{flab}^{\mathsf{ip}}$ and a DAG $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$. (Note that we can use $\mathsf{flab}^{\mathsf{ip}}$ as the labeling function since the maximal indegree of $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$ is 2.) $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K} = (\mathbb{V}_{\mathsf{ssdr}}^{\delta,K}, \mathbb{E}_{\mathsf{ssdr}}^{\delta,K})$ is a *source-to-sink-depth-robust* graph (Definition 2.6) defined as follows.

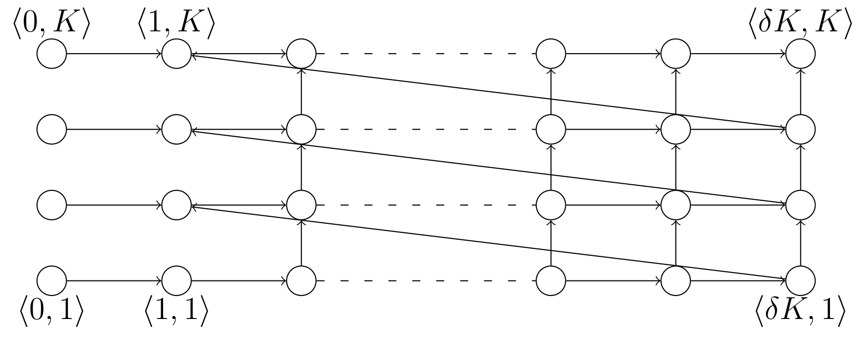


Figure 5.2: The graph $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ for $K = 4$

Nodes set: The set $\mathbb{V}^{\delta,K}_{\mathsf{ssdr}}$ has $K(1 + \delta K)$ vertices distributing across $1 + \delta K$ columns and K rows. For every $i \in \{0, \ldots, \delta K\}$ and every $j \in \{1, \ldots, K\}$, we use $\langle i, j\rangle$ to denote the node at column i and row j. The K nodes at column 0 are the source nodes and the K nodes at column δK are the sink nodes.

Edges set: The set $\mathbb{E}^{\delta,K}_{\mathsf{ssdr}}$ consists of 3 types of edges. The first type is called *horizontal edges*: For every i ($0 \leq i < \delta K$) and every j ($1 \leq j \leq K$), there is an edge from node $\langle i, j\rangle$ to node $\langle i + 1, j\rangle$. The second type is called *vertical edges*: For every i ($2 \leq i \leq \delta K$) and every j ($1 \leq j < K$), there is an edge from node $\langle i, j\rangle$ to node $\langle i, j + 1\rangle$. The third type is called *backward edges*: For every j ($1 \leq j < K$), there is an edge from node $\langle \delta K, j\rangle$ to node $\langle 1, j + 1\rangle$. In total, there are $\delta K^2 + \delta K \cdot (K - 1) < 2\delta K^2$ edges.

We prove a useful lemma showing that $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ is *source-to-sink-depth-robust.*

Lemma 5.1 *Fix any $K = 2^k \geq 4$ and $\delta \in \mathbb{N}$, the graph $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ is $(\frac{K}{4}, \frac{\delta K^2}{2})$-source-to-sink-depth-robust.*

Proof: Fix any nodes set S_{ssdr} in the graph $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}$ where $|S_{\mathsf{ssdr}}| \leq \frac{K}{4}$. Denote as

$$[\ell_1, r_1] \cup [\ell_2, r_2] \cup \cdots \cup [\ell_t, r_t]$$

the set of rows that have no intersection with S_{ssdr}, where

$$1 \le \ell_1 \le r_1 < \ell_2 \le r_2 < \cdots < \ell_t \le r_t \le K\,.$$

Let $i^* \in \{2, \ldots, \delta K\}$ be the first column in $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$ that has no intersection with S_{ssdr}. (Since $|S_{\mathsf{ssdr}}| \le K/4 < \delta K - 1$, such column must exist.) We construct a source-to-sink path P_{ssdr} in the graph $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K} - S_{\mathsf{ssdr}}$:

- For the first interval $[\ell_1, r_1]$, we construct a subpath (starting from the source node $\langle 0, \ell_1 \rangle$)

$$(\langle 0, \ell_1\rangle, \ldots, \langle \delta K, \ell_1\rangle, \langle 1, \ell_1+1\rangle, \ldots, \langle \delta K, \ell_1+1\rangle, \ldots, \langle 1, r_1\rangle, \ldots, \langle i^*, r_1\rangle)\ ,$$

  where $\langle i, j\rangle$ is the node at column i and row j. The length of the subpath is at least $(r_1 - \ell_1) \cdot \delta K + i^*$.

- For each interval $[\ell_j, r_j]$ ($2 \le j \le t-1$), we connect $\langle i^*, r_{j-1}\rangle$ to $\langle i^*, \ell_j\rangle$ through the column i^*, and construct a subpath

$$(\langle i^*, \ell_j\rangle, \ldots, \langle \delta K, \ell_j\rangle, \langle 1, \ell_j+1\rangle, \ldots, \langle \delta K, \ell_j+1\rangle, \ldots, \langle 1, r_j\rangle, \ldots, \langle i^*, r_j\rangle)\ .$$

  The length of the subpath is at least $(r_j - \ell_j) \cdot \delta K$.

- For the last interval $[\ell_t, r_t]$, we connect $\langle i^*, r_{t-1}\rangle$ to $\langle i^*, \ell_t\rangle$ through the column i^*, and construct a subpath (ending at the sink node $\langle \delta K, r_t\rangle$)

$$(\langle i^*, \ell_t\rangle, \ldots, \langle \delta K, \ell_t\rangle, \langle 1, \ell_t+1\rangle, \ldots, \langle \delta K, \ell_t+1\rangle, \ldots, \langle 1, r_t\rangle, \ldots, \langle \delta K, r_t\rangle)\ .$$

  The length of the subpath is at least $(r_t - \ell_t) \cdot \delta K + \delta K - i^*$.

Next we show that the length of P_{ssdr} is at least $\frac{\delta K^2}{2}$. Since $|S_{\mathsf{ssdr}}| \leq \frac{K}{4}$, the number of intervals t is no more than $K/4+1$ (as we can split the rows into at most $K/4+1$ intervals by removing $K/4$ rows). Moreover, the number of rows that have no intersection with S_{ssdr} is at least $K - K/4 = 3K/4$, that is,

$$\sum_{i=1}^{t}(r_i - \ell_i + 1) \geq \frac{3}{4} \cdot K.$$

Therefore the length of P_{ssdr} is at least

$$\begin{aligned}
|P_{\mathsf{ssdr}}| &\geq [(r_1 - \ell_1) \cdot \delta K + i^*] + \sum_{i=2}^{t-1}(r_i - \ell_i) \cdot \delta K + [(r_t - \ell_t) \cdot \delta K + \delta K - i^*] \\
&= (r_1 - \ell_1 + 1) \cdot \delta K + \sum_{i=2}^{t}(r_i - \ell_i) \cdot \delta K \\
&= \left[\sum_{i=1}^{t}(r_i - \ell_i + 1) - t + 1\right] \cdot \delta K \\
&\geq \left(\frac{3}{4} \cdot K - \frac{1}{4} \cdot K\right) \cdot \delta K = \frac{\delta K^2}{2}.
\end{aligned}$$

■

5.3 Wide-Block Labeling Functions: Pebbling Reducibility

In this section, we show that the labeling functions constructed in Section 5.2 satisfy pebbling reducibility with respect to depth-robust graphs. We will make use of the following notation.

Graph Composition: Given a graph $\mathbb{G}_1$ (with n_1 source nodes and n_2 sink nodes), and a graph $\mathbb{G}_2$ (with n_2 source nodes and n_3 sink nodes), we define $\mathbb{G}_1 \circ \mathbb{G}_2$ as the composition of $\mathbb{G}_1$ and $\mathbb{G}_2$, namely, we merge the ith sink node of $\mathbb{G}_1$ with the ith source node of $\mathbb{G}_2$ for each $i \in [n_2]$[2], and take the union of the rest parts of the graphs.

Theorem 5.1 *Fix $L = 2^\ell$, $W = 2^w \geq L$, and set $K := W/L$. Let $\mathcal{H}_{\mathsf{fix}}$ be any β_{fix}-small-block labeling functions. For any $\delta \in \mathbb{N}$, the labeling functions $\mathcal{H}_{\delta,w}$ constructed in Section 5.2 is $\beta_{\delta,w}$-pebbling-reducible w.r.t. (first-predecessor-distinct[3]) depth-robust graphs where*

$$\beta_{\delta,w}(\epsilon, \log|\mathbb{V}|) \geq \frac{\delta K^3}{8} \cdot \beta_{\mathsf{fix}}(\epsilon, \log|\mathbb{V}|)\,.$$

Here ϵ is in interval $(0,1]$ and $|\mathbb{V}|$ is the number of vertices in the graph.

Remark 5.2 (Generalization) *For the wide-block labeling functions constructed in Section 5.2, we make use of a specific graph $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$ that is source-to-sink depth robust. We emphasize, however, that any source-to-sink depth robust graphs suffice. In particular, by replacing $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$ with any 2-indegree DAG $\mathbb{G}^*$ where i) $\mathbb{G}^*$ has K source/sink nodes and ii) $\mathbb{G}^*$ is (e^*, d^*)-source-to-sink depth robust, the corresponding wide-block labeling functions is still β-pebbling-reducible, where*

$$\beta(\epsilon, \log|\mathbb{V}|) \geq e^* \cdot d^* \cdot \beta_{\mathsf{fix}}(\epsilon, \log|\mathbb{V}|)\,.$$

We leave finding new source-to-sink depth-robust graphs as an interesting direction for future work.

Remark 5.3 *Note that in Theorem 5.1, the pebbling reducibility only holds for depth-robust graphs. It is hard to directly link CMC and $\mathsf{cc}(\mathbb{G})$. The hardness lies in linking*

[2]We assume an implicit order for nodes in $\mathbb{G}_1$ and $\mathbb{G}_2$.

[3]See Section 2.5 for definition of first-predecessor-distinctness.

$\mathsf{cc}(\mathsf{Ext}_{\delta,W}(\mathbb{G}))$ *and* $\mathsf{cc}(\mathbb{G})$. *In particular, even if the gadget graph* $\mathbb{G}_{\delta,W}$ *has high CC, we do not know how to prove that* $\mathsf{cc}(\mathsf{Ext}_{\delta,W}(\mathbb{G})) \geq \mathsf{cc}(\mathbb{G}) \cdot \mathsf{cc}(\mathbb{G}_{\delta,W})$. *This is because we do not know how to transform a pebbling* P_1 *(of* $\mathsf{Ext}_{\delta,W}(\mathbb{G})$*) into a* legal *pebbling* P_2 *(of* $\mathbb{G}$*), and argue that* $\mathsf{cc}(\mathsf{P}_1)$ *is* at least $\mathsf{cc}(\mathsf{P}_2)$ *times* $\mathsf{cc}(\mathbb{G}_{\delta,W})$.

Proof: [of Theorem 5.1] Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be any (first-predecessor-distinct) (e, d)-depth-robust DAG with δ-indegree and single source/sink, let $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ be the graph functions built upon $\mathbb{G}$ and $\mathcal{H}_{\delta,w}$. It is sufficient to show that for every $\epsilon \in (0, 1]$,

$$\mathsf{CMC}_\epsilon(\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}) \geq \beta_{\mathsf{fix}}(\epsilon, \log|\mathbb{V}|) \cdot \frac{\delta K^3}{8} \cdot e \cdot (d-1)\,.$$

By opening the underlying graph structure of $\mathcal{H}_{\delta,w}$, we see that $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ is also a graph function built upon functions $\mathcal{H}_{\mathsf{fix}}$ and an extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ that has the following properties.

- **Nodes Expansion:** Every node $v \in \mathbb{V}$ in the original graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is expanded into K nodes, that is,

$$\mathsf{copy}(v) := \left(v^{(1)}, \ldots, v^{(K)}\right)\,.$$

- **Neighborhood Connection:** For every *non-source* node $v \in \mathbb{V} - \mathsf{src}(\mathbb{G})$, denote as $\mathsf{pred}(v) := (u_1, \ldots, u_\gamma)$ the predecessors of v in $\mathbb{G}$. In $\mathsf{Ext}_{\delta,K}(\mathbb{G})$, there is a subgraph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v)$ that connects

$$\mathsf{neighbor}(v) := \{\mathsf{copy}(u_1), \ldots, \mathsf{copy}(u_\gamma)\}$$

  to the set $\mathsf{copy}(v)$, where $\mathsf{neighbor}(v)$ (and $\mathsf{copy}(v)$) are the source nodes (and the sink nodes) of $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v)$, respectively. Note $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v)$ has the identical graph structure with the composition of $\mathbb{G}_{\mathsf{mix}}^{\gamma,K}$ and $\mathbb{G}_{\mathsf{ssdr}}^{\delta,K}$.

By first-predecessor-distinctness of $\mathbb{G}$ and by the graph structure of the MIX graph, it holds that $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is a predecessors-distinct graph with 2-indegree. Next, we will show that the extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is $(e,(d-1)\cdot\delta K)$-depth-robust for $K \in \{1,2\}$ and $(eK/4,(d-1)\cdot\delta K^2/2)$-depth-robust for $K \geq 4$. By Lemma 2.3, for any $K = 2^k \geq 1$, we have

$$\mathsf{cc}(\mathsf{Ext}_{\delta,K}(\mathbb{G})) \geq \frac{\delta K^3}{8} \cdot e \cdot (d-1)\,.$$

Thus by β_{fix}-pebbling reducibility of $\mathcal{H}_{\mathsf{fix}}$[4], we obtain Theorem 5.1.

Before proving the depth-robustness of the extension graph, we introduce a useful notation called *meta-node*. Intuitively, nodes set maps each vertex of the original graph $\mathbb{G}$ to a set of vertices in $\mathsf{Ext}_{\delta,K}(\mathbb{G})$.

Meta-Node: We define meta-node $\mathsf{nodes}(v)$ for every node $v \in \mathbb{V}$: For every *non-source* node $v \in \mathbb{V} - \mathsf{src}(\mathbb{G})$, we define $\mathsf{nodes}(v)$ as the set of vertices in the graph $\mathbb{G}_{\mathsf{mix}}^{\gamma,K}(v) \circ \mathbb{G}_{\mathsf{ssdr}}^{\delta,K}(v) - \mathsf{neighbor}(v)$; for every *source* node $v \in \mathsf{src}(\mathbb{G})$, we define $\mathsf{nodes}(v) := \mathsf{copy}(v)$. Note that for any $u, v \in \mathbb{V}$ such that $u \neq v$, the sets $\mathsf{nodes}(u)$ and $\mathsf{nodes}(v)$ are disjoint.

Depth-robustness of the Extension Graph: Next we show the depth robustness of the extension graph. We first consider the simpler case where $K \in \{1,2\}$. (In the following context, for a graph $\mathbb{G} = (\mathbb{V},\mathbb{E})$, we sometimes think $\mathbb{G} = \mathbb{V} \cup \mathbb{E}$ as the union of set $\mathbb{V}$ and $\mathbb{E}$ if there is no ambiguity.)

Lemma 5.2 *For any $K = 2^k \in \{1,2\}$ and (e,d)-depth robust DAG $\mathbb{G} = (\mathbb{V},\mathbb{E})$ that has maximal indegree $\delta \in \mathbb{N}$, the corresponding extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is $(e,(d-1)\cdot\delta K)$-depth-robust.*

[4]Note that β_{fix}-pebbling reducibility holds for multi-sources graphs.

Proof: For any nodes subset $S_{\mathsf{ext}} \subseteq \mathsf{Ext}_{\delta,K}(\mathbb{G})$ such that $|S_{\mathsf{ext}}| \leq e$, we show that $\mathsf{depth}(\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}) \geq (d-1) \cdot \delta K$, which finishes the proof.

First, we define a set

$$S := \{ v \in \mathbb{V} : |\, \mathsf{nodes}(v) \cap S_{\mathsf{ext}}| \geq 1 \} .$$

Since $\{\mathsf{nodes}(v)\}_{v\in\mathbb{V}}$ are disjoint and $|S_{\mathsf{ext}}| \leq e$, we have $|S| \leq e$. Then by (e,d)-depth robustness of $\mathbb{G}$, there exists a path $P = (v_1, \ldots, v_d)$ in the graph $\mathbb{G} - S$.

Next, given the path $P = (v_1, \ldots, v_d) \subseteq \mathbb{G} - S$, we show a path (with length at least $\delta K \cdot (d-1)$) in the graph $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$. For every $i \in [d-1]$, since $v_i \notin S$ and $v_{i+1} \notin S$, by definition of S, we have $(\mathsf{nodes}(v_{i+1}) \cup \mathsf{copy}(v_i)) \cap S_{\mathsf{ext}} = \emptyset$. Hence there exists a path (with length at least δK) from the node $v_i^{(1)} \in \mathsf{copy}(v_i)$ to the node $v_{i+1}^{(1)} \in \mathsf{copy}(v_{i+1})$. (The path starts from $v_i^{(1)}$, then comes to the first row of $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_{i+1}) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_{i+1})$, and goes through the first row until it reaches $v_{i+1}^{(1)}$.) By concatenating the $(d-1)$ paths, we obtain a path (with length at least $(d-1) \cdot \delta K$) in the graph $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$. ■

Next, we consider the more general case where $K \geq 4$.

Lemma 5.3 *For any $K = 2^k \geq 4$ and any (e,d)-depth robust DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with maximal indegree $\delta \in \mathbb{N}$, the corresponding extension graph $\mathsf{Ext}_{\delta,K}(\mathbb{G})$ is $(\frac{K}{4} \cdot e, \frac{\delta K^2}{2} \cdot (d-1))$-depth-robust.*

Proof: For any nodes subset $S_{\mathsf{ext}} \subseteq \mathsf{Ext}_{\delta,K}(\mathbb{G})$ such that $|S_{\mathsf{ext}}| \leq \frac{K}{4} \cdot e$, we show that $\mathsf{depth}(\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}) \geq \frac{\delta K^2}{2} \cdot (d-1)$, which finishes the proof.

Step 1: From S_{ext}, we first derive a set of nodes $S \subseteq \mathbb{V}$ in the graph $\mathbb{G}$, and find a long path in $\mathbb{G} - S$.

Claim 5.1 *Define a set*

$$S := \{ \, v \in \mathbb{V} : | \, \mathsf{nodes}(v) \cap S_{\mathsf{ext}}| \geq \frac{K}{4} \, \} \ ,$$

there exists a d-path[5] $P = (v_1, \ldots, v_d)$ *in the graph* $\mathbb{G} - S$.

Proof: We first show that $|S| \leq e$: Since $|S_{\mathsf{ext}}| \leq \frac{K}{4} \cdot e$ and $\{\mathsf{nodes}(v)\}_{v \in \mathbb{V}}$ are disjoint, we have

$$\frac{K}{4} \cdot e \geq |S_{\mathsf{ext}}| \geq \sum_{v \in S} |\mathsf{nodes}(v) \cap S_{\mathsf{ext}}| \geq |S| \cdot \frac{K}{4} \ ,$$

where the last inequality is from the definition of S. By dividing the terms by $\frac{K}{4}$, we have that $|S| \leq e$. Since the graph $\mathbb{G}$ is (e, d)-depth robust, we conclude that there is a d-path $P = (v_1, \ldots, v_d)$ in the graph $\mathbb{G} - S$. ■

Step 2: Given the path $P = (v_1, \ldots, v_d) \subseteq \mathbb{G} - S$, next in Lemma 5.4 we show that for every $i \in [d-1]$, there exists a long path from $\mathsf{copy}(v_i)$ to $\mathsf{copy}(v_{i+1})$ in the graph $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$, then by connecting the $d-1$ paths, we obtain a path with length at least $\frac{\delta K^2}{2} \cdot (d-1)$, hence finish the proof of Lemma 5.3. Note that the path extraction from $\mathsf{copy}(v_i)$ to $\mathsf{copy}(v_{i+1})$ consists of two steps: First we exploit the structure of SSDR graphs and obtain a long path ending at a node in $\mathsf{copy}(v_{i+1})$, then we exploit the structure of MIX graphs and connect $\mathsf{copy}(v_i)$ to the source node of the obtained path.

Lemma 5.4 *Given the path* $P = (v_1, \ldots, v_d) \subseteq \mathbb{G} - S$ *(obtained in Claim 5.1) and the graph* $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$, *there exists a nodes sequence* $(u_1, \ldots, u_d)$ *(where* $u_i \in \mathsf{copy}(v_i) - S_{\mathsf{ext}}$ *for every* $i \in [d]$*), such that for every* $i \in [d-1]$, *there is a path (with length at least* $\frac{\delta K^2}{2}$*) that connects* u_i *and* u_{i+1} *in* $\mathsf{Ext}_{\delta,K}(\mathbb{G}) - S_{\mathsf{ext}}$.

[5] A d-path is a path with d vertices.

Proof: We first show that there exists a path (with length at least $\frac{\delta K^2}{2}$) from some node $u_1 \in \mathsf{copy}(v_1) - S_{\mathsf{ext}}$ to some node $u_2 \in \mathsf{copy}(v_2) - S_{\mathsf{ext}}$. (The arguments for $u_2, \ldots, u_d$ will be similar.) The idea consists of two steps: First, we find a *long* source-to-sink path in $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$; second, we connect u_1 to the starting node of the source-to-sink path. We also require that the path does not intersect with S_{ext}.

Finding the Source-to-Sink Path: We first define a set S_{ssdr} and find a source-to-sink path in $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - S_{\mathsf{ssdr}}$.

Claim 5.2 *Define* $\mathsf{row} \subseteq [K]$ *as the set of row indices where* j *is in* row *if and only if the* j*th row of* $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) - \mathsf{neighbor}(v_2)$ *has intersection with* S_{ext}*. Define a set*

$$S_{\mathsf{ssdr}} := \{ \langle 0, j \rangle_{\mathsf{ssdr}} \}_{j \in \mathsf{row}} \cup \left(S_{\mathsf{ext}} \cap \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) \right) ,$$

where $\langle 0, j \rangle_{\mathsf{ssdr}}$ *is the source node of* $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$ *at row* j*. The graph* $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - S_{\mathsf{ssdr}}$ *has a source-to-sink path of* $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$ *with length at least* $\frac{\delta K^2}{2}$*.*

Proof: Since $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$ is $(\frac{K}{4}, \frac{\delta K^2}{2})$-source-sink-depth-robust for $K \geq 4$ (Lemma 5.1), it is sufficient to prove that $|S_{\mathsf{ssdr}}| \leq \frac{K}{4}$. In particular, we have

$$|S_{\mathsf{ssdr}}| \leq |S_{\mathsf{ext}} \cap \left(\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - \mathsf{neighbor}(v_2) \right) | = |S_{\mathsf{ext}} \cap \mathsf{nodes}(v_2)| < \frac{K}{4} .$$

The first *equality* holds as $\mathsf{nodes}(v_2) = \mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - \mathsf{neighbor}(v_2)$ for $v_2 \in \mathbb{V} - \mathsf{src}(\mathbb{G})$; the last inequality holds because $v_2 \notin S$. The first inequality holds because

$$\begin{aligned} |S_{\mathsf{ssdr}}| &\leq |\mathsf{row}| + |S_{\mathsf{ext}} \cap (\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - \mathsf{src}(\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2))| \\ &\leq |S_{\mathsf{ext}} \cap (\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) - \mathsf{neighbor}(v_2))| + |S_{\mathsf{ext}} \cap (\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - \mathsf{src}(\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2))| \\ &= |S_{\mathsf{ext}} \cap \left(\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) \circ \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - \mathsf{neighbor}(v_2) \right) | . \end{aligned}$$

The first inequality is by definition of S_{ssdr}; the second inequality is by definition of row; the last equality holds because $\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2) - \mathsf{src}(\mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2))$ and $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) - \mathsf{neighbor}(v_2)$ are disjoint. ■

Paths Connection: Next, we show how to connect an arbitrary node in $\mathsf{copy}(v_1) - S_{\mathsf{ext}}$ to the starting node of the source-to-sink path. Here we exploit the structure of MIX graphs.

Claim 5.3 *Denote as u_1 an arbitrary node in the non-empty set $\mathsf{copy}(v_1) - S_{\mathsf{ext}}$[6] and $P_{\mathsf{ssdr}}(v_2)$ the source-to-sink path obtained in Claim 5.2. The graph $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) - S_{\mathsf{ext}}$ has a path from u_1 to the starting node of $P_{\mathsf{ssdr}}(v_2)$.*

Proof: Denote as $\langle 0, j^* \rangle_{\mathsf{ssdr}} \in \mathbb{G}^{\delta,K}_{\mathsf{ssdr}}(v_2)$ the starting node of $P_{\mathsf{ssdr}}(v_2)$. The index j^* is not in the set row (defined in Claim 5.2) as the path $P_{\mathsf{ssdr}}(v_2)$ has no intersection with S_{ssdr} but $\{ \langle 0, j \rangle_{\mathsf{ssdr}} \}_{j \in \mathsf{row}} \subseteq S_{\mathsf{ssdr}}$. Hence the j^*th row of $\mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2) - \mathsf{neighbor}(v_2)$ has no intersection with S_{ext} by definition of row. Thus given any node $u_1 \in \mathsf{copy}(v_1) - S_{\mathsf{ext}}$, we can write u_1 as $\langle 0, i^* \rangle_{\mathsf{mix}} \in \mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2)$ for some $i^* \in [\gamma K]$, and the path

$$(\langle 0, i^* \rangle_{\mathsf{mix}}, \langle i^*, j^* \rangle_{\mathsf{mix}}, \langle i^*+1, j^* \rangle_{\mathsf{mix}} \ldots \langle \gamma K, j^* \rangle_{\mathsf{mix}}) \subseteq \mathbb{G}^{\gamma,K}_{\mathsf{mix}}(v_2)$$

connects u_1 (through row j^*) to the starting node of $P_{\mathsf{ssdr}}(v_2)$ (as $\langle \gamma K, j^* \rangle_{\mathsf{mix}} = \langle 0, j^* \rangle_{\mathsf{ssdr}}$). ■

Finally, using identical arguments, we can show that for every $i \in \{2, \ldots, d-1\}$, there exists a path (with length at least $\frac{\delta K^2}{2}$) from the node $u_i \in \mathsf{copy}(v_i) - S_{\mathsf{ext}}$ to some node $u_{i+1} \in \mathsf{copy}(v_{i+1}) - S_{\mathsf{ext}}$. Hence we finish the proof of Lemma 5.4. ■

From Lemma 5.4, we obtain Lemma 5.3. ■

From Lemma 5.2 and Lemma 5.3, we obtain Theorem 5.1. ■

[6]Note that $\mathsf{copy}(v_1) - S_{\mathsf{ext}}$ is non-empty because $|\mathsf{copy}(v_1) \cap S_{\mathsf{ext}}| \leq |\mathsf{nodes}(v_1) \cap S_{\mathsf{ext}}| < \frac{K}{4} < K = |\mathsf{copy}(v_1)|$. The first inequality holds as $\mathsf{copy}(v_1) \subseteq \mathsf{nodes}(v_1)$, the second inequality holds because $v_1 \notin S$.

5.4 iMHFs from Wide-Block Labeling Functions

In this section, from any depth-robust graph, we construct graph-based iMHFs based on the wide-block labeling functions built in Section 5.2.

Theorem 5.2 *Fix $L = 2^\ell$, $W = 2^w$ and set $K := W/L$. Let $\mathcal{H}_{\delta,w}$ be the wide-block labeling functions built in Section 5.2. For any (first-predecessor-distinct[7]) (e,d)-depth-robust DAG $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with δ-indegree and $N = 2^n$ vertices, the graph-based iMHFs family $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ is $(\mathsf{C}^{\parallel}_{\mathcal{F}}, \Delta_{\mathcal{F}}, 2\delta N K^2)$-memory hard, where for all $\epsilon \in [3 \cdot 2^{-L/10}, 1]$, it holds that*

$$\mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon) \geq \Omega\left(\epsilon \cdot e \cdot d \cdot \delta \cdot K^2 \cdot W\right), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O\left(\frac{\mathsf{st}(\mathbb{G}, N)}{e \cdot d}\right).$$

The graph $\mathbb{G}$ in [14] is a first-predecessor-distinct graph with $(\Omega(N/\log N), \Omega(N))$-depth-robustness and ST-complexity $\mathsf{st}(\mathbb{G}, N) = O(N^2/\log N)$, thus we obtain the following corollary.

Corollary 5.1 *Fix $L = 2^\ell$, $W = 2^w$ and set $K := W/L$. Let $\mathcal{H}_{2,w}$ be the labeling functions built in Section 5.2 and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be the 2-indegree DAG in [14] (with $N = 2^n$ vertices). The graph labeling functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{2,w}}$ is $(\mathsf{C}^{\parallel}_{\mathcal{F}}, \Delta_{\mathcal{F}}, O(NK^2))$-memory hard, where for all $\epsilon \in [3 \cdot 2^{-L/10}, 1]$, it holds that*

$$\mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon) \geq \Omega\left(\frac{\epsilon \cdot N^2 \cdot K^2 \cdot W}{\log N}\right), \qquad \Delta_{\mathcal{F}}(\epsilon) \leq O(1).$$

Proof: [of Theorem 5.2] By applying Theorem 4.1, Theorem 4.2, Theorem 4.3 and Theorem 5.1, we obtain that $\mathcal{H}_{\delta,w}$ is $\Omega(\epsilon\delta K^2 W)$-pebbling reducible w.r.t. depth-robust graphs and thus

$$\mathsf{C}^{\parallel}_{\mathcal{F}}(\epsilon) \geq \Omega\left(\epsilon \cdot e \cdot d \cdot \delta \cdot K^2 \cdot W\right).$$

[7]See Section 2.5 for definition of first-predecessor-distinctness.

Since the 2-degree graph $\mathbb{G}$ in [14] is $(\Omega(N/\log N), \Omega(N))$-depth-robust, we obtained the desired lower bound on $\mathsf{C}^{\|}_{\mathcal{F}}(\epsilon)$.

To argue efficiency gap, we show that $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ can be evaluated by a sequential algorithm with space-time complexity $O(\mathsf{st}(\mathbb{G}, N) \cdot \delta K^2 W)$ (for any ip and input x), thus $\Delta_{\mathcal{F}}(\epsilon) \leq O(\mathsf{st}(\mathbb{G}, N)/(ed))$.

We present the sequential algorithm for evaluating $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ in two steps. First, in Section 5.4.1, we show a simple sequential algorithm $\mathsf{A}_{\delta,w}$ for evaluating the labeling function $\mathcal{H}_{\delta,w}$. Second, in Section 5.4.2, we present a sequential algorithm for evaluating $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ by exploiting $\mathsf{A}_{\delta,w}$ and a sequential pebbling of $\mathbb{G}$ in a generic fashion. ■

5.4.1 Sequential Evaluation of $\mathcal{H}_{\delta,w}$.

We show a sequential algorithm that efficiently evaluates $\mathcal{H}_{\delta,w}$.

Lemma 5.5 *Fix $\delta \in \mathbb{N}$, $L = 2^\ell$, $W = 2^w$ and set $K := W/L$. $\mathcal{H}_{\delta,w}$ can be evaluated by a sequential algorithm in time complexity $\mathsf{T}_{\mathcal{H}}(\delta, w)$ and space complexity $\mathsf{S}_{\mathcal{H}}(\delta, w)$, where*

$$\mathsf{T}_{\mathcal{H}}(\delta, w) = 2\delta K^2\,, \qquad \mathsf{S}_{\mathcal{H}}(\delta, w) = (\delta+1)\cdot W\,.$$

Proof: For any $\gamma \in [\delta]$ and $\mathsf{ip} \in \mathbb{IP}$, we show that there is a sequential algorithm[8] $\mathsf{A}^{\mathsf{ip}}_{\gamma}$ that evaluates the labeling function $\mathsf{vlab}^{\mathsf{ip}}_{\gamma,w} : \{0,1\}^{\gamma W} \to \{0,1\}^{W}$ constructed in Section 5.2, where the time and space complexity are

$$\mathsf{Tm}(\mathsf{A}^{\mathsf{ip}}_{\gamma}) \leq (\gamma+\delta)\cdot K^2\,, \qquad \mathsf{Spc}(\mathsf{A}^{\mathsf{ip}}_{\gamma}) \leq (\delta+1)\cdot W\,.$$

Nodes Notations: To remove ambiguity, we use $\langle i, j\rangle_{\mathsf{mix}}$ to denote the node in $\mathbb{V}^{\gamma,K}_{\mathsf{mix}}$, and use $\langle i, j\rangle_{\mathsf{ssdr}}$ to denote the node in $\mathbb{V}^{\delta,K}_{\mathsf{ssdr}}$ at column i and row j.

[8]The algorithm's memory access pattern is independent of the input and the ideal primitive $\mathsf{ip} \in \mathbb{IP}$.

The Sequential Algorithm $\mathsf{A}^{\mathsf{ip}}_{\gamma}$: Let $\mathbf{x} = (x_1, \ldots, x_{\gamma K}) \in \{0,1\}^{\gamma KL}$ be any input vector. The algorithm first sets $x_j \in \{0,1\}^L$ as the label of node $\langle 0, j\rangle_{\mathsf{mix}} \in \mathbb{V}^{\gamma,K}_{\mathsf{mix}}$ $(1 \le j \le \gamma K)$ and computes the MIX function $\mathsf{mix}^{\mathsf{ip}}_{\gamma}(\mathbf{x})$ as follows.

Computing $\mathsf{mix}^{\mathsf{ip}}_{\gamma}(\mathbf{x})$. The algorithm computes $\mathsf{mix}^{\mathsf{ip}}_{\gamma}(\mathbf{x})$ through γK stages, where the ith stage (that takes K sequential steps) computes the labels for column i $(1 \le i \le \gamma K)$. We use $\ell_{i,j}$ to denote the label of node $\langle i, j\rangle_{\mathsf{mix}} \in \mathbb{V}^{\gamma,K}_{\mathsf{mix}}$.

- **Stage 1:** At the first step, the algorithm computes and stores the label $\ell_{1,1} := \mathsf{flab}^{\mathsf{ip}}(\ell_{0,1})$. At step $j \in \{2, \ldots, K\}$, the algorithm computes and stores the label $\ell_{1,j} := \mathsf{flab}^{\mathsf{ip}}(\ell_{0,1}, \ell_{0,j})$.

- **Stage i $(2 \le i \le \gamma K)$:** At step $j \in [K]$, the algorithm computes and stores the label $\ell_{i,j} := \mathsf{flab}^{\mathsf{ip}}(\ell_{0,i}, \ell_{i-1,j})$ and then forget the label $\ell_{i-1,j}$.

- **Output Phase:** The algorithm outputs $(\ell_{\gamma K,1}, \ldots, \ell_{\gamma K,K})$.

The sequential time complexity is $t_{\mathsf{mix}}(\gamma) = \gamma K^2$. The memory size is $s_{\mathsf{mix}}(\gamma) = (\gamma+1) \cdot K \cdot L = (\gamma+1) \cdot W$ as the algorithm only stores the input vector plus at most K labels at each step.

Let $\mathbf{y} = (y_1, \ldots, y_K)$ be the output vector of $\mathsf{mix}^{\mathsf{ip}}_{\gamma}(\mathbf{x})$. The algorithm then sets $y_j \in \{0,1\}^L$ as the label of node $\langle 0, j\rangle_{\mathsf{ssdr}} \in \mathbb{V}^{\delta,K}_{\mathsf{ssdr}}$ $(1 \le j \le K)$ and computes the SSDR function $\mathsf{ssdr}^{\mathsf{ip}}_{\delta}(\mathbf{y})$ as follows.

Computing $\mathsf{ssdr}^{\mathsf{ip}}_{\delta}(\mathbf{y})$. The algorithm computes $\mathsf{ssdr}^{\mathsf{ip}}_{\delta}(\mathbf{y})$ through K stages, where the jth $(1 \le j \le K)$ stage takes δK sequential steps and computes the labels of row j. We use $\ell_{i,j}$ to denote the label of node $\langle i, j\rangle_{\mathsf{ssdr}} \in \mathbb{V}^{\delta,K}_{\mathsf{ssdr}}$.

- **Stage 1:** At step i $(1 \le i \le \delta K)$, the algorithm computes and stores $\ell_{i,1} := \mathsf{flab}^{\mathsf{ip}}(\ell_{i-1,1})$. At the end of the stage, the algorithm forget the label $\ell_{0,1}$.

- **Stage** j **$(2 \leq j \leq K)$:** At step 1, the algorithm computes and stores $\ell_{1,j} := \mathsf{flab}^{\mathsf{ip}}(\ell_{0,j}, \ell_{\delta K, j-1})$ and then forget the label $\ell_{0,j}$. At step i $(2 \leq i < \delta K)$, the algorithm computes and stores $\ell_{i,j} := \mathsf{flab}^{\mathsf{ip}}(\ell_{i-1,j}, \ell_{i,j-1})$ and forget the label $\ell_{i,j-1}$ at row $j-1$. Finally, it computes and stores $\ell_{\delta K,j} := \mathsf{flab}^{\mathsf{ip}}(\ell_{\delta K-1,j}, \ell_{\delta K,j-1})$.

- **Output Phase:** The algorithm outputs $(\ell_{\delta K,1}, \dots, \ell_{\delta K,K})$.

The sequential time complexity is $t_{\mathsf{ssdr}}(\delta) = \delta K^2$. The memory size is $s_{\mathsf{ssdr}}(\delta) = (\delta+1) \cdot K \cdot L = (\delta+1) \cdot W$: When computing $\ell_{i,j}$ (where $1 \leq i \leq \delta K$ and $1 \leq j \leq K$), the algorithm stores the first $j-1$ labels of column δK, the last $K-j+1$ labels of column 0, the first $i-1$ labels of row j, and the last $\delta K - i + 1$ labels of row $j-1$. In total, there are at most $(\delta+1) \cdot K$ labels. ■

5.4.2 Sequential Evaluation of $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$

Next we show the algorithm for evaluating $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$.

Lemma 5.6 *Fix $\delta \in \mathbb{N}$, $L = 2^\ell$, $W = 2^w$ and set $K := W/L$. Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be any DAG with N vertices and maximal indegree δ. The graph functions $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$ can be evaluated by a sequential algorithm with space-time complexity $O(\mathsf{st}(\mathbb{G}, N) \cdot \delta K^2 W)$.*

Proof: Let P be a sequential pebbling strategy of $\mathbb{G}$ that uses t_{peb} steps and s_{peb} pebbles such that $t_{\mathsf{peb}} \cdot s_{\mathsf{peb}} = \mathsf{st}(\mathbb{G}, N)$. There is a straight-forward sequential algorithm for evaluating $\mathcal{F}_{\mathbb{G},\mathcal{H}_{\delta,w}}$: Whenever P stores pebbles on a set $\mathsf{P}_i \subseteq \mathbb{V}$, the algorithm stores the graph labels for P_i (which takes no more than $s_{\mathsf{peb}} \cdot W$ bits of memory); whenever P puts a pebble on a vertex $v \in \mathbb{V}$, the algorithm invokes the labeling function $\mathcal{H}_{\delta,w}$ and computes the graph label for vertex v. By Lemma 5.5, the computation takes at most $O(\delta K^2)$ sequential steps and $(s_{\mathsf{peb}} + \delta) \cdot W = O(s_{\mathsf{peb}} \cdot W)$ bits of memory. (Note that s_{peb} is at least δ because there exists a vertex v with indegree δ, and pebbling v

requires the pebbling of v's predecessor vertices at the previous step.) In summary, the time complexity is $O(t_{\mathsf{peb}} \cdot \delta K^2)$ and the space complexity is $O(s_{\mathsf{peb}} \cdot W)$, and thus the ST-complexity is $O(\mathsf{st}(\mathbb{G}, N) \cdot \delta K^2 W)$. ■

5.5 Instantiation

We provide an instantiation of iMHFs from wide-block labeling functions in Appendix C.

5.6 Conclusions and Open Problems

Our result takes the first step for developing *provably-secure* MHFs from *wide-block labeling functions*. However, this is not the end of the story and there are still many open problems waiting to be answered.

1. Can we provide security analyses for practical constructions, e.g., the wide-block labeling functions underlying SCRYPT/Argon2 schemes?

2. Can we give memory hardness proof for new metrics? For example, sustatined memory complexity [19] or bandwidth hardness [53].

3. Can we optimize the CMC for a given budget of evaluation time? If the sequential evaluation makes $N \cdot t$ calls to a primitive with output length L (where N is the number of graph nodes and t is the number of calls needed to evaluate the wide-block labeling function), the theoretical optimal CMC lower bound that one may achieve is $\mathsf{CMC} = \Omega((Nt)^2 L / \log N)$.

   Note that we have *already* made progress in maximizing CMC and achieved $\mathsf{CMC} = \Omega(N^2 t^{1.5} L / \log N)$. This is already much better than $\Omega(N^2 W / \log N)$ given by prior work in the random oracle model even if we assume i) the random oracle (with output length W) is instantiated from a construction with rate *one*, i.e., $t = W/L$, and ii) the attacker cannot improve CMC by exploiting higher level of granularity. However, there still exists a large gap between our scheme and the best-possible bound, and we think that one possible direction for a further breakthrough is to find a stronger *source-to-sink depth-robust graph*.

4. Can we construct a more robust *wide-block* labeling function? In particular, the pebbling reducibility of our *wide-block* labeling functions holds only for depth-robust graphs, thus generalizing this and linking CMC and CC directly is an interesting open problem.

Appendix A

Missing Proofs for the Analysis of `scrypt`

A.1 Proof of Claim 3.8

Proof: To prove this claim, we will show, by induction, that for every $j \geq k$ and $i \leq k$, $T_i^{\mathsf{ro}_k} = T_i^{\mathsf{ro}_j}$, and the execution of A^{ro_j} is identical to the execution of A^{ro_k} until the query T_k is first made.

The base of induction ($j = k$) holds trivially.

The inductive step is as follows. Suppose the statement is true for some $j \geq k$. We will show it for $j + 1$. We already established that if $\mathsf{ro}_j \notin \mathsf{colliding}_j$, then $T_i^{\mathsf{ro}_j} = T_i^{\mathsf{ro}_{j+1}}$ for every $i \leq j$, and is therefore equal to $T_i^{\mathsf{ro}_k}$ by the inductive hypothesis. Since ro_j and ro_{j+1} differ only in their answer to the query $T_j^{\mathsf{ro}_j} = T_j^{\mathsf{ro}_{j+1}}$, the execution of $A^{\mathsf{ro}_{j+1}}$ is identical to the execution of A^{ro_j} until this query is first made. Since $\mathsf{ro}_j \notin \mathsf{wrongOrder}_j$, this moment is no earlier than when the query T_k is made; therefore, until the point the query T_k is first made, the execution of $A^{\mathsf{ro}_{j+1}}$ is identical to the execution of A^{ro_j} and thus (by the inductive hypothesis) identically to the execution of A^{ro_k}.

The last part of the claim holds because $\mathsf{ro}_N \notin \mathsf{wrongOrder}_N$. ■

A.2 The Uniformity of Oracles $\{\mathsf{ro}_k\}$

Instead of proving ro_N is uniform, we prove a stronger result that ro_k is uniform for every k, which we will need later.

Remark A.1 *We stress that the deliberate design of the oracle sampling algorithm ensures that the random oracles are uniform. We do not know any other sampling algorithms that satisfy all of our needs for proving the theorem.*

Claim A.1 *For every k ($0 \leq k \leq N$), ro_k is uniform.*

Proof: We will prove this claim by induction. ro_0 is uniform by definition. Assume ro_j is uniform. Note that $\mathsf{ro}_{j+1} = \mathsf{ro}_j \in \mathsf{colliding}_j$ (by Definition 3.3) when $\mathsf{ro}_j \in \mathsf{colliding}_j$, hence it suffices to show that if ro_j is uniform over the complement of $\mathsf{colliding}_j$, then so is ro_{j+1}. We consider a mental experiment where ro_j is lazily sampled, one output at a time, from X_0 to inputs $X_1^{\mathsf{ro}_j}, \ldots, X_{N-1}^{\mathsf{ro}_j}, T_1^{\mathsf{ro}_j}, \ldots, T_{j-1}^{\mathsf{ro}_j}$, with outputs sampled uniformly from the set that do not cause ro_j to fall into $\mathsf{colliding}_j$. Since $T_j^{\mathsf{ro}_j}$ is distinct from all of these inputs (because $\mathsf{ro}_j \notin \mathsf{colliding}_j$), the value of $\mathsf{ro}_j(T_j^{\mathsf{ro}_j})$ does not affect whether ro_j is in $\mathsf{colliding}_j$, hence the output $S_j^{\mathsf{ro}_j}$ of ro_j on $T_j^{\mathsf{ro}_j}$ is uniform in $\{0, 1, \ldots, \lfloor 2^W/N \rfloor \cdot N - 1\}$ and independent of the rest of $T_j^{\mathsf{ro}_j}$. Finally, since we generate ro_{j+1} by replacing $S_j^{\mathsf{ro}_j}$ with $\mathrm{changeModn}(S_j^{\mathsf{ro}_j}, c_{j+1})$ for a uniformly random c_{j+1}, the distribution does not change.

■

A.3 Probability Analysis

We show an lower bound on the probability of $(E_{\mathsf{hard}} \cap E_{\mathsf{corr}} \cap E_{\mathsf{q}}) \cup \overline{E_{\mathsf{q}}}$. The key idea would be to prove upper bounds for the probability of events $\overline{E_{\mathsf{hard}}}$ and $\overline{E_{\mathsf{corr}}} \cap E_{\mathsf{q}}$.

However, It turns out to be hard to directly prove an upper bound on $\Pr[\overline{E_{\mathsf{hard}}}]$, that is, to show with high probability there are many hard challenges. Thus we define the following events instead.

Definition A.1 *Define every challenge c_k $(0 \le c_k < n)$ to be "bad" if $\mathsf{ro}_{k-1} \in \mathsf{bad}_{\mathsf{ro}}^{k-1} = \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$ (see Definitions 3.2 and 3.4). We denote by E^*_{hard} the event that the number of challenges that are hard or bad is at least $N(\frac{1}{2} - \epsilon)$. Let E_{conv} be the event that for every k $(0 \le k < n)$, $\mathsf{ro}_k \notin \mathsf{predictable}$ (see Definition 3.4).*

Fortunately, we observe that $E_{\mathsf{corr}} \cap E_{\mathsf{conv}}$ (Definition A.1 and Definition 3.7) implies that no challenge is bad, and therefore $E_{\mathsf{corr}} \cap E^*_{\mathsf{hard}} \cap E_{\mathsf{conv}} \Rightarrow E_{\mathsf{hard}}$: i.e., if there are more than $N(\frac{1}{2} - \epsilon)$ bad or hard challenges, and no challenge is bad, then there are more than $N(\frac{1}{2} - \epsilon)$ hard challenges. Thus, $E_{\mathsf{corr}} \cap E^*_{\mathsf{hard}} \cap E_{\mathsf{conv}} \Rightarrow E_{\mathsf{hard}} \cap E_{\mathsf{corr}}$. Therefore, it is now sufficient for us to argue that (with high probability) either there are many hard challenges, or there are many bad challenges. In particular, we will give a lower bound on $\Pr[E_{\mathsf{corr}} \cap E_{\mathsf{q}} \cap E^*_{\mathsf{hard}} \cap E_{\mathsf{conv}}]$, and thus obtain a lower bound for $\Pr[E_{\mathsf{hard}} \cap E_{\mathsf{corr}} \cap E_{\mathsf{q}}]$. To achieve this, we provide upper bounds on the probability of events $\overline{E^*_{\mathsf{hard}}}$, $\overline{E_{\mathsf{conv}}}$, and $\overline{E_{\mathsf{corr}}} \cap E_{\mathsf{q}}$, and then obtain the lower bound

$$\Pr[E_{\mathsf{corr}} \cap E_{\mathsf{q}} \cap E^*_{\mathsf{hard}} \cap E_{\mathsf{conv}}] \ge \Pr[E_{\mathsf{q}}] - \Pr[\overline{E_{\mathsf{corr}}} \cap E_{\mathsf{q}}] - \Pr[\overline{E^*_{\mathsf{hard}}}] - \Pr[\overline{E_{\mathsf{conv}}}] .$$

Remark A.2 *Note that E_{corr}, E^*_{hard}, E_{q}, and E_{conv} depend on the oracles ro_k, and we will bound their probability assuming ro_k is sampled uniformly random. Indeed, as we have proved before in Claim A.1, oracle ro_k is uniform for every k $(0 \le k \le N)$.*

The upper bound on $\Pr[\overline{E^*_{\mathsf{hard}}}]$. Recall that according to Definition A.1, E^*_{hard} is the event that the number of challenges that are hard or bad is at least $N(\frac{1}{2} - \epsilon)$.

Claim A.2 $\Pr[\overline{E^*_{\mathsf{hard}}}] \le e^{-2\epsilon^2 N}$.

Proof: Recall that a challenge c_k $(0 \le c_k < n)$ is "bad" if $\mathsf{ro}_{k-1} \in \mathsf{bad}^{k-1}_{\mathsf{ro}} = \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$. Define random variable $V_k = 0$ if challenge c_k is hard or bad, and $V_k = 1$ otherwise. We know that for *any* (particular fixing of the values of) $\mathsf{ro}_0, c_1, \dots, c_{k-1}$, which defines a particular fixing choices of $(V_1, \dots, V_{k-1}) = (v_1, \dots, v_{k-1})$, we have

$$\Pr_{c_k}[V_k = 1 | (V_1, \dots, V_{k-1}) = (v_1, \dots, v_{k-1})] \ge \frac{1}{2} ,$$

because if $\mathsf{ro}_{k-1} \notin \mathsf{bad}^{k-1}_{\mathsf{ro}} = \mathsf{colliding}_{k-1} \cup \mathsf{predictable}$, then Lemma 3.2 applies, and otherwise c_k is always bad. Hence we can now apply Claim 3.3. ■

The upper bound on $\Pr[\overline{E_{\mathsf{conv}}}]$**.** Recall from Definition A.1 that E_{conv} is the event that for every $0 \le k < N$, $\mathsf{ro}_k \notin \mathsf{predictable}$. Hence $\overline{E_{\mathsf{conv}}}$ means that there exists a k $(0 \le k < N)$, where $\mathsf{ro}_k \in \mathsf{predictable}$.[1] Using the same technique explained in the proof of Lemma 3.1, and by plugging in the ex-post-facto initial pebbling, the time lower bound on challenge-answering, the hint, and the predictor described above Definition A.1, we argue that with high probability (over ro_k), the predictor P described right above Definition 3.4 successfully predicts the output of ro_k on p_r distinct inputs given the m_r-bit input state and an additional $(p_r(2\log N + \log \mathsf{q} + 1) + \log \mathsf{q})$-bit hint. Note now that here

$$p_r = \lceil (m_r + 1 + \log \mathsf{q})/(W - 2\log N - \log \mathsf{q} - 1) + 1 \rceil .$$

More specifically, by following the same argument as in Lemma 3.1, we get a new bound that for each ro_k,

$$\Pr[\mathsf{ro}_k \in \mathsf{predictable}] \le 2(N-1)N^2\mathsf{q} \cdot 2^{-W} ,$$

[1]Note by definition of $\mathsf{predictable}$ (Definition 3.4), this also implies that $\mathsf{ro}_k \notin \mathsf{colliding}_0$.

where the probability is over the uniform choice of ro_k.

Finally, by the union bound over the N oracles $\{\mathsf{ro}_k\}$ (each uniformly distributed), we get $\Pr[\overline{E_{\mathsf{conv}}}] \leq 2\mathsf{q}(N-1)N^3 \cdot 2^{-W}$.

The upper bound on $\overline{E_{\mathsf{corr}}} \cap E_{\mathsf{q}}$. E_{corr} is the event that $\mathsf{ro}_k \notin \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$ (see Definitions 3.2 and 3.6) for every k, and $T_N^{\mathsf{ro}_N}$ is queried by A^{ro_N}. Intuitively, it means that labels are distinct, the challenge queries are in the correct order and the correct function output is generated.

We will finally show that $\Pr[\overline{E_{\mathsf{corr}}} \cap E_{\mathsf{q}}]$ is upper bounded by

$$\Pr[E_{\mathsf{q}}]-\chi_{\mathsf{q}}+(1.5N^3+N(N-1)+1+\mathsf{q}N^2)\cdot 2^{-W} \leq \Pr[E_{\mathsf{q}}]-\chi_{\mathsf{q}}+((\mathsf{q}+1)N^2+1.5N^3)\cdot 2^{-W},$$

where χ_{q} is the probability (for a uniform ro) that A^{ro} is successful and makes no more than q queries. For completeness, we provide the proof as follows. (Some of the contents are taken from the full version [54].)

First, we would like to consider event E_{corr} not for every oracle $\mathsf{ro}_0 \ldots \mathsf{ro}_N$, but only for ro_N (to save a factor of N in the union bound). Hence we prove the following claim.

Claim A.3 *Given adversary A, if $\mathsf{ro}_k \in \mathsf{colliding}_k \cup \mathsf{wrongOrder}_k$ for some k ($0 \leq k < N$), then $\mathsf{ro}_{k+1} \in \mathsf{colliding}_{k+1} \cup \mathsf{wrongOrder}_{k+1}$, and therefore $\mathsf{ro}_N \in \mathsf{colliding}_N \cup \mathsf{wrongOrder}_N$.*

Proof: If $\mathsf{ro}_k \in \mathsf{colliding}_k$, then $\mathsf{ro}_{k+1} = \mathsf{ro}_k$ and we finish the proof. Otherwise, let $T_{i_2}^{\mathsf{ro}_k}$ for $0 < i_2 \leq k$ be a violation of the correct order: the query $T_{i_1}^{\mathsf{ro}_k}$ does not occur before the first appearance of $T_{i_2}^{\mathsf{ro}_k}$ (for some $i_1 < i_2$) when A^{ro_k} is run; moreover, if there are multiple violations of the correct order, pick the one that occurs earliest. Note that $T_k^{\mathsf{ro}_k}$ is first queried by A^{ro_k} no earlier than $T_{i_2}^{\mathsf{ro}_k}$ is: if $i_2 = k$, this statement holds trivially, and if $i_2 < k$, this statement is true because otherwise the query of $T_k^{\mathsf{ro}_k}$ would be an earlier violation. Since $\mathsf{ro}_k \notin \mathsf{colliding}_k$ and $i_1, i_2 \leq k$, we have $T_{i_1}^{\mathsf{ro}_k} = T_{i_1}^{\mathsf{ro}_{k+1}}$ and

$T_{i_2}^{\mathsf{ro}_k} = T_{i_2}^{\mathsf{ro}_{k+1}}$. Moreover, since $T_k^{\mathsf{ro}_k}$ is first queried by A^{ro_k} no earlier than $T_{i_2}^{\mathsf{ro}_k}$, and the computation of A^{ro_k} and $A^{\mathsf{ro}_{k+1}}$ proceeds identically until T_k is first queried, the same violation of the correct order occurs in the computation of $A^{\mathsf{ro}_{k+1}}$: the value of $T_{i_1}^{\mathsf{ro}_{k+1}}$ is not queried before $T_{i_2}^{\mathsf{ro}_{k+1}}$ is. ■

Thus, to upper bound $\Pr[\overline{E_{\mathsf{corr}}}]$, it is enough to bound the probability that a uniformly chosen random oracle (ro_N) is in $\mathsf{colliding}_N \cup \mathsf{wrongOrder}_N$, or $A_N^{\mathsf{ro}_N}$ does not query T_N^{ro}.

Recall from Definition 3.2 that the set $\mathsf{colliding}_N = \mathsf{colliding}_N^* \cup \mathsf{roundingProblem}_{N-1}$. We first bound the size of $|\mathsf{colliding}_N^*|$.

Claim A.4 $|\mathsf{colliding}_N^*| \leq |\mathbb{RO}| \cdot 1.5N^3 2^{-W}$, *where* $\mathbb{RO}$ *is the set of oracles with input/output space* $\{0,1\}^W$.

Proof: To obtain an upper bound of $|\mathsf{colliding}_N^*|$, we consider three possible cases:

- There is a colliding pair of X values. Let i be the smallest, and j the smallest given i, such that $X_i^{\mathsf{ro}} = X_j^{\mathsf{ro}}$ and $0 \leq i < j < N$. By our choice of i and j, if $i > 0$, then $X_{i-1}^{\mathsf{ro}} \neq X_{j-1}^{\mathsf{ro}}$ and
$$\mathsf{ro}\left(X_{j-1}^{\mathsf{ro}}\right) = \mathsf{ro}\left(X_{i-1}^{\mathsf{ro}}\right) \ ,$$
and if $i = 0$, then
$$\mathsf{ro}\left(X_{j-1}^{\mathsf{ro}}\right) = X_0 \ .$$
For each i, j and distinct $X_1, \ldots, X_{j-1}$, we can partition all the oracles for which $X_1^{\mathsf{ro}} = X_1, \ldots, X_{j-1}^{\mathsf{ro}} = X_{j-1}$ into 2^W equal-size subsets according to X_j^{ro}, and only one of the subsets will contain oracles that have a collision between X_i and X_j. Thus, there are at most $|\mathbb{RO}|2^{-W}$ such oracles; and there are at most $N^2/2$ possible pairs (i, j).
- There is no collision among the X values, but there is a collision between and X and a T value. Let i be the smallest, and j be the smallest given i, such that

$X_i^{\mathsf{ro}} = T_j^{\mathsf{ro}}$ and $0 \le i < N$ and $0 < j \le N$. This means $X_i = \mathsf{ro}(T_{j-1}) \oplus X_{S_{j-1} \bmod N}$; thus $\mathsf{ro}(T_{j-1})$ has to be in the set $\{(X_i \oplus X_\ell)\}_{0 \le \ell < N}$. We can thus use the same partitioning argument as before, except now oracles with a collision can be in up to N of the subsets (one for each value of ℓ). Thus, for each i and j, there are at most $|\mathbb{RO}|N2^{-W}$ oracles that have a collision between X_i and T_j, and there are N^2 pairs (i, j).

- There are no collisions among the X values nor a collision between an X and a T value, but there is a collision between two T values. Let i be the smallest, and j be the smallest given i, such that $T_i^{\mathsf{ro}} = T_j^{\mathsf{ro}}$ and $0 < i < j \le N$. This means

  $$S_{i-1} \oplus X_{S_{i-1} \bmod N} = S_{j-1} \oplus X_{S_{j-1} \bmod N} \ ,$$

  i.e.,

  $$\mathsf{ro}(T_{i-1}^{\mathsf{ro}}) \oplus X_{S_{i-1} \bmod N} = \mathsf{ro}(T_{j-1}^{\mathsf{ro}}) \oplus X_{S_{j-1} \bmod N} \ ,$$

  and thus $\mathsf{ro}(T_{j-1}^{\mathsf{ro}})$ should be in the set $\{(\mathsf{ro}(T_{i-1}^{\mathsf{ro}}) \oplus X_{S_{i-1} \bmod N} \oplus X_\ell)\}_{0 \le \ell < N}$ in order to satisfy the equality. Using the same argument as in the previous paragraph, we know that for each pair (i, j), there are at most $|\mathbb{RO}|N2^{-W}$ oracles ro with such a collision. And there can be at most $N(N-1)/2$ such pairs (i, j).

Adding up the three cases, $|\mathsf{colliding}_N^*| \le |\mathbb{RO}|(.5N^2 + N^3 + .5N^2(N-1))2^{-W} = |\mathbb{RO}| \cdot 1.5N^3 2^{-W}$. ■

Second, we bound the number of oracles $\mathsf{ro} \in \mathsf{roundingProblem}_{N-1} \setminus \mathsf{colliding}_N^*$ (Definition 3.1), which is enough (together with the previous bound on the size of $\mathsf{colliding}_N^*$), to bound the size of $\mathsf{colliding}_N$.

Claim A.5 $|\mathsf{roundingProblem}_{N-1} \setminus \mathsf{colliding}_N^*| \le |\mathbb{RO}|N(N-1)2^{-W}$.

Proof: For each i, $0 \leq i \leq N-1$, we will bound the size of $\mathsf{roundingProblem}_i \setminus \mathsf{roundingProblem}_{i-1} \setminus \mathsf{colliding}^*_N$. The intuitive idea is that $\mathsf{ro}(T_i)$ can take only $N-1$ out of equiprobable 2^W values in order for ro to get into this set. However, to make this idea precise, we need to first fix T_i that does not collide with anything (otherwise, it is not true that all values are equiprobable). To do so, fix any sequence of W-bit strings $X_1, \ldots, X_{N-1}, S_0, \ldots S_{i-1}$ and let $T_i = S_{i-1} \oplus X_{S^{\mathsf{ro}}_{i-1} \bmod N}$. Partition the set of all oracles into subsets $\mathcal{H}(X_1, \ldots, X_{N-1}, S_0, \ldots S_{i-1}) = \{\mathsf{ro} \text{ s. t. } X^{\mathsf{ro}}_1 = X_1, \ldots, X^{\mathsf{ro}}_{N-1} = X_{N-1}, S^{\mathsf{ro}}_0 = S_0, \ldots S^{\mathsf{ro}}_{i-1} = S_{i-1}\}$. If T_i is equal to one of $X_0, \ldots, X_{N-1}, T_1, \ldots, T_{i-1}$, then every element of this subset is in $\mathsf{colliding}^*_N$. Otherwise, this subset can further partitioned into 2^W equally sized subsets depending on $\mathsf{ro}(T_i)$, and only elements of (at most $N-1$) subsets for which $\mathsf{ro}(T_i) > \lfloor 2^W/N \rfloor \cdot N - 1$ can be in $\mathsf{roundingProblem}_i \setminus \mathsf{roundingProblem}_{i-1}$. By the union bound over all i, the claim holds. ■

After bounding the size of $\mathsf{colliding}_N$, we need to provide an upper bound on the number of oracles $\mathsf{ro} \notin \mathsf{colliding}_N$, such that $A^{\mathsf{ro}_N}_N$ does not query T^{ro}_N, or ro is in $\mathsf{wrongOrder}_N$. We first provide an upper bound for the former case. More precisely, we prove the following.

Claim A.6 *Given adversary A, the number of oracles $\mathsf{ro} \notin \mathsf{colliding}^*_N$ such that E_{q} (see Definition 3.7) holds and A^{ro_N} does not query T^{ro}_N is no more than $|\mathbb{RO}|(\Pr[E_{\mathsf{q}}] - \chi_{\mathsf{q}} + 2^{-W})$, where χ_{q} is the probability (for a uniform ro) that A^{ro} is successful and makes no more than q queries.*

Proof: First, we will bound the number of oracles outside of $\mathsf{colliding}^*_N$ for which A successfully outputs S_N without querying T_n. We use a similar approach to the proof of Claim A.5 to make sure T_n is well-defined and can take any of 2^W equally likely values. Define the subset $\mathcal{H}(X_1, \ldots, X_{N-1}, S_0, \ldots S_{N-1})$ the same way as in Claim A.5. If T_n collides with one of $X_1, \ldots, X_{N-1}, T_1, \ldots T_{N-1}$, then every element in the subset is

$\mathsf{colliding}^*_N$. Otherwise, we partition this subset further into subsets H_α according to the answers α given to queries of A. What A does, including whether A queries T_n, depends only α, and if A does not query T_n, then H_α can be partitioned into 2^W equal-size parts according to the value of $\mathsf{ro}(T_n)$. Since the output of A is determined by α, A can be successful for only one of those parts. Thus, there are at most $|\mathbb{RO}| \cdot 2^{-W}$ oracles outside of $\mathsf{colliding}^*_N$ for which A is successful but does not query T_n.

On the other hand, there are $|\mathbb{RO}| \cdot (\Pr[E_\mathsf{q}] - \chi_\mathsf{q})$ oracles for which A^ro makes no more than q queries but fails. By combining the two parts above, the claim holds. ■

Finally, we proivde an upper bound on the number of oracles ro such that $\mathsf{ro} \in \mathsf{wrongOrder}_N \setminus \mathsf{colliding}^*_N$[2] (Definitions 3.6, 3.2) and E_q holds (i.e., A makes no more than q queries).

Claim A.7 *Given adversary A, the number of oracles ro such that E_q holds and $\mathsf{ro} \in \mathsf{wrongOrder}_N \setminus \mathsf{colliding}^*_N$ is no more than $|\mathbb{RO}|\mathsf{q}N^2 2^{-W}$.*

Proof: Let j, $0 < j \le N$ be the smallest value for which there exists some $0 \le i < j$ such that T_i^ro has not been queried by A^ro by the time T_j^ro is. Note that then T_{j-1}^ro has also not been queried by the time of the T_j^ro query (because either $j = 1$ or otherwise j would not be the smallest, since $j-1$ also satisfies the condition). In order for any given query t to be equal to T_j^ro, the value $S_{j-1}^\mathsf{ro} = \mathsf{ro}(T_{j-1}^\mathsf{ro})$ needs to at least satisfy $S_{j-1}^\mathsf{ro} = t \oplus X_c$ for some c. However, t is independent of the value S_{j-1}^ro as long as query T_{j-1}^ro has not been made by the time of query t. Thus, for every one of q possible queries, there are at most N values S_{j-1}^ro (out of 2^W possible ones) that will make this query equal to T_j^ro. (To formalize this argument, we need to make sure T_j^ro is well-defined and can take 2^W possible values, which we do in exactly the same way as in Claim A.6, using the fact that $\mathsf{ro} \notin \mathsf{colliding}^*_N$. We omit this formalization to avoid repetition.)

[2]Actually bounding $|\mathsf{wrongOrder}_N \setminus \mathsf{colliding}_N|$ is enough, but we prove a slightly stronger bound.

By the union bound over all q queries and N possible values of j, the claim holds. ■

From all above, $\Pr[\overline{E_{\mathsf{corr}}} \cap E_{\mathsf{q}}]$ is upper bounded by

$$\Pr[E_{\mathsf{q}}]-\chi_{\mathsf{q}}+(1.5N^3+N(N-1)+1+\mathsf{q}N^2)\cdot 2^{-W} \leq \Pr[E_{\mathsf{q}}]-\chi_{\mathsf{q}}+((\mathsf{q}+1)N^2+1.5N^3)\cdot 2^{-W} ,$$

where χ_{q} is the probability (for a uniform ro) that A^{ro} is successful and makes no more than q queries.

Final wrap-up. Recall that χ is the probability that A^{ro} succeeds and χ_{q} is the probability that A^{ro} succeeds and makes at most q queries. From all above, we have

$$\begin{aligned}
\Pr[E_{\mathsf{corr}} \cap E_{\mathsf{q}} \cap E^*_{\mathsf{hard}} \cap E_{\mathsf{conv}}] &\geq \Pr[E_{\mathsf{q}}] - \Pr[\overline{E_{\mathsf{corr}}} \cap E_{\mathsf{q}}] - \Pr[\overline{E^*_{\mathsf{hard}}}] - \Pr[\overline{E_{\mathsf{conv}}}] \\
&\geq \chi_{\mathsf{q}} - ((\mathsf{q}+1)N^2 + 1.5N^3 + 2\mathsf{q}(N-1)N^3)\cdot 2^{-W} - e^{-2\epsilon^2 N} \\
&\geq \chi_{\mathsf{q}} - 2\mathsf{q}N^4 \cdot 2^{-W} + (2\mathsf{q} - \frac{\mathsf{q}}{N} - \frac{1}{N} - 1.5)N^3 \cdot 2^{-W} - e^{-2\epsilon^2 N} \\
&\geq \chi_{\mathsf{q}} - 2\mathsf{q}N^4 \cdot 2^{-W} - e^{-2\epsilon^2 N} ,
\end{aligned}$$

because if $N \geq 2$ and $\mathsf{q} \geq 2$, then $1/N + 1.5 \leq 2 \leq \mathsf{q}$.

Note that $\chi_{\mathsf{q}} + \Pr[\overline{E_{\mathsf{q}}}] \geq \chi$. Therefore, $\Pr[(E_{\mathsf{corr}} \cap E_{\mathsf{q}} \cap E^*_{\mathsf{hard}} \cap E_{\mathsf{conv}}) \cup \overline{E_{\mathsf{q}}}] \geq \chi - 2\mathsf{q}N^4 \cdot 2^{-W} - e^{-2\epsilon^2 N}$. Combining this statement with the result of Lemma 3.3 and the discussion following Definition A.1, we get the result of Theorem 3.4.

Appendix B

Missing Proofs in Section 4.3

B.1 Proof of Lemma 4.1

Proof: Suppose without loss of generality that the nodes $\{1, \ldots, |\mathbb{V}|\}$ are in topological order. Let $\mathbf{x} = (x_1, \ldots, x_{n_s})$ be the non-colliding input vector (where $x_i \neq x_j$ for every $i \neq j$). For every source node $i \in [n_s]$, we define a virtual node (with index $-i$ and label $\ell_{-i} := x_i$) as node i's predecessor, that is, $\mathsf{pred}(i) := -i$ and $\mathsf{prelab}(i) := x_i$. Let E_{coll} be the event that at least two pre-labels collide. We show a predictor that whenever E_{coll} happens, there exists a hint (with hint space $[|\mathbb{V}|^2/2]$) such that the predictor can correctly guess an ideal primitive entry and thus by Lemma 2.1 and Lemma 2.2, the lemma holds.

The Hint. If E_{coll} happens, the hint is defined to be two node indices $u, v \in [|\mathbb{V}|]$ $(u < v)$ such that i) $\mathsf{prelab}(u) = \mathsf{prelab}(v)$ and ii) v is the minimal index $i \in [|\mathbb{V}|]$ such that $\mathsf{prelab}(i)$ is in the set $\{\mathsf{prelab}(1), \ldots, \mathsf{prelab}(i-1)\}$.

The Predictor. Given hint u, v, the predictor works as follows. Define set $S_1 := \mathsf{pred}(u) \cap \mathsf{pred}(v)$ and $S_2 := \mathsf{pred}(u) \cup \mathsf{pred}(v) - S_1$. Let $w \in [|\mathbb{V}|]$ be the node in S_2[1] with maximal

[1]S_2 is non-empty as $\mathsf{pred}(u) \neq \mathsf{pred}(v)$.

index. ($w > 0$ because at least one of u and v is a non-source node.) The predictor first computes labels (by topological order) for the set $\{-n_s, \ldots, -1, 1, \ldots, w-1\}$, which determine the value of $\mathsf{prelab}(w)$. (Note that $\mathsf{ip}(\mathsf{prelab}(w))$ was not queried during the computation, as otherwise there exists a vertex $u' < w$ where $\mathsf{prelab}(u') = \mathsf{prelab}(w)$, contradicting with the fact that $v > w$ is the minimal index $i \in [|\mathbb{V}|]$ such that $\mathsf{prelab}(i)$ is in the set $\{\mathsf{prelab}(1), \ldots, \mathsf{prelab}(i-1)\}$.)

Next the predictor extracts the label ℓ_w: If $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}$, since $\mathsf{prelab}(u) = \mathsf{prelab}(v)$, the predictor obtains ℓ_w from the label of the other node $w' < w$ in S_2 (e.g., if w is the b-th ($b \in \{1, 2\}$) predecessor of v (or u), then $w' < w$ is the b-th predecessor of u (or v respectively)). If $\mathbb{IP} = \mathbb{RP}$, the predictor computes values

$$X_u := \bigoplus_{z \in \mathsf{pred}(u) - (S_1 \cup \{w\})} \ell_z$$

and

$$X_v := \bigoplus_{z \in \mathsf{pred}(v) - (S_1 \cup \{w\})} \ell_z \,.$$

Since $\mathsf{prelab}(u) = \mathsf{prelab}(v)$ and $w \in S_2$ is the predecessor of only one node out of u, v, we have $X_u \oplus X_v \oplus \ell_w = 0$, thus w's label is $\ell_w = X_u \oplus X_v$.

Finally, given ℓ_w and $\mathsf{prelab}(w)$, the predictor can predict the value $\mathsf{ip}(\mathsf{prelab}(w))$. In particular, $\mathsf{cf}(\mathsf{prelab}(w)) = \ell_w$, $\mathsf{rp}(\mathsf{prelab}(w)) = \mathsf{prelab}(w) \oplus \ell_w$, and if $\mathbb{IP} = \mathbb{IC}$, we define x as the last L-bit string of $\mathsf{prelab}(w)$, then we have $\mathsf{ic}(\mathsf{prelab}(w)) = x \oplus \ell_w$.

In summary, we showed a predictor that whenever pre-label collision exists, there exists a small hint where the predictor (on input the hint) can correctly predict an ideal primitive entry. Thus by Lemma 2.1 and Lemma 2.2, the lemma holds. ■

B.2 Proof of Lemma 4.2

Proof: Suppose without loss of generality that the nodes $\{1, \ldots, |\mathbb{V}|\}$ are in topological order. Define E^* as the event that the following two conditions both hold:

1. The pre-labels are distinct from each other.
2. The ex-post-facto pebbling is illegal.

By Lemma 4.1 it is sufficient to prove that $\Pr[E^*] \leq \epsilon_{\mathsf{legal}}(\mathbb{IP})$ (where the randomness is over the uniform choices of ip and A's coins r). Without loss of generality we fix r to be the optimal randomness that maximizes $\Pr[E^*]$. We show a predictor that whenever E^* happens, there exists a hint (with hint space $[2\mathsf{q}|\mathbb{V}|]$) such that the predictor (on input the hint) can correctly predict an ideal primitive entry and thus by Lemma 2.1 and Lemma 2.2, we have $\Pr[E^*] \leq \epsilon_{\mathsf{legal}}(\mathbb{IP})$.

<u>*The Hint.*</u> For the fixed random coins r, and any choice of ip, denote as $\mathsf{P} = (\mathsf{P}_0, \ldots, \mathsf{P}_{t_{\mathsf{peb}}})$ the ex-post-facto pebbling. If E^* happens, then there exists a round $i \in [t_{\mathsf{peb}}]$, and a node $v \in \mathsf{P}_i \setminus \mathsf{P}_{i-1}$, such that a node $u \in \mathsf{pred}(v)$ is not in P_{i-1}. Moreover, by Claim 4.1, there is a correct call for v in round i. For simpler reference, we use $C(v)$ to denote the corresponding correct call.[2] We choose the minimal round i if there are multiple rounds that satisfy the property. If v has two predecessors which are all outside P_{i-1}, we pick u as the predecessor that has larger index. The hint is defined to be the query index of $C(v)$ (denote as $q(v)$), node index v, and an indicator $b \in \{1, 2\}$ showing that u is the b-th predecessor of v.

<u>*The Predictor.*</u> Given hint $(q(v), v, b)$ (which determines the predecessor node u), the predictor simulates $\mathsf{A}^{\mathsf{ip}}(\mathbf{x}; r)$ until A makes the call $C(v)$ (i.e. the $q(v)$-th oracle query by A) in round i, and works as follows:

[2]We can think $C(v)$ as an encoding of the call which captures the order index of the call in the trace, as well as the type and input/output of the call.

- If $\mathbb{IP} = \mathbb{CF}/\mathbb{IC}$, or $\mathbb{IP} = \mathbb{RP}$ and u is the only predecessor of v, then after simulating $\mathsf{A}^{\mathsf{ip}}(x; r)$ until round i, P will predict $(\mathsf{prelab}(u), \mathsf{ip}(\mathsf{prelab}(u)))$ as follows: P will first obtain $\mathsf{prelab}(v)$ from the call input (or output) if the call $C(v)$ is forward (or backward), respectively. Then P will extract ℓ_u from $\mathsf{prelab}(v)$ and bit b.

  Next P will compute the labels of nodes from 1 to $u - 1$ according to topological order, until the ℓ-labels of u's parents are updated, which also determines the value of $\mathsf{prelab}(u)$.

  Finally, if $\mathbb{IP} = \mathbb{CF}$, the predictor outputs $(\mathsf{prelab}(u), \mathsf{ip}(\mathsf{prelab}(u)) = \ell_u)$; if $\mathbb{IP} = \mathbb{IC}/\mathbb{RP}$, let x be the last L-bit string of $\mathsf{prelab}(u)$, the predictor outputs $(\mathsf{prelab}(u), \mathsf{ip}(\mathsf{prelab}(u)) = x \oplus \ell_u)$.

- If $\mathbb{IP} = \mathbb{RP}$ and v has two predecessors, let $w \in \mathsf{pred}(v)$ be the other predecessor (besides u). After simulating $\mathsf{A}^{\mathsf{ip}}(x; r)$ until round i, P will predict $(\mathsf{prelab}(u), \mathsf{ip}(\mathsf{prelab}(u)))$ as follows: P will first compute the labels of nodes from 1 to $u-1$ according to topological order, until the ℓ-labels of u's parents are updated, which also determines the value of $\mathsf{prelab}(u)$.

  Next, P will updated the labels for *all ancestors* of node w according to topological order until ℓ_w is updated. Followed by that, P will obtain $\mathsf{prelab}(v)$ from the call input (or output) if the call $C(v)$ is forward (or backward), respectively. Finally, P extracts $\ell_u = \ell_w \oplus \mathsf{prelab}(v)$, and outputs $(\mathsf{prelab}(u), \mathsf{ip}(\mathsf{prelab}(u)) = \ell_u \oplus \mathsf{prelab}(u))$.

Correctness of the Predictor. To argue that the predictor will correctly predict $(\mathsf{prelab}(u), \mathsf{ip}(\mathsf{prelab}(u))$ without querying $\mathsf{ip}(\mathsf{prelab}(u))$, we need to prove two facts.

1. In the simulation of $\mathsf{A}^{\mathsf{ip}}(x; r)$, there was no correct call for u before the start of round i. (Recall that the call $C(v)$ is in round i.)

2. If $\mathbb{IP} = \mathbb{RP}$ and v has two predecessors u and w, it holds that there is no path from u to w. (Otherwise P will query $\mathsf{ip}(\mathsf{prelab}(u))$ when computing ℓ_w according to topological order.)

3. When P computes the labels of nodes from 1 to $u-1$ according to topological order, the predictor will never query $\mathsf{ip}(\mathsf{prelab}(u))$.

The last fact easily holds true because pre-labels are distinct. The first fact is proved as follows: Assume for contradiction that a correct call for u exists before the start of round i. Since $C(v)$ is a critical call for u in trace $\mathsf{A}(\sigma_{i-1})$ (as $C(v)$ is a correct call for $v \in \mathsf{succ}(u)$ in round i and v is in P_i), by definition of ex-post-facto pebbling, we have $u \in \mathsf{P}_{i-1}$, contradicting with the fact that $u \notin \mathsf{P}_{i-1}$.

The second fact is proved as follows: If $w < u$, the fact trivially holds as the nodes are in topological order. If $w > u$, then w was already pebbled before round i, as otherwise our choice of u would be w because w is not in P_{i-1} and has larger index than u. Moreover, by the first fact, there was no correct call for u (and thus u was never pebbled) before round i. Assume for contradiction that u has a path to w, since u was never pebbled before round i but w was pebbled *before* round i, there must exist a round $i^* < i$ and a node v^* in the path from u to w, such that v^* is in $\mathsf{P}_{i^*} \setminus \mathsf{P}_{i^*-1}$ while a node $u^* \in \mathsf{pred}(v^*)$[3] is not in P_{i^*-1}, contradicting with the fact that the round index i is minimal.

In summary, we showed a predictor that whenever E^* happens, there exists a small hint $h \in \mathcal{H}$ (where $|\mathcal{H}| \leq 2\mathsf{q}|\mathbb{V}|$), such that the predictor (on input h) correctly predicts an ideal primitive entry. Thus by Lemma 2.1 and Lemma 2.2, the lemma holds. ■

[3] u^* is in the path from u to w.

B.3 Proof of Claim 4.2

Proof: When pre-labels are distinct and the ex-post-facto pebbling is legal, we prove by induction that in any round $\gamma \geq i$ of $\mathsf{A}(\mathbf{x}, r)$ (which is the $(\gamma - i)$-th round of $\mathsf{A}(\sigma_i)$), P recognizes all correct calls for nodes in $\mathsf{P}_i \cup \cdots \cup \mathsf{P}_\gamma$ *when simulating* $\mathsf{A}(\sigma_i)$. The induction is in the increasing order of rounds.

For a value $\gamma \geq i$, suppose the claim holds for every round k $(i \leq k \leq \gamma)$, that is, in round k (which is the $(k - i)$-th round of $\mathsf{A}(\sigma_i)$), P recognizes all correct calls for nodes in $\mathsf{P}_i \cup \cdots \cup \mathsf{P}_k$ *when simulating* $\mathsf{A}(\sigma_i)$. (The induction basis where $\gamma = i$ holds trivially because P does not need to check calls for round i *when simulating* $\mathsf{A}(\sigma_i)$.)

We now prove that the claim holds for $k = \gamma + 1$. For any vertex $v \in \mathsf{P}_i \cup \cdots \cup \mathsf{P}_{\gamma+1}$, let $C(v)$ denote any correct call for v in round $\gamma + 1$ (which is the $(\gamma + 1 - i)$-th round of $\mathsf{A}(\sigma_i)$). (Note that the call $C(v)$ cannot be a correct call for another node $u \neq v$ because pre-labels are distinct.) We consider following cases:

1. v is in P_i. If the call $C(v)$ is the *first* correct call for v in $\mathsf{A}(\sigma_i)$, P will recognize it via the hint C_i, then $\mathsf{prelab}(v)$, $\mathsf{aftlab}(v)$ and ℓ_v would be updated. If $C(v)$ is not the first correct call, P can recognize the correct call by checking the consistency between the query input and $\mathsf{prelab}(v)$ if the call is forward, or between the query input and $\mathsf{aftlab}(v)$ if the call is backward.

2. If $v \notin \mathsf{P}_i$ but $v \in \mathsf{P}_{i+1} \cup \cdots \cup \mathsf{P}_\gamma$, let j be the minimal index in $[i + 1, \gamma]$ such that v is in P_j. Hence we have $v \in \mathsf{P}_j \setminus \mathsf{P}_{j-1}$ and by Claim 4.1, there is a correct call for v in round j. Then by inductive hypothesis, in round j $(i < j \leq \gamma)$, P already recognized the correct call for $v \in \mathsf{P}_j$ and updated $\mathsf{prelab}(v)$ and $\mathsf{aftlab}(v)$. Thus the predictor will recognize the call $C(v)$ by checking the consistency between call input and $\mathsf{prelab}(v)$ (or $\mathsf{aftlab}(v)$) if the call is forward (or backward), respectively.

3. If $v \notin \mathsf{P}_i \cup \cdots \cup \mathsf{P}_\gamma$ but $v \in \mathsf{P}_{\gamma+1}$, then we argue that $\mathsf{prelab}(v)$ was updated previously since the ℓ-labels of v's predecessors have been updated:

   - For each predecessor vertex $u \in \mathsf{pred}(v) \cap \mathsf{P}_i$, by assumption of the claim statement, ℓ_u was updated after the first critical call for u (which happened no later than checking the call $C(v)$ where $v \in \mathsf{succ}(u)$ and $v \in \mathsf{P}_{\gamma+1}$).
   - For each predecessor vertex $u \in \mathsf{pred}(v) \setminus \mathsf{P}_i$, since the ex-post-facto pebbling is legal and $v \in \mathsf{P}_{\gamma+1}$, there must exists a round γ' ($i < \gamma' \leq \gamma$) such that u is in $\mathsf{P}_{\gamma'} \setminus \mathsf{P}_{\gamma'-1}$. By Claim 4.1, there is a correct call $C(u)$ for $u \in \mathsf{P}_{\gamma'}$ in round γ'. By inductive hypothesis, P has recognized the call $C(u)$ and updated the label ℓ_u.

   In summary, since $\mathsf{prelab}(v)$ was updated previously, P will recognize the call by checking the consistency between $\mathsf{prelab}(v)$ and the call input if $C(v)$ is forward; or querying the oracle and checking the consistency between $\mathsf{prelab}(v)$ and the call output if $C(v)$ is backward.

In summary, in round $\gamma + 1$ (which is the $(\gamma + 1 - i)$-th round of $\mathsf{A}(\sigma_i)$), P correctly recognized all correct calls for nodes in $\mathsf{P}_i \cup \cdots \cup \mathsf{P}_{\gamma+1}$. By induction on γ, the claim holds. ■

Appendix C

Instantiations of Graph-based iMHFs

In this chapter, we instantiate the small/wide-block labeling functions (constructed in Section 4 and Section 5) with fixed-key AES and construct a graph-based iMHF using DRSample graph [17]. We choose DRSample graph as the underlying depth-robust graph because it is relatively practical and has near-optimal depth-robustness guarantee [17].

The goal of an MHF is to enforce the use of large amounts of memory for any strategy that efficiently computes the function. Moreover, the description size of the function should be as small as possible. To evaluate the effectiveness of our construction, we tune the parameters so that the best attack so far still requires a large memory buffer (i.e. 256MB), and we measure the sequential time cost and the description size of the MHF. Note that the best attack is a simple sequential algorithm, thus we can use it as our default sequential algorithm and there is no efficiency gap between the default algorithm and the best attack so far.

Next, we describe the best attack and measure the performance of our iMHFs constructions.

The Best Attack. The best attack towards DRSample graph was proposed by Blocki et al. [32]. They used a greedy pebbling strategy proposed by Boneh et al. [13]. For a DAG $\mathbb{G}$ with N vertices (where the vertices are sorted in topological order), the greedy pebbling strategy $\mathsf{GP}(\mathbb{G})$ works as follows: At step i ($1 \leq i \leq N$), after pebbling vertex i, the strategy only keeps pebbles on vertices which have edges to the vertex set $[i+1, \ldots, N]$. Blocki et al. [32] showed that the greedy pebbling strategy can pebble the DRSample graph with N sequential steps and $O(N/\log N)$ pebbles.

Lemma C.1 (Theorem 2 in [32]) *For a randomly sampled DRSample graph $\mathbb{G}$ with $N = 2^n$ vertices, for all $\delta > 0$, we have*

$$\Pr\left[\mathsf{st}(\mathsf{GP}(\mathbb{G}), N) > (1+\delta)\left(\frac{2N^2}{n}\right)\right] \leq \exp\left(\frac{-2\delta^2 N}{3n} + n\ln 2\right) .$$

Note that the performance of the attack scales with the graph parameter N, and $2.6N/\log N$ is a reasonable estimation on the attack's space complexity in practice.

Performance. We tune the parameters so that the best attack so far (i.e. the greedy pebbling attack [32, 13]) still requires a large memory buffer (i.e. 256MB). For the graph-based iMHFs built from small-block labeling functions, the graph parameter is $N = 2^{28}$, the MHF description size is $N \log N \approx 940\text{MB}$,[1] and the default sequential algorithm makes $N = 2^{28}$ AES calls (which takes ≈ 0.01s on an Intel i7-6700 CPU [57]).

For the graph-based iMHFs built from wide-block labeling functions, we set message block size as $W = 64L = 8192$ bits. The graph parameter is $N = 2^{21}$, the MHF description size is $N \log N \approx 5.5\text{MB}$, and the default sequential algorithm makes at most $4NW^2/L^2 = 2^{35}$ AES calls (which takes ≈ 1.5s on an Intel i7-6700 CPU [57]).

[1]The MHF description stores the information of N random edges in the DRSample graph, where the ith ($1 \leq i \leq N$) edge points to node i, and we require $\log N$ bits to store the start node of the edge.

In summary, the iMHFs built from small-block labeling functions is faster, while the iMHFs built from wide-block labeling functions has smaller description size.

Bibliography

[1] S. J. Vaughan-Nichols, *Password site lastpass warns of data breach*, 2015.

[2] L. Krantz, *Harvard says data breach occurred in june*, 2015.

[3] A. Peterson, *E-trade notifies 31,000 customers that their contact info may have been breached in 2013 hack*, 2015.

[4] K. D. Harris and A. General, *California data breach report*, *Retrieved August* **7** (2016) 2016.

[5] F. Pigni, M. Bartosiak, G. Piccoli, and B. Ives, *Targeting target with a 100 million dollar data breach*, *Journal of Information Technology Teaching Cases* **8** (2018), no. 1 9–23.

[6] N. Provos and D. Mazieres, *Bcrypt algorithm*, in *USENIX*, 1999.

[7] B. Kaliski, *Pkcs# 5: Password-based cryptography specification version 2.0*, tech. rep., 2000.

[8] J. Blocki, L. Ren, and S. Zhou, *Bandwidth-hard functions: Reductions and lower bounds*, *IACR Cryptology ePrint Archive* **2018** (2018) 221.

[9] S. Nakamoto *et. al.*, *Bitcoin: A peer-to-peer electronic cash system*, .

[10] J. Alwen and V. Serbinenko, *High parallel complexity graphs and memory-hard functions*, in *47th ACM STOC* (R. A. Servedio and R. Rubinfeld, eds.), pp. 595–603, ACM Press, June, 2015.

[11] J. Alwen, B. Chen, C. Kamath, V. Kolmogorov, K. Pietrzak, and S. Tessaro, *On the complexity of scrypt and proofs of space in the parallel random oracle model*, in *EUROCRYPT 2016, Part II* (M. Fischlin and J.-S. Coron, eds.), vol. 9666 of *LNCS*, pp. 358–387, Springer, Heidelberg, May, 2016.

[12] J. Alwen and J. Blocki, *Efficiently computing data-independent memory-hard functions*, in *CRYPTO 2016, Part II* (M. Robshaw and J. Katz, eds.), vol. 9815 of *LNCS*, pp. 241–271, Springer, Heidelberg, Aug., 2016.

[13] D. Boneh, H. Corrigan-Gibbs, and S. E. Schechter, *Balloon hashing: A memory-hard function providing provable protection against sequential attacks*, in *ASIACRYPT 2016, Part I* (J. H. Cheon and T. Takagi, eds.), vol. 10031 of *LNCS*, pp. 220–248, Springer, Heidelberg, Dec., 2016.

[14] J. Alwen, J. Blocki, and K. Pietrzak, *Depth-robust graphs and their cumulative memory complexity*, in *EUROCRYPT 2017, Part III* (J. Coron and J. B. Nielsen, eds.), vol. 10212 of *LNCS*, pp. 3–32, Springer, Heidelberg, Apr. / May, 2017.

[15] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, *Scrypt is maximally memory-hard*, in *EUROCRYPT 2017, Part III* (J. Coron and J. B. Nielsen, eds.), vol. 10212 of *LNCS*, pp. 33–62, Springer, Heidelberg, Apr. / May, 2017.

[16] J. Blocki and S. Zhou, *On the depth-robustness and cumulative pebbling cost of Argon2i*, in *TCC 2017, Part I* (Y. Kalai and L. Reyzin, eds.), vol. 10677 of *LNCS*, pp. 445–465, Springer, Heidelberg, Nov., 2017.

[17] J. Alwen, J. Blocki, and B. Harsha, *Practical graphs for optimal side-channel resistant memory-hard functions*, in *ACM CCS 17* (B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, eds.), pp. 1001–1017, ACM Press, Oct. / Nov., 2017.

[18] J. Alwen and B. Tackmann, *Moderately hard functions: Definition, instantiations, and applications*, in *TCC 2017, Part I* (Y. Kalai and L. Reyzin, eds.), vol. 10677 of *LNCS*, pp. 493–526, Springer, Heidelberg, Nov., 2017.

[19] J. Alwen, J. Blocki, and K. Pietrzak, *Sustained space complexity*, in *EUROCRYPT 2018, Part II* (J. B. Nielsen and V. Rijmen, eds.), vol. 10821 of *LNCS*, pp. 99–130, Springer, Heidelberg, Apr. / May, 2018.

[20] T. Dryja, Q. C. Liu, and S. Park, *Static-memory-hard functions, and modeling the cost of space vs. time*, in *TCC 2018, Part I* (A. Beimel and S. Dziembowski, eds.), vol. 11239 of *LNCS*, pp. 33–66, Springer, Heidelberg, Nov., 2018.

[21] C. Forler, S. Lucks, and J. Wenzel, "Catena: A memory-consuming password scrambler." Cryptology ePrint Archive, Report 2013/525, 2013. `http://eprint.iacr.org/2013/525`.

[22] H. Wu, *Pomelo: A password hashing algorithm*, tech. rep., Technical report, 2014. available at https://password-hashing. net , 2014.

[23] K. Pintér, *Gambit–a sponge based, memory hard key derivation function, Submission to Password Hashing Competition (PHC)* (2014).

[24] B. Cox, *Twocats (and skinnycat): A compute time and sequential memory hard password hashing scheme, Password Hashing Competition. v0 edn* (2014).

[25] D. Chang, A. Jati, S. Mishra, and S. K. Sanadhya, *Rig: A simple, secure and flexible design for password hashing*, in *International Conference on Information Security and Cryptology*, pp. 361–381, Springer, 2014.

[26] A. Peslyak, "yescrypt-a password hashing competition submission. submission to password hashing competition (phc)(2014)."

[27] L. C. Almeida, E. R. Andrade, P. S. Barreto, and M. A. Simplicio, *Lyra: Password-based key derivation with tunable memory and processing costs*, *Journal of Cryptographic Engineering* **4** (2014), no. 2 75–89.

[28] A. Biryukov, D. Dinu, and D. Khovratovich, *Argon2: new generation of memory-hard functions for password hashing and other applications*, in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 292–302, IEEE, 2016.

[29] C. Percival and S. Josefsson, *The scrypt password-based key derivation function*, tech. rep., 2016.

[30] J. Alwen and J. Blocki, *Towards practical attacks on argon2i and balloon hashing*, in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 142–157, IEEE, 2017.

[31] J. Alwen, P. Gazi, C. Kamath, K. Klein, G. Osang, K. Pietrzak, L. Reyzin, M. Rolínek, and M. Rybár, *On the memory-hardness of data-independent password-hashing functions*, in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 51–65, ACM, 2018.

[32] J. Blocki, B. Harsha, S. Kang, S. Lee, L. Xing, and S. Zhou, "Data-independent memory hard functions: New attacks and stronger constructions." Cryptology ePrint Archive, Report 2018/944, 2018. `http://eprint.iacr.org/2018/944`.

[33] C. Percival, *Stronger key derivation via sequential memory-hard functions*, in *BSDCan 2009*, 2009.

[34] Charles Lee, *Litecoin*, 2011.

[35] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function." RFC 7914 (Informational), Aug., 2016.

[36] "Password hashing competition." `https://password-hashing.net/`.

[37] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2 password hash." Version 1.3, 2016. `https://www.cryptolux.org/images/0/0d/Argon2.pdf`.

[38] C. Dwork, M. Naor, and H. Wee, *Pebbling and proofs of work*, in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, vol. 3621 of *Lecture Notes in Computer Science*, pp. 37–54, Springer, 2005.

[39] B. Chen and S. Tessaro, *Memory-hard functions from cryptographic primitives*, in *CRYPTO 2019*, Aug., 2019.

[40] M. S. Paterson and C. E. Hewitt, *Comparative schematology*, in *Record of the Project MAC conference on concurrent systems and parallel computation*, pp. 119–127, ACM, 1970.

[41] S. A. Cook, *An observation on time-storage trade off*, in *Proceedings of the fifth annual ACM symposium on Theory of computing*, pp. 29–33, ACM, 1973.

[42] S. Dziembowski, T. Kazana, and D. Wichs, *One-time computable self-erasing functions*, in *TCC 2011* (Y. Ishai, ed.), vol. 6597 of *LNCS*, pp. 125–143, Springer, Heidelberg, Mar., 2011.

[43] R. Gennaro and L. Trevisan, *Lower bounds on the efficiency of generic cryptographic constructions*, in *41st FOCS*, pp. 305–313, IEEE Computer Society Press, Nov., 2000.

[44] R. Gennaro, Y. Gertner, J. Katz, and L. Trevisan, *Bounds on the efficiency of generic cryptographic constructions*, *SIAM J. Comput.* **35** (2005), no. 1 217–246.

[45] A. Evans Jr, W. Kantrowitz, and E. Weiss, *A user authentication scheme not requiring secrecy in the computer*, *Communications of the ACM* **17** (1974), no. 8 437–442.

[46] R. Morris and K. Thompson, *Password security: A case history*, *Communications of the ACM* **22** (1979), no. 11 594–597.

[47] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, *Proofs of space*, in *CRYPTO 2015, Part II* (R. Gennaro and M. J. B. Robshaw, eds.), vol. 9216 of *LNCS*, pp. 585–605, Springer, Heidelberg, Aug., 2015.

[48] G. Ateniese, I. Bonacina, A. Faonio, and N. Galesi, *Proofs of space: When space is of the essence*, in *SCN 14* (M. Abdalla and R. D. Prisco, eds.), vol. 8642 of *LNCS*, pp. 538–557, Springer, Heidelberg, Sept., 2014.

[49] C. Dwork, A. Goldberg, and M. Naor, *On memory-bound functions for fighting spam*, in *CRYPTO 2003* (D. Boneh, ed.), vol. 2729 of *LNCS*, pp. 426–444, Springer, Heidelberg, Aug., 2003.

[50] C. Dwork, M. Naor, and H. Wee, *Pebbling and proofs of work*, in *CRYPTO 2005* (V. Shoup, ed.), vol. 3621 of *LNCS*, pp. 37–54, Springer, Heidelberg, Aug., 2005.

[51] L. Ren and S. Devadas, *Proof of space from stacked expanders*, in *TCC 2016-B, Part I* (M. Hirt and A. D. Smith, eds.), vol. 9985 of *LNCS*, pp. 262–285, Springer, Heidelberg, Oct. / Nov., 2016.

[52] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, *Moderately hard, memory-bound functions*, *ACM Transactions on Internet Technology (TOIT)* **5** (2005), no. 2 299–327.

[53] L. Ren and S. Devadas, *Bandwidth hard functions for ASIC resistance*, in *TCC 2017, Part I* (Y. Kalai and L. Reyzin, eds.), vol. 10677 of *LNCS*, pp. 466–492, Springer, Heidelberg, Nov., 2017.

[54] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, "Scrypt is maximally memory-hard." Cryptology ePrint Archive, Report 2016/989, 2016. `http://eprint.iacr.org/2016/989`.

[55] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, *Journal of the American statistical association* **58** (1963), no. 301 13–30.

[56] C. Percival and S. Josefsson, *The scrypt password-based key derivation function*, .

[57] *AES-NI SSL performance study*, 2018. `https://calomel.org/aesni_ssl_performance.html`.