NON-INTERACTIVE PROOFS: WHAT ASSUMPTIONS ARE SUFFICIENT?

by
Zhengzhong Jin

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
July 2022

Abstract

A non-interactive proof system allows a prover to convince a verifier that a statement is true by sending a single round of messages. In this thesis, we study under what assumptions can we build non-interactive proof systems with succinct verification and zero-knowledge. We obtain the following results.

- **Succinct Arguments:** We construct the first non-interactive succinct arguments (SNARGs) for $\mathcal{P}$ from standard assumptions. Our construction is based on the polynomial hardness of Learning with Errors (LWE).
- **Zero-Knowledge:** We build the first non-interactive zero-knowledge proof systems (NIZKs) for $\mathcal{NP}$ from sub-exponential Decisional Diffie-Hellman (DDH) assumption in the standard groups, without use of groups with pairings.

To obtain our results, we build SNARGs for batch-$\mathcal{NP}$ from LWE and correlation intractable hash functions for TC^0 from sub-exponential DDH assumption, respectively, which may be of independent interest.

Thesis Readers

Dr. Abhishek Jain (Primary Advisor)
Associate Professor
Department of Computer Science
Johns Hopkins University

Dr. Xin Li
Associate Professor
Department of Computer Science
Johns Hopkins University

Dr. Sanjam Garg
Associate Professor
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Dedicated to my parents.

Acknowledgements

I would like to sincerely thank my advisors Abhishek Jain and Xin Li. There are no words that can adequately express my gratitude to them. I started my Ph.D. with a project on error-correcting codes for edit errors with Xin and I have learned a lot about theoretical computer science since then. Abhishek introduced me to the world of theoretical cryptography and spent a great amount of time discussing with me about a variety of topics. Their broad knowledge, enthusiasm for research, and critical attitude towards science deeply affected me. Without their guidance and constant support, this thesis would not be possible. I also would like to thank other professors I discussed with at Johns Hopkins. They include Matthew D. Green, Amitabh Basu, and Michael Dinitz. Thank them for being on my GBO committee and providing many helpful suggestions.

I would like to thank Sanjam Garg for hosting me for a visit at UC Berkeley during my fifth year and I would like to thank him for many insightful discussions. I also want to thank Guru Vamsi Policharla, Yinuo Zhang, Aarushi Goel, Mingyuan Wang, Arka Rai Choudhuri, Sruthi Sekar, Sina Shiehian, Jaiden Fairoze, and Bhaskar Roberts for making Berkeley an enjoyable place to stay.

I would like to thank Amit Sahai for inviting me for a short visit to UCLA, and I also want to thank him for his valuable advice and insightful discussions. I would like to thank Rex Fernando, Alexis Korb, Paul Lou, Riddhi Ghosal, and Nathan Manohar for making my visit so much fun.

I would like to thank my brilliant collaborators Kuan Cheng, Ke Wu, James Bartusek, Brent Carmer, Tancrède Lepoint, Fermi Ma, Tal Malkin, Alex J. Malozemoff, Mariana Raykova, Vipul Goyal, Prabhanjan Ananth, Giulio Malavolta, Alireza Farhadi, MohammadTaghi Hajiaghayi, Aviad Rubinstein, Saeed Seddighin, Gabrielle Beck, Arka Rai Choudhuri, Aarushi Goel, Aditya Hegde, and Gabriel Kaptchuk. I have learned a lot from them and I want to thank them for sharing their knowledge with me.

I would like to thank other researchers I discussed with. They are Yuval Ishai, Prabhanjan Ananth, Vinod Vaikuntanathan, Huijia Lin, Qipeng Liu and Jiaheng Zhang. I apologize if I miss someone. I want to thank them for many informative discussions.

I would like to thank my labmates and friends at Johns Hopkins. They include Kuan Cheng, Arka Rai Choudhuri, Aarushi Goel, Ke Wu, Yu Zheng, Cong Gao, Zeng Zhang, Chang Lou, Alishah Chator, Yuxin He, Gabrielle Beck, Tushar Jois, Gijs Van Laer, Gabriel Kaptchuk, Zaoxing Liu, Shiwei Weng, Yigong Hu, Zhihao Bai, Hang Zhu, Zeyu Zhang, Song Li, Xuan Wu, Zhuolong Yu, Zifeng Kang, Mingqing Kang, Jianjia Yu, and Jingfeng Wu, Their company is a great source of happiness during my Ph.D. and I would like to thank them for that.

I would like to thank my family, my mother and my father, for their constant support.

Contents

List of Figures

Chapter 1

Introduction

A proof allows one man to convince the other that some statement is true. Proofs are so fundamental that their intuition can be traced back to classical Greece. More recently, proofs are studied as the following notion of non-interactive proof systems in cryptography.

A non-interactive proof system for a langauage $\mathcal{L} \subseteq \{0,1\}^*$ consists of a prover and a verifier. For example, $\mathcal{L}$ can be the set of all true mathematical theorem statements encoded as binary strings. The prover takes a statement $x \in \{0,1\}^*$ as input, and sends a single round of the message (a proof) to convince the verifier that $x \in \mathcal{L}$. Then the verifier will accept or reject the proof for the statement x. The semantics of such a non-interactive proof system are characterized by the following completeness and soundness properties. *Completeness* says that if the prover behaves honestly, then the proof should always be accepted by the verifier, while *soundness* requires that if the statement x is false ($x \notin \mathcal{L}$), then any cheating prover can not provide an acceptable proof.

The reader may notice that there is a trivial non-interactive proof system for $\mathcal{NP}$: the prover can send the witness of the statement directly to the verifier. Then the verifier checks whether the proof is a witness for the statement. Indeed, in this thesis, we are mainly interested in the non-interactive proof systems with the

following additional properties that are very useful in cryptography: zero-knowledge and succinctness.

Succinct Verification. The succinct verification requires that the verification process is much quicker than deciding the statement x itself. Specifically, if x can be decided by a deterministic or non-deterministic Turing machine in T steps, then we require that the verification process is a deterministic Turing machine that runs in $T^{o(1)}$ steps.

The de facto model for such proof systems also allows for an initial setup that samples a CRS and distributes it to the parties. Furthermore, the soundness guarantee is computational, i.e., it only holds against computationally-bounded provers [1]. The key benefit of such proof systems is that they can be used as short certificates for the correctness of long computations that anyone can verify.

There are many applications of succinct non-interactive arguments (SNARGs). One example is computation delegation: a client wishes to evaluate a program (say, represented as a Turing machine) on an input but does not have the necessary computational resources. Therefore, it delegates the computation to an untrusted server who provides the output together with a SNARG proof. Finally, to ensure the output is correctly computed rather than some garbage, the client can verify the proof in time significantly less than computing the program on its own. Other applications include popular real-world systems such as blockchains [2].

Zero-Knowledge. Knowledge [3] is a fundamental concept in cryptography. Intuitively, one could become more computationally powerful if he obtains a proof for a statement. For example, factorization is not known to have polynomial-time algorithm. However, if the statement x represents "N is a multiplication of two primes", and let the proof be the factorization of N, then anyone with the proof knows how to factor N.

Zero-knowledge is a seemingly contradictory feature that requires that an honestly

generated proof should reveal nothing but convinces the verifier that $x \in \mathcal{L}$. In other words, it says that anything that one can infer from the proof can also be computed efficiently without the proof. It is known that zero-knowlege is impossible for non-trivial languages in the plain model. Hence, the traditional (and de facto) model for NIZKs allows for a trusted setup algorithm, which generates a common reference string (CRS) and provides it to the prover and verifier.

Non-interactive zero-knowledge proof systems (NIZKs) are very important in cryptography and have many applications in hiding something. Some examples are digital signatures [4, 5], advanced encryption schemes [6, 7], and blockchains [8].

Why Assumptions Matter? Non-interactive zero-knowledge and succinct non-interactive arguments are unlikely to be constructed *unconditionally*. Specifically, NIZKs with statistical zero-knowledge and soundness are unlikely to exist for all of $\mathcal{NP}$, unless the polynomial-time hierarchy collapses [9]. Also, SNARGs with soundness against unbounded cheating prover are unlikely to exists for $\mathcal{NP}$, given the non-deterministic time hierarchy theorem [10].

Therefore, computational assumptions are used to build NIZKs and SNARGs. Under computational assumptions, we only require the zero-knowledge or the soundness property to hold against *polynomial-time* adversaries.

There is a bunch of mathematical problems whose hardness has been extensively studied. People believe in their hardness and refer to them as "standard assumptions". Typical standard assumptions include quadratic residuosity assumption, Diffie-Hellman assumptions [11], bilinear map assumptions [12], and learning with errors (LWE) assumptions [13]. Fearing that a concrete assumption may be broken some day[1], cryptographers may also base their constructions on generic assumptions, such as the existence of one-way functions or the existence of public-key encryption.

A central problem in cryptography is how to base cryptography on sound assump-

[1] "Cryptographers seldom sleep well." ([14], Section 1.1)

tions. Specifically, can we base cryptographic constructions on standard assumptions? Further, can we base them on generic assumptions? In the region of non-interactive proof systems, we study the following question.

Under what assumptions can we build non-interactive proof systems with zero-knowledge or succinct verification?

Next, we discuss the prior works on this problem for SNARKs and NIZKs, respectively.

Succinct Non-Interactive Arguments. Starting from [15], there is a large body of work that constructs SNARGs for $\mathcal{NP}$ (see, e.g., [15–22]. These schemes are either in the Random Oracle model or require non-falsifiable assumptions [23]. Despite some of these schemes form the basis of efficient implementations used in practice, the underlying assumptions they use are non-standard.

How about SNARGs for $\mathcal{NP}$ from standard assumptions? Gentry and Wichs [24] showed that adaptive sound SNARGs for $\mathcal{NP}$ with a black-box soundness reduction to falsifiable assumptions are impossible. However, this negative result does not imply any difficulty in constructing SNARGs for languages within $\mathcal{P}$.

For deterministic computations, prior constructions are known based on assumptions related to obfuscation or multilinear maps [25–31]. Despite recent breakthrough on basing iO from well-founded assumptions [32], these assumptions are still relatively less well-understood compared to standard assumptions.

For the protocols without iO, Canetti et. al [33] constructed SNARGs based on a very strong assumption, namely, the existence of fully homomorphic encryption with optimal circular security. Later, the beautiful work by Jawale, Kalai, Khurana and Zhang [34] improves their assumption to the sub-exponential hardness of LWE. However, since this line of works are based on the GKR interactive delegation protocol for bounded-depth computation [35], both works are targeting at this sub-class of $\mathcal{P}$.

Another line of research initiated by [36, 37] and continuing with [38–42] constructed designated-verifier SNARGs for *entire* $\mathcal{P}$ and various sub-classes of $\mathcal{NP}$ (such as SNARGs for batch-$\mathcal{NP}$) based on standard assumptions, by leveraging the *no-signaling* techniques. The main drawback of these schemes is that the verifier needs a "secret key" corresponding to the CRS to verify the proof. Recently, [43] eliminates this drawback by using a new falsifiable but non-standard assumption on groups with bilinear maps.

In conclusion, constructing SNARGs for $\mathcal{P}$ from standard assumptions remains an open problem.

Non-Interactive Zero-Knowledge Proof Systems. Non-interactive zero-knowledge proof systems are constructed for general languages in $\mathcal{NP}$ for the first time by the work [44]. Their construction relies on the quadratic residuosity assumption. Later, a series of works built NIZK from other assumptions. By now, NIZKs for $\mathcal{NP}$ are known from most of the standard assumptions known to imply public-key encryption – this includes factoring related assumptions [45, 46], bilinear maps [47–49], and more recently, learning with errors (LWE) [33, 50].

Notable exceptions to this list are standard assumptions related to the discrete logarithm problem, such as the Decisional Diffie-Hellman (DDH) assumption. The problem of basing NIZKs on DDH in the pairing-free groups has been open for more than three decades.

From a conceptual viewpoint, an answer to the above question would shed further light on the cryptographic complexity of NIZKs relative to public-key encryption. Specifically, can we build NIZKs from public-key encryption?

A recent beautiful work of Brakerski et al. [51] made significant progress towards building NIZKs from DDH. However, they additionally rely on the hardness of the learning parity with noise (LPN) problem. Namely, they construct NIZKs assuming

that DDH and LPN are *both* hard. NIZKs based on the *sole* hardness of DDH, however, still remain elusive.

1.1 Our Results

1.1.1 Succinct Non-Interactive Arguments

We first construct SNARGs for all polynomial-time deterministic computations based on the hardness of LWE against polynomial-time adversaries. Our construction is in the common *random* string model and achieves adaptive soundness.

Theorem 1.1.1 (Informal)**.** *Assuming the hardness of LWE, for every polynomial $T = T(\lambda)$, there exists a publicly-verifiable non-interactive delegation scheme with adaptive soundness for any time T Turing machine. The verifier running time, size of the CRS and proof are all $\mathsf{poly}(\log T, \lambda)$ while the prover running time is $\mathsf{poly}(T, \lambda)$.*

Our result also extends, with the same parameters, to delegation of RAM computation.

1.1.2 Non-Interactive Zero-Knowledge Proof Systems

We also construct (statistical) NIZK arguments for $\mathcal{NP}$ based on the sub-exponential hardness of DDH against polynomial-time adversaries in standard groups.

Theorem 1.1.2 (Informal)**.** *Assuming sub-exponential hardness of DDH, there exist (statistical) NIZK arguments for $\mathcal{NP}$ in the common random string model.*

Our NIZK achieves adaptive, multi-theorem statistical zero knowledge and non-adaptive soundness. By relaxing the zero-knowledge guarantee to be computational, we can achieve adaptive soundness. In this thesis, we also construct statistical Zap arguments from sub-exponential DDH assumption. [2] We defer the formal result statement for Zaps to Section 5.6.

[2]Following [52], by standard complexity leveraging, our statistical NIZK and Zap arguments can

Our results in Theorem 1.1.2 rely on the assumption that *polynomial-time* adversaries cannot distinguish Diffie-Hellman tuples from random tuples in standard $\mathbb{Z}_q^*$ group with better than sub-exponentially small advantage. Alternatively, if we also assume hardness against sub-exponential time adversaries, then we can instantiate Theorem 1.1.2 using Elliptic curves over $\mathbb{F}_p$ with a prime $p > 3$ (see Section 5.7). To the best of our knowledge, our assumption is unaffected by known attacks on the discrete logarithm problem.[3]

Discussion. While our primary focus is on constructing NIZKs from DDH, we note that our constructions enjoy certain properties that have previously not been achieved even using bilinear maps. Our NIZK constructions rely on a common *random* string setup, unlike prior schemes based on bilinear maps that require a common *reference* string for achieving statistical ZK [48, 49]. In particular, statistical NIZKs in the common random string model were previously only known from LWE (or circular-secure FHE) [33, 50], and statistical Zap arguments were previously only known from (quasi-polynomial) LWE [57, 58].

Our results also shed light on the understanding of the power of groups with bilinear maps relative to non-pairing groups in cryptography. There are (at least) two prominent examples where bilinear maps have traditionally had an edge – advanced encryption schemes such as identity-based [59] and attribute-based encryption [60, 61] (and more broadly, functional encryption [60, 62, 63]), and NIZKs. For the former, the gap has recently started to narrow in some important cases; see, e.g., [64]. Our results can help us understand whether such gap is inherent for NIZKs based on standard

be upgraded (without changing our assumption) to achieve adaptive soundness for all instances of a priori (polynomially) bounded size. For the "unbounded-size" case, [53] proved the impossibility of statistical NIZKs where adaptive soundness is proven via a black-box reduction to falsifiable assumptions [23].

[3]There are well-known attacks for discrete logarithm over $\mathbb{Z}_q^*$ that require sub-exponential time and achieve constant success probability [54, 55]. However, as observed in [56], a 2^t time algorithm with constant successful probability does not necessarily imply a polynomial time attack with 2^{-t} successful probability.

assumptions.[4]

1.2 Our Methodology: More Interaction

The general methodology throughout this thesis is as follows.

To construct a cryptographic primitive A,
we first construct an "interactive" variant of A, and then "round-collapse" it.

Specifically, constructing the cryptographic primitive A directly could be a challenging problem. Hence, our general methodology is to first relax our goal to construct an "interactive" variant of A. By leveraging the power of interaction, we may construct the primitive easily. Finally, we find some way to collapse the rounds in our interactive variant and thus obtain the "non-interactive" version of A.

Our methodology is inspired by Fiat-Shamir paradigm, whose focus is on non-interactive proof systems. Since Fiat-Shamir transformation is also central to the constructions in this thesis, we recall it as follows.

Fiat-Shamir Paradigm. Specifically, Fiat-Shamir transformation [66] is a method that transforms any public-coin interactive protocol into a non-interactive protocol. The idea is to replace the verifier's messages with the hash values of the transcript of the protocol so far. Fiat-Shamir transformation was firstly proven secure in the Random Oracle model [66]. Later, a line of works [33, 50, 51, 56, 65, 67, 68] instantiate Fiat-Shamir in the standard model via the framework of *correlation intractability*.

Next, we show how to apply our methodology to non-interactive proof systems, with succinct verification or zero-knowledge properties, respectively.

[4] If we allow for non-standard assumptions (albeit those not known to imply public-key encryption), then this gap is not inherent, as demonstrated by [56, 65].

1.2.1 Succinct Non-interactive Arguments

We first explain the main challenge in constructing SNARGs for $\mathcal{P}$. The line of research [36–42] uses *no-signaling PCPs* to build designated-verifier SNARGs for all of $\mathcal{P}$. Removing the designated-verifier constraint is a difficult problem until the recent work [43], which removes the designated-verification using a new falsifiable assumption in bilinear groups. Roughly speaking, the difficulty arises in the transformation from PCPs to SNARGs. Prior works encrypt the verifier's challenges in PCPs, and then have the prover homomorphically compute the PCP responses. This leads to the designated-verification natural of the prior constructions.

Hence, to build public-verifiable SNARGs for $\mathcal{P}$, we use the Fiat-Shamir paradigm, which gives us public-verifiability for free. Towards this, the natural attempt is to instantiate the Kilian's protocol [69] with no-signaling PCPs. However, Kilian's protocol is an argument system, whereas the known instantiations of the Fiat-Shamir paradigm in the standard model requires the underlying interactive protocol to be statistically sound. Indeed, this is how the recent works [33, 34] achieve their results. In this work, we observe that a slight extension of statistical soundness, which we call it as *somewhere statistical soundness*, also allows Fiat-Shamir paradigm via the correlation intractability framework. Roughly speaking, we allow the underlying protocol to be in the CRS model, and the protocol is only statistical sound for a subset of constraints that are secretly specified by the CRS. However, Kilian's protocol instantiated with no-signaling PCPs is not known to satisfy this soundness property.

Our Method. To overcome this obstacle, our key idea is to apply our methodology. Namely, instead of using PCPs, which are essentially non-interactive, we first make use of more interactions to build a multi-round succinct interactive argument, with more rounds than Kilian's protocol, but it achieves somewhere statistical soundness. Then we apply the Fiat-Shamir transformation via correlation intractability framework to

collapse its rounds and thus obtain SNARGs.

Succinct Interactive Arguments for Batch-$\mathcal{NP}$. Specifically, our first step is to build a public-coin succinct interactive protocol for batch-$\mathcal{NP}$ with somewhere statistical soundness. Informally speaking, such an argument system allows an efficient prover to interact with a public-coin verifier in multiple rounds to convince him a batch of k $\mathcal{NP}$ statements, with size smaller than the combined witness length. Moreover, somewhere statistical soundness requires that there is an index i^* hidden in the CRS such that if the i^*-th statements is false, then even unbounded cheating prover can not find an accepting proof.

Looking forward, we will construct SNARGs for $\mathcal{P}$ in a generic way from SNARGs for batch-$\mathcal{NP}$. The high level idea is that, to verify a polynomial-time computation, it suffices to verify that each step of the computation is correct. The latter can be naturally expressed as a batch-$\mathcal{NP}$ instance.

SNARGs for Batch-$\mathcal{NP}$. To build SNARGs for batch-$\mathcal{NP}$, we apply Fiat-Shamir transformation to the succinct interactive protocol with the recent correlation intractable hash functions [68], and thus obtain SNARGs for batch-$\mathcal{NP}$.

Theorem 1.2.1 (Informal)**.** *Assuming the hardness of LWE, there exists a SNARG for batch-$\mathcal{NP}$ with the following parameters: in order to prove k instances of a language $\mathcal{L}$ whose $\mathcal{NP}$-relation can be decided by a Turing machine in time T, the size of the CRS and proof are $\mathsf{poly}(\log k, \log T, n, m, \lambda)$, the prover running time is $\mathsf{poly}(k, T, n, m, \lambda)$ and the verifier running time is $\mathsf{poly}(\log k, \log T, n, m, \lambda) + \mathsf{poly}(k, n, \lambda)$, where n is the length of a single instance and m is the length of a single witness.*

We note that our scheme is in the common random string. In contrast, [70] requires a common *reference* string.

1.2.2 Non-interactive Zero-Knowledge Proof Systems

We now how to use the same general methodology to build NIZKs from sub-exponential DDH. [33] gave a general framework for building NIZKs via Fiat-Shamir paradigm. The idea is firstly building a special interactive protocol called *trapdoor sigma protocol*, and then instantiating Fiat-Shamir via the correlation intractability framework. Looking forward, to build NIZKs from DDH, we can use the known trapdoor sigma protocol for $\mathcal{NP}$ from DDH in [51], and thus the main challenge is how to instantiate the correlation intractability from DDH assumption.

We first recall the correlation intractability framework, which instantiates the hash function in Fiat-Shamir transformation.

Correlation-Intractatable Hash Functions. Roughly speaking, a family of hash functions $\{\mathsf{Hash}(k,\cdot)\}_k$ is said to be correlation intractable for a relation class $\mathcal{R}$ if for any $R \in \mathcal{R}$, given a hash key k sampled by a key generation algorithm, an adversary cannot find an input x such that $(x, \mathsf{Hash}(k,x)) \in R$. In the sequel, we focus on searchable relations where R is associated with a circuit C and $(x,y) \in R$ if and only if $y = C(x)$.

A sequence of works [56, 65, 67, 71, 72] have constructed CIH for various classes of (not necessarily efficiently searchable) relations from well-defined, albeit strong assumptions that are not well understood. Recently, Canetti et al. [33] constructed CIH for all efficiently searchable relations from circular-secure fully homomorphic encryption. Subsequently, Peikert and Shiehian [50] obtained a similar result based on standard LWE. However, those constructions relies on lattice-related assumptions. If we want to build NIZKs from DDH, we need CIH from DDH assumption.

Recently, Brakerski et al. [51] demonstrated a new approach for constructing CIH from (rate-1) trapdoor hash functions (TDH). This raises hope for potential instantiations of CIH – ideally for all efficiently searchable relations – from other

standard assumptions (such as DDH). So far, however, this approach has yielded CIH only for relations that can be approximated by constant-degree polynomials over $\mathbb{Z}_2$ due to limitations of known results for TDH. This severely restricts the class of compatible trapdoor sigma protocols that can be used for constructing NIZKs via the CIH framework. Indeed, Brakerski et al. rely crucially on LPN to construct such sigma protocols. However, the trapdoor sigma protocols from DDH are not known to be compatible with the relations approximable by constant-degree polynomials.

Our Method: Round Collapsing, *Twice*. We apply our general methodology to overcome the above challenge. Here, the cryptographic primitive we are interested in is CIH or TDH for a larger class of functions. Specifically, following our methodology, we first generalize the notion of TDHs to *interactive trapdoor hashing protocols* (ITDH). Crucially relying on the interaction, we expand the class of functions that can be computed to constant-depth threshold circuits (TC^0) under DDH assumption.

Next, we invent a new technique that collapses the rounds of ITDH to construct CIH. Using this approach, we construct a CIH for TC^0 based on sub-exponential DDH. Finally, we observe that the trapdoor sigma protocol from DDH is compatible with CIH for TC^0.

Overall, our approach for constructing NIZKs involves **two stages of round collapsing** – we first collapse rounds of ITDH to construct CIH, and then use CIH to collapse rounds of trapdoor sigma protocols to obtain NIZKs.

Interactive Trapdoor Hashing Protocol. Specifically, an ITDH for a function family F is an interactive protocol between two parties – a sender and a receiver – where the sender holds an input x and the receiver holds a function $f \in F$. At the end of the protocol, the parties obtain an additive secret-sharing of $f(x)$. An ITDH must satisfy the following key properties:

- The sender must be *laconic* in that the length of each of its messages (consisting

of a hash value) is independent of the input length.

- The receiver's messages must *hide* the function f (the exact formulation of this property is nuanced).

Assuming DDH, we construct a constant-round ITDH protocol for TC^0 circuits. While ITDH for TC^0 suffices for our main application, our approach can be generalized to obtain a polynomial-round ITDH for P/poly.

Theorem 1.2.2 (Informal)**.** *Assuming DDH, there exists a constant-round ITDH for* TC^0*.*

We view ITDH as a natural generalization of TDH that might allow for a broader pool of applications. While our present focus is on the class of computations, it is conceivable that the use of interaction might enable additional properties in the future that are not possible (or harder to achieve) in the non-interactive setting.

CIH for TC^0 from Sub-exponential DDH. By introducing a new technique round-collapsing the above ITDHs, we expand the class of searchable relations that CIH can support without relying on LWE. Specifically, we construct CIH for constant-depth threshold circuits from sub-exponential DDH.

Theorem 1.2.3 (Informal)**.** *Assuming sub-exponential hardness of DDH against polynomial-time attackers, there exists a CIH for* TC^0*.*

In fact, we can trade-off between the hardness assumption on DDH and the depth of the circuits that CIH can support. Assuming sub-exponential hardness of DDH against *sub-exponential time* adversaries, we can obtain CIH for threshold circuits of depth $O(\log \log n)$. Moreover, assuming *exponential* hardness of DDH against polynomial time adversaries (which can be conjectured to hold in elliptic curve groups), we can obtain CIH for log-depth threshold circuits, i.e., TC^1. We refer the reader to Section 5.4.4 for details.

Expanding the class of relations for CIH, in turn, expands the class of compatible trapdoor sigma protocols. In particular, we show that trapdoor sigma protocols for $\mathcal{NP}$ compatible with CIH from Theorem 1.2.3 can be built from DDH. This allows us to construct NIZK and Zap arguments in Theorem 1.1.2.

While our primary interest is using CIH for constructing NIZKs (and Zap arguments), we note that recent works (e.g., [33, 73]) have also explored applications of CIH to succinct arguments [15, 69], verifiable delay functions [74] and establishing hardness of complexity classes such as PPAD [75]. Our constructions of CIH may therefore be of independent interest for applications beyond NIZKs.

1.3 Organization

The rest of this thesis is organized as follows.

- We introduce notations and preliminaries in Chapter 2.
- In Chapter 3, we recall the background of Fiat-Shamir paradigm and correlation intractability.
- In Chapter 4 we present the construction of SNARGs for $\mathcal{P}$ from LWE.
- In Chapter 5 we present the constructions of NIZKs and Zaps from sub-exponential DDH.

Chapter 2

Preliminaries

Notations. We introduce the following notations.

- For any positive integer $N \in \mathbb{Z}, N > 0$, denote $[N] = \{1, 2, \ldots, N\}$.
- A binary relation $\mathcal{R}$ is a subset of $\{0,1\}^* \times \{0,1\}^*$.
- For any integer $R > 0$, and $x \in \mathbb{Z}_R$, $0 \leq x < R$, the indicator vector $\mathbb{1}_x$ of x is a vector in $\{0,1\}^R$, where the $(x+1)^{\text{th}}$ position is 1, and all other coordinates are zero.
- For any positive integer n, any vector $x = (x_1, x_2, \ldots, x_n)$, and any subset $S \subseteq [n]$, we denote $x|_S = \{x_i\}_{i \in S}$.

Statistical Distance. For any two discrete distributions P, Q, the statistical distance between P and Q is defined as $\mathsf{SD}(P,Q) = \sum_i \Big| \Pr[P = i] - \Pr[Q = i] \Big| /2$ where i takes all the values in the support of P and Q.

Hamming Distance. Let n be an integer, and S be a set, and $x = (x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_n)$ be two tuples in S^n, the Hamming distance $\mathsf{Ham}(x,y)$ is defined as $\mathsf{Ham}(x,y) = |\{i \mid x_i \neq y_i\}|$.

Threshold Gate. Let $x_1, x_2, \ldots, x_n$ be n binary variables. A threshold gate is defined

as the following function:

$$\mathsf{Th}_t(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \sum_{i\in[n]} x_i \geq t \\ 0 & \text{Otherwise} \end{cases}$$

Not-Threshold Gate. A not-threshold gate $\overline{\mathsf{Th}}_t$ is the negation of a threshold gate.

Threshold Circuits and TC^0. A threshold circuit is a directed acyclic graph, where each node either computes a threshold gate of unbounded fan-in or a negation gate.

In this work, for any constant L, we use TC^0_L to denote the class of L-depth polynomial-size threshold circuits. When the depth L is not important or is clear from the context, we omit it and simply denote the circuit class TC^0_L as TC^0. The not-threshold gate is universal for TC^0, since we can convert any threshold circuit of constant depth to a constant depth circuit that only contains not-threshold gates. The conversion works as follows: for each negation gate, we convert it to a not-threshold gate with a single input and threshold $t = 1$. For each threshold gate, we convert it to a not-threshold gate with the same input and threshold and then compose it with a negation gate, where the negation gate can be implemented as a not-threshold gate.

2.1 Cryptographic Assumptions

2.1.1 Number-Theoretic Assumptions

Discrete Logarithm Assumption. In the following, we state the discrete logarithm (DL) assumption.

Definition 2.1.1 (Discrete Logarithm)**.** *A prime-order group generator is an algorithm $\mathcal{G}$ that takes the security parameter λ as input, and outputs a tuple $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of prime order $p(\lambda)$, and g is a generator of $\mathbb{G}$. We say that the DL problem is hard in $\mathcal{G}$, if for any n.u. PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\lambda)$ such that,*

$$\Pr\left[(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda), h \leftarrow \mathbb{G}, x \leftarrow \mathcal{A}(1^\lambda, \mathbb{G}, p, g, h) : g^x = h\right] \leq \nu(\lambda).$$

We say that the DL is sub-exponentially hard in $\mathcal{G}$, if there exists a constant $0 < c < 1$ such that for any n.u. PPT adversary, the success probability is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

Decisional Diffie-Hellman Assumption. In the following, we state the decisional Diffie-Hellman (DDH) assumption.

Definition 2.1.2 (Decisional Diffie-Hellman)**.** *Let $\mathcal{G}$ be a prime-order group generator (as in Definition 2.1.1). We say that $\mathcal{G}$ satisfies the DDH assumption if for any n.u. PPT distinguisher $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that*

$$\Big|\Pr\left[(\mathbb{G},p,g)\leftarrow\mathcal{G}(1^\lambda),a,b\leftarrow\mathbb{Z}_p:\mathcal{D}(1^\lambda,\mathbb{G},p,g,g^a,g^b,g^{ab})=1\right]-$$
$$\Pr\left[(\mathbb{G},p,g)\leftarrow\mathcal{G}(1^\lambda),a,b,c\leftarrow\mathbb{Z}_p:\mathcal{D}(1^\lambda,\mathbb{G},p,g,g^a,g^b,g^c)=1\right]\Big|\leq\nu(\lambda)$$

We say that $\mathcal{G}$ satisfies the sub-exponential DDH assumption, if there exists a constant $0 < c < 1$ such that for any n.u. PPT distinguisher, the advantage $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

2.1.2 Lattice Assumptions

Another central cryptographic assumption we will require in our work is the Learning with Error (LWE) assumption that we define below.

Definition 2.1.3 (Learning with Errors Assumption)**.** *For any positive integers n, q, any $\mathbf{s} \in \mathbb{Z}^n$, and any error distribution χ over $\mathbb{Z}$, the LWE (Learning with Error) distribution $A_{\mathbf{s},\chi}$ is defined by uniformly sampling a vector $\mathbf{a}$, and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $e \leftarrow \chi$.*

The $\mathsf{LWE}_{n,q,\chi}$ assumption states that no non uniform PPT adversary can distinguish, with non-negligible probability, between (i) the distribution $A_{\mathbf{s},\chi}$ for a single $\mathbf{s} \leftarrow \mathbb{Z}_q^n$; and (ii) the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

A standard instantiation of LWE chooses χ as *discrete Gaussian* distribution over $\mathbb{Z}$ with parameters $r = 2\sqrt{n}$. For this parameterization, LWE is at least as hard as quantumly approximating some "short vector" problem on n-dimensional lattices in the worst case to $\tilde{O}(q\sqrt{n})$ factors [76, 77]. There are also classical reductions for different parameterizations [78, 79].

2.2 Non-Interactive Proof Systems

We recall the syntax and security properties associated with non-interactive proof systems in the common random string (CRS) model.

A non-interactive proof system for an $\mathcal{NP}$ language $\mathcal{L}$ with associated relation $\mathcal{R}$ is a tuple of algorithms $\Pi = (\mathsf{CGen}, \mathsf{P}, \mathsf{V})$ described as follows.

- $\mathsf{CGen}(1^\lambda)$: It takes as input the security parameter λ, and outputs a common random string crs.
- $\mathsf{P}(\mathsf{crs}, x, w)$: It takes as input a common random string crs, an instance $x \in \mathcal{L}$, a witness ω, and outputs a proof π.
- $\mathsf{V}(\mathsf{crs}, x, \pi)$: It takes as input a common random string crs, an instance x, a proof π, and decides to accept (output 1) or reject (output 0) the proof.

We now define various properties of non-interactive proof systems that we consider in this thesis.

- **Completeness:** For any instance $x \in \mathcal{L}$, and any witness ω of x, we have

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{CGen}(1^\lambda), \pi \leftarrow \mathsf{P}(\mathsf{crs}, x, \omega) : \mathsf{V}(\mathsf{crs}, x, \pi) = 1\right] = 1.$$

- **Computational Soundness:** For any n.u. PPT cheating prover P^*, there exists a negligible function $\nu(\lambda)$ such that

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{CGen}(1^\lambda), (x, \pi) \leftarrow \mathsf{P}^*(\mathsf{crs}) : x \notin \mathcal{L} \wedge \mathsf{V}(\mathsf{crs}, x, \pi) = 1\right] \leq \nu(\lambda).$$

We say that the proof system achieves sub-exponential computational soundness, if there exists a constant $0 < c < 1$ such that for any n.u. PPT cheating prover, the success probability is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

We refer to the above as **adaptive** soundness. If we modify the above definition s.t. the adversary chooses the statement x before obtaining the CRS, then the resulting notion is referred to as **non-adaptive** soundness.

- **Argument of Knowledge:** There exists a PPT extractor $E = (E_1, E_2)$ such that, for any n.u. PPT prover P^*, there exists a negligible function $\nu(\lambda)$ such that

$$\Pr\left[x \leftarrow \mathsf{P}^*(1^\lambda), (\widetilde{\mathsf{crs}}, \mathsf{td}) \leftarrow E_1(1^\lambda), \pi \leftarrow \mathsf{P}^*(\widetilde{\mathsf{crs}}), \omega \leftarrow E_2(\mathsf{td}, x, \pi) : \mathcal{R}(x, \omega) = 1\right] \geq \Pr\left[x \leftarrow \mathsf{P}^*(1^\lambda), \mathsf{crs} \leftarrow \mathsf{CGen}(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) : \mathsf{V}(\mathsf{crs}, x, \pi) = 1\right] - \nu(\lambda).$$

We say that the proof system achieves sub-exponential non-adaptive argument of knowledge, if there exists a constant $0 < c < 1$ such that for any n.u. PPT cheating prover, $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

We refer to the above as **non-adaptive** argument of knowledge. If we modify the above definition such that the adversary chooses the statement x after obtaining the CRS, and $\mathsf{SD}(\mathsf{crs}, \widetilde{\mathsf{crs}}) \leq \nu(\lambda)$, then the resulting notion is referred to as **adaptive** argument of knowledge. (Note that this definition is slightly stronger in the sense that we require the CRS output by CGen and E_1 to be statistically close.)

- **Adaptive Statistical Witness Indistinguishability (SWI):** For any unbounded adversary $\mathcal{A}$, there exists a negligible function $\nu(\lambda)$ such that

$$|\Pr[\mathsf{Expr}_0 = 1] - \Pr[\mathsf{Expr}_1 = 1]| \leq \nu(\lambda),$$

where Expr_b, for every $b \in \{0, 1\}$, is defined as the following experiment:

Experiment Expr_b**:**

- $\mathsf{crs} \leftarrow \mathsf{CGen}(1^\lambda)$.
- $(x, \omega_0, \omega_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs})$.
- If $\mathcal{R}(x, \omega_0) \neq 1$ or $\mathcal{R}(x, \omega_1) \neq 1$, output 0 and halt.
- $\pi \leftarrow \mathsf{P}(\mathsf{crs}, x, \omega_b)$.
- Output $\mathcal{A}(\mathsf{crs}, \pi)$.

We say that the proof system satisfies adaptive **computational** witness indistinguishability if the above condition holds for any non-uniform PPT adversary.

- **Adaptive Statistical Zero Knowledge (SZK):** There exists a simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ such that for any unbounded adversary $\mathcal{A}$ and polynomial $Q(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Big| \Pr\left[\mathsf{crs} \leftarrow \mathsf{CGen}(1^\lambda) : \mathcal{A}^{\mathsf{Real}(\cdot,\cdot)}(1^\lambda, \mathsf{crs}) = 1\right] - \Pr\left[(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{S}_1(1^\lambda) : \mathcal{A}^{\mathsf{Ideal}(\cdot,\cdot)}(1^\lambda, \mathsf{crs}) = 1\right] \Big| \leq \nu(\lambda)$$

where the oracles $\mathsf{Real}(\cdot,\cdot)$ and $\mathsf{Ideal}(\cdot,\cdot)$ are defined as follows, and $\mathcal{A}$ makes at most $Q(\lambda)$ queries to each oracle.

$\underline{\mathsf{Real}(x, \omega)}$

- If $\mathcal{R}(x, \omega) \neq 1$ output $\perp$.
- Otherwise, output $\pi \leftarrow \mathsf{P}(\mathsf{crs}, x, \omega)$.

$\underline{\mathsf{Ideal}(x, \omega)}$

- If $\mathcal{R}(x, \omega) \neq 1$ output $\perp$.
- Otherwise, output $\pi \leftarrow \mathsf{S}_2(\mathsf{td}, x)$.

We say that the proof system satisfies adaptive **computational** zero knowledge if the above condition holds for any non-uniform PPT adversary.

2.3 Statistical Zap Arguments

Zaps [80] are two-round witness indistinguishable proof systems with a public-coin verifier message. Below, we define statistical Zap arguments, i.e., Zaps that achieve statistical WI property and computational soundness.

A statistical Zap argument for an $\mathcal{NP}$ language L is a two-round protocol (P, V) with a public-coin verifier message that satisfies the following properties:

- **Completeness:** For every $x \in L$ and witness ω for x, we have that

$$\Pr\left[\mathsf{Out}_{\mathsf{V}}\left(\mathsf{P}(1^\lambda, x, \omega) \leftrightarrow \mathsf{V}(1^\lambda, x)\right) = 1\right] = 1$$

  where $\mathsf{Out}_{\mathsf{V}}(e)$ is the output of V in a protocol execution e.

- **Computational Soundness:** For any non-uniform PPT prover P^*, there exists a negligible function $\nu(\cdot)$ such that for any $x \notin L$, we have that

$$\Pr\left[\mathsf{Out}_{\mathsf{V}}\left(\mathsf{P}^*(1^\lambda, x) \leftrightarrow \mathsf{V}(1^\lambda, x)\right) = 1\right] \leq \nu(\lambda)$$

  The above is referred to as **non-adaptive** soundness. If we modify the above definition s.t. the adversary chooses the statement x after receiving the verifier's message, then the resulting notion is referred to as **adaptive** soundness.

- **Statistical Witness Indistinguishability:** For any unbounded verifier V^*, there exists a negligible function $\nu(\cdot)$ such that for every $x \in L$, and witnesses ω_1, ω_2 for x, we have that

$$\mathsf{SD}\left(\mathsf{Trans}\left(\mathsf{P}(1^\lambda, x, \omega_1) \leftrightarrow \mathsf{V}^*(1^\lambda, x)\right), \mathsf{Trans}\left(\mathsf{P}(1^\lambda, x, \omega_2) \leftrightarrow \mathsf{V}^*(1^\lambda, x)\right)\right) \leq \nu(\lambda)$$

  where $\mathsf{Trans}(e)$ is the transcript of a protocol execution e.

2.4 Building Blocks

In this thesis, we will use the following build blocks.

2.4.1 Two-Round Oblivious Transfer

Definition 2.4.1. *A statistical sender-private oblivious transfer (OT) is a tuple of algorithms* $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$*:*

- $\mathsf{OT}_1(1^\lambda, b)$*: On input security parameter* λ*, a bit* $b \in \{0,1\}$*,* OT_1 *outputs the first round message* ot_1 *and a state* st*.*
- $\mathsf{OT}_2(1^\lambda, \mathsf{ot}_1, m_0, m_1)$*: On input security parameter* λ*, a first round message* ot_1*, two bits* $m_0, m_1 \in \{0,1\}$*,* OT_2 *outputs the second round message* ot_2*.*
- $\mathsf{OT}_3(1^\lambda, \mathsf{ot}_2, \mathsf{st})$*: On input security parameter* λ*, the second round message* ot_2*, and the state generated by* OT_1*,* OT_3 *outputs a message* m*.*

We require the following properties:

- **Correctness:** *For any* $b, m_0, m_1 \in \{0,1\}$*,*

$$\Pr[(\mathsf{ot}_1, \mathsf{st}) \leftarrow \mathsf{OT}_1(1^\lambda, b), \mathsf{ot}_2 \leftarrow \mathsf{OT}_2(1^\lambda, \mathsf{ot}_1, m_0, m_1), m \leftarrow \mathsf{OT}_3(1^\lambda, \mathsf{ot}_2, \mathsf{st}) : m = m_b] = 1$$

- **Statistical Sender Privacy:** *There exists a negligible function* $\nu(\lambda)$ *and an deterministic exponential time extractor* OTExt *such that for any (potential maliciously generated)* ot_1*,* $\mathsf{OTExt}(1^\lambda, \mathsf{ot}_1)$ *outputs a bit* $b \in \{0,1\}$*. Then for any* $m_0, m_1 \in \{0,1\}$*, we have*

$$\mathsf{SD}\left(\mathsf{OT}_2(1^\lambda, \mathsf{ot}_1, m_0, m_1), \mathsf{OT}_2(1^\lambda, \mathsf{ot}_1, m_b, m_b)\right) \leq \nu(\lambda)$$

- **Pseudorandom Receiver's Message:** *For any $b \in \{0,1\}$, let ot_1 be the first round message generated by $\mathsf{OT}_1(1^\lambda, b)$. For any n.u. PPT adversary $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that, for any $\lambda \in \mathbb{N}$,*

$$\left| \Pr\left[\mathcal{D}(1^\lambda, \mathsf{ot}_1) = 1\right] - \Pr\left[u \leftarrow \{0,1\}^{|\mathsf{ot}_1|} : \mathcal{D}(1^\lambda, u) = 1\right] \right| \leq \nu(\lambda)$$

Furthermore, we say that the OT satisfies the sub-exponential pseudorandom receiver's message property, if there exists a constant $0 < c < 1$ such that for any n.u. PPT adversary, the advantage $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

Lemma 2.4.2. *Assuming DDH, there exists a two-round oblivious transfer.*

A two-round oblivious transfer from DDH was constructed by [81]. Their construction satisfies correctness and statistical sender-privacy. Further, the receiver's message in their scheme is (sub-exponentially) pseudorandom, assuming (sub-exponential) DDH.

2.4.2 Rate-1 Trapdoor Hash Functions

We recall the notion of (rate-1) trapdoor hash functions (TDH) introduced in [82]. For our constructions, we require TDH with an "enhanced correctness" property, as defined in [51].

Previously, [82] constructed TDH for index predicates. Their construction was later generalized by [51] to linear functions and constant degree polynomials over $\mathbb{Z}_2$. In this work, we consider a further generalized family of functions, namely, linear functions over $\mathbb{Z}_R$, where R is a polynomial in the security parameter.

Definition 2.4.3 (Linear Function Family)**.** *$\mathcal{F} = \{\mathcal{F}_{n,R}\}_{n,R}$ is a family of linear functions over $\mathbb{Z}_R$ if every $f \in \mathcal{F}_{n,R}$ is of the form:*

$$f(x_1, x_2, \ldots, x_n) = (a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n) \bmod R,$$

where $0 \leq a_i < R$ *for every* $i \in \{0, \ldots, n\}$.

Definition. A trapdoor hash function for $\mathcal{F}$ is a tuple of algorithms $\mathsf{TDH} = (\mathsf{HKGen}, \mathsf{EKGen}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{Dec})$ described as follows:

- $\mathsf{HKGen}(1^\lambda, 1^n, 1^R)$: The hash key generation algorithm takes as input a security parameter λ, input length n, and a modulo R. It outputs a hash key hk.
- $\mathsf{EKGen}(\mathsf{hk}, f)$: The encoding key generation algorithm takes as input a hash key hk and a circuit $f \in \mathcal{F}_n$, and it outputs an encoding key ek together with a trapdoor td.
- $\mathsf{Hash}(\mathsf{hk}, x)$: The hashing algorithm takes as input a hash key hk and an input value x, and it outputs a hash value $\mathsf{h} \in \{0,1\}^\eta$.
- $\mathsf{Enc}(\mathsf{ek}, x)$: The encoding algorithm takes as input an encoding key ek and an input $x \in \{0,1\}^n$, and it outputs an encoding $\mathsf{e} \in \mathbb{Z}_R$.
- $\mathsf{Dec}(\mathsf{td}, h)$: The decoding algorithm takes as input a trapdoor td and the hash value h, and it outputs a value $\mathsf{d} \in \mathbb{Z}_R$.

We require TDH to satisfy the following properties:

- **Compactness:** The bit-length η of a hash value h is *independent* of n, and is a fixed polynomial in λ. For simplicity, in this work, we require that $\eta \leq \lambda$.
- **τ-Enhanced Correctness:** For any $\lambda, n, R \in \mathbb{N}$, any string $\mathsf{h} \in \{0,1\}^{\eta(\lambda)}$, any $f \in \mathcal{F}_{n,R}$, and any hk output by $\mathsf{HKGen}(1^\lambda, 1^n, 1^R)$, we have

$$\Pr\left[(\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{EKGen}(\mathsf{hk}, f) : \forall x \text{ s.t. } \mathsf{Hash}(\mathsf{hk}, x) = \mathsf{h}, f(x) = (\mathsf{e} + \mathsf{d}) \bmod R\right] \geq 1 - \tau(\lambda),$$

  where $\mathsf{e} = \mathsf{Enc}(\mathsf{ek}, x), \mathsf{d} = \mathsf{Dec}(\mathsf{td}, \mathsf{h})$, and the probability is over the randomness of EKGen.

- **Function Privacy:** There exists a simulator Sim and a negligible function $\nu(\lambda)$ such that, for any polynomials n and R in the security parameter λ, there exists a constant $c < 1$ such that for any $\lambda \in \mathbb{N}$, any function $f \in \mathcal{F}_{n,R}$, and any n.u. PPT adversary $\mathcal{D}$,

$$\Big| \Pr\Big[\mathsf{hk} \leftarrow \mathsf{HKGen}(1^\lambda, 1^n, 1^R), (\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{EKGen}(\mathsf{hk}, f) : \mathcal{D}(1^\lambda, (\mathsf{hk}, \mathsf{ek})) = 1\Big] - \Pr\Big[\mathsf{hk} \leftarrow \mathsf{HKGen}(1^\lambda, 1^n, 1^R), \widetilde{\mathsf{ek}} \leftarrow \mathsf{Sim}(1^\lambda, 1^n, 1^R) : \mathcal{D}(1^\lambda, (\mathsf{hk}, \widetilde{\mathsf{ek}})) = 1\Big] \Big| \leq \nu(\lambda)$$

  We say that the TDH achieves sub-exponential function privacy, if there exists a constant $0 < c < 1$ such that for any n.u. PPT adversary, the advantage $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

Theorem 2.4.4. *Assuming sub-exponential DDH, for any inverse polynomial τ in the security parameter λ, there exists a* TDH *construction for the linear function family $\mathcal{F} = \{\mathcal{F}_{n,R}\}_{n,R}$ with $\tau(\lambda)$-enhanced correctness and sub-exponential function privacy.*

The proof of this theorem follows via a simple modification of the TDH construction in [82]. For completeness, we present it here.

Construction of Trapdoor Hash for Linear Functions over $\mathbb{Z}_R$. In the following construction, we use a hash function $\phi : \mathbb{G} \to \{0,1\}^*$, which is sampled from a hash function family Φ. Here we can use a pseudo-random function as in the previous work [51], or k-wise independent hash functions.

- $\mathsf{HKGen}(1^\lambda, 1^n, 1^R)$:
  - Generate a group $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$.
  - For each $i \in [n]$, sample an element uniformly at random from the group $\mathbb{G}$, $h_i \leftarrow \mathbb{G}$.
  - Output $\mathsf{hk} = (\mathbb{G}, p, g, \{h_i\}_{i\in[n]}, R)$.

- $\mathsf{EKGen}(\mathsf{hk}, f)$:
  - Let the linear function

    $f(x_1, x_2, \ldots, x_n) = (a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n) \bmod R$, where $0 \leq a_i < R$.

  - Sample a secret $s \leftarrow \mathbb{Z}_p$ uniformly at random. For each $i \in [n]$, let $f_i = h_i^s \cdot g^{a_i}$.
  - Set the parameters $\tau' = \lceil \log_2(2R^2 \cdot (n+1)/\tau) \rceil + 1$, and $T = 2^{\tau'} \lceil \ln(2/\tau) \rceil + 1$.
  - Let Φ be a family of pseudo-random functions from $\mathbb{G}$ to $\{0,1\}^{\tau'}$. Sample $\phi \leftarrow \Phi$.
  - Output $(\mathsf{ek} = (\phi, \{f_i\}_{i \in [n]}), \mathsf{td} = (s, a_0))$.
- $\mathsf{Hash}(\mathsf{hk}, x)$:
  - Let $x = (x_1, x_2, \ldots, x_n)$, where $0 \leq x_i < R$, for each $i \in [n]$.
  - Output $\mathsf{h} = \prod_{i \in [n]} h_i^{x_i}$
- $\mathsf{Enc}(\mathsf{ek}, x)$:
  - Let $h_e \leftarrow \prod_{i \in [n]} f_i^{x_i}$.
  - Output $\mathsf{DLog}(g, \phi, R, h_e)$.
- $\mathsf{Dec}(\mathsf{td}, \mathsf{h})$:
  - Let $h_d = h^s \cdot g^{-a_0}$.
  - Output $(-\mathsf{DLog}(g, \phi, R, h_d)) \bmod R$.

Lemma 2.4.5. *Let S be an integer, and $0 < \tau < 1$ be a real number. Let $\Phi = \{\phi : \mathbb{G} \to \{0,1\}^{\tau'}\}$ be a pseudo-random hash function family with output length*

Distributed Discrete Logarithm $\mathsf{DLog}(g, \phi, R, h)$

- Let $i = 0$
- Loop while $i \leq T$
  - If $\phi(h \cdot g^i) = 0$, output $(i \bmod R)$.
  - Let $i = i + 1$.

  Output 0.

Figure 2-1. Description of the distributed discrete logarithm algorithm DLog.

$\tau' = \lceil \log_2(2S/\tau) \rceil + 1$, *and set parameter* $T = 2^{\tau'} \lceil \ln(2/\tau) \rceil + 1$. *Then, for any cyclic group* $\mathbb{G}$ *of order* p *with generator* g, *and any* $h \in \mathbb{G}$, *we have*

$$\Pr_{\phi \leftarrow \Phi} \left[\forall 0 \leq x < S, (\mathsf{DLog}(g, \phi, R, h \cdot g^x) - \mathsf{DLog}(g, \phi, R, h)) \bmod R = x \right] > 1 - \tau - \mathsf{negl}(\lambda).$$

Furthermore, if S *is a polynomial in* n, *and* τ *is an inverse polynomial in* n, *then running time of the algorithm* DLog *is also a polynomial in* n.

Proof. The proof follows the same strategy as Proposition B.2 in [51]. Since the input to ϕ is fixed before ϕ is sampled, we can switch ϕ to a random function. For ease of presentation, let the event E denote $\forall 0 \leq x < S, (\mathsf{DLog}(g, \phi, R, h \cdot g^x) - \mathsf{DLog}(g, \phi, R, h)) \bmod R = x$. We consider three cases:

- **Case 1:** There exists a $0 \leq i < S$ such that $\phi(h \cdot g^i) = 0$. In this case, E may not hold, hence we bound the probability of this case. By the union bound, we bound it by $S/2^{\tau'}$.
- **Case 2:** For any $0 \leq i < S, \phi(h \cdot g^i) \neq 0$, and there exists a $S \leq i \leq T$ such that $\phi(h \cdot g^i) = 0$. In this case, E always holds.
- **Case 3:** For any $i \leq T, \phi(h \cdot g^i) \neq 0$. In this case, E may not hold. Hence, we

bound the probability of this case. Since the hash function ϕ is random, we bound the probability by $(1-2^{-\tau'})^T$.

In total, we have

$$\Pr_{\phi}[E] \geq 1 - \left(\frac{S}{2^{\tau'}} + \left(1-2^{-\tau'}\right)^T\right) > 1 - (\tau/2 + \tau/2) \geq 1 - \tau.$$

□

Lemma 2.4.6 (τ-Enhanced Correctness for TDH)**.** *The above construction of* TDH *satisfies τ-enhanced correctness.*

Proof. For any polynomial $R = R(n)$, any hash value $\mathsf{h} \in \mathbb{G}$, and any linear function $f \in \mathcal{F}_{n,R}$, and any hash key hk output by $\mathsf{HKGen}(1^\lambda, 1^n, 1^R)$,

$$\begin{aligned}
&\Pr_{\phi,\{f_i\}_{i\in[n]}}[\forall x : \mathsf{Hash}(\mathsf{hk}, x) = \mathsf{h}, f(x) = (\mathsf{e}+\mathsf{d}) \bmod R] \\
=&\Pr_{\phi\leftarrow\Phi}[\forall x : \mathsf{Hash}(\mathsf{hk}, x) = \mathsf{h}, f(x) = (\mathsf{DLog}(g,\phi,R,h_e) - \mathsf{DLog}(g,\phi,R,h_d)) \bmod R] \\
\geq&\Pr_{h\leftarrow\Phi}\left[\forall 0 \leq y \leq R^2\cdot(n+1), (\mathsf{DLog}(g,\phi,R,h_d\cdot g^y) - \mathsf{DLog}(g,\phi,R,h_d)) \bmod R = y\right] \\
>&1-\tau
\end{aligned}$$

The second line follows from the definition of e and d. The third line follows from

$$\begin{aligned}
h_e &= \prod_{i\in[n]} f_i^{x_i} = \prod_{i\in[n]} (h_i^s \cdot g^{a_i})^{x_i} = \prod_{i\in[n]} (h_i^{x_i})^s \cdot g^{\sum_i a_i x_i} = (h^s \cdot g^{-a_0}) \cdot g^{a_0+\sum_i a_i x_i} \\
&= h_d \cdot g^{a_0+\sum_i a_i x_i}.
\end{aligned}$$

Let $y = a_0 + \sum_{i\in[n]} a_i x_i$, then we have $0 \leq y < R^2\cdot(n+1)$. The forth line follows from Lemma 2.4.5. □

Lemma 2.4.7 (Sub-exponential Function Privacy)**.** *Assuming sub-exponential hardness of DDH, the construction of* TDH *satisfies sub-exponential function privacy.*

The proof of this lemma follows in the same manner as the proof of Theorem 4.2 in [82] since the ek is the same as in the construction of [82].

2.4.3 Low-degree Extensions

For any field $\mathbb{H}$ and any extension field $\mathbb{F}$ of $\mathbb{H}$, any index $(i_1, i_2, \ldots, i_m) \in \mathbb{H}^m$, let $\widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_m}$ be the following polynomial over $\mathbb{F}[x_1, x_2, \ldots, x_m]$.

$$\widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_m}(x_1, x_2, \ldots, x_m) = \frac{\prod_{j_1 \in \mathbb{H}\setminus\{i_1\}}(x_1 - j_1) \cdot \prod_{j_2 \in \mathbb{H}\setminus\{i_2\}}(x_1 - j_1) \ldots \prod_{j_m \in \mathbb{H}\setminus\{i_m\}}(x_m - j_m)}{\prod_{j_1 \in \mathbb{H}\setminus\{i_1\}}(i_1 - j_1) \cdot \prod_{j_2 \in \mathbb{H}\setminus\{i_2\}}(i_1 - j_1) \ldots \prod_{j_m \in \mathbb{H}\setminus\{i_m\}}(i_m - j_m)}$$

For any string $x \in \{0,1\}^n$, where $n = |\mathbb{H}|^m$, we identify the set $\mathbb{H}^m$ with the index set $[n]$. Then we define the low-degree extension of x, $\mathsf{LDE}(x)$, as the following polynomial in $\mathbb{F}[x_1, x_2, \ldots, x_m]$,

$$\mathsf{LDE}(x) = \sum_{i_1,i_2,\ldots i_m \in \mathbb{H}} x_{i_1,i_2,\ldots,i_m} \cdot \widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_m}(x_1, x_2, \ldots, x_m).$$

2.4.4 Somewhere Extractable Commitment

In this subsection, we define *somewhere extractable commitments*. A somewhere extractable commitment has a key with two computationally indistinguishable modes: (i) In the *normal mode*, the key is *uniformly random*; and (ii) in the *trapdoor mode*, the key is generated according to a subset S denoting the coordinates of the committed message.

Furthermore, we require the following properties.

- **Efficiency:** We require that the size of the CRS and commitment roughly grow with $|S|$.
- **Extraction:** The trapdoor mode commitment key is associated with a trapdoor td, such that given the trapdoor, one can extract the message on coordinates in S. Note that the extraction implies the statistical binding property for the coordinates in S.
- **Local Opening:** We allow the prover to generate a *local opening* for any single coordinate of the message. The local opening needs to have a small size, which

only grows poly-logarithmically with the total length of the message. Moreover, we require that the value from the local opening should be consistent with the extracted value.

We note that this notion is essentially the same as somewhere statistical binding hash [83], except that we explicitly require an extraction property (although as we will see, this property is already satisfied by the construction of [83]). This notion is also similar to the notion of somewhere-extractable linearly homomorphic commitment in [70], except that here we do not require linear homomorphism property, but we further require local opening property.

We now move to the formal definition. A somewhere extractable commitment scheme is a tuple of algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext})$ described below.

- $\mathsf{Gen}(1^\lambda, 1^N, 1^{|S|})$: On input a security parameter, the length of the message N, and the size of a subset $S \subseteq [N]$, the "normal mode" key generation algorithm outputs a *uniformly random* commitment key K.
- $\mathsf{TGen}(1^\lambda, 1^N, S)$: On input a security parameter, the length of the message N, an extraction subset $S \subseteq [N]$, the "trapdoor mode" key generation algorithm outputs a commitment key K^* and a trapdoor td.
- $\mathsf{Com}(K, \mathbf{m} \in \{0,1\}^N; r)$: On input the commitment key K, a vector $\mathbf{m} = (m_1, m_2, \ldots, m_N) \in \{0,1\}^N$, and the random coins r, it outputs a commitment c.
- $\mathsf{Open}(K, \mathbf{m}, i, r)$: On input the commitment key K, a vector $\mathbf{m} = (m_1, m_2, \ldots, m_N) \in \{0,1\}^N$, an index $i \in [N]$, and the random coins r, the opening algorithm outputs a *local opening* π_i to m_i.
- $\mathsf{Verify}(K, c, m_i, i, \pi_i)$: On input the commitment key K, a commitment c, a bit $m_i \in \{0,1\}$, and a local opening π_i, the verification algorithm decides to accept (output 1) or reject (output 0) the local opening.

- $\mathsf{Ext}(c, \mathsf{td})$: On input a commitment c, and the trapdoor td generated by the trapdoor key generation algorithm TGen with respect to the subset S, the extraction algorithm outputs an extraction string m_S^* on the subset S.

Furthermore, we require the commitment scheme to satisfy the following properties.

- **Succinct CRS:** The size of the CRS is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Succinct Commitment:** The size of the commitment c is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Succinct Local Opening:** The size of the local opening $\pi_i \leftarrow \mathsf{Open}(K, m, i, r)$ is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Succinct Verification:** The running time of the verification algorithm is bounded by $\mathsf{poly}(\lambda, |S|, \log N)$.
- **Key Indistinguishability:** For any non-uniform PPT adversary $\mathcal{A}$ and any polynomial $N = N(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Big| \Pr\left[S \leftarrow \mathcal{A}(1^\lambda, 1^N), K \leftarrow \mathsf{Gen}(1^\lambda, 1^N, 1^{|S|}) : \mathcal{A}(K) = 1\right] - \Pr\left[S \leftarrow \mathcal{A}(1^\lambda, 1^N), (K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S) : \mathcal{A}(K^*) = 1\right] \Big| \leq \nu(\lambda).$$

- **Opening Completeness.** For any commitment key K, any message $\mathbf{m} = (m_1, \ldots, m_N) \in \{0,1\}^N$, any randomness r, and any index $i \in [N]$, we have

$$\Pr\left[c \leftarrow \mathsf{Com}(K, \mathbf{m}; r), \pi_i \leftarrow \mathsf{Open}(K, \mathbf{m}, i, r) : \mathsf{Verify}(K, c, m_i, i, \pi_i) = 1\right] = 1.$$

- **Extraction Correctness.** For any subset $S \subseteq [N]$, any trapdoor key $(K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)$, any commitment c, any index $i \in [N]$, any bit $m_{i^*} \in \{0,1\}$, and any proof π_{i^*}, we have

$$\Pr\left[\mathsf{Verify}(K, c, m_{i^*}, i^*, \pi_{i^*}) = 1 \Rightarrow \mathsf{Ext}(c, \mathsf{td})|_{i^*} = m_{i^*}\right] = 1.$$

  Since the extracted value $\mathsf{Ext}(c, \mathsf{td})|_{i^*}$ is unique, the extraction correctness implies statistical binding property.

Theorem 2.4.8. *There exists a construction of somewhere extractable commitment from LWE.*

Proof Sketch. Theorem 2.4.8 is implicit in [83]. We briefly recall the construction of the somewhere statistical binding hash in [83] here. For the ease of presentation, we only describe the construction for $|S| = 1$ as in [83]. The construction for general S can be obtained by using multiple copies of such commitments.

The commitment key consists of a fully homomorphic encryption of the index i^* in the set S. To hash a message $(m_1, m_2, \ldots, m_N)$, they build a Merkle Tree, where each node of the Merkle Tree is associated with a ciphertext. The leaf nodes contains the encryption of m_i's, and for the path from m_{i^*} to the root, the ciphertext contains an encryption of m_{i^*}. This is achieved by homomorphically evaluating a circuit that selects the left or the right child according to i^* on each node of the Merkle Tree. Since the fully homomorphic encryption ciphertext is computationally indistinguishable with uniformly random string, we can use uniformly random string in the "normal mode". The local opening follows from the Merkle Tree structure.

The extraction property is implicitly satisfied by the construction. Specifically, the trapdoor corresponds to the secret key of the fully homomorphic encryption. Given the secret key, we can decrypt the root node to extract m_{i^*}. □

2.4.5 No-Signaling Somewhere Extractable Commitments

We consider here a slight variant of no-signaling somewhere extractable (NS-SE) commitments introduced in the work of [84]. The no-signaling property, as described in the technical overview is imposed on the extractor of the SE commitment scheme. Intuitively, an extractor for an SE scheme is said to be *computationally no-signaling* if for any sets $S' \subseteq S$, where S is of size at most L, the extracted values corresponding to the indices in S' have computationally indistinguishable marginal distributions

whether extracted on set S or S'.

Definition 2.4.9. *The extractor of an* SECOM *commitment scheme* $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext})$ *is no-signaling if for any* $S' \subseteq S \subseteq [N]$*, where* $|S| \leq L$*, and any* PPT *adversary* $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\left| \Pr\left[\mathcal{D}_2(K^*, c, \vec{y}, z) \;\middle|\; \begin{array}{l} (K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S') \\ (c, z) \leftarrow \mathcal{D}_1(K^*) \\ \vec{y} := \mathsf{Ext}(c, \mathsf{td}) \end{array} \right] - \Pr\left[\mathcal{D}_2(K^*, c, \vec{y}_{S'}, z) \;\middle|\; \begin{array}{l} (K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S) \\ (c, z) \leftarrow \mathcal{D}_1(K^*) \\ \vec{y} := \mathsf{Ext}(c, \mathsf{td}) \end{array} \right] \right| \leq \mathsf{negl}(\lambda)$$

We will refer to SECOM schemes satisfying the above definition to be an L-no-signaling NS-SECOM commitment.

Theorem 2.4.10 ([84])**.** *Given* L *instances of an* SECOM *commitment scheme* $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext})$ *with locality parameter 1, one can construct an* L*-no-signaling* NS-SECOM*.*

Construction. We first sketch the construction from [84], and then present details. For simplicity we consider here the case that $S := \{s_1, \cdots, s_L\}$ has size exactly L. The rough idea is to generate L different commitment keys $K' = (K_1, \cdots, K_L)$ such that to commit to a vector $\vec{m}$, one produces L commitments $\mathsf{Com}(K_i, \vec{m})$ (with different randomness for each i). For the trapdoor key generation algorithm, $K'^* = (K_1^*, \cdots, K_L^*)$, where each K_i^* is generated for the single element set $\{s_i\}$. Therefore the size of the keys and commitment in the L-no-signaling NS-SECOM are larger by a multiplicative factor of L.

Next, we represent the full construction and security proof.

Theorem 2.4.11. *Given* ℓ *instances of an* SECOM *commitment scheme* $\mathsf{SECOM} = (\mathsf{SECOM.Gen}, \mathsf{SECOM.TGen}, \mathsf{SECOM.Com}, \mathsf{SECOM.Open}, \mathsf{SECOM.Verify}, \mathsf{SECOM.Ext})$ *with locality parameter 1, one can construct an* L*-no-signaling* NS-SECOM*.*

Proof. We construct the L-no-signaling $\mathsf{NS\text{-}SECOM}$ as follows.

- $\underline{\mathsf{TGen}(1^\lambda, 1^N, S)}$: The key generation algorithm generates a 1-SECOM key for each element in S, and also generates the remaining $(L - |S|)$-SECOM keys. Then it outputs a random shuffle of all the generated keys.
  - Let $S = \{s_1, s_2, \ldots, s_{|S|}\}$. For each $i \in [|S|]$, let $(K_i^*, \mathsf{td}_i) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^N, \{s_i\})$.
  - For $[L] \setminus [|S|]$, $K_i^* \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^N, \phi)$, where ϕ is the empty set.
  - Let $\pi : [L] \to [L]$ be a random shuffle. Output $K^* = \{K^*_{\pi(i)}\}_{i\in[L]}$, and $\mathsf{td} = (\{\mathsf{td}_i\}_{i\in[|S|]}, \pi)$.
- $\underline{\mathsf{Com}(K, \mathbf{m} \in \{0,1\}^N; r)}$: The commitment algorithm commits the message $\mathbf{m}$ for each key specified in K.
  - Parse $r = r_1, r_2, \ldots, r_L$, and $K = \{K'_i\}_{i\in[L]}$.
  - For each $i \in [L]$, compute $c_i \leftarrow \mathsf{SECOM.Com}(K'_i, \mathbf{m}; r_i)$. Output $c = \{c_i\}_{i\in[L]}$.
- $\underline{\mathsf{Ext}(c, \mathsf{td})}$: The extraction algorithm extracts for each element in S. It uses π to recover the order.
  - Parse $c = \{c_i\}_{i\in[L]}$, and $\mathsf{td} = (\{\mathsf{td}_i\}_{i\in[|S|]}, \pi)$.
  - For each $i \in [|S|]$, let $m_i^* \leftarrow \mathsf{SECOM.Ext}(c_{\pi(i)}, \mathsf{td}_i)$. Output $m_S^* = \{m_i^*\}_{i\in[|S|]}$.

The local opening algorithm Open and the local verification algorithm Verify repeat the same algorithms in SECOM for L times. We omit the details here.

To prove the aforementioned construction is L-no-signaling, we build a series of hybrids. For any two sets $S' \subseteq S$,

- Hyb_0: This hybrid uses S to generate the commitment key.

– $(K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)$, $(c, z) \leftarrow \mathcal{D}_1(K^*)$, $\vec{y} := \mathsf{Ext}(c, \mathsf{td})$, Output $\mathcal{D}_2(K^*, c, \vec{y}|_{S'}, z)$.

- Hyb_1: This bybrid is almost the same as Hyb_0, except that, we rearrange the elements in S as $\{s_1, s_2, \ldots, s_{|S|}\}$ such that the first $|S'|$ elements are exactly S', i.e. $S' = \{s_1, s_2, \ldots, s_{|S'|}\}$. Furthermore, we replace the computation of $\vec{y}|_{S'}$ as direct computation without using $\{\mathsf{td}_i\}_{i=|S'|+1,\ldots,|S|}$.

  – $(K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S)$, $(c, z) \leftarrow \mathcal{D}_1(K^*)$.

  – Parse $c = \{c_i\}_{i\in[L]}$. For each $i \in [|S'|]$, let $y'_i := \mathsf{SECOM.Ext}(c_{\pi(i)}, \mathsf{td}_i)$. Output $\mathcal{D}_2(K^*, c, \{y'_i\}_i, z)$.

  This hybrid is identical to the hybrid Hyb_0, because the random shuffle π completely hides the order we represent S.

- $\mathsf{Hyb}_2^{j^*}$: This hybrid is almost the same as Hyb_1, except that we replace the TGen as follows.

  – For each $i \in |S|$, if $i \leq |S'|+j^*$, then let $K^*_i \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^N, \phi)$ and $\mathsf{td}_i = \phi$. For each $|S'|+j^* < i \leq |S|$, let $(K^*_i, \mathsf{td}_i) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^N, \{s_i\})$.

  – For $[L] \setminus [|S|]$, $K^*_i \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^N, \phi)$, where ϕ is the empty set.

  – Let $\pi : [L] \to [L]$ be a random shuffle. Output $K^* = \{K^*_{\pi(i)}\}_{i\in[L]}$, and $\mathsf{td} = (\{\mathsf{td}_i\}_{i\in[|S|]}, \pi)$.

  Hyb_2^1 is computationally indistinguishable with Hyb_1. Furthermore, $\mathsf{Hyb}_2^{j^*}$ is computationally indistinguishable with $\mathsf{Hyb}_2^{j^*+1}$. This follows from the key indistinguishability of SECOM.

- $\mathsf{Hyb}_3 = \mathsf{Hyb}_2^{|S|-|S'|}$. This hybrid is identical to the following hybrid.

  – $(K^*, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^N, S')$, $(c, z) \leftarrow \mathcal{D}_1(K^*)$, $\vec{y} := \mathsf{Ext}(c, \mathsf{td})$, Output $\mathcal{D}_2(K^*, c, \vec{y}, z)$.

By the hybrid argument, Hyb_0 and Hyb_3 are computationally indistinguishable. We finish the proof. □

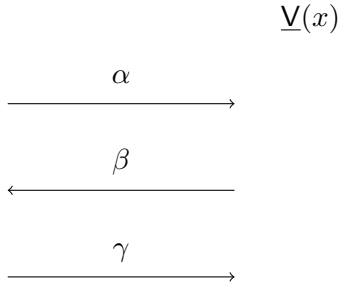
Preservation of succinct local opening. In our work, we will require local opening of the L-no-signaling NS-SECOM to be succinct. From the above construction it is clear that if the underlying SECOM has a succinct local opening, then the size of the succinct opening of the L-no-signaling NS-SECOM is larger by a multiplicative factor of L - one simply provides succinct local openings to each of the L underlying SECOMs.

Chapter 3

Background: Fiat-Shamir

At a very high level, the Fiat-Shamir transform [66] is a round collapsing transformation that allows one to start with an *interactive proof* of a specified structure, and transform it into a *non-interactive argument* in the CRS model. Specifically, the starting interactive proof system (P, V) must be public coin i.e. a protocol where the verifier only sends random coins as its messages.

The transformation is defined with respect to some hash function family $\mathcal{H}$, where the sampled hash function, $h \leftarrow \mathcal{H}$, is set to be the CRS. The prover can then derive the verifier's messages non-interactively by applying h on the protocol transcript. Consider the following interactive protocol between the prover P and verifier V establishing that $x \in \mathcal{L}$, where V's message β is a uniformly random string:



To generate a non-interactive proof, P computes $\beta = h(x, \alpha)$ with the resultant proof being (α, γ). V can recompute β (from x and α) and check if the transcript $(x, \alpha, \beta, \gamma)$ is accepting. Note that the total communication from the prover to the

verifier remains unchanged by the transformation. Therefore, when proof size is the main concern in the non-interactive setting, it is important to start with an interactive protocol that already satisfies the communication requirements.

3.1 Soundness of Fiat-Shamir transform

Initially, the soundness (i.e. inability of a cheating prover to generate an accepting proof when $x \notin \mathcal{L}$) of the Fiat-Shamir transform was proven by modeling the hash family as a random oracle. An exciting line of recent results have shown that for several applications [33, 50, 51, 56, 65, 67, 72, 85, 86], the Fiat-Shamir transformation can be soundly instantiated by a hash function family that is *correlation intractable*.

First, let us see why there is a need to "re-prove" the soundness in the transformed non-interactive protocol. Unlike in the interactive setting, where the prover has no control over the verifier message β, in the transformed protocol, a cheating prover could try various values of α as inputs to h until it arrives on a β it finds favorable. Specifically, for every $x \notin \mathcal{L}$, and every α, we define the set of "bad" βs,

$$\mathcal{B}_{x,\alpha} \coloneqq \left\{ \beta \;\middle|\; \exists \gamma \text{ s.t. } \mathsf{V}(x, \alpha, \beta, \gamma) = 1 \right\}.$$

Intuitively, these are the set of verifier challenges that could lead the verifier to accept, even if the statement is not in the language. We want it to be computationally intractable to find an α such that $h(x, \alpha) \in \mathcal{B}_{x,\alpha}$, i.e. hard to find α that would result in a *bad* verifier challenge. This is exactly what *correlation intractability* of a hash family captures. Specifically, we define the correlation intractability as follows. We start by describing a hash family $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$, which is defined by the two following algorithms:

- Gen: a PPT algorithm that on input the security parameter 1^λ, outputs key k.
- Hash: a *deterministic* polynomial algorithm than on input a key $k \in \mathsf{Gen}(1^\lambda)$, and an element $x \in \{0,1\}^{n(\lambda)}$ outputs an element $y \in \{0,1\}^\lambda$.

Given a hash family $\mathcal{H}$, we are now ready to define what it means for $\mathcal{H}$ to be correlation intractable.

Definition 3.1.1 (Correlation Intractability [87])**.** *A hash family* $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ *is said to be correlation intractable (*CI*) for a relation family* $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ *if the following property holds:*

For every PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $R \in \mathcal{R}_\lambda$*,*

$$\Pr_{\substack{k\leftarrow\mathcal{H}.\mathsf{Gen}(1^\lambda)\\ x\leftarrow\mathcal{A}(k)}}[(x, \mathcal{H}.\mathsf{Hash}(k, x)) \in R] \leq \mathsf{negl}(\lambda).$$

Assume for the moment that there exists at most one *bad* verifier challenge for every pair (x, α). Then one can define a function $f(\cdot) \coloneqq \mathsf{BAD}(\cdot)$ that on input (x, α) outputs the unique $\beta \in \mathcal{B}_{x,\alpha}$ (if it exists). If $\mathcal{H}$ is a CIH for f, then any cheating prover producing an accepting transcript (α, γ) for $x \notin \mathcal{L}$ must break the correlation intractability of $\mathcal{H}$, since by definition $h(x, \alpha) \in \mathcal{B}_{x,\alpha}$. For any set $\mathcal{B}_{x,\alpha}$ where $|\mathcal{B}_{x,\alpha}|$ is *polynomially bounded*, one can set $f_i(\cdot) \coloneqq \mathsf{BAD}(\cdot, i)$ to output the i-th element of $\mathcal{B}_{x,\alpha}$. By a simple application of union bound, one can observe that it remains computationally intractable for an adversary to find an α such that $h(x, \alpha)$ is the output of any f_i, and thereby remains intractable to output *any* element in $\mathcal{B}_{x,\alpha}$.

The above ideas can be extended to multi-round protocols that additionally satisfy certain properties such as *round-by-round soundness* [33].

CIH for Efficiently Verifiable Product Relations. We take the following definitions of product relations, and efficiently verifiable relations, from [68].

Definition 3.1.2 (Product Relation, Definition 3.1 [68])**.** *A relation* $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ *is a product relation, if for any* x*, the set* $R_x = \{y \mid (x, y) \in R\}$ *is the Cartesian product of several sets* $S_{1,x}, S_{2,x}, \ldots, S_{t,x}$*, i.e.*

$$R_x = S_{1,x} \times S_{2,x} \times \ldots \times S_{t,x}.$$

Definition 3.1.3 (Efficient Product Verifiability, Definition 3.3 [68])**.** *A relation R is efficiently product verifiable, if there exists a circuit C such that, for any x, the sets $S_{1,x}, S_{2,x} \dots S_{t,x}$ (in Definition 3.1.2) satisfy that, for any i, $y_i \in S_{i,x}$ if and only if $C(x, y_i, i) = 1$.*

Definition 3.1.4 (Product Sparsity, Definition 3.4 [68])**.** *A relation $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ has sparsity ρ, if for any x, the sets $S_{1,x}, S_{2,x}, \dots, S_{t,x}$ (in Definition 3.1.2) satisfies $|S_{i,x}| \leq \rho|\mathcal{Y}|$.*

[68] show that for efficient product verifiable relations, there exists a CIH assuming only the hardness of LWE.

Theorem 3.1.5 (CIH for Efficient Product Verifiable Relations, Theorem 5.5 [68])**.** *Let $R \subseteq \mathcal{X} \times \mathcal{Y}^t$ be a T-time product verifiable relation with sparsity at most $1 - \epsilon$, for $\epsilon \geq \lambda^{-O(1)}$. Then, if $t > \lambda/\epsilon$, there exists a hash family $\mathcal{H} = \{\mathcal{H}_\lambda : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda^{t_\lambda}\}_\lambda$ that is correlation intractable for R under LWE assumption. Furthermore, $\mathcal{H}$ only depends on $(\mathcal{X}_\lambda, \mathcal{Y}_\lambda, T_\lambda, t_\lambda, \epsilon)$, and can be evaluated in time $\mathsf{poly}(\log |X|, t, T)$.*

Chapter 4

SNARGs for $\mathcal{P}$ from LWE

4.1 Technical Overview

Towards our goal of achieving publicly verifiable delegation schemes for all polynomial time computations, we depart significantly from prior approaches in the designated-verifier setting. We leverage advances in the instantiation of the Fiat-Shamir transformation, recently applied in the context of publicly verifiable delegation schemes for bounded-depth computation [33, 34]. We start with an overview of the necessary background before describing the main ideas underlying our work.

4.1.1 Background

Fiat-Shamir Instantiation for Product Relations. While we described above an extension to *any* polynomially bounded $|\mathcal{B}_{x,\alpha}|$, this approach no longer works when $\mathcal{B}_{x,\alpha}$ is super-polynomial. Looking ahead, the set of bad challenges we consider in our work will *not* be of a polynomially bounded size, and therefore the ideas discussed above do not suffice. Instead, we will borrow upon the recent exciting work of [68]. Their work consider sets $\mathcal{B}_{x,\alpha}$ that for every x and α, can be represented as a Cartesian product of t sets,

$$\mathcal{B}_{x,\alpha} = \mathcal{B}^{(1)}_{x,\alpha} \times \cdots \times \mathcal{B}^{(t)}_{x,\alpha},$$

where each $\mathcal{B}^{(i)}_{x,\alpha}$ can be efficiently verified, i.e. there is a circuit C that on input (x, α), β_i and i outputs 1 if and only $\beta_i \in \mathcal{B}^{(i)}_{x,\alpha}$. For such sets, [68] show that one can construct CI hash families assuming only the hardness of LWE even if $|\mathcal{B}_{x,\alpha}|$ is not polynomially bounded.

Main Barriers. Since the Fiat-Shamir transformation preserves prover communication, it is imperative that our starting interactive protocol already has low communication. A natural candidate for such an interactive protocol is the public coin succinct argument system for NP by Kilian [69] with total communication smaller than the size of the witness. A recent work of [88], however, established non-trivial barriers to instantiating the hash function in the Fiat-Shamir transformation of Kilian's protocol.

There is in fact a broader point to consider: Kilian's protocol is an *argument*, i.e. its soundness holds only against computationally bounded cheating provers. In general, successful applications of the Fiat-Shamir paradigm when used in conjunction with CIH, have been largely limited to starting with interactive *proofs*, i.e. protocols for which even a computationally unbounded adversary cannot convince a verifier of the validity of a false statement. In fact there are examples of certain interactive arguments that are not sound on the application of the Fiat-Shamir transformation (see e.g. [89, 90]).

For this reason, the state of the art non-interactive delegation schemes that follow this approach [33, 34] rely upon known interactive delegation schemes with unconditional soundness – in particular, the scheme of [91] for bounded-depth computations. The only other known interactive delegation scheme is for bounded-space computations [92] (with verification time and communication sublinear in the number of computation steps). As such, it is unclear how to use this approach to achieve our goal of non-interactive delegation for *all* polynomial time computations.

Our Work. In light of the challenges described above, we take a different approach. We choose to view the problem of delegation of deterministic computations through

the lens of *SNARGs for batch-$\mathcal{NP}$*. Here the prover is trying to convince the verifier of the veracity of k different statements for an NP language, with communication smaller than the combined length of the witnesses for all the statements. This is an independently interesting problem, and has seen recent progress in the non-interactive setting based on standard assumptions [70].

More specifically, we reduce the task of constructing delegation schemes for polynomial-time computations to the task of constructing SNARGs for batch-$\mathcal{NP}$. We then use the Fiat-Shamir methodology to construct SNARGs for batch-$\mathcal{NP}$. As is to be expected, the same challenges as discussed earlier in the context of using the Fiat-Shamir transformation apply to the problem of constructing SNARGs for batch-$\mathcal{NP}$ as well. Indeed, presently interactive batch *proofs* are only known for UP (a subset of $\mathcal{NP}$ for which each statement has a unique witness) [92–94], and it is an open problem to construct batch proofs for $\mathcal{NP}$. Nevertheless, as we will discuss later in Section 4.1.3, we will build upon the "*dual-mode methodology*" from the recent work of [70] to circumvent these challenges and construct SNARGs for batch-$\mathcal{NP}$ (with necessary security and efficiency properties that we discuss below) based on LWE via the Fiat-Shamir methodology. For now, however, we simply assume that such SNARGs for batch-$\mathcal{NP}$ exist and proceed to describe the main ideas underlying our construction of a delegation scheme for polynomial-time computations.

We remark that some works [39, 43] have previously studied both of these problems – delegations schemes for deterministic computations and SNARGs for batch-$\mathcal{NP}$ –and used common tools and techniques to solve both the problems. We make this connection more explicit by reducing the problem of delegation of deterministic computations to SNARGs for batch-$\mathcal{NP}$. A similar approach was taken in the work of [92] who consider the problem of batch verifying *interactive proofs* in the setting of (unconditionally sound) interactive delegation for bounded space computation. At a very high level, in their work, the prover sends several intermediate steps of the

computation and then batches proofs that these intermediate steps were computed correctly. We use a similar blueprint; however, our focus is on the non-interactive setting, and all polynomial-time computations. Furthermore, we require stronger efficiency – poly-logarithmic dependence on the number of computation steps, as opposed to sublinear in [92].

4.1.2 Delegating Polynomial-Time Computations

We start our discussion with the problem of delegating the computation of a Turing machine. Here, for a Turing machine $\mathcal{M}$ and input x, the prover produces a proof Π to convince the verifier that $\mathcal{M}$ accepts x within T steps, with the requirement that both the proof size $|\Pi|$ and the verifier's running time are $\mathsf{polylog}(T)$. As stated earlier, we want to cast the problem of delegation as a problem of SNARGs for batch-NP. Intuitively, a SNARGs for batch-NP allows the prover to prove that k statements $x_1, \ldots, x_k$ *all* belong to an $\mathcal{NP}$ language $\mathcal{L}$ such that communication cost is "small" (we defer the exact communication requirements to later).

To cast the delegation problem as a SNARGs for batch-$\mathcal{NP}$, we look at the intermediate states of the Turing machine computation. Let st_i be the encoding of the state of $\mathcal{M}$ and its tapes after exactly i steps of the computation. We want to prove that for every $i \in [T-1]$, $\mathsf{st}_{i+1} = \mathsf{Step}(\mathsf{st}_i)$, where Step is the *deterministic* algorithm computing the state transition of a single step. The states st_i are thus a "witness" to the entire computation. On the surface, this already appears to be a batch problem of T instances, but an observant reader may notice that for each i, the witnesses for i and $i+1$ "overlap", specifically the overlapping state st_{i+1}. If this overlap of witnesses are not ensured, then a cheating prover could use witnesses $(\mathsf{st}_i, \mathsf{st}_{i+1})$ and $(\mathsf{st}'_{i+1}, \mathsf{st}'_{i+2})$ such that $\mathsf{st}_{i+1} = \mathsf{Step}(\mathsf{st}_i) \wedge \mathsf{st}'_{i+2} = \mathsf{Step}(\mathsf{st}'_{i+1})$ but $\mathsf{st}_{i+1} \neq \mathsf{st}'_{i+1}$. This is clearly undesirable since the overlap of witnesses is necessary to establish *continuity* in the computation - otherwise a cheating prover is proving T independent

statements, unhelpful to establish correctness of computation. Unfortunately, the notion of SNARGs for batch-$\mathcal{NP}$ we have described does not enforce any constraints across statements.

We overcome this problem as follows:

- *Step 1*: First, the prover commits to *all* the internal states $\mathsf{st}_1||\cdot||\mathsf{st}_T$. Let this committed value be c.
- *Step 2*: The prover now proves that for every $i \in [T]$, $(x, c, i) \in \mathcal{L}$, where $\mathcal{L}$ is defined by the relation circuit C below.

C

Statement: x, c, i
Witness: $\mathsf{st}_i, \mathsf{st}_{i+1}, \mathsf{open}_{\mathsf{st}_i}, \mathsf{open}_{\mathsf{st}_{i+1}}$
Output: Output 1 if and only if the following verify

1. check if $\mathsf{Com.Verify}(c, \mathsf{st}_i, \mathsf{open}_{\mathsf{st}_i}) \stackrel{?}{=} 1$.
2. check if $\mathsf{Com.Verify}(c, \mathsf{st}_{i+1}, \mathsf{open}_{\mathsf{st}_{i+1}}) \stackrel{?}{=} 1$.
3. if $i = 1$, check if st_1 encodes input x.
4. if $i = T - 1$, check if st_T is the *accept* state.
5. Check if $\mathsf{Step}(\mathsf{st}_i) = \mathsf{st}_{i+1}$.

Here, $\mathsf{open}_{\mathsf{st}_i}$ corresponds to a proof of opening that st_i was indeed the i-th vector that was committed to in c.

Weaker Goal: Bounded-Space Computation. For now, we consider the weaker goal of bounded-space computation since it already highlights the main challenges. Specifically, we allow the total communication and the verification time to be $\mathsf{poly}(\log T, |\mathsf{st}|)$, i.e., grow with the size of the internal state. We shall later see how to go beyond the space constraints.

Towards achieving this weaker goal, we establish some efficiency properties of the commitment scheme used in Step 1:

- *Size of the commitment:* Enforcing the same communication constraints as above on the size of the commitment - we have that the commitment to a vector of size $T \cdot |\mathsf{st}|$ is at most $\mathsf{poly}(\log T, |\mathsf{st}|)$, i.e. the commitment is *succinct*.

- *Size of the opening:* Looking ahead, we will require that the size of C, the relation circuit for $\mathcal{L}$, to also be at most $\mathsf{poly}(\log T, |\mathsf{st}|)$. This means that the commitment must have a *succinct local opening* opening $\mathsf{open}_{\mathsf{st}_i}$ to st_i, since the opening is a part of the witness (and thus contributes to $|C|$). Succinctness here means that the size of the opening is $\mathsf{poly}(\log T, |\mathsf{st}|)$.

Let us now shift our focus to Step 2. Our initial idea is to use a SNARG for batch-$\mathcal{NP}$ to prove all the instances (x, c, i) for the relation circuit C. Note, however, that the verifier of a SNARG for batch-$\mathcal{NP}$ inevitably runs in time $\Omega(T)$ for T instances as it needs to at least read the T instances. This is prohibitive for us since we require the verifier of our delegation scheme to run in time polylogarithmic in T.

Using SNARGs for Batch-Index. To overcome this issue, our key observation is that the statements above are *identical* except for the index i. Hence, the verifier in our case need not suffer from the $\Omega(T)$ running time barrier, since the number of instances T provides all the necessary information about the instances. This motivates us to adopt the following useful abstraction we call *SNARGs for batch-index*. Formally, the index language is defined as follows,

$$\mathcal{L}^{\mathsf{idx}} = \{(C, i) \mid \exists w \text{ s.t. } C(i, w) = 1\}$$

where C represents a circuit, and i an index. In a SNARG for batch-index, the prover tries to convince the verifier that $(C, 1), \cdots, (C, T) \in \mathcal{L}^{\mathsf{idx}}$.

To implement Step 2, we can set the index language such that C contains hard-coded the *common inputs* across all T instances, namely, the commitment c and input x, and only takes in as input an index i. The witness to C remains unchanged.

Towards achieving the relaxed goal of delegating bounded-space computation, we allow the proof size and the verification time of SNARGs for T statements to be $\mathsf{poly}(\log T, |C|)$, as long as $|C| = \mathsf{poly}(|\mathsf{st}|)$. We note that this already rules out using existing SNARGs for batch-$\mathcal{NP}$ based on standard assumptions [70] since the proof size in their scheme depends on $\sqrt{T}$).

Security. Let us now turn our attention to the security of our approach. Along the way, we will establish the required security properties from the commitment scheme and SNARGs for batch-index.

Since the commitment used in Step 1 is succinct, for every i there could always *exist* states st'_i and st'_{i+1} with corresponding local commitment openings to c such that $\mathsf{Step}(\mathsf{st}'_i) = \mathsf{st}'_{i+1}$ even if it is computationally hard to find them. Thus for all i it may always be the case that $(C, i) \in \mathcal{L}^{\mathsf{idx}}$, making soundness of the SNARGs for batch-index a vacuous notion. The fix is to use *somewhere statistical binding commitments* [83] such that for a commitment key generated on input i^* there is a *unique* (except with negligible probability) local opening to $\mathsf{st}'_{i^*}, \mathsf{st}'_{i^*+1}$. Thus, if $\mathsf{Step}(\mathsf{st}'_{i^*}) \neq \mathsf{st}'_{i^*+1}$, then $(C, i^*) \notin \mathcal{L}^{\mathsf{idx}}$.

In more detail, let S_i be the set of indices corresponding to st_i in the vector $\mathsf{st}_1||\cdots||\mathsf{st}_T$. We will require the somewhere statistical binding property to be at indices $S_i \cup S_{i+1}$ when the commitment key is generated in the *trapdoor mode*[1] on input $S_i \cup S_{i+1}$. We shall shortly see why this is the case. In fact, looking forward, we will actually require something stronger. Namely, generating a key in trapdoor mode on input $S_i \cup S_{i+1}$ produces a trapdoor that allows for *unique extraction* at positions $S_i \cup S_{i+1}$ even for a commitment produced by an *unbounded cheating prover*. We refer to this as the *somewhere extractable* property, and use the shorthand SE to refer to it in the sequel.

[1]Keys generated in this mode (only in the security proof) are computationally indistinguishable from keys in the *normal mode*.

From the discussion above, by setting the SE commitment to be extractable for $\mathsf{st}_i, \mathsf{st}_{i+1}$, the best that one can hope for in terms of SNARGs for batch-index soundness is that a cheating prover is not able to produce an accepting proof when the i-th statement is false, i.e. $\mathsf{st}_{i+1} \neq \mathsf{Step}(\mathsf{st}_i)$. This motivates a notion of *somewhere soundness* where the CRS for the SNARGs for batch-index is generated on an index i such that it is hard for a cheating prover to produce an accepting proof when $(C, i) \notin \mathcal{L}^{\mathsf{idx}}$.

The work of [70] considered *non-adaptive* security for non-interactive batch arguments, where the statements are fixed *before* the CRS is generated. In our approach, however, a cheating prover gets to choose the commitment c, which is hardcoded into the circuit C, effectively allowing it to adaptively pick the statements *after* the CRS is generated. Unfortunately, as observed in [39], there are significant barriers to achieving *full adaptivity*, where the cheating prover can choose the statements *after* the CRS is generated.

We overcome this seeming conundrum by considering an intermediate notion of security that we call *semi-adaptive somewhere soundness*. We explain it here for the case of index language. Intuitively, the cheating prover must declare an index i^* of its choice before the CRS is generated. However, it can choose the circuit C after viewing the CRS. The soundness guarantee states that it will not be able to produce an accepting batch proof if $(C, i^*) \notin \mathcal{L}$. More specifically, for any computationally bounded cheating prover P^*,

$$\Pr\left[\begin{array}{l|l} \Pi \text{ accepting} \\ (C, i^*) \notin \mathcal{L} & \begin{array}{l} i^* \leftarrow \mathsf{P}^* \\ \mathsf{crs}^* \leftarrow \mathsf{TrapdoorMode}(i^*) \\ (\Pi, C) \leftarrow \mathsf{P}^*(\mathsf{crs}^*) \end{array} \end{array}\right] < \mathsf{negl}(\lambda)$$

Now that we have seemingly fixed the issues raised above, how do we prove that the above scheme is secure? A natural proof strategy is the following: (1) set the index i for the trapdoor generation of the CRS for SNARGs, and index $S_i \cup S_{i+1}$ for the commitment key for the SE commitment; (2) extract $\widetilde{\mathsf{st}}_i$ and $\widetilde{\mathsf{st}}_{i+1}$ from the SE

commitment using the trapdoor; (3) if $\mathsf{Step}(\widetilde{\mathsf{st}}_i) \neq \widetilde{\mathsf{st}}_{i+1}$, but the proof Π is accepting, output (Π, C) as the cheating proof of the SNARGs.

Let us see why this is the case. From the somewhere extractability property of the SE commitment, we know that other than with negligible probability, the extracted value is the *only* valid opening. So, if $\mathsf{Step}(\widetilde{\mathsf{st}}_i) \neq \widetilde{\mathsf{st}}_{i+1}$, then $(C', i) \notin \mathcal{L}$, and therefore we can break the soundness of the SNARGs for batch-index.

Local vs global soundness. By the above, we are guaranteed that if the proof is accepting, then the i-th instance must be true: $(C, i) \in \mathcal{L}$, i.e. it must be the case that the extracted values do indeed satisfy C. This gives us a *local soundness* guarantee (i-th statement is true), but for the entire computation to be true, we want local soundness to hold simultaneously for all $i \in [T]$, i.e. we want *global soundness*. If one stops to think about this, our argument above for the soundness of the i-th instance crucially relied on extractability at position i and $i + 1$. For simultaneous local soundness to hold, we would require extractability at *all* positions, which is not achievable in a succinct manner (the commitment size would grow with T instead of $\mathsf{poly}(\log T)$ as desired). One might propose an alternate hybrid strategy where one starts by proving the first instance is locally sound, then switch to proving the same for the second instance and so on. But a *local witness*, i.e. the extracted value in each case, could satisfy C even though there exists no *global witness*, i.e. $\mathcal{M}$ does not accept x in T steps.

This problem is not new to our setting, and is in fact well documented in delegation literature starting with [37] - either in the construction of no-signaling PCPs [36–40, 42], or more recently in the use of so-called quasi-arguments [43] to construct delegation schemes.

No-signaling commitments. We take a slightly different approach and describe the notion of no-signaling with respect to SE commitments as done very recently in [84]. Specifically, an extractor for an SE scheme is said to be *computationally*

no-signaling if for any sets S and S', both of size at most L, the extracted values in the intersection $S \cap S'$ have computationally indistinguishable marginal distributions whether extracted on set S or S'. Specifically, for any computationally bounded adversary $\mathcal{A}$, the following distributions are computationally indistinguishable:

$$\left\{ (c, y_{S \cap S'}) \;\middle|\; \begin{array}{l} (K, \mathsf{td}) \leftarrow \mathsf{TrapdoorMode}(L, S) \\ c \leftarrow \mathcal{A}(K) \\ y = \mathsf{Ext}(\mathsf{td}, c) \end{array} \right\}$$

and

$$\left\{ (c, y_{S \cap S'}) \;\middle|\; \begin{array}{l} (K, \mathsf{td}) \leftarrow \mathsf{TrapdoorMode}(L, S') \\ c \leftarrow \mathcal{A}(K) \\ y = \mathsf{Ext}(\mathsf{td}, c) \end{array} \right\}$$

[84] also describe a generic compiler that transforms *any* SE scheme to a no-signaling one (NS-SE) without additional assumptions, thereby preserving the assumptions from the underlying SE commitment. We observe that the transformation also preserves the desired efficiency requirements. For completeness, we discuss the transformation in the technical sections of our paper.

It is the no signaling property of the SE, in conjunction with the SNARGs for batch-index, that finally gives us a delegation scheme. Consider two experiments:

$\underline{\mathsf{EXP}_1}$**:** (a) the BARG CRS is generated on input 1, the NS-SE key generated on $S_1 \cup S_2$; (b) extract st_1 and st_2 from the NS-SE and output it along with proof Π if Π is accepting.

$\underline{\mathsf{EXP}_2}$**:** (a) the BARG CRS is generated on input 2, the NS-SE key generated on $S_2 \cup S_3$; (b) extract st'_2 and st'_3 from the NS-SE and output it along with proof Π if Π is accepting.

By our earlier argument, due of the (local) soundness of BARG, we have already established that st_2 is consistent with st_1 in EXP_1, and st'_2 with st'_3 in EXP_2. By the description of C, we additionally know that the start state st_1 is consistent with the input x, where by consistent we mean that it is the unique correct state at step 1 with

respect to x. Now, the no-signaling property of the SE commitment scheme ensures that st_2 and st'_2 have computationally indistinguishable distributions. This suffices to ensure that st_2 and st'_2 must both be consistent with x, since otherwise there is an efficient distinguisher - compute $\widetilde{\mathsf{st}}_2$ from x and see which of the two it matches. Therefore, by the fact that st'_2 is consistent with st'_3, st'_3 is consistent with x. By a hybrid argument we can extend this approach all the way to st_T establishing that st_T is indeed consistent with x.

Remark 4.1.1. *Note that in each experiment the adversary could choose to output a different x, but the above distinguishing check is done with respect to the x output by the adversary, guaranteeing that the extracted st_i is consistent with the x that the adversary output.*

From the proof size of the underlying BARG scheme, the total proof size for the delegation scheme is $\mathsf{poly}(|\mathsf{st}|, \log T)$. The same is true of the size of the CRS, which depends on $|C|$, and thus only on st in our setting. This ensures that the above scheme is a *delegation scheme for space bounded computation* with a short CRS. Unlike prior work [43], our CRS is already "small", and therefore we do not need an additional bootstrapping step to reduce the CRS size.

Beyond bounded space computation. To go beyond delegation for bounded space computation, we use ideas from prior works [38, 39, 43]. The main insight is to simulate a Turing machine $\mathcal{M}$ with large space via a RAM machine $\mathcal{R}$, where the RAM machine has access to a large untrusted external memory but a small internal memory. A digest of the external memory, in the form of the root of the hash tree, is stored in the internal memory. This has two benefits: (i) the root of the hash tree is small ($\mathsf{poly}(\lambda)$), and thus can be stored in the internal memory; and (ii) the hash tree allows for authenticated access, both read and write, to the external memory where the proof size logarithmic in the size of the external memory. Applying these ideas to our bounded space computation, we achieve a RAM delegation protocol. Since the

size of the CRS is small in the bounded space computation, it continues to be so in the RAM delegation setting.

The notion of RAM delegation we achieve is similar to that considered in [43]. Here a prover is convincing the verifier that a RAM machine $\mathcal{R}$ starting at configuration x (including the large external memory) transitions to configuration y in T steps where the verifier is only given digests h_x and h_y of the two configurations. The notion of security is that a computationally bounded cheating prover, other than with negligible probability, should not be able to produce a configuration x, digest h and proof Π such that: (i) Π is accepting for the digests $(\mathsf{h}_x, \mathsf{h})$ where h_x is digest for configuration x; and (ii) h is *not* the digest of the (unique) configuration of $\mathcal{R}$, T steps after x. We refer the reader to [43] for a detailed discussion of the various notions of RAM delegation considered in prior works.

4.1.3 SNARGs for Batch-$\mathcal{NP}$

Now that we have constructed a delegation scheme for polynomial-time computations assuming the existence of SNARGs for batch-index, we revisit the problem of constructing such a primitive. In fact, we will consider the more general case of constructing SNARGs for batch-$\mathcal{NP}$.

Recall that in a SNARGs for batch-$\mathcal{NP}$, a prover wants to convince a verifier of the veracity of k statements $(x_1, \cdots, x_k)$ in $\mathcal{L}$ by producing a non-interactive batch proof that is publicly verifiable, such that if *any* of the k instances are false (i.e. $\exists i$ s.t. $x_i \notin \mathcal{L}$), then a computationally bounded cheating prover should not be able to generate an accepting proof. If the witness length is $m = m(|x|)$, we require the communication to be *smaller* than $k \cdot m$.

Prior Work. We know of only two solutions to this problem based on falsifiable assumptions: (i) [43] construct such SNARGs for batch-$\mathcal{NP}$ relying on a new non-standard hardness assumption on groups with bilinear maps; and (ii) more recently,

[70] construct the same by assuming the hardness of the quadratic residuosity (QR) assumption in addition to either the hardness of Learning with Errors (LWE) problem, or sub-exponential hardness of the decisional Diffie-Hellman (DDH) problem. In the context of this paper, of particular interest to us is the work of [70] since they follow the Fiat-Shamir instantiation approach.

As discussed earlier in the context of non-interactive delegation schemes for polynomial-time computation, there are challenges to starting with an interacting *argument* if we want to go the "Fiat-Shamir instantiation" approach. Instead of tackling the (seemingly harder) problem of constructing interactive batch proofs for $\mathcal{NP}$, [70] choose an alternate starting point to apply the Fiat-Shamir transform. They introduce the notion of a *dual-mode SNARGs for batch-$\mathcal{NP}$* in the common reference string (CRS) model. The CRS in such protocols can be generated in two *computationally indistinguishable* modes - *normal mode* and *trapdoor mode*. (We have already seen a flavor of this notion when describing the delegation scheme.) For an honest protocol execution, the CRS is generated in the normal mode, while trapdoor mode is used in the proof of soundness. Specifically, in the trapdoor mode, an index i is specified during CRS generation such that if $x_i \notin \mathcal{L}$, then even a computationally unbounded cheating prover cannot provide an accepting proof.

Such a protocol provides two complementary benefits: (i) building a SNARG for batch-$\mathcal{NP}$ is easier than building a batch proof; and (ii) it allows for the possibility of instantiating the Fiat-Shamir transform in the trapdoor mode. [70], building on the Spartan protocol [95], construct such a dual-mode SNARGs for batch-$\mathcal{NP}$ with non-adaptive soundness. They then show that the specific dual-mode SNARGs for batch-$\mathcal{NP}$ constructed is Fiat-Shamir *compatible*, i.e. there exists a hash function family such that the Fiat-Shamir transformation when instantiated with this family is a sound SNARGs for batch-$\mathcal{NP}$. The size of the batch proof in their protocol is $\tilde{O}((|C| + \sqrt{k|C|}) \cdot \lambda)$ where $|C|$ is the size of the relation circuit for $\mathcal{L}$, λ is the security

parameter and $\widetilde{O}$ hides factors that are poly-logarithmic in $|C|$ and k.

Our Work. In our overview of the delegation scheme for polynomial-time computations, we identified both security and efficiency properties for SNARGs for batch-$\mathcal{NP}$ we deemed essential for the construction of said delegation scheme. The properties achieved by the non-interactive batch arguments scheme of [70], however, do not meet these requirements.

To construct SNARGs for batch-$\mathcal{NP}$ with the desired properties, we adopt the same "dual mode methodology" introduced in [70], but deviate from their approach in a couple of crucial aspects. First, instead of building upon the Spartan protocol, we work directly with *probabilistic checkable proofs* (PCPs), in a manner conceptually similar to Kilian's protocol. Next, we leverage this change of approach to recurse over the number of statements (akin to [92]), allowing us to depend only poly-logarithmically on the number of instances. We will elaborate on these points below.

To summarize our improvements over the BARG in [70]: (i) we achieve the stronger security notion of *semi-adaptive somewhere soundness*, (ii) we improve upon the size of the batch proofs to incur only poly-logarithmic dependence on k, (iii) we simplify the underlying assumptions - we no longer additionally require the quadratic residuosity (QR) assumption.[2] Our protocol is also conceptually simpler.

The rest of this section is organized as follows: first, we describe our construction of SNARGs for batch-index. Next, we discuss how to extend our result to obtain SNARGs for batch-$\mathcal{NP}$.

4.1.3.1 SNARGs for Batch-Index

Recall that in a SNARG for batch-index, the prover is trying to prove that $(C, 1), \ldots, (C, k) \in \mathcal{L}^{\mathsf{idx}}$. Our goal is to construct such SNARGs for batch-index with proof size

[2]The reliance on QR assumption in [70] stems from their use of SE commitments that are required to additionally satisfy some homomorphism properties. Our approach does not require such properties; in particular, we can use known constructions of SE commitments from LWE.

and verification time $\mathsf{poly}(\log k, |C|)$.

Our construction involves the use of PCPs, namely, proofs where the verification procedure only needs to query a few locations of the PCP to be reasonably convinced of the validity of the statement. For now, consider a PCP where the length of the PCP is $m = \mathsf{poly}(|C|)$ and the query Q is of size $\mathsf{polylog}(m)$, where C is the relation circuit for $\mathcal{L}$. The verification procedure only takes in $\mathsf{PCP}|_Q$, the values of the PCP at the locations specified by Q.

The prover generates k PCPs $\mathsf{PCP}_1, \cdots, \mathsf{PCP}_k$ using the corresponding witnesses $w_1, \cdots, w_k$. It then arranges the PCPs in rows, and commits to them in a column-wise fashion. On receiving the commitment, the verifier sends the PCP query Q to the prover, who then opens the commitments of the corresponding columns. To be convinced of the proof, the verifier checks if (i) the commitment openings are valid; and (ii) all k PCP proofs verify. Note that the same Q is used for *all* PCPs. As in the case of our delegation scheme, we use a *succinct* SE *commitment with local opening* to commit to each column.

This high level overview is represented in Figure 4-1.

Delegating the Verification, Recursively. The main efficiency bottleneck in the above approach is the third round message consisting of commitment openings that require at least k bits of communication (length of the message committed). In order to achieve only poly-logarithmic dependence on k, we delegate the verification process to the prover, building on ideas from prior works [92].

Specifically, we observe that the verification of the commitment openings and the PCP responses constitutes a new index language defined w.r.t. the following relation circuit VerifyC: it takes as input an index i, and verifies the commitment openings and the PCP responses for the i-th instance. The PCP query Q and the corresponding commitments $c|_Q$ are hardwired in VerifyC.

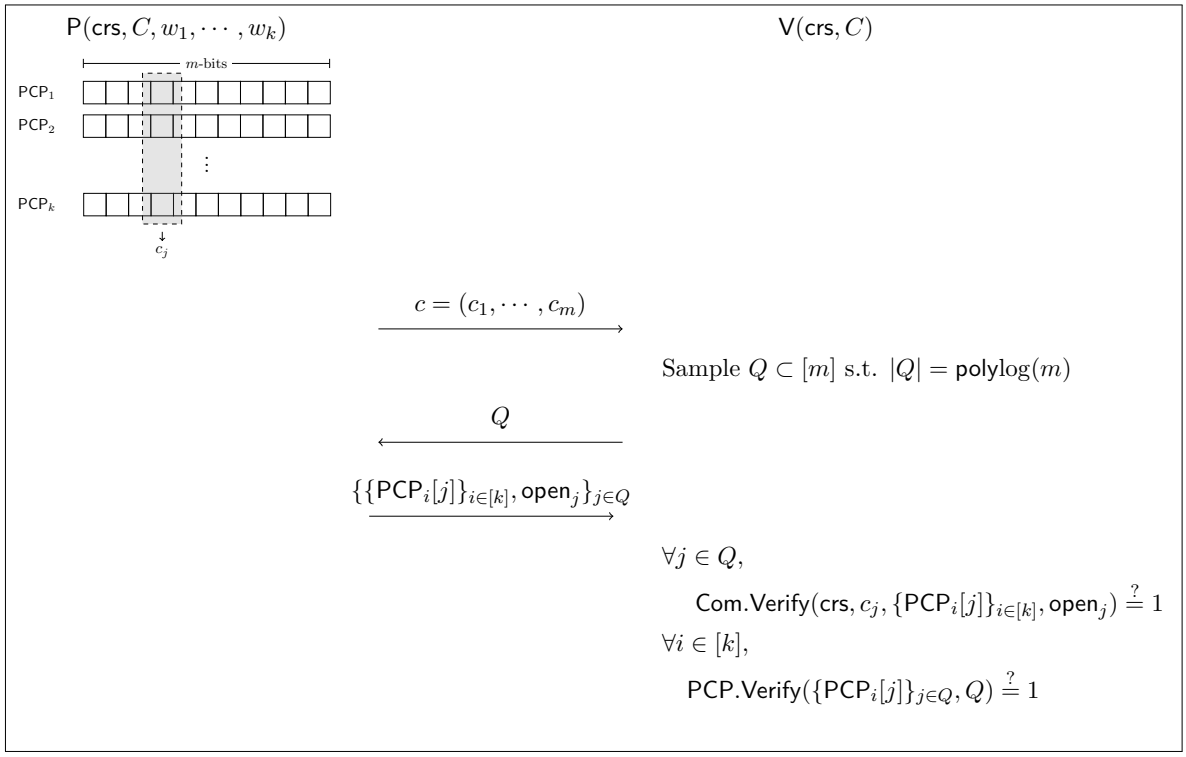


Figure 4-1. High level overview of initial approach

VerifyC

Hardcoded: The commitments on the coordinates specified by Q: $c|_Q, Q$.
Input Instance: An index i
Witness: The PCP responses for the i-th instance $\{\mathsf{PCP}[j]\}_{j\in Q}$, and commitment openings $\{\mathsf{open}_j\}_{j\in Q}$.
Output: Output 1 if and only if the following verify

1. Verify commitment openings: $\forall j \in Q$,
$$\mathsf{Com.Verify}(c_j, i, \mathsf{PCP}[j], \mathsf{open}_j) \stackrel{?}{=} 1.$$
2. Verify PCP proofs,
$$\mathsf{PCP.Verify}(i, \{\mathsf{PCP}[j]\}_{j\in Q}, Q) \stackrel{?}{=} 1.$$

To delegate the verification work to the prover, we no longer require the prover to explicitly send the openings. Instead, the prover provides another BARG that convinces the verifier that $\mathsf{VerifyC}(i, \cdot)$ is satisfiable for all i. If we can ensure that the verification time of the new SNARGs for batch-index is smaller, then we can apply

this idea *recursively* until the commitment openings are small enough to send directly.

A naive implementation of this strategy, however, does not provide any benefit since at every recursion level, the new BARG still has k instances, and thus the verifier still needs at least $\Omega(k)$ time to verify.

Grouping the Instances. To save on verification time, our first idea is to reduce the number of instances by "grouping" two instances together. More specifically, we group the indices $1, 2, \ldots, k$ into $k/2$ pairs $(1, 2), (3, 4), \ldots, (k-1, k)$, and use the following "grouped" new circuit NewRel as the relation circuit for the new index language. The circuit NewRel takes as input the new "grouped" instance $(2i-1, 2i)$ and the new witness (ω, ω'), and checks whether $\mathsf{VerifyC}(2i-1, \omega)$ and $\mathsf{VerifyC}(2i, \omega')$ both output 1.

$$\mathsf{NewRel}((2i-1, 2i), (\omega, \omega')) = \mathsf{VerifyC}(2i-1, \omega) \wedge \mathsf{VerifyC}(2i, \omega').$$

In this manner, we halve the number of instances at each recursion level and thus the recursion ends in $\log k$ levels.

Unfortunately, the above idea also does not save on verification time. The problem is that the relation circuit size grows exponentially with the number of levels. To see this, let us denote the relation circuit at the L-th recursion level as NewRel_L and the verification circuit at L-th level as $\mathsf{VerifyC}_L$. Then since NewRel_L contains two copies of $\mathsf{VerifyC}_L$, we have $|\mathsf{NewRel}_L| \geq 2|\mathsf{VerifyC}_L|$. Furthermore, since $\mathsf{VerifyC}_L$ contains the PCP verification circuit $\mathsf{PCP.Verify}$ for the relation NewRel_{L-1} at the previous level, we have that $|\mathsf{VerifyC}_L| \geq |\mathsf{NewRel}_{L-1}|$. Combining them, we obtain $|\mathsf{NewRel}_L| \geq 2|\mathsf{NewRel}_{L-1}|$. Since we have $\log k$ levels in total, in the last level $L = \log k$, the new relation circuit size becomes at least $\Omega(2^{\log k}|C|) = \Omega(k|C|)$. Thus, the verifier would still run in time $\Omega(k)$.

PCPs with Fast Online Verification. To resolve the above problem, we take a closer look at the PCP verification algorithm. The work of [20] considers *Linear*

PCPs where the verification algorithm is split into two parts: an input-oblivious query phase, and a very fast online verification phase. The query phase only takes the relation circuit C as input, and outputs some queries Q and a "short" state st. The online-verification phase takes as input an instance x and the state st, and runs in $\widetilde{O}(|x|)$ time to decide to accept or reject. We observe that the same property also holds for many existing *standard* PCPs. As an example, in this work, we show that the PCP in [92] can be slightly modified to satisfy almost the same property, except that we allow the online-verification time to be $\mathsf{poly}(|x|, \log|C|)$ in order to be general enough.

We now use the above property of the PCP verification to remove the dependence on $|\mathsf{NewRel}_{L-1}|$ in the size of the new relation circuit NewRel_L at level L. Since the query phase is oblivious to the instance x – which in our case corresponds to the index i or $(2i-1, 2i)$ etc. – this part can be *shared* across all the instances at a recursion level. Namely, we have the verifier execute the input-oblivious query phase only *once* for *all* the instances, and then use the same state st and the query Q for the online-verification phase of all instances. Then we replace $\mathsf{PCP.Verify}$ in our verification circuit $\mathsf{VerifyC}$ with the PCP online verification phase that contains the state st hardwired. Since the online verification phase of PCP runs in time $\mathsf{poly}(|x_L|, \log|\mathsf{NewRel}_{L-1}|)$ where x_L is the length of the instance at the L-th recursion level, we have now improved the size of the new relation circuit $|\mathsf{NewRel}_L|$ to $\mathsf{poly}(|x_L|, \log|\mathsf{NewRel}_{L-1}|)$.

Avoiding Instance Length Growth. An observant reader may notice that even the above improvement does not fully solve our problem. This is because the instance length grows at every recursion level. At the top level, the instances are $1, 2, \ldots, k$, each of which has bit-length $\log k$. Then in the first recursion level, the instances become $(1, 2), (3, 4), \ldots, (k-1, k)$, which have length $2\log k$. At the bottom level $L = \log k$, the instance length becomes $\Omega(2^{\log k}) = \Omega(k)$. Therefore, the size of the new relation circuit $|\mathsf{NewRel}|$ and the verification time at the bottom level is still $\Omega(k)$.

Thus, it would seem that we have made no progress.

In order to prevent the growth of the instance length during recursion, we crucially exploit the nature of the index language. Specifically, instead of providing $(2i-1, 2i)$ as input to the circuit NewRel, we simply provide i as the input instance and then require NewRel to generate $(2i-1, 2i)$ *on its own*. Then, it feeds $2i-1$ and $2i$ to two copies of the PCP online verification circuit $\mathsf{VerifyC}$.

In this way, the instance length at each recursion level stays $\log k$, while the size of the NewRel only increases by $\mathsf{polylog}\ k$ for the computation of $(2i-1, i)$. Hence, the relation circuit size $|\mathsf{NewRel}_L|$ at recursion level L is only $\mathsf{poly}(\log k, \log |\mathsf{NewRel}_{L-1}|)$. This allows us to bound $|\mathsf{NewRel}_L| = \mathsf{poly}(\log k, |C|)$ for all levels.

Summary of Our Construction. In summary, our construction of SNARGs for batch-index uses $\log k$ levels of recursion. In each level, the number of instances is half of the previous level. At each level, the prover commits to the PCP proofs for all instances in a "column-wise" fashion, and sends them to the verifier. Then the verifier uses the input-oblivious PCP query generation algorithm to compute query Q and a short state st once for all the instances. Next, both the parties recursively execute a new SNARGs for the "grouped" relation circuit NewRel with st hardwired. Finally, when they reach the the last level of recursion, the prover sends the witness directly to the verifier.

We now briefly analyze the efficiency of the construction. The verification process consists of two parts:

- PCP query generation algorithm at each level, which runs in time $\mathsf{poly}(\lambda, \log k, |\mathsf{NewRel}_L|)$.
- At the final level, directly verify the relation circuit NewRel_L, where $L = \log k$. This step takes time $O(|\mathsf{NewRel}_L|)$.

Recall that $|\mathsf{NewRel}_L| = \mathsf{poly}(\lambda, \log k, \log |\mathsf{NewRel}_{L-1}|)$, where at the top level, NewRel_0

$= C$. Hence, we can bound the circuit size $|\mathsf{NewRel}_L|$ at each level by $\mathsf{poly}(\lambda, \log k, |C|)$, and hence the total verification time is bounded by $\mathsf{poly}(\lambda, \log k, |C|)$.

The prover, at recursion level L, computes the witness and instance for the next recursion level in time $\mathsf{poly}(\lambda, k, |\mathsf{NewRel}_L|)$. Hence, the prover runs in time $\mathsf{poly}(\lambda, k, |C|)$ in total.

The CRS consists of $\log k$ SE commitment keys for each level of the recursion. By the specific instantiations of the SE commitment scheme[83] (and the correlation intractable hash family [68]), we have that for each level of the recursion, the CRS is of size $\mathsf{poly}(\log k, |C|)$, for a total of $\mathsf{poly}(\log k, |C|)$.

Security. Recall that our goal is to achieve *semi-adaptive somewhere soundness*, which requires that a cheating prover after specifying an index i should not be able to produce an accepting proof when $(C, i) \notin \mathcal{L}$, where the CRS is generated in the trapdoor mode on index i. Taking the approach in [70] of *dual mode* proofs, we extend the same definition to the interactive setting, but here we allow the adversary to be unbounded once the index to the trapdoor is fixed. Specifically, for any (potentially) cheating prover P^*,

$$\Pr\left[\begin{array}{l|l} \Pi \text{ accepting} \\ (C, i^*) \notin \mathcal{L} & \begin{array}{l} i^* \leftarrow \mathsf{P}^* \\ \mathsf{crs}^* \leftarrow \mathsf{TrapdoorMode}(i^*) \\ (\Pi, C) \leftarrow \langle \mathsf{P}^*(\mathsf{crs}^*), \mathsf{V} \rangle \end{array} \end{array}\right] < \mathsf{negl}(\lambda)$$

where $\langle \mathsf{P}^*(\mathsf{crs}^*), \mathsf{V} \rangle$ indicates the interaction between the cheating prover P^*, and verifier V with output the proof Π and the circuit the prover chooses C. Note that the *mode indistinguishability*, i.e. ability to distinguish between the CRS generated for two different indices i and j is still *computational*.

Thus for security, in the trapdoor mode, the SE key for each column is generated on a set $\{i\}$, ensuring that the prover is uniquely bound to PCP_i *before* it sees the queries Q. This then allows us to rely on the (statistical) soundness of the PCP at index i.

Applying the Fiat-Shamir Transform. Given our interactive protocol, we want to

compress it to a non-interactive protocol via the Fiat-Shamir transform. As discussed earlier, crucial to this transformation is defining the set of bad verifier challenges.

$$\mathcal{B}_{C,c} = \{Q \ \mid \ \mathsf{PCP.Verify}(C, \mathsf{PCP}_i|_Q) = 1 \ \wedge \ (C,i) \notin \mathcal{L}\}$$

where PCP_i is extracted using the trapdoor for the SE commitment with the commitment key in the trapdoor mode generated on index i.

If we are able to demonstrate that the above set is a Cartesian product of efficiently verifiable sets, then we can apply the [68] result directly, achieving a result based on LWE. At a very high level, this simply follows from the soundness amplification by parallel repetition in the PCP. Specifically, for the desired parameters, the PCPs we consider have soundness $(1-\varepsilon)$ for $\varepsilon = 1/\mathsf{poly}(\log|C|)$. Since we want the soundness to be negligible, we amplify soundness by parallel repetition, generating $t = \lambda/\varepsilon$ sets of queries $Q_1, \cdots, Q_t$ for the same PCP. This gives the desired negligible soundness as $(1-\varepsilon)^t = 2^{-\Omega(\lambda)}$. Thus we have the following set of bad challenges for negligible soundness,

$$\mathcal{B}_{C,c} = \mathcal{B}^{(1)}_{C,c} \times \cdots \times \mathcal{B}^{(t)}_{C,c}$$

where for each i, $\mathcal{B}^{(i)}_{C,c} = \{Q_i \mid \mathsf{PCP.Verify}(C, \mathsf{PCP}_i|_{Q_i}) = 1 \ \wedge \ (C,i) \notin \mathcal{L}\}$. Each $\mathcal{B}^{(i)}_{C,c}$ is also clearly efficiently verifiable since the $\mathsf{PCP.Verify}$ can be used to verify if queries $Q_i \in \mathcal{B}^{(i)}_{C,c}$. One also needs to verify that $(C,i) \notin \mathcal{L}$, which can be done after extracting PCP_i if we require further properties from the underlying PCP. We note that while the above description is not fully technically precise, it is helpful in providing the main ideas, and we refer the reader to the technical section for the details.

It should be noted that the unrolled recursion is a multi-round protocol, while the above argument considers the set of bad challenges for a single level of recursion. [33] showed that this approach suffices if the protocol is *round-by-round sound* which in this case intuitively means that if the (batch) claim at one level of the recursion is false, then other than with negligible probability (over the verifier's random coins), the

claim remains false in the next level of recursion. Here, when we set the commitment key to be generated in the trapdoor mode for index i, if at the j-th level of recursion $(C^{(j)}, i)$ is false, then other than with negligible probability over the choice of PCP queries Q, $(C^{(j+1)}, \lceil i/2 \rceil)$ is also false.

4.1.3.2 SNARGs for Batch-$\mathcal{NP}$

SNARGs for Batch-$\mathcal{NP}$ from SNARGs for Batch-Index. We now describe how to transform any SNARGs for batch-index into a SNARGs for batch-$\mathcal{NP}$. Recall that in a SNARGs for batch-$\mathcal{NP}$ with the language $\mathcal{L}$, the prover is trying to convince the verifier of the validity of k statements $x_1, \cdots, x_k$, i.e. $\mathcal{L}$ has a relation circuit $\mathcal{R}$ such that if $x_i \in \mathcal{L}$, there exists a witness w_i such that $\mathcal{R}(x_i, w_i) = 1$. Contrast this with our discussed notion of SNARGs for batch-index, where there is a *single* circuit C that takes in inputs i and witness w_i, and outputs 1 if $C(i, w_i) = 1$.

An immediate idea is to set $w_i' = (x_i, w_i)$ such that C implements the relation circuit $\mathcal{R}$. Since our SNARGs for batch-index allows the prover to choose any witness, the above idea allows a cheating prover to choose new statements different from $x_1, \cdots, x_k$, thus the soundness does not translate. The next natural idea is to hardcode the statements $x_1, \cdots, x_k$ into the circuit C, which now only takes in input (i, w_i), but still implements $\mathcal{R}$. While we have solved our earlier issue, we have introduced a new one since C now grows linearly in k, and from the efficiency of the SNARGs, so does the size of the proof.

We solve the communication issue as before, by having the prover arrange the *statements* in rows, and commit to them column-wise using an SE commitment scheme. C now hardcodes the commitment c instead, where the witness additionally consists of x_i along with a proof of (local) opening. We require the prover to use fixed randomness (e.g. 0) to compute the commitment. This allows the verifier to check that the prover has indeed committed to the correct statements.

Improving parameters for SNARGs for batch-$\mathcal{NP}$. Note that our constructed batch arguments for $\mathcal{NP}$ has proof size $\mathsf{poly}(\log k, |C|)$. These parameters sufficed for the construction of our delegation scheme for polynomial-time computations since there the circuit size $|C| = \mathsf{poly}(\lambda)$, where λ is the security parameter (which we have not included thus far in our discussion to avoid notation clutter). But in the case of SNARGs for batch-$\mathcal{NP}$, we want to remove the dependence on $|C|$, since the circuit may be large.

To do so, we leverage our delegation scheme for deterministic polynomial-time computations. Consider the $\mathcal{NP}$ language $\mathcal{L} = \{x \mid \exists w \text{ s.t. } \mathcal{M}(x, w) \text{ outputs 1 in } T \text{ steps }\}$. The high-level idea is the following:

- The prover generates k delegation proofs $\{\Pi_i\}_{i\in[k]}$ that $\mathcal{M}$ outputs 1 for each of the inputs $(x_1, w_1), \ldots, (x_k, w_k)$. By the efficiency of the delegation scheme, each of these proofs are of size $\mathsf{poly}(\lambda, \log T, |x|, |w|)$.
- Next, the prover computes a SNARGs for batch-index to prove that the delegation verifier will accept (x_i, w_i, Π_i) for every i.

The $|C|$ in the BARG now corresponds to the size of the delegation verifier circuit, which is only $\mathsf{poly}(\lambda, \log T, |w|)$. This gives us the desired efficiency properties.

4.2 SNARGs for Batch-$\mathcal{NP}$

This section is organized as follows.

- In section 4.2.1, we define SNARGs for batch-$\mathcal{NP}$ in the circuit model.
- Next, in section 4.2.2, we define and construct PCP with fast online verification which will be necessary for our construction of SNARGs for batch-$\mathcal{NP}$.
- In Section 4.2.3, we define SNARGs for batch-index. We then construct them

generically from PCP with fast online verification, and somewhere extractable commitments.

- Finally, in section 4.2.4, we construct SNARGs for batch-$\mathcal{NP}$ in the circuit model generically from SNARGs for batch-index, and somewhere extractable commitments.

4.2.1 Definition

Circuit Satisfiability Language. Let SAT be the following language

$$\mathsf{SAT} = \{(C, x) \mid \exists\ w \text{ s.t. } C(x, w) = 1\},$$

where $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is a Boolean function, and $x \in \{0,1\}^n$ is an instance.

A non-interactive batch argument for SAT is a protocol between a prover and a verifier. The prover and the verifier first agree on a circuit C, and a series of T instances $x_1, x_2, \ldots, x_T$. Then the prover sends a single message to the verifier and tries to convince the verifier that $(C, x_1), (C, x_2), \ldots, (C, x_T) \in \mathsf{SAT}$.

More formally, such a protocol is specified by a tuple of algorithms $(\mathsf{Gen}, \mathsf{TGen}, \mathsf{P}, \mathsf{V})$ that work as follows.

- $\mathsf{Gen}(1^\lambda, 1^T, 1^{|C|})$: On input a security parameter λ, the number of instances T, and the size of the circuit C, the CRS generation algorithm outputs crs.
- $\mathsf{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$: On input a security parameter λ, the number of instances T, the size of the circuit C and an index i^*, the trapdoor CRS generation algorithm outputs crs^*.
- $\mathsf{P}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \omega_1, \omega_2, \ldots, \omega_T)$: On input crs, a circuit C, and T instances $x_1, x_2, \ldots, x_T$ and their corresponding witnesses $\omega_1, \omega_2, \ldots, \omega_T$, the prover algorithm outputs a proof π.

- $\mathsf{V}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \pi)$: On input crs, a circuit C, a series of instances $x_1, x_2, \ldots, x_T$, and a proof π, the verifier algorithm decides to accept (output 1) or reject (output 0).

Furthermore, we require the aforementioned algorithms to satisfy the following properties.

- **Succinct Communication.** The size of π is bounded by $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Compact CRS.** The size of crs is bounded by $\mathsf{poly}(\lambda, \log T, |C|)$.
- **Succinct Verification.** The verification algorithm runs in time $\mathsf{poly}(\lambda, T, n) + \mathsf{poly}(\lambda, \log T, |C|)$. Moreover, it can be split into the following two parts[3]:
  - **Pre-processing:** There exists a deterministic algorithm $\mathsf{PreVerify}(\mathsf{crs}, x_1, x_2, \ldots, x_T)$ that takes as input the CRS, and T instances $x_1, x_2, \ldots, x_T$, and outputs a short sketch c, where $|c| = \mathsf{poly}(\lambda, \log T, |x_1|)$.
  - **Online Verification:** There exists an online verification algorithm $\mathsf{OnlineVerify}(\mathsf{crs}, c, C, \pi)$ that takes as input the sketch c, a circuit C, and a proof π, and outputs 1 (accepts) or 0 (rejects). Furthermore, the running time of the online verification algorithm is $\mathsf{poly}(\lambda, |C|, |c|, |\pi|) = \mathsf{poly}(\lambda, \log T, |C|)$.
- **CRS Indistinguishability.** For any non-uniform PPT adversary $\mathcal{A}$, and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Big| \Pr\left[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T) : \mathcal{A}(\mathsf{crs}) = 1\right] - \Pr\left[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs}^* \leftarrow \mathsf{TGen}(1^\lambda, 1^T, i^*) : \mathcal{A}(\mathsf{crs}^*) = 1\right] \Big| \leq \nu(\lambda).$$

[3]We note this is a stronger property than previously considered. However, its is natural, and our construction achieves this property.

- **Completeness.** For any circuit C, any T instances $x_1, \ldots, x_T$ such that $(C, x_1), (C, x_2), \ldots, (C, x_T) \in \mathsf{SAT}$ and witnesses $\omega_1, \omega_2, \ldots, \omega_T$ for $(C, x_1), (C, x_2), \ldots, (C, x_T)$, we have

$$\Pr\Big[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T, 1^{|C|}), \pi \leftarrow \mathsf{P}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \omega_1, \omega_2, \ldots, \omega_T) : \mathsf{V}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \pi) = 1\Big] = 1.$$

- **Semi-Adaptive Somewhere Soundness.** For any non-uniform PPT adversary $\mathcal{A}$, and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda) \leq \nu(\lambda)$, where $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda)$ is defined as

$$\Pr\Big[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs}^* \leftarrow \mathsf{TGen}(1^\lambda, 1^T, i^*), (C, x_1, x_2, \ldots, x_T, \Pi) \leftarrow \mathcal{A}(\mathsf{crs}^*) : i^* \in [T] \wedge (C, x_{i^*}) \notin \mathsf{SAT} \wedge \mathsf{V}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \Pi) = 1\Big].$$

- **Somewhere Argument of Knowledge.** There exists a PPT extractor E such that, for any non-uniform PPT adversary $\mathcal{A}$, and any polynomial $T = T(\lambda)$, there exists a negligible function $\nu(\lambda)$ such that

$$\Pr\Big[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs}^* \leftarrow E(1^\lambda, 1^T, i^*), (C, x_1, x_2, \ldots, x_T, \Pi) \leftarrow \mathcal{A}(\mathsf{crs}^*), \omega \leftarrow E(C, x_1, x_2, \ldots, x_T, \Pi) : C(x_{i^*}, \omega) = 1\Big] \geq \Pr\Big[i^* \leftarrow \mathcal{A}(1^\lambda, 1^T), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T), (C, x_1, x_2, \ldots, x_T, \Pi) \leftarrow \mathcal{A}(\mathsf{crs}^*) : \mathsf{V}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \Pi) = 1\Big] - \nu(\lambda).$$

  Moreover, the CRS generated by the extractor $\mathsf{crs}^* \leftarrow E(1^\lambda, 1^T, i^*)$ and the CRS in real execution $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T)$ are computationally indistinguishable.

4.2.2 PCP with Fast Online Verification

In this subsection, we define PCPs with a *fast online verification property*. At a high level, such a property requires that for any PCP for the circuit satisfiability language

$$\mathsf{C\text{-}SAT} = \{x \mid \exists w : C(x, w) = 1\},$$

the verification algorithm can be split into two parts: (i) a query algorithm Q which generates the PCP queries that depend on C but are independent of x; and (ii) an online verification algorithm D, which depends on x but its running time grows only *polylogarithmically* in $|C|$ and polynomially in $|x|$. Previously, such a property is implicit in the construction of the linear PCPs in [20]. In this work, we focus on the original definition of PCPs (as opposed to linear PCPs).

More formally, for any Boolean circuit $C : \{0,1\}^{|x|} \times \{0,1\}^{|w|} \to \{0,1\}$, a PCP with fast online verification for $\mathsf{C\text{-}SAT}$ is a tuple of polynomial-time algorithms $(\mathsf{P}, \mathsf{Q}, \mathsf{D})$, with the following syntax.

- $\mathsf{P}(1^\lambda, C, x, \omega)$: The prover algorithm takes as input a security parameter λ, the circuit C, an instance x and its witness ω, and outputs a PCP proof $\pi \in \{0,1\}^*$.
- $\mathsf{Q}(1^\lambda, C, r)$: On input the security parameter λ, the circuit C, and the random coin r, the query algorithm generates a subset $Q \subseteq [|\pi|]$, and a state st.
- $\mathsf{D}(x, \mathsf{st}, \pi')$: On input an instance x, a state st, and a binary string $\pi' \in \{0,1\}^{|Q|}$, the online verification algorithm D *deterministically* decides to accept (output 1) or reject (output 0).

Furthermore, we require the following properties of the PCP.

- **Completeness.** For any circuit C, any instance $x \in \mathsf{C\text{-}SAT}$, and any witness ω for x, we have

$$\Pr_r \left[\pi \leftarrow \mathsf{P}(1^\lambda, C, x, \omega), (Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^\lambda, C, r) : \mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1\right] = 1.$$

- $\rho(\lambda)$**-Soundness.** For any circuit C, and any $x \notin \mathsf{C\text{-}SAT}$, and any string $\pi^* \in \{0,1\}^*$,

$$\Pr_r \left[(Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^\lambda, C, r) : \mathsf{D}(x, \mathsf{st}, \pi^*|_Q) = 1\right] \leq \rho(\lambda).$$

- **Polynomial Proof Size.** The size of the proof π is bounded by $\mathsf{poly}(\lambda, |C|)$.
- **Small Query Complexity.** The size of the set Q is bounded by $\mathsf{poly}(\lambda, \log |C|)$.
- **Succinct Verification.** The state st can be represented in $\mathsf{poly}(\lambda, |x|, \log |C|)$ bits, and the online verification algorithm runs in time $\mathsf{poly}(\lambda, |x|, \log |C|)$. The query algorithm Q runs in time $\mathsf{poly}(\lambda, |C|)$.
- **ρ-Proof of Knowledge.** For any PCP proof π^*, there exists a deterministic polynomial time extractor E such that, if $\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^\lambda, C, r) : \mathsf{D}(x, \mathsf{st}, \pi^*|_Q) = 1] > \rho(\lambda)$, then $\Pr[\omega \leftarrow E(\pi^*) : C(x, \omega) = 1] = 1$.

Lemma 4.2.1. *There exists a PCP with fast online verification for the* C-SAT *language with ρ-soundness, and ρ-proof of knowledge property, where $\rho = 1 - 1/\mathsf{poly}(\log |C|)$.*

Proof Sketch. We show that the PCP in [96], and the probabilistic checkable interactive proofs in [92], can be modified to obtain a PCP with fast online verification. For any circuit C, by the Cook-Levin Theorem, there exists a 3-CNF ϕ such that for any x, $\phi(x, \cdot)$ is satisfiable if and only if $x \in \mathsf{C\text{-}SAT}$. Furthermore, for any witness ω of $x \in \mathsf{C\text{-}SAT}$, we can derive a witness y for $\phi(x, \cdot)$, where $|y| = O(|C|)$.

Parameters and Ingredients. Let $\mathbb{H}$ be a field of size $\mathsf{poly}\log|C|$, and let $\mathbb{F}$ be a large enough extension field of $\mathbb{H}$ with size $\mathsf{poly}(\log |C|)$. Let $m_x = \log_{|\mathbb{H}|} |x|$, and $m_y = \log_{|\mathbb{H}|} |y|$. Let $m' = \log_{|\mathbb{H}|}(|x| + |y|)$, and $n = |x|$.

Let $I : \mathbb{H}^{m'} \to \{0, 1\}, K : \mathbb{H}^{m'} \to \mathbb{H}^{\max(m_x, m_y)}$ be the following polynomials.

$$I(i) = \begin{cases} 1 & i \leq n, \\ 0 & \text{Otherwise.} \end{cases} \qquad K(i) = \begin{cases} i & i \leq n, \\ i - n & \text{Otherwise.} \end{cases}$$

where we identify the index set $[|x|+|y|]$ with $\mathbb{H}^m$. Let $\widetilde{I}, \widetilde{K}$ be the extension of I, K to $\mathbb{F}$, respectively. Then $\widetilde{I}$ and $\widetilde{K}$ has degree at most $\mathsf{poly}(m')$. Let $\widetilde{x} = \mathsf{LDE}(x), \widetilde{y} = \mathsf{LDE}(y)$ be the low-degree extension of x, y over $\mathbb{F}$, respectively.

Let $\widetilde{P}(i_1, i_2, i_3, b_1, b_2, b_3)$ be the following polynomial.

$$\widetilde{P}(i_1, i_2, i_3, b_1, b_2, b_3) = \prod_{j \in \{1,2,3\}} \left(\widetilde{I}(i_j) \cdot \widetilde{x}(\widetilde{K}(i_j)) + (1 - \widetilde{I}(i_j)) \cdot \widetilde{y}(\widetilde{K}(i_j)) - b_j \right) \quad (4.1)$$

Let $C' : \mathbb{H}^{3m'+3} \to \{0,1\}$ be a circuit such that $C'(i_1, i_2, i_3, b_1, b_2, b_3) = 1$ if and only if $b_1, b_2, b_3 \in \{0,1\}$ and $(x_{i_1} = b_1) \vee (x_{i_2} = b_2) \vee (x_{i_3} = b_3)$ is a clause in the 3-CNF ϕ, and let $\widetilde{C} : \mathbb{F}^{3m'+3} \to \mathbb{F}$ be the extension of C' to $\mathbb{F}$. Then we have that $\phi(x, \cdot)$ is satisfiable, if and only if there exists a $\widetilde{y}$ such that the following polynomial $F(z)$ of $3m' + 3$ variables is a zero polynomial:

$$F(z) = \sum_{i_1, i_2, i_3 \in \mathbb{H}^{m'}, b_1, b_2, b_3 \in \{0,1\}} \widetilde{C}(i_1, i_2, i_3, b_1, b_2, b_3) \cdot \widetilde{P}(i_1, i_2, i_3, b_1, b_2, b_3) \cdot \widetilde{\mathsf{Eq}}_{i_1, i_2, i_3, b_1, b_2, b_3}(z)$$

Construction Sketch. The PCP construction is the unrolling of the following interactive protocol consisting of two parts.

- **Low-Degree Testing:** The prover sends $\widetilde{y} = \mathsf{LDE}(y)$. The verifier performs a low-degree test on $\widetilde{y}$.
- **Sumcheck:** Then the verifier sends a random $z^* \in \mathbb{F}^{3m'+3}$. The prover and the verifier then execute a sumcheck protocol to prove $F(z^*) = 0$. Let

  $$\widetilde{\phi}_{z^*}(i_1, i_2, i_3, b_1, b_2, b_3) = \mathsf{LDE}(\{\widetilde{\mathsf{Eq}}_{i_1, i_2, i_3, b_1, b_2, b_3}(z^*)\}_{i_1, i_2, i_3 \in \mathbb{H}^{m'}, b_1, b_2, b_3 \in \{0,1\}})$$

  be the low-degree extension of the linear coefficients $\widetilde{\mathsf{Eq}}_{i_1, i_2, i_3, b_1, b_2, b_3}(z^*)$. Then the prover and the verifier run the sumcheck protocol for the sum

  $$\sum_{i_1, i_2, i_3 \in \mathbb{H}^{m'}, b_1, b_2, b_3 \in \{0,1\}} \widetilde{C}(i_1, i_2, i_3, b_1, b_2, b_3) \cdot \widetilde{P}(i_1, i_2, i_3, b_1, b_2, b_3) \cdot \widetilde{\phi}_{z^*}(i_1, i_2, i_3, b_1, b_2, b_3) = 0.$$

  At the end of the sumcheck protocol, the verifier obtains a random point $(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*) \in \mathbb{F}^{3m'+3}$ (corresponding to its messages in the protocol) and a value $v \in \mathbb{F}$. The verifier then checks whether

  $$\widetilde{C}(i_1^*, i_2^*, i_3^*, i_1^*, b_2^*, b_3^*) \cdot \widetilde{P}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*) \cdot \widetilde{\phi}_{z^*}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*) = v. \quad (4.2)$$

We now describe how to fit this PCP construction into our definition of PCP with fast online verification.

- $\mathsf{PCP.Q}(1^\lambda, C, r)$: We now show that the PCP queries can be generated independently of x. The PCP query consists of the queries in (i) the low-degree testing of $\widetilde{y}$; and (ii) the sumcheck. The low-degree testing queries only query some values of $\widetilde{y}$. Hence, these queries are generated independently of x. The sumcheck protocol is public-coin. Therefore, the queries in sumcheck can also be generated independent of x.

  In addition, for the sumcheck, we do the following "preprocessing" to save time in online verification. For $\widetilde{C}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*)$, we evaluate it directly, and store the resultant value in the state st. Furthermore, to help the online verification algorithm (described below) compute $\widetilde{P}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*)$ in time only polylogarithmic in $|C|$, we compute $\widetilde{K}(i_j^*), \widetilde{I}(i_j^*)$ and $\{\widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_{m_x}}(\widetilde{K}(i_j^*))\}_{i_1,i_2,\ldots,i_{m_x}\in\mathbb{H}}$ for $j \in \{1,2,3\}$ in time $\mathsf{poly}(C)$, and also store the resultant values in the state st. Finally, we compute and store $\widetilde{\phi}_{z^*}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*)$ in st.

  Since there are $O(n)$ number of field elements in the state st, the size of st is bounded by $\mathsf{poly}(\lambda, |x|, \log|C|)$.

- $\mathsf{PCP.D}(x, \mathsf{st}, \pi')$: We will show that given the state st, the verification runs in $\mathsf{poly}(|x|, \log|C|)$ time.

  For the low-degree testing, the verifier performs the same verification procedure as the underlying low-degree testing. This takes time $\mathsf{poly}(\log|C|)$.

  For the sumcheck, the verifier performs the same checks as in the underlying sumcheck protocol. At the end, the verifier uses the "preprocessed" values of $\widetilde{C}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*)$ present in the state st. To compute $\widetilde{P}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*.b_3^*)$, the verifier will obtain each term in Equation 4.1. For $\widetilde{K}(i_j^*), \widetilde{I}(i_j^*)$, the verifier can obtain them from the state st. For $\widetilde{y}(\widetilde{K}(i_j^*))$, the verifier obtains it from the

PCP response π'. For $\widetilde{x}(\widetilde{K}(i_j^*))$, the verifier computes it by the definition of low-degree extension (see Section 2.4.3) using $\{\widetilde{\mathsf{Eq}}_{i_1,i_2,\ldots,i_{m_x}}(\widetilde{K}(i_j^*))\}_{i_1,i_2,\ldots,i_{m_x}}$ in the state st. Now the verifier obtains all terms in Equation 4.1, and hence can compute $\widetilde{P}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*)$. Finally, the verifier obtains $\widetilde{\phi}_{z^*}(i_1^*, i_2^*, i_3^*, b_1^*, b_2^*, b_3^*)$ from the state st, and verifies Equation 4.2.

For the running time, the computation of the low-degree extension $\widetilde{x}(\widetilde{K}(i_j^*))$ takes time $O(|x| \cdot \mathsf{poly}(\log |C|))$. Hence, the online verification takes time $\mathsf{poly}(|x|, \log |C|)$ in total.

By the above running time analysis, the succinct verification property is satisfied. The small query complexity follows from the small query complexity of the low-degree testing and the sumcheck protocol. Since the sumcheck has $O(m')$-rounds, and the prover sends $O(1)$ elements in $\mathbb{F}$ in each round, the unrolled proof has size $|\mathbb{F}|^{O(m')} = \mathsf{poly}(|C|)$. Hence, polynomial proof size property follows.

The completeness and soundness follows from the completeness and soundness of the zero-testing and the sumcheck protocol. The proof of knowledge property follows from the decoding of $\widetilde{y}$. □

Next, we define the *bad* relation for any PCP with fast online verification with an eye towards our BARG construction we describe next. As described in the overview in section 5.1, we commit several PCP proofs "columnwise" using a somewhere extractable commitmentand apply a CIH to these commitments to obtain the query PCP Q.

In the soundness proof, we first switch the commitment key to the trapdoor mode. The *bad* relation is defined with respect to the trapdoor td of the commitment. Specifically, we can use td to extract a PCP proof π from the commitment. Now given the extracted proof π, we define a query Q to be bad, when $\pi|_Q$ is accepting but we cannot extract a witness from π. However, the verification algorithm not only needs Q, but also the state st. To resolve this issue, in the following definition, we have the

CIH output the randomness r. We then use this randomness to generate Q and st via $\mathsf{PCP.Q}$.

Definition 4.2.2 (Bad relation for PCP)**.** *Let* $\mathsf{SECOM} = (\mathsf{SECOM.Gen}, \mathsf{SECOM.TGen}, \mathsf{SECOM.Com}, \mathsf{SECOM.Open}, \mathsf{SECOM.Verify}, \mathsf{SECOM.Ext})$ *be a somewhere extractable commitment scheme, and* $\mathsf{PCP} = (\mathsf{P}, \mathsf{Q}, \mathsf{D})$ *be any PCP with fast online verification, we define the bad relation* $\mathcal{R} = \{\mathcal{R}_\lambda\}_\lambda$ *for* PCP *as follows.*

For any instance length $n = n(\lambda)$*, witness length* $m = m(\lambda)$*, proof length* $\ell = \ell(\lambda)$*, and a parameter* $T = T(\lambda)$*, we define the bad relation for* PCP *as* $\mathcal{R}_\lambda = \{R_{\lambda,x,\mathsf{td}}\}$*, where* td *is obtained from* $(K^*, \mathsf{td}) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^T, i^*)$ *for a index* $i^* \in [T]$*, and*

$$R_{\lambda,x,\mathsf{td}} = \{((C, c), r) \mid C(x, E(\pi)) \neq 1 \wedge \mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1\},$$

where $(Q, \mathsf{st}) = \mathsf{Q}(1^\lambda, C, r)$*,* $c = \{c_q\}_{q\in[\ell]}, \pi = \{\mathsf{SECOM.Ext}(c_q, \mathsf{td})\}_{q\in[\ell]}$*,* $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ *is a Boolean circuit, and* x *is a string of length* n*, and* E *is the proof of knowledge extractor.*

Theorem 4.2.3 (CIH for PCP)**.** *There exists a PCP with fast online verification* $(\mathsf{P}, \mathsf{Q}, \mathsf{D})$ *and a hash family* $\mathcal{H}$ *such that,* $\mathcal{H}$ *is correlation intractable for its bad relation family* $\mathcal{R} = \{\mathcal{R}_\lambda\}_\lambda$ *(in Definition 4.2.2). Furthermore,* $\mathcal{H}$ *can be evaluated in time* $\mathsf{poly}(\lambda, \log T, |C|)$

Proof. Intuitively, we will take the PCP in Lemma 4.2.1, and repeat its verification several times in parallel (with independent randomness), and apply Theorem 3.1.5 to the resulting PCP.

Let $\mathsf{PCP}' = (\mathsf{PCP}'.\mathsf{P}, \mathsf{PCP}'.\mathsf{Q}, \mathsf{PCP}'.\mathsf{D})$ be the $(1-\epsilon)$-sound PCP with fast online verification from Lemma 4.2.1, where $\epsilon = 1/\mathsf{poly}(\log |C|)$. We build a new $\mathsf{PCP} = (\mathsf{P}, \mathsf{Q}, \mathsf{D})$ as follows.

- P is the same as $\mathsf{PCP}'.\mathsf{P}$.

- $\mathsf{Q}(1^\lambda, C, r)$: Parse $r = (r_1, r_2, \ldots, r_t)$, where $t = \lambda/\epsilon$.
  - For each $i \in [t]$, let $(Q_i, \mathsf{st}_i) = \mathsf{PCP}'.\mathsf{Q}(1^\lambda, C, r_i)$.
  - Output $Q = (Q_1, Q_2, \ldots, Q_t), \mathsf{st} = (\mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_t)$.
- $\mathsf{D}(x, \mathsf{st}, \pi')$ Parse $\pi' = (\pi'_1, \pi'_2, \ldots, \pi'_t)$, and $\mathsf{st} = (\mathsf{st}_1, \mathsf{st}_2, \ldots, \mathsf{st}_t)$.
  - For each $i \in [t]$, verify if $\mathsf{PCP}'.\mathsf{D}(x, \mathsf{st}_i, \pi'_i) = 1$.
  - If all verification passes, then output 1 (accept). Otherwise output 0 (reject).
- Proof of knowledge Extractor E: We use the proof of knowledge extractor of PCP' as the extractor for PCP.

The resultant PCP satisfies $\rho = (1-\epsilon)^t = 2^{-\Omega(\lambda)}$-soundness and ρ-proof of knowledge property. By construction, for each security parameter λ, instance x, and trapdoor td, $R_{\lambda,x,\mathsf{td}}$ is a product relation, since the bad relation for PCP is the product of the bad relations for PCP'. The bad relation for PCP' is efficiently verifiable in time $\mathsf{poly}(\lambda, \log T, |C|)$.

To demonstrate sparsity, for any instance x and extracted PCP proof π, if $C(x, E(\pi)) \neq 1$, then $\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^\lambda, C, r) : \mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1] \leq \rho$, otherwise this contradicts the ρ-soundness of PCP. Since our construction is a parallel repetition,

$$\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{Q}(1^\lambda, C, r) : \mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1] = \Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{PCP}'.\mathsf{Q}(1^\lambda, C, r) : \mathsf{PCP}'.\mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1]^t.$$

Hence, if the left hand is bounded by ρ, then we have

$$\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{PCP}'.\mathsf{Q}(1^\lambda, C, r) : \mathsf{PCP}'.\mathsf{D}(x, \mathsf{st}, \pi|_Q) = 1] \leq 1 - \epsilon.$$

Hence, the relation $R_{\lambda,\mathsf{td}}$ has sparsity $(1-\epsilon)$. Therefore, by Theorem 3.1.5, there exists a correlation intractable hash family $\mathcal{H}$ for $\mathcal{R}$. □

4.2.3 SNARGs for Index Languages

Index Language. Let the index language be the following language

$$\mathcal{L}^{\mathsf{idx}} = \{(C, i) \mid \exists\ w \text{ s.t. } C(i, w) = 1\},$$

where C is a Boolean function, and i is an index.

Non-interactive batch arguments for the index language is a special case of BARGs for general circuit satisfiability when the instances $x_1, x_2, \ldots, x_T$ are simply the indices $1, 2, \ldots, T$. We therefore omit $x_1, x_2, \ldots, x_T$ as inputs to the prover and the verifier algorithms $\mathsf{BARG.P}, \mathsf{BARG.V}$ (and also as an output of the adversary $\mathcal{A}$ describing the semi-adaptive somewhere soundness property). Furthermore, since the verifier does not need to read the instances, there is no pre-processing in this case, and the succinct verification property requires the verifier to run in time $\mathsf{poly}(\lambda, \log T, |C|)$.

In this subsection, we mainly prove the following theorem.

Theorem 4.2.4 (BARGs for Index Language)**.** *Assuming LWE, there exist batch arguments for the index language with succinct verification property.*

We proceed to describe the construction of the BARGs for the index language.

Ingredients. Our construction is recursive. We use an index L to index the level of recursion. Note that we can assume without loss of generality that T is a power of two by padding the last instance $(2^{\lceil \log_2 T \rceil} - T)$ times. The BARGs at the L-level can handle $T = 2^L$ instances. To construct $\mathsf{BARG}_L = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{P}, \mathsf{V})$ at the L-level, we need the following ingredients.

- A somewhere extractable commitment (section 2.4.4) $\mathsf{SECOM} = (\mathsf{SECOM.Gen}, \mathsf{SECOM.TGen}, \mathsf{SECOM.Com}, \mathsf{SECOM.Open}, \mathsf{SECOM.Verify}, \mathsf{SECOM.Ext})$.
- A PCP with fast online verification scheme $\mathsf{PCP} = (\mathsf{PCP.P}, \mathsf{PCP.Q}, \mathsf{PCP.D})$ with proof length $\ell = \ell(\lambda, |C|)$ from Theorem 4.2.3.

- A CIH (definition 3.1.1) $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ for the bad relation of PCP with fast online verification from Theorem 4.2.3.

- A non-interactive batch arguments at the $(L-1)$-level $\mathsf{BARG}_{L-1} = (\mathsf{BARG}_{L-1}.\mathsf{Gen}, \mathsf{BARG}_{L-1}.\mathsf{TGen}, \mathsf{BARG}_{L-1}.\mathsf{P}, \mathsf{BARG}_{L-1}.\mathsf{V})$.

Construction. We proceed to describe the construction. In the base case $L = 0$ and $T = 1$, we have the prover send the witness directly to the verifier, and have the verifier verify the witness. When $L \geq 1$, we reduce the batch argument to verify a batch of $T/2$ instances, and apply the $(L-1)$-level BARG recursively. In more detail, we construct BARG for $L \geq 1$ as follows.

Circuit $\mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}(i, (\vec{\rho}, \pi', \vec{\rho}', \pi''))$

Output $\mathsf{VerifyC}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}(2i-1, \vec{\rho}, \pi') \wedge \mathsf{VerifyC}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}(2i, \vec{\rho}', \pi'')$.

Figure 4-2. The grouped new circuit, where the ungrouped new circuit VerifyC is depicted in Figure 4-3.

- $\underline{\mathsf{Gen}(1^\lambda, 1^{T=2^L}, 1^{|C|})}$: The CRS generation algorithm generates a CRS, which contains (i) a somewhere extractable commitment key; (ii) a CRS for the smaller non-interactive batch arguments BARG_{L-1}; and (ii) a key for the CIH $\mathcal{H}$.

  - Let $K \leftarrow \mathsf{SECOM.Gen}(1^\lambda, 1^T, 1^1), \mathsf{crs}' \leftarrow \mathsf{BARG}_{L-1}.\mathsf{Gen}(1^\lambda, 1^{T'}, 1^{|\mathsf{NewRel}|})$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$, where $T' = T/2$.

  - Let $\mathsf{crs} = (K, \mathsf{crs}', \mathcal{H}.k)$ and output crs.

- $\underline{\mathsf{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)}$: The trapdoor CRS generation algorithm generates the trapdoor CRS as follows.

  - Generate $(K^*, \mathsf{td}) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$,

Circuit $\mathsf{VerifyC}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}(i,\vec{\rho},\pi')$

Hardwired: The commitment key K, the set Q, the commitments $\{c_q\}_{q\in Q}$, and the state st for PCP verification.

Parse the input $\vec{\rho}=\{\rho_q\}_{q\in Q}$, and $\pi'=\{\pi'_q\}_{q\in Q}$.

- For each $q\in Q$, verify the opening π'_q to the commitment c_q. Specifically, verify
$$\forall q\in Q, \mathsf{SECOM.Verify}(K,c_q,\pi'_q,i,\rho_q)=1.$$
- Verify π' is accepted by the PCP online verification, i.e. verify $\mathsf{PCP.D}(\mathsf{st},i,\pi')=1$.
- If all verification passes, then output 1 (accept), otherwise output 0 (reject).

Figure 4-3. The ungrouped new circuit.

  - Let $\mathsf{crs}^{*\prime}\leftarrow\mathsf{BARG.TGen}(1^\lambda,1^{T'},1^{|\mathsf{NewRel}|},\lfloor(i^*+1)/2\rfloor)$, and $\mathcal{H}.k\leftarrow\mathcal{H}.\mathsf{Gen}(1^\lambda)$.
  - Let $\mathsf{crs}^*=(K^*,\mathsf{crs}^{*\prime},\mathcal{H}.k)$, and output crs^*.

- $\underline{\mathsf{P}(\mathsf{crs},C,\omega_1,\omega_2,\ldots,\omega_T)}$: The prover algorithm first commits to all PCP strings in a "columnwise" manner, and then applies the CIH to the commitment.
  - For each $i\in[T]$, compute the PCP proof $\pi_i\leftarrow\mathsf{PCP.P}(1^\lambda,C,i,\omega_i)$ for i-th instance (C,i).
  - Committing the $\pi=\{\pi_i\}_{i\in[T]}$ "columnwise",
$$\forall q\in[\ell], c_q\leftarrow\mathsf{SECOM.Com}(K,\{\pi_i|_q\}_{i\in[T]};r_q),$$
    with uniformly random r_q.
  - Applying the CIH to $c=\{c_q\}_{q\in[\ell]}$, $r\leftarrow\mathcal{H}.\mathsf{Hash}(\mathcal{H}.k,(C,c))$, and let $(Q,\mathsf{st})\leftarrow\mathsf{PCP.Q}(1^\lambda,C,r)$.
  - For each $i\in[T]$, let $\vec{\rho}_i$ be the opening of $\pi_i|_Q$. Specifically,
$$\vec{\rho}_i=\{\mathsf{SECOM.Open}(K,\{\pi_i|_q\}_{i\in[T]},i,r_q)\}_{q\in Q}.$$

– Compute a smaller BARG proof, let

$$\Pi' \leftarrow \mathsf{BARG}'.\mathsf{P}(\mathsf{crs}', \mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}, \{\vec{\rho}_{2i-1}, \pi_{2i-1}|_Q, \vec{\rho}_{2i}, \pi_{2i}|_Q\}_{i\in[T']}),$$

where NewRel is depicted in Figure 4-2.

– Output the proof $\Pi = (c, \Pi')$.

- $\underline{\mathsf{V}(\mathsf{crs}, C, \Pi)}$: The verification algorithm parses the proof Π as the commitment and the proof for the smaller BARG, then it utilizes the fast online verification property of the PCP to delegate the online verification to the smaller BARG.

  – Parse $\Pi = (c, \Pi')$. Applying CIH to c, let $r \leftarrow \mathcal{H}.\mathsf{Hash}(\mathcal{H}.k, (C, c))$.

  – Generate the PCP query, $(Q, \mathsf{st}) \leftarrow \mathsf{PCP}.\mathsf{Q}(1^\lambda, C, r)$.

  – Verify the smaller BARG, output $\mathsf{BARG}_{L-1}.\mathsf{V}(\mathsf{crs}', \mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}, \Pi')$.

Before analysing the efficiency, we first bound the size of the circuit $\mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}$.

Lemma 4.2.5. *In the construction of* NewRel *in Figure 4-2, we have*

$$|\mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}| = \mathsf{poly}(\lambda, \log T, \log C).$$

Proof. By the construction of NewRel, since $2i-1$ and $2i$ can be computed by a circuit of size $O(\log T)$, we have $|\mathsf{NewRel}| = \tilde{O}(|\mathsf{VerifyC}|)$. For $\mathsf{VerifyC}$, we analyse the size of the circuit computing each step.

- First, $\mathsf{VerifyC}$ verifies if the opening π'_q is accepted with respect to c_q. By the succinct verification property of the somewhere extractable commitment, this step can be computed by a circuit of size $\mathsf{poly}(\lambda, \log T) \cdot |Q|$. By the small query complexity of the PCP, this term is bounded by $\mathsf{poly}(\lambda, \log T, \log C)$.

- Second, $\mathsf{VerifyC}$ verifies the PCP proof. By the succinct verification property of PCP, this step can be computed by a circuit of size $\mathsf{poly}(\lambda, \log T, \log C)$.

Since each step of $\mathsf{VerifyC}$ can be computed by a circuit of size $\mathsf{poly}(\lambda, \log T, \log C)$, which completes the proof. □

Lemma 4.2.6 (Succinct Proof)**.** *The aforementioned construction satisfies the succinct proof property.*

Proof. We analyse the length of the proof recursively. By construction, we have

$$|\Pi_L| = |c| + |\Pi_{L-1}|,$$

where Π_{L-1} is the proof length of the $(L-1)$-level BARG. By the succinct commitment property, the size of c is bounded by $\mathsf{poly}(\lambda, \log T) \cdot \ell$. Since the length of the PCP proof is bounded by $\mathsf{poly}(|C|)$, we have that $\ell = \mathsf{poly}(|C|)$. Hence, $|\Pi_L| = \mathsf{poly}(\lambda, \log T, |C|) + |\Pi_{L-1}|$. Hence, recursively applying Lemma 4.2.5, we have $\Pi_L = \mathsf{poly}(\lambda, \log T, |C|)$. □

Lemma 4.2.7 (Succinct Verification)**.** *The aforementioned construction satisfies succinct verification property.*

Proof. Let Time_L and Time_{L-1} be the verification time for BARG_L and BARG_{L-1} respectively. Then we have

$$\mathsf{Time}_L = \mathsf{Time}_{\mathcal{H}.\mathsf{Hash}} + \mathsf{Time}_{\mathsf{PCP.Q}} + \mathsf{Time}_{L-1},$$

where $\mathsf{Time}_{\mathcal{H}.\mathsf{Hash}}$ is the running time of $\mathcal{H}.\mathsf{Hash}$, and $\mathsf{Time}_{\mathsf{PCP.Q}}$ is the running time of $\mathsf{PCP.Q}$. From Theorem 4.2.3, $\mathsf{Time}_{\mathcal{H}.\mathsf{Hash}} = \mathsf{poly}(\lambda, \log T, |C|)$. From the PCP construction, $\mathsf{PCP.Q}$ needs to be a polynomial time algorithm. Hence, we have $\mathsf{Time}_{\mathsf{PCP.Q}} = \mathsf{poly}(\lambda, |C|)$. Therefore,

$$\mathsf{Time}_L = \mathsf{poly}(\lambda, \log T, |C|) + \mathsf{Time}_{L-1}.$$

Recursively applying this equation, we obtain $\mathsf{Time} = \mathsf{poly}(\lambda, \log T, |C|)$. □

Lemma 4.2.8 (Compact CRS)**.** *The aforementioned construction satisfies compact CRS property.*

Proof. By construction,

$$|\mathsf{crs}_L| = |K| + |\mathsf{crs}_{L-1}| + |\mathcal{H}.k|,$$

where crs_L and crs_{L-1} are the CRS of the BARG at the L-the level and $(L-1)$-th level respectively. From succinct CRS property of the commitment scheme, we have $|K| = \mathsf{poly}(\lambda, \log T)$. The size of the hash key is bounded by the running time of $\mathcal{H}.\mathsf{Hash}$, which is $\mathsf{poly}(\lambda, \log T, |C|)$. Hence, we have

$$|\mathsf{crs}_L| = \mathsf{poly}(\lambda, \log T, |C|) + |\mathsf{crs}_{L-1}|.$$

Recursively applying this equation, by Lemma 4.2.5, we obtain $|\mathsf{crs}| = \mathsf{poly}(\lambda, \log T, |C|)$.

□

Lemma 4.2.9 (CRS indistinguishability)**.** *The aforementioned construction satisfies CRS indistinguishability.*

Proof. Since the construction is recursive, we prove the CRS indistinguishability by induction. In the base case, when $T = 1$, the CRS generated by Gen and TGen is clearly indistinguishable. Now, assuming BARG_{L-1} satisfies CRS indistinguishability, we prove the CRS indistinguishability of BARG by the following hybrid arguments.

- $\underline{\mathsf{Hyb}_0}$: Let $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T, 1^{|C|})$. Output crs.
- $\underline{\mathsf{Hyb}_1}$: Let $(K^*, \mathsf{td}) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$, $\mathsf{crs}' \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T'}, 1^{|\mathsf{NewRel}|})$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$. Output $\mathsf{crs} = (K^*, \mathsf{crs}', \mathcal{H}.k)$.

  This hybrid is computationally indistinguishable with Hyb_0, from the key indistinguishability of the commitment scheme SECOM.
- $\underline{\mathsf{Hyb}_2}$: Let $(K^*, \mathsf{td}) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$, $\mathsf{crs}^{*\prime} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T'}, 1^{|\mathsf{NewRel}|}, \lfloor (i^*+1)/2 \rfloor$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$. Output $\mathsf{crs} = (K^*, \mathsf{crs}', \mathcal{H}.k)$

This hybrid is computationally indistinguishable with Hyb_1, from the CRS indistinguishability of the smaller BARG. This hybrid is identical to $\mathsf{crs}^* \leftarrow \mathsf{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)$.

By the hybrid argument, we finish the proof. □

Lemma 4.2.10 (Completeness)**.** *The aforementioned construction satisfies the completeness property.*

Proof. We prove the completeness by induction. For the base case $L = 0, T = 1$, since the prover sends the witness directly, the completeness is satisfied. Now, assuming the BARG_{L-1} satisfies the completeness, we need to show the completeness of BARG_L. Since the verification algorithm of BARG_L invokes BARG_{L-1} and verifies Π', it suffices to show that $\{\vec{\rho}_{2i-1}, \pi_{2i-1}|_Q, \vec{\rho}_{2i}, \pi_{2i}|_Q\}_{i\in[T']}$ are the witnesses for $\mathsf{NewRel}_{[K,Q,\{c_q\}_{q\in Q},\mathsf{st}]}$. According to the construction, to prove this we only need to show $(\vec{\rho}_{2i-1}, \pi_{2i-1}|_Q)$ is a witness for $\mathsf{VerifyC}_{[K,c,C]}$. This follows from the completeness of the local opening and completeness of the PCP. □

Before proving the semi-adaptive somewhere soundness, we first show in Lemma 4.2.11 that any adversary for the semi-adaptive somewhere soundness of BARG_L can be used to build a new adversary for the semi-adaptive somewhere soundness of BARG_{L-1}. Then in Lemma 4.2.12, we will apply this Lemma recursively to prove the semi-adaptive somewhere soundness of BARG_L.

Lemma 4.2.11. *For any level L and non-uniform PPT adversary $\mathcal{A}$ for the semi-adaptive somewhere soundness of BARG_L with advantage $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda)$, there exists a non-uniform PPT adversary $\mathcal{A}'$ for BARG_{L-1} such that $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}'}(\lambda) \geq \mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda) - \mathsf{negl}(\lambda)$. Furthermore, let $\mathsf{Time}_{\mathcal{A}}$ and $\mathsf{Time}_{\mathcal{A}'}$ be the running time of $\mathcal{A}$ and $\mathcal{A}'$ respectively. Then $\mathsf{Time}_{\mathcal{A}'} = \mathsf{Time}_{\mathcal{A}} + \mathsf{poly}(\lambda)$.*

Proof. We build the adversary $\mathcal{A}'$ trying to break the semi-adaptive somewhere soundness for the underlying $(L-1)$-level BARG scheme BARG_{L-1} as follows.

- $\mathcal{A}'(1^\lambda, 1^{T'})$: Invoke the adversary $i^* \leftarrow \mathcal{A}(1^\lambda, 1^T)$. Output $i^{*\prime} = \lfloor (i^*+1)/2 \rfloor$.
- $\mathcal{A}'(\mathsf{crs}^{*\prime})$: The adversary generates crs for $\mathcal{A}$, and obtains the proof from $\mathcal{A}$.
  - Generate $(K^*, \mathsf{td}) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$, and $\mathcal{H}.k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$.
  - Compose the CRS for BARG, let $\mathsf{crs}^* = (K^*, \mathsf{crs}^{*\prime}, \mathcal{H}.k)$. Feed it to $\mathcal{A}$, $(C, \Pi) \leftarrow \mathcal{A}(\mathsf{crs}^*)$.
  - Parse $\Pi = (c, \Pi')$. Applying CIH to c, $r \leftarrow \mathcal{H}.\mathsf{Hash}(\mathcal{H}.k, c)$, and $(Q, \mathsf{st}) \leftarrow \mathsf{PCP.Q}(1^\lambda, C, r)$.
  - Output $(\mathsf{NewRel}_{[K, Q, \{c_q\}_{q\in Q}, \mathsf{st}]}, \Pi')$.

To argue $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}'}(\lambda) \geq \mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda) - \mathsf{negl}(\lambda)$, we build the following adversary $\mathcal{B}$ for the correlation intractable hash family $\mathcal{H}$.

First, the adversary $\mathcal{B}$ invokes the adversary $\mathcal{A}$ by executing $i^* \leftarrow \mathcal{A}(1^\lambda, 1^{T=2^L})$. Then it generates a SECOM key $(K^*, \mathsf{td}) \leftarrow \mathsf{SECOM.TGen}(1^\lambda, 1^T, \{i^*\})$. The adversary then chooses the bad relation $R_{\lambda, i^*, \mathsf{td}} \in \mathcal{R}_\lambda$ to break the correlation intractability. Next, $\mathcal{B}$ is given a CIH key $\mathcal{H}.k$. $\mathcal{B}$ generates the CRS for the smaller SNARG $\mathsf{crs}^{*\prime} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T'=T/2}, 1^{|\mathsf{NewRel}|}, \lfloor (i^*+1)/2 \rfloor)$, and composes the CRS $\mathsf{crs} = (K^*, \mathsf{crs}^{*\prime}, \mathcal{H}.k)$, and feeds it to $\mathcal{A}$. Let $(C, \Pi) \leftarrow \mathcal{A}(\mathsf{crs})$. Parse $\Pi = (c, \Pi')$. Output (C, c).

By the correlation intractability, on the one hand we have that

$$\Pr\left[\mathcal{H}.k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda), (C, c) \leftarrow \mathcal{B}(\mathcal{H}.k) : ((C, c), \mathcal{H}.\mathsf{Hash}(\mathcal{H}.k, (C, c))) \in R_{\lambda, i^*, \mathsf{td}}\right] \leq \mathsf{negl}(\lambda). \tag{4.3}$$

But on the other hand,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}}(\lambda) &= \Pr\left[(C, i^*) \notin L \wedge \mathsf{V}(\mathsf{crs}, C, \Pi) = 1\right] \\ &= \Pr\left[(C, i^*) \notin L \wedge \mathsf{V}(\mathsf{crs}, C, \Pi) = 1 \wedge (\mathsf{NewRel}, i^{*\prime}) \in L\right] + \\ &\qquad \Pr\left[(C, i^*) \notin L \wedge \mathsf{V}(\mathsf{crs}, C, \Pi) = 1 \wedge (\mathsf{NewRel}, i^{*\prime}) \notin L\right].\end{aligned}$$

By the definition of $\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}'}(\lambda)$, we have that

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A}'}(\lambda) \geq \Pr\left[(C, i^*) \notin L \wedge \mathsf{V}(\mathsf{crs}, C, \Pi) = 1 \wedge (\mathsf{NewRel}, i^{*\prime}) \notin L\right]$$

Hence, it suffices to bound the first term $\Pr[(C, i^*) \notin L \wedge \mathsf{V}(\mathsf{crs}, C, \Pi) = 1 \wedge (\mathsf{NewRel}, i^{*\prime}) \in L]$. We have

$$\begin{aligned}\Pr[(C, i^*) \notin L \wedge (\mathsf{NewRel}, i^{*\prime}) \in L] &= \Pr[(C, i^*) \notin L \wedge \mathsf{NewRel}(i^{*\prime}, \cdot) \text{ is satisfiable}] \\ &\leq \Pr[(C, i^*) \notin L \wedge \mathsf{VerifyC}(i^*, \cdot) \text{ is satisfiable}].\end{aligned}$$

Note that in the last term, if $\mathsf{VerifyC}(i^*, \cdot)$ is satisfiable, then there exists $\vec{\rho}, \pi'$ such that $\vec{\rho}$ is the opening of π' and the PCP verification of π' accepts. By the extraction correctness of SECOM, π' here should be equal to the $\{\mathsf{SECOM.Ext}(c_q, \mathsf{td})\}_{q \in [\ell]}$ in the bad relation $R_{\lambda, i^*, \mathsf{td}}$ definition. Hence, $\mathsf{PCP.D}(i^*, \mathsf{st}, \pi|_Q) = 1$, which implies that $((C, c), r) \in R_{\lambda, i^*, \mathsf{td}}$. By correlation intractability (Equation 4.3), this event is bounded by a negligible probability, which completes the proof. □

Lemma 4.2.12 (Semi-adaptive Somewhere Soundness)**.** *The aforementioned construction satisfies the semi-adaptive somewhere soundness.*

Proof. We recursively apply the Lemma 4.2.11 $L = \log_2 T$ times. At the end, we obtain a polynomial time adversary for the 0-level base case construction. Since the base case protocol has the prover send the witness directly, it is statistically sound. Hence, we reach a contradiction, which completes the proof. □

Remark 4.2.13. *The proof of Lemma 4.2.12 can be extended to show somewhere argument of knowledge property. We provide a proof sketch here. For the base case $L = 0, T = 1$, since the prover sends the witness, we can extract the witness from the*

proof directly. For any $L \geq 1$, we firstly use the extractor of SECOM *to extract the PCP proof π as in the soundness proof, and then apply the proof of knowledge property of the PCP to obtain a witness.*

4.2.4 SNARGs for batch-$\mathcal{NP}$

In this subsection, we present a generic approach to generalize the SNARGs for batch-index and obtain SNARGs for batch-$\mathcal{NP}$.

Theorem 4.2.14. *If there exists a SNARGs for batch-index* $\mathsf{BARG}' = (\mathsf{BARG}'.\mathsf{Gen}, \mathsf{BARG}'.\mathsf{TGen}, \mathsf{BARG}'.\mathsf{P}, \mathsf{BARG}'.\mathsf{V})$ *for the index language $\mathcal{L}^{\mathsf{idx}}$, then there exists a SNARGs* $\mathsf{BARG} = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{P}, \mathsf{V})$ *for* SAT *with succinct verification property.*

Proof Sketch. We construct the BARG as follows.

Circuit $C'_{[K,c,C]}(i, \{\rho_j\}_{j\in[n]}, x, \omega)$

Hardwired: The commitment key K, the commitments $c = \{c_j\}_{j\in[n]}$, and a circuit C.

- Verify the opening $\{\rho_j\}_{j\in[n]}$ and the instance x are the i-th coordinate of c.
$$\forall j \in [n], \text{ verify } \mathsf{SECOM.Verify}(K, c_j, x[j], i, \rho_j) = 1.$$
- Verify the C-SAT: verify if $C(x, \omega) = 1$.

If all verification passes, then output 1. Otherwise, output 0.

Figure 4-4. The new circuit C' for batch argument.

- $\underline{\mathsf{Gen}(1^\lambda, 1^T, 1^{|C|})}$: When generating the CRS, in addition to generating the CRS for BARG', we also generate a somewhere extractable commitment key K.
  - Let $\mathsf{crs}' \leftarrow \mathsf{BARG}'.\mathsf{Gen}(1^\lambda, 1^T, 1^{|C'|})$, where the circuit C' is depicted in Figure 4-4.

- Let $K \leftarrow \mathsf{SECOM.Gen}(1^\lambda, 1^T, 1^1)$ be a somewhere extractable commitment key with set size 1.
- Output $\mathsf{crs} = (K, \mathsf{crs}')$.

- $\underline{\mathsf{TGen}(1^\lambda, 1^T, 1^{|C|}, i^*)}$: In the trapdoor CRS generation algorithm, we generate the trapdoor CRS of BARG', and also generate the trapdoor somewhere extractable commitment key which is extractable at i^*-th coordinate.
  - Let $\mathsf{crs}^{*\prime} \leftarrow \mathsf{BARG}'.\mathsf{Gen}(1^\lambda, 1^T, 1^{|C'|}, i^*)$ be the trapdoor CRS of BARG' for i^*.
  - Let $K^* \leftarrow \mathsf{SECOM.Gen}(1^\lambda, 1^T, 1^1, \{i^*\})$ be the trapdoor somewhere extractable commitment key.
  - Output $\mathsf{crs}^* = (K^*, \mathsf{crs}^{*\prime})$.

- $\underline{\mathsf{P}(\mathsf{crs}, C, x_1, \ldots, x_T, \omega_1, \ldots, \omega_T)}$: The prover first uses the somewhere extractable commitment to commit all the instances, and obtain a short commitment c. Then we have the prover use BARG' to prove the statement: "for each $i \in [T]$, there exists an accepting local opening x_i at the i-th coordinate for the commitment c, and there exists a witness ω_i such that $C(x_i, \omega_i) = 1$".
  - Parse $\mathsf{crs} = (K, \mathsf{crs}')$, where K is a somewhere extractable commitment key, and crs' is a CRS for BARG'.
  - Recall that $n = |x_1|$ is the length of the instances. Commit the instances by
    $$\forall j \in [n], c_j \coloneqq \mathsf{SECOM.Com}(K, x_1[j], x_2[j], \ldots, x_T[j]; 0)^{[4]},$$
    where $x_i[j]$ is the j-th bit of the string x_i. Let $c = \{c_j\}_{j \in n}$.

[4]The commitment is computed with fixed randomness, without loss of generality we fix this to be 0.

– For each $i \in [T], j \in [n]$, generate the opening $\rho_{i,j} \leftarrow \mathsf{SECOM.Open}(K, (x_1[j], \ldots, x_T[j]), i)$.

– Generate batch argument with witness $(\{\rho_{i,j}\}_{j\in[n]}, x_i, \omega_i)$,

$$\Pi' \leftarrow \mathsf{BARG'.P}\left(\mathsf{crs}', C'_{[K,c,C]}, \{\{\rho_{i,j}\}_{j\in[n]}, x_i, \omega_i\}_{i\in[T]}\right).$$

– Output $\Pi = \Pi'$.

- $\underline{\mathsf{V}(\mathsf{crs}, C, x_1, x_2, \ldots, x_T, \Pi)}$: The verifier parses $\Pi = \Pi'$ and $\mathsf{crs} = (K, \mathsf{crs}')$. We construct the following pre-processing and online verification algorithms.

  – **Pre-processing** $\mathsf{PreVerify}(\mathsf{crs}, x_1, x_2, \ldots, x_T)$**:** Commit the instances by

  $$\forall j \in [n], c_j \coloneqq \mathsf{SECOM.Com}(K, x_1[j], x_2[j], \ldots, x_T[j]; 0).$$

  Output the short sketch $c = \{c_j\}_{j\in[n]}$.

  – **Online Verification** $\mathsf{OnlineVerify}(\mathsf{crs}, c, C, \Pi')$**:**
  Output $\mathsf{BARG'.V}(\mathsf{crs}', C'_{[K,c,C]}, \Pi')$.

The succinct communication property follows directly from the succinct communication property of the BARGs for the index language. The succinct verification property follows from the construction. The compact CRS property is also satisfied, since the underlying BARGs for index language satisfy compact CRS property, and $|K|$ is also bounded by $\mathsf{poly}(\lambda, \log T)$ from the succinct CRS property of somewhere extractable commitment. The CRS indistinguishability proprty follows from the key indistinguishability of somewhere extractable commitment and the CRS indistinguishability of $\mathsf{BARG'}$.

To prove semi-adaptive somewhere soundness, for any adversary $\mathcal{A}$ for the semi-adaptive somewhere soundness of BARG, we construct a new adversary $\mathcal{A}'$ for $\mathsf{BARG'}$, as follows: $\mathcal{A}'$ first invokes $\mathcal{A}$, and obtain a index i^* which it outputs directly. Then, $\mathcal{A}'$ receives a CRS crs', generates a trapdoor commitment key K^* which is extractable

at the i^*-th coordinate, and feeds $\mathsf{crs} = (K = K^*, \mathsf{crs}')$ to $\mathcal{A}$. Next, $\mathcal{A}$ outputs $(C, x_1, x_2, \ldots, x_T, \Pi)$. $\mathcal{A}'$ computes $c = \mathsf{PreVerify}(\mathsf{crs}, x_1, x_2, \ldots, x_T)$, and outputs $(C'_{[K,c,C]}, \Pi)$. The adversary $\mathcal{A}'$ simulates the environment for the adversary $\mathcal{A}$. Furthermore, if $(C, x_{i^*}) \notin \mathsf{SAT}$, then by the extraction correctness of the commitment, we have $(C', i^*) \notin \mathcal{L}^{\mathsf{idx}}$. Hence, if the attack of $\mathcal{A}$ succeeds, then $\mathcal{A}'$ also succeeds. Since the underlying BARG' is somewhere sound, we prove the semi-adaptive somewhere soundness of BARG. □

4.3 SNARGs for $\mathcal{P}$

We follow the notions of delegation, for both Turing Machines and RAM, as defined in [43] who further show that their notion of RAM delegation implies Turing Machine delegation. This allows us to focus on constructing RAM delegation schemes for the rest of the paper.

4.3.1 Turing Machine Delegation

Consider a Turing machine $\mathcal{M}$. A publicly verifiable non-interactive delegation scheme for $\mathcal{M}$ consists of the following polynomial time algorithms:

- $\mathsf{Del.S}$ - randomized setup algorithm that on input security parameter 1^λ, time bound T and input length n outputs a pair of public keys - prover key pk and verifier key vk.
- $\mathsf{Del.P}$ - deterministic prover algorithm that on input prover key pk and an input $x \in \{0,1\}^n$ outputs a proof Π.
- $\mathsf{Del.V}$ - deterministic verifier algorithm that on input verifier key pk, input $x \in \{0,1\}^n$ and proof Π outputs either 0 or 1.

For any Turing machine $\mathcal{M}$, we define the corresponding language $\mathcal{U}_\mathcal{M}$ below,

$$\mathcal{U}_\mathcal{M} \coloneqq \Big\{(x, T) \;\Big|\; \mathcal{M} \text{ accepts } x \text{ within } T \text{ steps}\Big\}$$

Definition 4.3.1. *A publicly verifiable non-interactive delegation scheme* $(\mathsf{Del.S}, \mathsf{Del.P}, \mathsf{Del.V})$ *for* $\mathcal{M}$ *with setup time* $T_{\mathsf{S}} = T_{\mathsf{S}}(\lambda, T)$ *and proof length* $L_\Pi = L_\Pi(\lambda, T)$.

Completeness. *For every* $\lambda, T, n \in \mathbb{N}$ *such that* $n \leq T \leq 2^\lambda$, *and* $x \in \{0,1\}^n$ *such that* $(x, T) \in \mathcal{U}_\mathcal{M}$,

$$\Pr\left[\mathsf{Del.V}(\mathsf{vk}, x, \Pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Del.S}(1^\lambda, T, n) \\ \Pi \coloneqq \mathsf{Del.P}(\mathsf{pk}, x) \end{array} \right] = 1$$

Efficiency. *In the completeness experiment above,*

- $\mathsf{Del.S}$ *runs in time* T_{S}.
- $\mathsf{Del.P}$ *runs in time* $\mathsf{poly}(\lambda, T)$ *and outputs a proof of length* L_Π.
- $\mathsf{Del.V}$ *runs in time* $O(L_\Pi) + n \cdot \mathsf{poly}(\lambda)$.

Soundness. *For every* PPT *adversary* $\mathcal{A}$ *and pair of polynomials* $T = T(\lambda)$ *and* $n = n(\lambda)$ *there exists a negligible function* $\mathsf{negl}(\mathrm{cot})$ *such that for every* $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{Del.V}(\mathsf{vk}, x, \Pi) = 1 \\ (x, T) \notin \mathcal{U}_\mathcal{M} \end{array} \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Del.S}(1^\lambda, T, n) \\ (x, \Pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

4.3.2 RAM Delegation

A RAM machine of word size ℓ is modeled as a deterministic machine with random access to memory of size 2^ℓ where the local state of the machine has size only $O(\ell)$. At each time step, the machine updates its local state by either reading or writing a single memory. At any given time, the memory and the local state together represent the configuration cf of the machine. For simplicity, we assume that the machine has no input outside of its local state and memory, and the word size ℓ will correspond to the security parameter λ.

A publicly verifiable non-interactive delegation scheme for $\mathcal{R}$ consists of the following polynomial time algorithms:

$\mathsf{RDel.S}$ - randomized setup algorithm that on input security parameter 1^λ, time bound T outputs a triple of public keys - prover key pk, verifier key vk and a

digest key dk.

$\mathsf{RDel.D}$ - deterministic digest algorithm that on input digest key dk and configuration cf outputs a digest h.

$\mathsf{RDel.P}$ - deterministic prover algorithm that on input prover key pk and a pair of source and destination configurations $\mathsf{cf}, \mathsf{cf}'$ outputs a proof Π.

$\mathsf{RDel.V}$ - deterministic verifier algorithm that on verifier key pk, pair of digests h, h' and proof Π outputs either 0 or 1.

For any machine $\mathcal{R}$, we define the corresponding language $\mathcal{U}_{\mathcal{R}}$ below,

$$\mathcal{U}_{\mathcal{R}} \coloneqq \left\{ (\ell, \mathsf{cf}, \mathsf{cf}', T) \;\middle|\; \mathcal{R} \text{ with word size } \ell \text{ transitions from } \mathsf{cf} \text{ to } \mathsf{cf}' \text{ in } T \text{ steps} \right\}$$

Definition 4.3.2. *A publicly verifiable non-interactive delegation scheme* $(\mathsf{RDel.S}, \mathsf{RDel.D}, \mathsf{RDel.P}, R\mathsf{Del.V})$ *for* $\mathcal{R}$ *with setup time* $T_{\mathsf{S}} = T_{\mathsf{S}}(\lambda, T)$ *and proof length* $L_{\Pi} = L_{\Pi}(\lambda, T)$.

Completeness. *For every* $\lambda, T \in \mathbb{N}$ *such that* $n \leq T \leq 2^{\lambda}$, *and* $\mathsf{cf}, \mathsf{cf}' \in \{0,1\}^*$ *such that* $(\lambda, \mathsf{cf}, \mathsf{cf}', T) \in \mathcal{U}_{\mathcal{R}}$,

$$\Pr\left[\mathsf{RDel.V}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \Pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^{\lambda},) \\ \mathsf{h} \coloneqq \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}) \\ \mathsf{h}' \coloneqq \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}') \\ \Pi \coloneqq \mathsf{Del.P}(\mathsf{pk}, \mathsf{cf}, \mathsf{cf}') \end{array} \right] = 1$$

Efficiency. *In the completeness experiment above,*

- $\mathsf{RDel.S}$ *runs in time* T_{S}.
- $\mathsf{RDel.D}$ *on input* cf *runs in time* $|\mathsf{cf}| \cdot \mathsf{poly}(\lambda)$ *and outputs a digest of length* λ.
- $\mathsf{RDel.P}$ *runs in time* $\mathsf{poly}(\lambda, T, |\mathsf{cf}|)$ *and output a proof of length* L_{Π}.
- $\mathsf{RDel.V}$ *runs in time* $O(L_{\Pi}) + \mathsf{poly}(\lambda)$.

Collision resistance. *For every* PPT *adversary* $\mathcal{A}$ *and pair of polynomials* $T = T(\lambda)$*there exists a negligible function* $\mathsf{negl}(\mathrm{cot})$ *such that for every* $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{cf} \neq \mathsf{cf}' \\ \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}) = \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}') \end{array} \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^{\lambda}, T, n) \\ (\mathsf{cf}, \mathsf{cf}') \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

Soundness. *For every* PPT *adversary* $\mathcal{A}$ *and pair of polynomials* $T = T(\lambda)$*there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{RDel.V}(\mathsf{vk},\mathsf{h},\mathsf{h}',\Pi)=1 & (\mathsf{pk},\mathsf{vk},\mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^\lambda,T,n) \\ (\lambda,\mathsf{cf},\mathsf{cf}',T)\in\mathcal{U}_{\mathcal{R}} & (\mathsf{cf},\mathsf{cf}',\mathsf{h},\mathsf{h}',\Pi) \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{vk},\mathsf{dk}) \\ \mathsf{h}=\mathsf{RDel.D}(\mathsf{dk},\mathsf{cf}) & \\ \mathsf{h}'\neq\mathsf{RDel.D}(\mathsf{dk},\mathsf{cf}') & \end{array}\right] \leq \mathsf{negl}(\lambda)$$

As discussed in [43], the notion of RAM delegation considered in their work is different from those in prior works [38, 39] - namely that in prior works the adversary was not required to output the full configuration explicitly, only that it was difficult to produce accepting proofs for two different statements (h,h') and $(\mathsf{h},\mathsf{h}'')$ that share the same initial digest. We refer the reader to [43] for a more detailed comparison of the notions.

The following theorem establishes that RAM delegation implies Turing machine delegation for the definitions described above.

Theorem 4.3.3 ([43])**.** *Suppose that for any RAM machine there exists a publicly verifiable non-interactive delegation scheme with setup time* T'_{S} *and proof length* L'_{Π}*. Then for any Turing machine there exists a publicly verifiable non-interactive delegation scheme with setup time* T_{S} *and proof length* L_{Π} *where* $T_{\mathsf{S}}(\lambda,T) = T'_{\mathsf{S}}(\lambda,T')$*,* $L_{\Pi}(\lambda,T) = L'_{\Pi}(\lambda,T')$ *for* $T' = O(T)$*.*

4.3.3 Hash Tree

For going beyond space bounded computation, we recall the definition of hash trees as defined in [43].

A *hash tree* consists of the following algorithms:

$\mathsf{HT.Gen}$ - randomized algorithm that on input the security parameter 1^λ outputs a hash key dk

$\mathsf{HT.Hash}$ - deterministic algorithm that on input the hash key dk and string $D \in \{0,1\}^L$ outputs a hash tree tree and a root rt.

$\mathsf{HT.Read}$ - deterministic algorithm that on input hash tree tree and memory location ℓ outputs a bit b along with a proof Π.

$\mathsf{HT.Write}$ - deterministic algorithm that on input hash tree tree, memory location ℓ and bit b outputs a new tree tree', a new root rt' along with a proof Π.

$\mathsf{HT.VerRead}$ - deterministic algorithm on input hash key dk, root rt, memory location ℓ, bit b and proof Π outputs either 0 or 1.

$\mathsf{HT.VerWrite}$ - deterministic algorithm on input hash key dk, root rt, memory location ℓ, bit b, new root rt' and proof Π outputs either 0 or 1.

Definition 4.3.4 (Hash Tree)**.** *A hash tree scheme* ($\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Read}, \mathsf{HT.Write}, \mathsf{HT.VerRead}, \mathsf{HT.VerWrite}$) *satisfies the following properties:*

Completeness. *For every* $\lambda \in \mathbb{N}, D \in \{0,1\}^L$ *for* $L \leq 2^\lambda$ *and* $\ell \in [L]$*:*

$$\Pr\left[\begin{array}{l|l} \mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}, \ell, b, \Pi) = 1 \\ D[\ell] = b \end{array} \begin{array}{l} \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda) \\ (\mathsf{tree}, \mathsf{rt}) \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, D) \\ (b, \Pi) \coloneqq \mathsf{HT.Read}(\mathsf{tree}, \ell) \end{array}\right] = 1$$

Efficiency. *In the completeness experiment, the running time of* $\mathsf{HT.Hash}$ *is* $|D| \cdot \mathsf{poly}(\lambda)$*. The length of the root* rt*, and proofs produced by* $\mathsf{HT.Read}$ *and* $\mathsf{HT.Write}$ *are* $\mathsf{poly}(\lambda)$*.*

Soundness of Read. *For every polynomial size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{l|l} b_1 \neq b_2 \\ \mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}, \ell, b_1, \Pi_1) = 1 \\ \mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}, \ell, b_2, \Pi_2) = 1 \end{array} \begin{array}{l} \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda) \\ (\mathsf{rt}, \ell, b_1, \Pi_1, b_2, \Pi_2) \leftarrow \mathcal{A}(\mathsf{dk}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Soundness of Write. *For every polynomial size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{rt}_1 \neq \mathsf{rt}_2 \\ \mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}, \ell, b, \mathsf{rt}_1, \Pi_1) = 1 \\ \mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}, \ell, b, \mathsf{rt}_2, \Pi_2) = 1 \end{array} \begin{array}{l} \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda) \\ (\mathsf{rt}, \ell, b, \mathsf{rt}_1, \Pi_1, \mathsf{rt}_2, \Pi_2) \leftarrow \mathcal{A}(\mathsf{dk}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Theorem 4.3.5 ([97])**.** *From any family of collision resistant hash functions, one can construct a hash tree scheme.*

4.3.4 Protocol

The protocol follows the construction of the base case for RAM delegation in [43]. The crucial difference in our setting is that we are able to reduce the computation to an instance of a non-interactive BARG for index languages with a short CRS. The benefit of this is that we no longer have to do the bootstrapping since the CRS is already small. Our proof also closely follows the proof structure of their construction, although the no-signaling properties used in our proof are derived from the commitment rather than the underlying argument scheme.

RAM machine steps to circuit satisfiability. We use the translation from a *single* step of the machine $\mathcal{R}$ as described in [43]. Without loss of generality, assume that every step of $\mathcal{R}$ consists of a *single read operation, followed by a single write operation*. Therefore, a single step can be decomposed into the following *deterministic* polynomial time algorithms:

StepR**:** On input the local state st of $\mathcal{R}$, outputs the memory location ℓ that $\mathcal{R}$ while in state st would read from.

StepW**:** On input the local state st and bit b, outputs a bit b', memory location ℓ' and state st' such that $\mathcal{R}$ while in state st on reading bit b would write b' to location ℓ' and then transition to new local state st'.

We denote by φ the circuit representing a single step of $\mathcal{R}$, i.e. given a pair of digests $\mathsf{h} = (\mathsf{st}, \mathsf{rt})$, $\mathsf{h}' = (\mathsf{st}', \mathsf{rt}')$, bit b and proof Π, Π' there exists an efficiently computable w (given $(\mathsf{h}, \mathsf{h}', b, \Pi, \Pi')$) such that $\varphi(\mathsf{h}, \mathsf{h}', b, \Pi, \Pi', w) = 1$ *if and only if*

$$\ell = \mathsf{StepR}(\mathsf{st})$$

$$(b', \ell', \mathsf{st}'') = \mathsf{StepW}(\mathsf{st}, b)$$

$$\mathsf{st}' = \mathsf{st}''$$

$$\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}, \ell, b, \Pi) = 1$$

$$\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}, \ell', b', \mathsf{rt}', \Pi') = 1$$

From the efficiency of the hash tree scheme, there exists a φ such that the above can be represented as a formula of $L = \mathsf{poly}(\lambda)$ variables.

We will use φ_i to denote the i-th step in the above formula ϕ. Note that the subscript will be helpful in our discussion of security, but the circuits themselves are identical for all i.

For T steps of $\mathcal{R}$, we then have the following formula ϕ over $M := O(L \cdot T)$ variables:

$$\phi\left(\mathsf{h}_0, \{\mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i\}_{i\in[T]}\right) := \bigwedge_{i\in[T]} \varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i)$$

Note that the above formula is *not an index language*. This is because for all i, φ_i and φ_{i+1} share a part of the witness, something not handled by the index language since we would have to ensure that the (partial) witness is the *same*. As described in the technical overview, we handle this by using a $\mathsf{NS\text{-}SECOM}$ to commit to the witnesses, and then prove for each i that values in the commitment satisfy the clause ϕ_i. The no-signaling property will help ensure consistency of the shared witness across different clauses φ_i and φ_{i+1}.

The components we require for our delegation scheme are listed below:

- An L-no-signaling-SECOM commitment scheme (Definition 2.4.9) $\mathsf{NS\text{-}SECOM} = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{Com}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Ext})$.
- A non interactive batch argument (Section 4.2.1) for an index language $(\mathsf{BARG.Gen}, \mathsf{BARG.TGen}, \mathsf{BARG.P}, \mathsf{BARG.V})$ (Section 4.2.3).

We present our RAM delegation scheme in Figure 4-5.

[5] We overload notation here to specify opening to many bits.

[6] Since these are not a single bit, we overload notation and skip the index i that is passed to the Verify algorithm.

Theorem 4.3.6. *Assuming the hardness of the Learning with Errors (*LWE*) (Definition 2.1.3), for every polynomial* $T = T(\lambda)$*, the protocol in Figure 4-5 is a publicly-verifiable non-interactive RAM delegation scheme (Definition 4.3.2) with CRS size, proof size and verifier time all* $\mathsf{poly}(\lambda, \log T)$ *while the prover running time is* $\mathsf{poly}(\lambda, T)$*.*

The assumptions required for our construction follow from the assumptions of the underlying primitives. We focus on proving the efficiency and security of our protocol below.

4.3.4.1 Efficiency

Before proving security, we prove that the language above is indeed an index language, and that the efficiency criteria for a RAM delegation scheme are satisfied.

Claim 4.3.7. *For all* $i \in [T]$*,* $(C_{\mathsf{index}}, i) \in \mathcal{L}$*.*

Proof. This just follows from the construction in Figure 4-6. The witness corresponds to the inputs as in the delegation protocol. □

Claim 4.3.8. $|C_{\mathsf{index}}| = \mathsf{poly}(\lambda, \log T)$

Proof. $|C_{\mathsf{index}}|$ consists of the following:

- The hardcoded commitment key K: $|K| = L \cdot \mathsf{poly}(\lambda, \log M) = \mathsf{poly}(\lambda, \log T)$.
- The hardcoded commitment c: $|c| = L \cdot \mathsf{poly}(\lambda, \log M) = \mathsf{poly}(\lambda, \log T)$ (same as key size).
- The hardcoded circuit φ: $|\varphi_i| = O(L) = \mathsf{poly}(\lambda)$.
- Size of openings as a part of the witness of size $L \cdot \mathsf{poly}(\lambda, \log M) = \mathsf{poly}(\lambda, \log T)$.
- Verifier circuit for commitment of size $\mathsf{poly}(\lambda, \log M) = \mathsf{poly}(\lambda, \log T)$.

□

CRS size. The CRS consists of the commitment key K, the BARG CRS crs and the digest key dk. By the corresponding properties of the underlying scheme we have: $|K| + |\mathsf{crs}| + |\mathsf{dk}| = \mathsf{poly}(\lambda, \log T) + \mathsf{poly}(\lambda, \log T, |C_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$.

Proof length. The proof consists of the commitment c and the BARG proof: $|c| + |\Pi| = \mathsf{poly}(\lambda, \log T) + \mathsf{poly}(\lambda, \log T, |C_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$

Verifier time: The verification time is the time taken to compute the circuit $|C_{\mathsf{index}}|$ and verify the BARG proof: $\mathsf{poly}(\lambda, \log T, |C_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$.

4.3.4.2 Security Proof

Let us assume for the sake of contradiction that the soundness of the above scheme does not hold. Then fix $\mathcal{A}$ and T such that there exists a polynomial $\mathsf{p}(\cdot)$ where, for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathsf{RDel.V}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \Pi) = 1 & (\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^\lambda, T, n) \\ (\lambda, \mathsf{cf}, \mathsf{cf}', T) \in \mathcal{U}_{\mathcal{R}} & (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \\ \mathsf{h} = \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}) & \\ \mathsf{h}' \neq \mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf}') & \end{array}\right] \geq \frac{1}{\mathsf{p}(\lambda)}. \tag{4.4}$$

We now use an averaging argument to fix a *bad* digest key dk. Specifically, a digest key dk is *bad* if Equation (4.4) holds with probability at least $1/2\mathsf{p}(\lambda)$ when dk is sampled by $\mathsf{RDel.S}$. By an averaging argument the fraction of such *bad* digest keys must be at least $1/2\mathsf{p}(\lambda)$. Therefore, conditioned on a fixed *bad* digest key dk^* we have,

$$\Pr\left[\begin{array}{l|l} \mathsf{RDel.V}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \Pi) = 1 & (\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^\lambda, T, n)|_{\mathsf{dk}=\mathsf{dk}^*} \\ (\lambda, \mathsf{cf}, \mathsf{cf}', T) \in \mathcal{U}_{\mathcal{R}} & (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}^*) \\ \mathsf{h} = \mathsf{RDel.D}(\mathsf{dk}^*, \mathsf{cf}) & \\ \mathsf{h}' \neq \mathsf{RDel.D}(\mathsf{dk}^*, \mathsf{cf}') & \end{array}\right] \geq \frac{1}{\mathsf{p}(\lambda)}. \tag{4.5}$$

where $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^\lambda, T, n)|_{\mathsf{dk}=\mathsf{dk}^*}$ is the setup algorithm $\mathsf{RDel.S}$ conditioned on the digest key output being dk^*. Moving forward, for ease of notation, the fixed bad dk^* will be denoted simply by dk.

Next, we change the setup algorithm in both the underlying commitment scheme ($\mathsf{NS\text{-}SECOM}$) and the batch argument (BARG) to be the trapdoor setup algorithm with the $\emptyset$ as the argument. We want to argue that the distribution of $\mathcal{A}$'s output $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}')$ does not change by more than a negligible probability. Specifically,

Claim 4.3.9. *For all* PPT *distinguisher* $\mathcal{D}$*,*

$$\left|\Pr\left[\begin{array}{l|l} \mathcal{D}(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}') = 1 & \begin{array}{l}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.S}(1^\lambda, T, n)|_{\mathsf{dk}} \\ (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})\end{array}\end{array}\right] - \Pr\left[\begin{array}{l|l} \mathcal{D}(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}') = 1 & \begin{array}{l}K \leftarrow \mathsf{TGen}(1^\lambda, 1^M, \emptyset) \\ \mathsf{crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^T, 1^{|C_{\mathsf{index}}|}, \emptyset) \\ \mathsf{pk} \coloneqq (K, \mathsf{crs}, \mathsf{dk}),\ \mathsf{vk} \coloneqq (K, \mathsf{crs}) \\ (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})\end{array}\end{array}\right]\right| \leq \mathsf{negl}(\lambda). \tag{4.6}$$

where in each experiment above $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}') = \perp$ *if* $\mathsf{RDel.V}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \Pi) = 0$.

Proof. Consider Hyb_0 to be the experiment

$$\begin{aligned}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) &\leftarrow \mathsf{RDel.S}(1^\lambda, T, n)|_{\mathsf{dk}} \\ (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) &\leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})\end{aligned}$$

with output $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}')$. Similarly, Hyb_1 is the experiment

$$\begin{aligned}K &\leftarrow \mathsf{TGen}(1^\lambda, 1^M, \emptyset) \\ \mathsf{crs} &\leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^T, 1^{|C_{\mathsf{index}}|}, \emptyset) \\ \mathsf{pk} &\coloneqq (K, \mathsf{crs}, \mathsf{dk}),\ \mathsf{vk} \coloneqq (K, \mathsf{crs}) \\ (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) &\leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})\end{aligned}$$

with output $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}')$. It then suffices to show that $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$. For this we introduce an intermediate hybrid Hyb' that has the same output, but the experiment is:

$$\begin{aligned}K &\leftarrow \mathsf{Gen}(1^\lambda, 1^M) \\ \mathsf{crs} &\leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^T, 1^{|C_{\mathsf{index}}|}, \emptyset)\end{aligned}$$

$$\mathsf{pk} := (K, \mathsf{crs}, \mathsf{dk}),\ \mathsf{vk} := (K, \mathsf{crs})$$

$$(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}).$$

Note that for all of the above experiments, set $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi)$ to $\perp$ if Π is not accepting.

We now prove the following:

$\mathsf{Hyb}_0 \approx \mathsf{Hyb}'$: We rely on the *key-indistinguishability* property of the BARG scheme. Specifically, if there exists a PPT distinguisher $\mathcal{D}$ that distinguishes Hyb_0 and Hyb', we can construct an adversary $\mathcal{B}$ (with a bad dk as an additional argument) as below,

$\underline{\mathcal{B}(1^\lambda, \mathsf{dk})}$:

1. Send $\emptyset$ to the BARG key indistinguishability.
2. On obtaining crs from the challenger, run the rest of the experiment.
3. Check if Π is accepting. If not, set $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi)$ to $\perp$.
4. Output $\mathcal{D}(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}')$.

Depending on the response received from the challenger, the experiment corresponds either to Hyb_0 or Hyb', and $\mathcal{B}$ succeeds if $\mathcal{D}$ succeeds.

$\mathsf{Hyb}' \approx \mathsf{Hyb}_1$: We rely on the *key-indistinguishability* property of the NS-SECOM scheme. The proof follows identically as in the above case.

This completes the proof. □

If $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}') \neq \perp$, i.e. the proof is accepting, then starting with configuration cf we define the "*true*" configuration $\overline{\mathsf{cf}}_i$ after i-steps of computation. Formally, for each $i \in [0, T]$, $\overline{\mathsf{cf}}_i$ is the *unique configuration* such that $(\lambda, \mathsf{cf}, \overline{\mathsf{cf}}_i, i) \in \mathcal{U}_{\mathcal{R}}$. The corresponding digest $\overline{\mathsf{h}}_i$ is defined to be $\mathsf{RDel.D}(\mathsf{dk}, \overline{\mathsf{cf}}_i)$. We will refer to such configurations (resp. digests) to be the "true" configuration (resp. digest).

We define below the event and experiment that will be relevant to the analysis.

CHEAT: The event that $\mathsf{BARG.Vf}(\mathsf{crs}, C_{\mathsf{index}}, \pi) = 1$, and $\mathsf{h} = \overline{\mathsf{h}}_0$ but $\mathsf{h}' \neq \overline{\mathsf{h}}_T$.

EXP_i: Let S_i denote the set of ϕ's variables that represent the variables for the i-th clause - $\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i$. EXP_i then corresponds to the experiment where the keys for the commitment scheme and the batch argument are generated with trapdoor for S_i and i respectively. More formally

$$K \leftarrow \mathsf{TGen}(1^\lambda, 1^M, S_i), \ \mathsf{crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^T, 1^{|C_{\mathsf{index}}|}, i)$$

If the proof π is non-accepting, the output of the experiment is $\perp$. Otherwise the output is $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}')$ along with the extracted value

We claim that in the above experiment, the adversary still cheats with an inverse polynomial probability.

Claim 4.3.10. *For all $i \in [T]$,*

$$\Pr_{\mathsf{EXP}_i}[\mathsf{CHEAT}] \geq \frac{1}{\mathsf{poly}(\lambda)} \tag{4.7}$$

Proof. This follows in an identical manner to Claim 4.3.9 from the key indistinguishability of the underlying schemes, and the fact that the event CHEAT is efficiently checkable. □

Claim 4.3.11. *For all $i \in [T]$,*

$$\Pr_{\mathsf{EXP}_i}\Big[\ \mathsf{CHEAT} \implies \varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 1\ \Big] \geq 1 - \mathsf{negl}(\lambda) \tag{4.8}$$

where $\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i$ is extracted from the commitment using the trapdoor.

Proof. Given that CHEAT holds, we know that the proof Π output by $\mathcal{A}$ is accepting. We first use the semi-adaptive somewhere soundness of the BARG to show that the $(C_{\mathsf{index}}, i) \in \mathcal{L}$. If not, we construct an adversary $\mathcal{B}$ as below

$\underline{\mathcal{B}(1^\lambda, \mathsf{dk})}$:

1. Send i to the BARG challenger and receive crs.

2. Compute the rest of the experiment EXP_i using crs.

3. Return $(C_{\mathsf{index}}, \Pi)$.

Thus, we have $(C_{\mathsf{index}}, i) \in \mathcal{L}$ which means that there is an accepting witness to C_{index} in Figure 4-6. Now, from the somewhere statistical binding property of the $\mathsf{NS\text{-}SECOM}$ for S_i, we have that other than with negligible probability the extracted values $(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i)$ (along with their opening proofs) are the only valid local openings to c on S_i. Therefore, since $(C_{\mathsf{index}}, i) \in \mathcal{L}$ and $(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i)$ is the only valid opening to c we have that other than with negligible probability, C_{index} for index i has a *unique* (partial) witness $(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i)$.

This in turn implies (by the construction of C_{index}), we have $\varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$. □

Therefore we have

$$\Pr_{\mathsf{EXP}_i}\left[\begin{array}{l}\mathsf{CHEAT} \wedge \\ \varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 0\end{array}\right] < \mathsf{negl}(\lambda) \tag{4.9}$$

In EXP_1, φ_1 is consistent with h, therefore $\mathsf{h} = \mathsf{h}_0$ which gives us,

$$\Pr_{\mathsf{EXP}_1}\left[\begin{array}{l}\mathsf{CHEAT} \wedge \\ \mathsf{h}_0 \neq \overline{\mathsf{h}}_0\end{array}\right] < \mathsf{negl}(\lambda) \tag{4.10}$$

In EXP_T, φ_T is consistent with h', therefore $\mathsf{h}' = \mathsf{h}_T$ which, because the adversary is cheating, gives us

$$\Pr_{\mathsf{EXP}_T}\left[\begin{array}{l}\mathsf{CHEAT} \wedge \\ \mathsf{h}_T = \overline{\mathsf{h}}_T\end{array}\right] < \mathsf{negl}(\lambda) \tag{4.11}$$

We now want to use the no-signaling property to claim that in both EXP_i and EXP_{i+1} it must be the case that the extracted h_i is the corresponding "true" digest at the i-step of the execution.

Claim 4.3.12. *For all* $i \in [T-1]$

$$\left|\Pr_{\mathsf{EXP}_i}\left[\begin{array}{l}\mathsf{CHEAT} \wedge \\ \mathsf{h}_i = \overline{\mathsf{h}}_i\end{array}\right] - \Pr_{\mathsf{EXP}_{i+1}}\left[\begin{array}{l}\mathsf{CHEAT} \wedge \\ \mathsf{h}_i = \overline{\mathsf{h}}_i\end{array}\right]\right| < \mathsf{negl}(\lambda) \tag{4.12}$$

Proof. We will prove this using a sequence of hybrids. Let Hyb_0 be the distribution in EXP_i with output $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \mathsf{h}_i)$ where h_i is extracted from the NS-SECOM commitment scheme using the trapdoor for S_i. Similarly, let Hyb_1 be the distribution in EXP_i with output $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \mathsf{h}_i)$ where h_i is extracted from the NS-SECOM commitment scheme using the trapdoor for S_{i+1}.

It suffices to show these two distributions are indistinguishable since given $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \mathsf{h}_i)$ one can compute $\overline{\mathsf{h}}_i$ and check if $\overline{\mathsf{h}}_i = \mathsf{h}_i$. If the probability of the check succeeded differed in the Hyb_0 and Hyb_1 by a non-negligible amount, we would have an efficient distinguisher.

We introduce an intermediate distribution Hyb' where the experiment is:

$$\begin{aligned} K &\leftarrow \mathsf{TGen}(1^\lambda, 1^M, S_{i+1}) \\ \mathsf{crs} &\leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^T, 1^{|C_{\mathsf{index}}|}, i) \\ \mathsf{pk} &:= (K, \mathsf{crs}, \mathsf{dk}),\ \mathsf{vk} := (K, \mathsf{crs}) \\ (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \Pi) &\leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}). \end{aligned}$$

with the output being $(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \mathsf{h}_i)$.

$\mathsf{Hyb}_0 \approx \mathsf{Hyb}'$**:** We rely on the *no-signaling* property of the commitment scheme NS-SECOM. Specifically, if there exists a PPT distinguisher $\mathcal{D}$ that distinguishes Hyb_0 and Hyb', we can construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ (with a bad dk as non-uniform advice) as below,

$\underline{\mathcal{B}_1(1^\lambda, \mathsf{dk})}$:

1. On input K, run the rest of the experiment with K as the NS-SECOM key.

2. Separate c from $\mathcal{A}$'s proof.
3. Set $z := (\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}')$ as the auxiliary input, setting z to $\perp$ if Π is not accepting.
4. Output (c, z).

$\underline{\mathcal{B}_2(1^\lambda, \mathsf{dk})}$:

1. On input K, c, z, h_i, output $\mathcal{D}(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \mathsf{h}_i)$.

Depending on the response received from the challenger, the experiment corresponds either to Hyb_0 or Hyb', and $\mathcal{B}$ succeeds if $\mathcal{D}$ succeeds.

$\mathsf{Hyb}' \approx \mathsf{Hyb}_1$: We rely on the *key-indistinguishability* property of the BARG scheme. Specifically, if there exists a PPT distinguisher $\mathcal{D}$ that distinguishes Hyb' and Hyb_0 we construct an adversary $\mathcal{B}$ (with a bad dk as non-uniform advice) as below,

$\underline{\mathcal{B}(1^\lambda, \mathsf{dk})}$:

1. Send $i, i+1$ to the BARG key indistinguishability challenger.
2. On receiving crs run the rest of the experiment using received crs.
3. Use the NS-SECOM to extract h_i.
4. Output $\mathcal{D}(\mathsf{cf}, \mathsf{cf}', \mathsf{h}, \mathsf{h}', \mathsf{h}_i)$.

Note that we have directly reduced to the key indistinguishability of the BARG between keys generated for i and $i+1$, while our security definition only states that it is computationally intractable to differentiate between the key generated in the normal mode, and the key generated for an index i. It is easy to see that our definition also implies key indistinguishability on any two inputs i and j (with the normal mode as the intermediate hybrid experiment).

This completes the proof. $\square$

From the above equations, given that $\mathsf{h}_T \neq \overline{\mathsf{h}}_T$ (other than with negligible probability), it must be the case that there exists an i^* such that the input digest is "true", while the output digest is not "true". Formally, from equations (4.7) to (4.12), we have that there exists $i^* \in [T]$ such that

$$\Pr_{\mathsf{EXP}_{i^*}}\left[\begin{array}{l}\mathsf{CHEAT} \wedge \\ \mathsf{h}_{i^*-1} = \overline{\mathsf{h}}_{i^*-1} \wedge \\ \mathsf{h}_{i^*} \neq \overline{\mathsf{h}}_{i^*} \wedge \\ \varphi_i(\mathsf{h}_{i^*-1}, \mathsf{h}_{i^*}, b_{i^*}, \Pi_{i^*}, \Pi'_{i^*}, w_{i^*}) = 1\end{array}\right] \geq \frac{1}{T \cdot \mathsf{poly}(\lambda)} = \frac{1}{\mathsf{poly}(\lambda)} \tag{4.13}$$

Let $i \coloneqq i^*$ as described above. Then for $(\mathsf{st}_{i-1}, \mathsf{rt}_{i-1}) \coloneqq \mathsf{h}_{i-1}$,

$$\ell_i \coloneqq \mathsf{StepR}(\mathsf{st}_{i-1})$$

$$(b'_i, \ell'_i, \mathsf{st}_i) \coloneqq \mathsf{StepW}(\mathsf{st}_{i-1}, b_i)$$

Next, compute the corresponding "true" variables starting from $(\overline{\mathsf{st}}_{i-1}, \overline{D}_{i-1}) \coloneqq \overline{\mathsf{cf}}_{i-1}$:

$$(\overline{\mathsf{tree}}_i, \overline{\mathsf{rt}}_i) \coloneqq \mathsf{HT.Hash}(\mathsf{dk}, \overline{D}_{i-1})$$

$$\overline{\ell}_i \coloneqq \mathsf{StepR}(\overline{\mathsf{st}}_{i-1})$$

$$(\overline{b}_i, \overline{\Pi}_i) \coloneqq \mathsf{HT.Read}(\overline{\mathsf{tree}}_{i-1}, \overline{\ell}_i)$$

$$(\overline{b}'_i, \overline{\ell}'_i, \overline{\mathsf{st}}_i) \coloneqq \mathsf{StepW}(\overline{\mathsf{st}}_{i-1}, \overline{b}_i)$$

$$(\overline{\mathsf{tree}}_i, \overline{\mathsf{rt}}_i, \overline{\Pi}'_i) \coloneqq \mathsf{HT.Read}(\overline{\mathsf{tree}}_{i-1}, \overline{\ell}'_i, \overline{b}'_i)$$

We describe two events below.

CHEAT_1**:** the event that the following is true

$$b_i \neq \overline{b}_i$$

$$\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_{i-1}, \ell_i, b_i, \Pi_i) = 1$$

$$\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_{i-1}, \ell_i, \overline{b}_i, \overline{\Pi}_i) = 1$$

CHEAT_2 the event that the following is true

$$\mathsf{rt}_i \neq \overline{\mathsf{rt}}_i$$
$$\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}_{i-1}, \ell'_i, b'_i, \mathsf{rt}'_i, \Pi'_i) = 1$$
$$\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}_{i-1}, \ell'_i, b'_i, \overline{\mathsf{rt}}'_i, \overline{\Pi}'_i) = 1$$

As observed in [43], the following claim establishes that by the completeness of the hash tree scheme, either the read or the write operation. We have reproduced the proof here for completeness.

Claim 4.3.13.

$$\Pr_{\mathsf{EXP}_i}\left[\begin{array}{l}\mathsf{CHEAT} \wedge \\ \mathsf{h}_{i-1} = \overline{\mathsf{h}}_{i-1} \wedge \\ \mathsf{h}_i \neq \overline{\mathsf{h}}_i \wedge \\ \varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 1\end{array}\right] \implies \mathsf{CHEAT}_1 \vee \mathsf{CHEAT}_2 \tag{4.14}$$

Proof. This follows from the completeness of the hash tree scheme. Specifically, by definition, we have that $\overline{\mathsf{h}}_{i-1} = (\overline{\mathsf{st}}_{i-1}, \overline{\mathsf{rt}}_{i-1})$ and $\overline{\mathsf{h}}_i = (\overline{\mathsf{st}}_i, \overline{\mathsf{rt}}_i)$. By the completeness of the hash tree scheme we have,

$$\mathsf{HT.VerRead}(\mathsf{dk}, \overline{\mathsf{rt}}_{i-1}, \overline{\ell}_i, \overline{b}_i, \overline{\Pi}_i) = 1$$
$$\mathsf{HT.VerWrite}(\mathsf{dk}, \overline{\mathsf{rt}}_{i-1}, \overline{\ell}'_i, \overline{b}'_i, \overline{\mathsf{rt}}'_i, \overline{\Pi}'_i) = 1$$

Now from the claim, we have that $\mathsf{h}_{i-1} = \overline{\mathsf{h}}_{i-1}$, which means that $(\mathsf{st}_{i-1}, \mathsf{rt}_{i-1}) = (\overline{\mathsf{st}}_{i-1}, \overline{\mathsf{rt}}_{i-1})$. Since StepR is a deterministic algorithm, we also have $\ell_i = \overline{\ell}_i$. Finally, since $\varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$ we have,

$$\mathsf{h}_i = (\mathsf{st}_i, \mathsf{rt}_i)$$
$$\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_{i-1}, \ell_i, b_i, \Pi_i) = 1$$
$$\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}_{i-1}, \ell'_i, b'_i, \mathsf{rt}'_i, \Pi'_i) = 1$$

We consider two cases below to complete the proof. Note that in both the cases, the required proofs from the events verify, so we omit them in the discussion below.

Case $b_i \neq \overline{b}_i$**:** CHEAT_1 holds.

Case $b_i \neq \overline{b}_i$**:** Then, because StepW is deterministic, we have that $(\ell'_i, b'_i, \mathsf{st}_i) = (\overline{\ell}'_i, \overline{b}'_i, \overline{\mathsf{st}}_i)$. Combining this with the fact that $\mathsf{h}_i \neq \overline{\mathsf{h}}_i$, it must be the case that $\mathsf{rt}_i \neq \overline{\mathsf{rt}}_i$. This ensures that CHEAT_2 holds in this case.

□

From the above claims, since $\mathsf{CHEAT}_1 \vee \mathsf{CHEAT}_2$ happens with probability $1/\mathsf{poly}(\lambda)$. We can now construct an adversary $\mathcal{B}$ for the hash tree scheme that breaks either the soundness of read (when CHEAT_1 holds) or the soundness of write (when CHEAT_2 holds). Specifically,

$\underline{\mathcal{B}(1^\lambda)}$:

1. Receive dk from the HT challenger.
2. Runs EXP_i (from the above claim either CHEAT_1 or CHEAT_2 hold).
3. If CHEAT_1 holds, output $(\mathsf{rt}_{i-1}, \ell_i, b_i, \Pi_i, \overline{b}_i, \overline{\Pi}_i)$.
4. Else, if CHEAT_2 holds, output $(\mathsf{dk}, \mathsf{rt}_{i-1}, \ell'_i, b'_i, \mathsf{rt}'_i, \overline{\mathsf{rt}}'_i, \overline{\Pi}'_i)$

Correctness follows from the description of the events. Since we have that the received dk is *bad* with probability $1/2\mathsf{p}(\lambda)$, $\mathcal{B}$ successfully breaks the soundness of the hash tree scheme. This completes the security proof.

4.4 Application

In this section, we show how to use batch arguments for $\mathcal{NP}$ and the Turing machine delegation scheme to build more efficient batch arguments for $\mathcal{NP}$ by removing the dependence on the circuit size. The idea is to delegate the verification of an $\mathcal{NP}$

instance to the Turing machine, and then use the batch argument to prove that the verifier in the delegation scheme will accept the delegation proof for all instances.

In more detail, we use batch arguments for SAT and a Turing machine delegation scheme to construct BARGs for the following $\mathcal{NP}$ language

$$\mathcal{L} = \{x \mid \exists \omega : \mathcal{M}(x, \omega) \text{ outputs 1 (accepts) in } T \text{ steps}\},$$

where $\mathcal{M}$ is a Turing machine. The proof size and CRS size are polynomial in $\lambda, |w|$, and $\log T$.

Theorem 4.4.1. *Let* $\mathsf{Del} = (\mathsf{Del.S}, \mathsf{Del.P}, \mathsf{Del.V})$ *be a Turing machine delegation scheme, and* $\mathsf{BARG}' = (\mathsf{BARG}'.\mathsf{Gen}, \mathsf{BARG}'.\mathsf{TGen}, \mathsf{BARG}'.\mathsf{P}, \mathsf{BARG}'.\mathsf{V})$ *be a batch argument for* SAT *with efficient online verification property. Then we can construct a batch argument* $\mathsf{BARG} = (\mathsf{Gen}, \mathsf{TGen}, \mathsf{P}, \mathsf{V})$ *with efficient online verification property for* $\mathcal{L}$ *with the following efficiency. Let* $n = |x|$ *be the length of the instances,* $m = |\omega|$ *be the length of the witnesses, and* k *be the number of instances.*

- **CRS size:** *The size of the CRS is* $\mathsf{poly}(\lambda, n, m, \log k, \log T)$.
- **Proof size:** *The size of the argument is* $\mathsf{poly}(\lambda, n, m, \log k, \log T)$.
- **Efficient Online Verification:** *The running time of the offline preprocessing is* $\mathsf{poly}(\lambda, k, n)$*, and the online verification algorithm runs in* $\mathsf{poly}(\lambda, n, m, \log k, \log T)$.

Proof. We build the BARG as follows. We will use a circuit $\mathsf{DelVerify}$ in Figure 4-7.

- $\underline{\mathsf{Gen}(1^\lambda, 1^k, 1^T)}$: Generate a CRS for the Turing machine delegation scheme, and a CRS for the batch argument for $\mathsf{C\text{-}SAT}$.
  - Let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Del.S}(1^\lambda, T)$, and $\mathsf{crs}' \leftarrow \mathsf{BARG}'.\mathsf{Gen}(1^\lambda, 1^k, 1^{|\mathsf{DelVerify}|})$.
  - Output $\mathsf{crs} = ((\mathsf{pk}, \mathsf{vk}), \mathsf{crs}')$.

- $\underline{\mathsf{TGen}(1^\lambda, 1^k, 1^T, i^*)}$: The trapdoor CRS generation algorithm generates $(\mathsf{pk}, \mathsf{vk})$ in the same way as Gen, and generates a trapdoor CRS for BARG'.

  - Let $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Del.S}(1^\lambda, T)$, and $\mathsf{crs}^{*\prime} \leftarrow \mathsf{BARG}'.\mathsf{Gen}(1^\lambda, 1^k, 1^{|\mathsf{DelVerify}|}, i^*)$.
  - Output $\mathsf{crs}^* = ((\mathsf{pk}, \mathsf{vk}), \mathsf{crs}^{*\prime})$.

- $\underline{\mathsf{P}(\mathsf{crs}, x_1, \ldots, x_k, \omega_1, \ldots, \omega_k)}$: The prover first delegates each instance using Turing machine delegation, and then uses batch arguments to prove all the delegated instances verify.

  Parse $\mathsf{crs} = ((\mathsf{pk}, \mathsf{vk}), \mathsf{crs}')$, where $(\mathsf{pk}, \mathsf{vk})$ is the CRS for Turing machine delegation, and crs' is the CRS for the batch argument.

  - Delegate the computation of $\mathcal{M}$ using Turing machine delegation.

    $$\forall i \in [k], \Pi_i \leftarrow \mathsf{Del.P}(\mathsf{pk}, x_i, \omega_i).$$

  - Use batch arguments to generate the poof.

    $$\Pi' \leftarrow \mathsf{BARG}'.\mathsf{P}(\mathsf{crs}', \mathsf{DelVerify}_{[(\mathsf{pk},\mathsf{vk})]}, x_1, \ldots, x_k, \{(\omega_i, \Pi_i)\}_{i\in[k]}).$$

  - Output $\Pi = \Pi'$.

- $\underline{\mathsf{V}(\mathsf{crs}, x_1, \ldots, x_k, \Pi)}$: The verifier parses $\Pi = \Pi'$, and verifies

  $$\mathsf{BARG}'.\mathsf{V}(\mathsf{crs}', \mathsf{DelVerify}, x_1, \ldots, x_k, \Pi') = 1.$$

The CRS size is $|(\mathsf{pk}, \mathsf{vk})| + |\mathsf{crs}'|$. Since the delegation scheme has CRS size $\mathsf{poly}(\lambda, \log T)$, and crs' is bounded by $\mathsf{poly}(\lambda, \log k, |\mathsf{DelVerify}|) = \mathsf{poly}(\lambda, \log k, \mathsf{poly}(\lambda, \log k, n, \log T, m))$, the CRS size is bounded by $\mathsf{poly}(\lambda, \log T, n, m)$. The proof size is the same as the proof size of BARG', where the $\mathsf{DelVerify}$ size $|x|+|\omega|+|\Pi|+\mathsf{poly}(\lambda, \log T) = \mathsf{poly}(\lambda, n, m, \log T)$. Hence, the proof size is bounded by $\mathsf{poly}(\lambda, \log k, |\mathsf{DelVerify}|) = \mathsf{poly}(\lambda, n, m, \log k, \log T)$. Finally, since the verification algorithm is the same as the

verification of BARG'. The efficient online verification property follows from the same property of the underlying BARG'.

The completeness follows from the completeness of the delegation scheme Del and the batch argument BARG'. The CRS indistinguishability follows the same property of BARG'.

To prove the semi-adaptive somewhere soundness, we require the somewhere argument of knowledge property of the underlying batch argument BARG', which allows us extraction a witness for the i^*-th instance from an accepting proof (See Remark 4.2.13).

Given somewhere argument of knowledge property, we can extract the witness $(\omega_{i^*}, \Pi'_{i^*})$ for the i^*-th instance from any cheating prover for BARG, and output Π'_{i^*} as the attacking proof for the underlying delegation scheme Del, with input (x_{i^*}, ω_{i^*}). Since the delegation scheme is sound, this proves the soundness of our batch argument construction. □

RAM Delegation

$\underline{\mathsf{RDel.S}(1^\lambda, T)}$**:** Generate the public parameters for the underlying primitives

$$K \leftarrow \mathsf{Gen}(1^\lambda, 1^M, 1^L),\ \mathsf{crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^T, 1^{|C_{\mathsf{index}}|}),\ \mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda).$$

Ouput $(\mathsf{pk} := (K, \mathsf{crs}, \mathsf{dk}),\ \mathsf{vk} := (K, \mathsf{crs}),\ \mathsf{dk})$.

$\underline{\mathsf{RDel.D}(\mathsf{dk}, \mathsf{cf} = (\mathsf{st}, D))}$**:** Compute the hash tree,

$$(\mathsf{tree}, \mathsf{rt}) := \mathsf{HT.Hash}(\mathsf{dk}, D)$$

Output $\mathsf{h} := (\mathsf{st}, \mathsf{rt})$.

$\underline{\mathsf{RDel.P}((\mathsf{pk}, \mathsf{dk}), \mathsf{cf}, \mathsf{cf}')}$**:** Prover emulates $\mathcal{R}$ for T steps from cf to cf' to obtain the satisfying assignment for ϕ as follows: define

$$(\mathsf{st}_0, D_0) := \mathsf{cf}, \quad (\mathsf{tree}_0, \mathsf{rt}_0) := \mathsf{HT.Hash}(\mathsf{dk}, D_0), \quad \mathsf{h}_0 := (\mathsf{st}_0, \mathsf{rt}_0)$$

Then for every $i \in [T]$

$$\ell_i = \mathsf{StepR}(\mathsf{st}_{i-1}), \quad (b_i, \Pi_i) := \mathsf{HT.Read}(\mathsf{tree}_{i-1}, \ell_i)$$
$$(b'_i, \ell'_i, \mathsf{st}_i) := \mathsf{StepW}(\mathsf{st}_{i-1}, b_i), \quad (\mathsf{tree}_i, \mathsf{rt}_i, \Pi'_i) := \mathsf{HT.Write}(\mathsf{tree}_{i-1}, \ell'_i, b'_i)$$
$$\mathsf{h}_i := (\mathsf{st}_i, \mathsf{rt}_i)$$

and then compute (efficiently) w_i be such that $\varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$.

Compute the no-signaling commitment to $(\mathsf{h}_0, \{\mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i\}_{i\in[T]})$

$$c \leftarrow \mathsf{Com}\left(K, \left(\mathsf{h}_0, \{\mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i\}_{i\in[T]}\right); R\right)$$

For every $i \in [T]$, compute the local opening to the commitment:
For $A \in \{\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i\}$,

$$\rho_A := \mathsf{Open}(K, A, R)^5$$

Compute the circuit C_{index} as described in Figure 4-6, and then compute the proof of the underlying BARG

$$\Pi := \mathsf{BARG.P}\left(\mathsf{crs}, C_{\mathsf{index}}, \left\{\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i, \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \rho_{b_i}, \rho_{\Pi_i}, \rho_{\Pi'_i}, \rho_{w_i}\right\}_{i\in[T]}\right)$$

Output (c, Π).

$\underline{\mathsf{RDel.V}(\mathsf{vk}, \mathsf{h}, \mathsf{h}', \Pi)}$**:** Given c and K, compute C_{index} (as in Figure 4-6) and output 1 if and only if

$$\mathsf{BARG.V}\left(\mathsf{crs}, C_{\mathsf{index}}, \Pi\right) = 1$$

Figure 4-5. RAM delegation scheme

Circuit C_{index}

Hardwired: K, c, φ
Input: $i, \mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i, \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \rho_{b_i}, \rho_{\Pi_i}, \rho_{\Pi'_i}, \rho_{w_i}$
Output: Output 1 if and only if

1. Verify commitment openings:
   (a) $\mathsf{Verify}(K, c, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}) = 1$[6]
   (b) $\mathsf{Verify}(K, c, \mathsf{h}_i, \rho_{\mathsf{h}_i}) = 1$
   (c) $\mathsf{Verify}(K, c, b_i, \rho_{b_i}) = 1$
   (d) $\mathsf{Verify}(K, c, \Pi_i, \rho_{\Pi_i}) = 1$
   (e) $\mathsf{Verify}(K, c, \Pi'_i, \rho_{\Pi'_i}) = 1$
   (f) $\mathsf{Verify}(K, c, w_i, \rho_{w_i}) = 1$
2. $\varphi_i(\mathsf{h}_{i-1}, \mathsf{h}_i, b_i, \Pi_i, \Pi'_i, w_i) = 1$

Figure 4-6. Circuit

Circuit $\mathsf{DelVerify}_{[(\mathsf{pk},\mathsf{vk})]}(x, (\omega, \Pi))$

Hardwired: The CRS $(\mathsf{pk}, \mathsf{vk})$ for the Turing machine delegation scheme.
Output $\mathsf{Del.V}(\mathsf{pk}, (x, \omega), \Pi)$.

Figure 4-7. The new circuit C' for batch argument.

Chapter 5

NIZKs from Sub-exponential DDH

5.1 Technical Overview

Our constructions rely on the correlation-intractability framework for instantiating the Fiat-Shamir paradigm. We start by recalling this framework.

Main Challenges. As mentioned earlier, the recent work of Brakerski et al. [51] leverages the compactness properties of (rate-1) trapdoor hash functions to build CIH for functions that can be approximated by a distribution on constant-degree polynomials. While this is a small class, [51] show that by relying on the LPN assumption, it is possible to construct trapdoor sigma protocols where the bad challenge function has probabilistic constant-degree representation. By collapsing the rounds of this protocol, they obtain NIZKs for $\mathcal{NP}$ from LPN and DDH (or other standard assumptions that suffice for constructing TDH).

We now briefly discuss the main conceptual challenges in buildings NIZKs based solely on DDH. On the one hand, (non-pairing) group-based assumptions seem to have less structure than lattice assumptions; for example, we can only exploit linear homomorphisms. Hence it is not immediately clear how to construct rate-1 trapdoor hash functions from DDH beyond (probabilistic) linear functions or constant-degree polynomials (a constant-degree polynomial is also a linear function of its monomials).[1]

[1]The breakthrough work of [98] shows that in the case of homomorphic secret-sharing, it is in fact

On the other hand, it seems that we need CIH for more complicated functions in order to build NIZKs from (only) DDH via the CIH framework.

Indeed, the bad challenge function in trapdoor sigma protocols involves (at least) *extraction* from the commitment scheme used in the protocol, and it is unclear whether such extraction can be represented by probablistic constant-degree polynomials when the commitment scheme is constructed from standard group-based assumptions. For example, the decryption circuit for the ElGamal encryption scheme [99] (based on DDH) is in a higher complexity class, and is not known to have representation by probabilistic constant-degree polynomials. Indeed, there are known lower-bounds for functions that can be approximated by probabilistic polynomials. Specifically, [100–103] proved that approximating a n fan-in majority gate by probabilistic polynomials over binary field with a small constant error requires degree at least $\Omega(\sqrt{n})$.

Roadmap. We overcome the above dilemma by exploiting the power of interaction.

- In Section 5.1.1, we introduce the notion of interactive trapdoor hashing protocols (ITDH) – a generalization of TDH to multi-round interactive protocols. We show that despite increased interaction, ITDH can be used to build CIH. Namely, we devise a round-collapsing approach to construct CIH from ITDH.

- We next show that ITDH can capture a larger class of computations than what can be supported by known constructions of TDH. Namely, we construct a constant-round ITDH protocol for TC^0 where the sender is laconic (Section 5.1.2).

- Finally, we demonstrate that using DDH, it is possible to construct trapdoor sigma protocols where the bad challenge function can be computed in low depth. Using such sigma protocols, we build multi-theorem (statistical) NIZK and

possible to go beyond linear homomorphisms in traditional groups. The communication complexity of the sender in their scenario, however, grows with the input length and is *not* compact as in the case of TDH.

statistical Zap arguments for $\mathcal{NP}$ (Sections 5.1.3 and 5.1.4, respectively).

5.1.1 Interactive Trapdoor Hashing Protocols

We start by providing an informal definition of ITDH and then describe our strategy for constructing CIH from ITDH.

Defining ITDH. An L-level ITDH is an interactive protocol between a "sender" and a "receiver", where the receiver's input is a circuit f and the sender's input is a string x. The two parties jointly compute $f(x)$ by multiple rounds of communication that are divided into L *levels*. Each level $\ell \in [L]$ consists of two consecutive protocol messages – a receiver's message, followed by the sender's response:

- First, the receiver uses f (and prior protocol information) to compute a key k_ℓ and trapdoor td_ℓ. It sends the key k_ℓ to the sender.
- Upon receiving this message, the sender computes a hash value h_ℓ together with an encoding e_ℓ. The sender sends h_ℓ to the receiver but keeps e_ℓ to herself. (The encoding e_ℓ can be viewed as sender's "private state" used for computing the next level message.)

Upon receiving the level L (i.e., final) message h_L from the sender, the receiver computes a decoding value d using the trapdoor. The function output $f(x)$ can be recovered by computing $\mathsf{e} \oplus \mathsf{d}$, where e is the *final* level encoding computed by the sender. We require the following properties from ITDH:

- **Compactness**: The sender's message in every level must be *compact*. Specifically, for every level $\ell \in [L]$, the size of the hash value h_ℓ is bounded by the security parameter, and is independent of the length of the sender's input x and the size of the circuit f.
- **Approximate Correctness**: For an overwhelming fraction of the random tapes

for the receiver, for any input x, the Hamming distance between $\mathsf{e} \oplus \mathsf{d}$ and $f(x)$ must be small. Note that this is an *adaptive* definition in that the input x is chosen after the randomness for the receiver is fixed.

- **Leveled Function Privacy**: The receiver's messages computationally hide the circuit f. Specifically, we require that the receiver's message in every level can be simulated without knowledge of the circuit f. Moreover, we allow the privacy guarantee to be *different* for each level by use of different security parameters for different levels.

As we discuss in Section 5.2.1, barring some differences in syntax, trapdoor hash functions can be viewed as 1-level ITDH. We refer the reader to the technical sections for a formal definition of ITDH.

CIH from ITDH. We now describe our round-collapsing strategy for constructing CIH from ITDH. Given an L-level ITDH for a circuit family $\mathcal{C}$, we construct a family of CIH for relations searchable by $\mathcal{C}$ as follows:

- **Key Generation:** The key generation algorithm uses the function-privacy simulator for ITDH to compute a simulated receiver message for *every* level. It outputs a key k consisting of L simulated receiver messages (one for each level) as well as a random mask mask.

- **Hash Function:** Given a key k and an input x, the hash function uses the ITDH sender algorithm on input x to perform an ITDH protocol execution "in its head." Specifically, for every level $\ell \in [L]$, it reads the corresponding receiver message in the key k and uses it to computes the hash value and the encoding for that level. By proceeding in a level-by-level fashion, it obtains the final level encoding e. It outputs $\mathsf{e} \oplus \mathsf{mask}$.

We now sketch the proof for correlation intractability. For simplicity, we first

consider the case when $L = 1$. We then extend the proof strategy to the multi-level case.

For $L = 1$, the proof of correlation intractability resembles the proof in [51]. We first switch the simulated receiver message in the CIH key to a "real" message honestly computed using a circuit $C \in \mathcal{C}$. Now, suppose that the adversary finds an x such that $\mathsf{Hash}(\mathsf{k}, x) = C(x)$. Then by approximate correctness of ITDH, $C(x) \approx \mathsf{e} \oplus \mathsf{d}$, where the " $\approx$ " notation denotes closeness in Hamming distance. This implies that $\mathsf{e} \oplus \mathsf{d} \approx \mathsf{e} \oplus \mathsf{mask}$, and thus $\mathsf{d} \approx \mathsf{mask}$. However, once we fix the randomness used by the receiver, d *only depends on* h. Since h is compact, the value d is exponentially "sparse" in its range. Therefore, the probability that $\mathsf{d} \approx \mathsf{mask}$ is exponentially small, and thus such an input x exists with only negligible probability.

Let us now consider the multi-level case. Our starting idea is to switch the simulated receiver messages in the CIH key to "real" messages in a level-by-level manner. However, note that the honest receiver message at each level depends on the hash value sent by the sender in the previous level, and at the time of the key generation of the CIH, the sender's input has not been determined. Hence, it is not immediately clear how to compute the honest receiver message at each level without knowing the sender's input.

To get around this issue, at each level ℓ, we first simply *guess* the sender's hash value $\mathsf{h}_{\ell-1}$ in the previous level $(\ell - 1)$, and then switch the simulated receiver message in level ℓ to one computed honestly using the ITDH receiver algorithm on input $\mathsf{h}_{\ell-1}$. To ensure this guessing succeeds with high probability, we rely on the *compactness* of the hash values. Specifically, let λ_ℓ denote the security parameter for the ℓ^{th} level in ITDH (as mentioned earlier, we allow the security parameters for each level to be different). Then the guessing of the level $(\ell - 1)$ hash value succeeds with probability $2^{-\lambda_{\ell-1}}$. We set $\lambda_{\ell-1}$ to be sublinear in λ, where λ is the security parameter for CIH. Then, when we reach the final level, all our guesses are successful with probability

$2^{-(\lambda_1+\lambda_2+\ldots+\lambda_L)}$, which is sub-exponential in λ. Since the probability of $\mathsf{d} \approx \mathsf{mask}$ can be exponentially small in λ, we can still get a contradiction.

However, the above argument assumes the function privacy is perfect, which is not the case. Indeed, at every level, we must also account for the adversary's distinguishing advantage when we switch a simulated message to a real message. In order to make the above argument go through, we need the distinguishing advantage to be a magnitude smaller than $2^{-\lambda_{\ell-1}}$ (for every ℓ). That is, we require ITDH to satisfy sub-exponential leveled functional privacy. Now, the distinguishing advantage can be bounded by $2^{-\lambda_\ell^c}$, where $0 < c < 1$ is a constant. Once we choose λ_ℓ large enough, then $2^{-\lambda_\ell^c}$ can be much smaller than $2^{-\lambda_{\ell-1}}$, and thus the above argument goes through as long as L is not too large.

In particular, there is room for trade-off between the number of levels in ITDH that we can collapse and the amount of leveled function privacy required. If we wish to rely on *polynomial time* and sub-exponential advantage assumptions, then the above transformation requires the number of levels to be *constant*. If we allow for *sub-exponential time* (and sub-exponential advantage) assumptions, then the above transformation can work for up to $O(\log \log \lambda)$ levels. We refer the reader to Section 5.4.4 for more details.

5.1.2 Constructing ITDH

We now provide an overview of our construction of constant-round ITDH for TC^0. Let *not-threshold* gate be a gate that computes a threshold gate and then outputs its negation. Since not-threshold gates are universal for threshold circuits, it suffices for our purpose to consider circuits that consist of only not-threshold gates.

The starting template for our construction consists of the following natural two-step approach reminiscent of classical secure computation protocols [104]:

- **Step 1 (Depth-1 Circuits):** First, we build an ITDH for a simple circuit family where each circuit is simply a *single* layer of layer of not-threshold gates.
- **Step 2 (Sequential Composition):** Next, to compute circuits with larger depth, we *sequentially compose* multiple instances of ITDH from the first step, where the output of the i^{th} ITDH is used as an input in the $(i+1)^{\text{th}}$ ITDH.

Input Passing. While natural, the above template doesn't work straight out of the box. Recall that the protocol output in any ITDH execution is "secret shared" between the sender and the receiver, where the sender holds the final level encoding e, and the receiver holds the decoding d. Then the first challenge in the sequential composition is how to continue the circuit computation when the result of the previous ITDH $\mathsf{e} \oplus \mathsf{d}$ is not known to the sender and the receiver.

A plausible way to resolve this challenge is for the receiver to simply send the decoding in the i^{th} ITDH to the sender so that the latter can compute the output, and then use it as input in the $(i+1)^{\text{th}}$ ITDH. However, this leaks intermediate wire values (of the TC^0 circuit that we wish to compute) to the sender, thereby compromising function privacy. Note that the reverse strategy of requiring the sender to send the encoding to the receiver (to allow output computation) also does not work since it violates the compactness requirement on the sender's messages to the receiver.

To overcome this challenge, we keep the secret-sharing structure of the output in every ITDH intact. Instead, we extend the functionality of ITDH for depth-1 threshold circuits so that the output of the i^{th} ITDH can be computed *within* the $(i+1)^{\text{th}}$ ITDH. Specifically, we first construct an ITDH for a circuit family $\mathcal{T}^{\oplus}$ where every circuit consists of a single layer of Xor-*then*-Not-Threshold gates. Such a gate first computes xor of its input with a vector pre-hardwired in the gate description, and then computes a not-threshold gate over the xor-ed value.

This allows for resolving the above problem as follows: the final-level encoding

from the i^{th} ITDH constitutes the sender's input in the $(i+1)^{\text{th}}$ ITDH. On the other hand, the decoding in the i^{th} ITDH is used as the pre-hardwired string in the circuit computed by the $(i+1)^{\text{th}}$ ITDH.

ITDH for a single Xor-*then*-Not-Threshold Gate. We now describe the main ideas for computing a *single* Xor-*then*-Not-Threshold gate. Our ideas readily extend to the case where we want to compute a single layer of such gates.

To construct an ITDH for a single Xor-*then*-Not-Threshold gate, we only rely on trapdoor hash functions (TDH) for linear functions. Crucially, however, we use *interaction* to go beyond computing linear functions. At a high-level, we first "decompose" an Xor-*then*-Not-Threshold gate as the composition of two linear functions. We then use TDH for computing each of these linear functions separately. Finally, we "compose" the two TDH executions sequentially to obtain a 2-level ITDH for an Xor-*then*-Not-Threshold gate.

An observant reader may wonder how we decompose a Xor-*then*-Not-Threshold gate into computation of linear functions. Indeed, the composition of linear functions is still a linear function, while such a threshold gate involves non-linear computation. As we will soon see, our decomposition strategy crucially relies on "offline" processing by the parties on the intermediate values between the two TDH executions. This introduces the desired non-linearity in the computation.

Let the vector $x \in \{0,1\}^n$ be the input to the Xor-*then*-Not-Threshold gate, $y \in \{0,1\}^n$ be the binary vector pre-hardwired in the gate description, and let t be the threshold. The Xor-*then*-Not-Threshold gate computes whether the number of 1's in $x \oplus y$ is smaller than t. To compute such a gate, we proceed in the following three simple steps:

- *Xor*: First, xor the input vector x with y, where y is part of the gate description.
- *Summation*: Second, sum the elements in the vector $x \oplus y$ over $\mathbb{Z}$.

- *Comparison*: Finally, compare the summation with the threshold t.

We now describe how to express each step as a linear function. For the first step, let x_i and y_i be two bits at (say) the i^{th} coordinate of x and y, respectively. Then $x_i \oplus y_i = 1$ if and only if $x_i = 0 \wedge y_i = 1$ or $x_i = 1 \wedge y_i = 0$. Hence, $x_i \oplus y_i = (1 - x_i) \cdot y_i + x_i \cdot (1 - y_i)$. Since y_i is part of the circuit description, the right hand side is a linear function of x_i over $\mathbb{Z}$.

In the second step, we simply sum over all the coordinates of $x \oplus y$. Since the summation is a linear function, and the first step is also linear, composing these two linear functions, we obtain a linear function of x over $\mathbb{Z}$. Then we can use a TDH for linear functions for this task. We note, however, that the construction of TDH in [51, 82] only works for linear functions over $\mathbb{Z}_2$. We therefore extend their construction to arbitrary polynomial modulus. In our case, since the summation cannot be more than n, it suffices to choose the modulo $(n + 1)$.

We now proceed to express the comparison in the final step as a linear function. Suppose the summation value from the second step is $\mathsf{sum} \in \{0, 1, 2, \ldots, n\}$ and we want to compare it with a threshold t. Let $\mathbb{1}_{\mathsf{sum}}$ denote the indicator vector of sum, i.e., $\mathbb{1}_{\mathsf{sum}} = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$, where the $(\mathsf{sum}+1)^{\text{th}}$ coordinate is 1, and all other coordinates are 0. Then, we have that

$$\mathsf{sum} < t \iff \langle \mathbb{1}_{\mathsf{sum}}, \mathbb{1}_{<t} \rangle = 1,$$

where $\mathbb{1}_{<t} = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$ is a vector with 1's on the first t-coordinates, and 0's on the remaining coordinates. We can therefore express the comparison in the final step as an inner product of $\mathbb{1}_{\mathsf{sum}}$ and $\mathbb{1}_{<t}$, which is a linear function of $\mathbb{1}_{\mathsf{sum}}$. This means that we can again use a TDH for linear functions for performing this computation.

Note, however, that the sender and the receiver do not directly obtain the summation value sum after the first TDH execution. Indeed, after the first TDH execution,

the sender obtains an encoding e and the receiver obtains a decoding d such that $(\mathsf{e} + \mathsf{d}) \bmod R = \mathsf{sum}$. Thus, we need a mechanism to perform the final step even though neither party holds sum.[2]

Fortunately, we can still express the comparison $(\mathsf{e} + \mathsf{d}) \bmod R < t$ as

$$(\mathsf{e} + \mathsf{d}) \bmod R < t \Leftrightarrow \langle \mathbb{1}_{\mathsf{e}}, \mathbb{1}_{\mathsf{d},<t} \rangle = 1,$$

where $\mathbb{1}_{\mathsf{d},<t} = \sum_{j=0}^{t-1} \mathbb{1}_{(j-\mathsf{d}) \bmod R}$. The above equation follows from the fact that checking $(\mathsf{e} + \mathsf{d}) \bmod R < t$ is equivalent to check whether there exists a $j \in \{0, 1, \ldots, t-1\}$ such that $(\mathsf{e} + \mathsf{d}) \bmod R = j$, which is equivalent to checking $\mathsf{e} = (j - \mathsf{d}) \bmod R$. Note that by this equation, we express the comparison as a linear function of $\mathbb{1}_{\mathsf{e}}$ over $\mathbb{Z}_2$. Hence, the comparison in the final step can be computed by another TDH.

Between the two executions of TDH, the sender processes e from the first TDH to obtain $\mathbb{1}_{\mathsf{e}}$, and use it as the input to the second TDH. Similarly, the receiver processes d from the first TDH to obtain $\mathbb{1}_{\mathsf{d},<t} = \sum_{j=0}^{t-1} \mathbb{1}_{(j-\mathsf{d}) \bmod R}$, and use the linear function $\langle \cdot, \mathbb{1}_{\mathsf{d},<t} \rangle$ as the input to the second TDH. Note that this intermediate processing is non-linear, since computing the indicator vector can be done by several equality checks, and the equality check is not a linear function. Hence, it introduces the necessary non-linearity in the computation, but is done "outside" of the TDH execution.

Controlling the Error. We now discuss another issue that arises in the implementation of our template. Recall that an ITDH guarantees only approximate correctness, i.e., the xor of the final-level encoding e and decoding d is "close" (in terms of Hamming distance) to the true function output. Then, in a sequential composition of an ITDH protocol, *each* execution only guarantees approximate correctness. This means that the errors could *spread* across the executions, ultimately causing *every output bit of the final execution to be incorrect*. For example, suppose a coordinate of the output for an intermediate execution is flipped and later, the computation of every output bit

[2]For reasons as discussed earlier, the straightforward idea of simply requiring one of the two parties to send their secret share to the other party (for computing sum) does not work.

depends on this flipped output bit. In this case, every output bit could be incorrect.

To overcome this issue, we observe that any circuit can be converted to a new circuit that satisfies a "parallel" structure demonstrated in Figure 5-1.

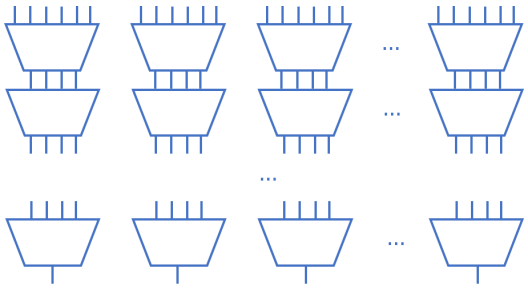


Figure 5-1. Parallel structure. The top (resp., bottom) layer corresponds to input (resp., output) wires.

In such circuits, each output bit only depends on the input to *one* parallel execution. Hence, the spreading of one Hamming error is controlled in one parallel execution. This allows us to prove approximate correctness of the sequential composition.

5.1.3 Constructing NIZKs

Armed with our construction of CIH, we now sketch the main ideas underlying our construction of (statistical) multi-theorem NIZK for $\mathcal{NP}$. We proceed in the following two steps:

1. First, using CIH for TC^0, we construct a non-interactive witness indistinguishable (NIWI) argument for $\mathcal{NP}$ in the common random string model. Our construction satisfies either statistical WI and non-adaptive soundness, or computational WI and adaptive soundness.

2. We then transform the above NIWI into an adaptive, *multi-theorem* NIZK for $\mathcal{NP}$ in the common random string model via a variant of the Feige-Lapidot-Shamir (FLS) "OR-trick" [46].[3] Our NIZK satisfies either statistical ZK and

[3]By using "programmable" CIH, one could directly obtain NIZKs in the first step. However, the

non-adaptive soundness, or computational ZK and adaptive soundness. Crucially, unlike the classical FLS transformation, our transformation does *not* require "CRS switching" in the security proof and hence works for both statistical and computational ZK cases seamlessly while preserving the distribution of the CRS in the underlying NIWI.

Statistical NIZKs. In the remainder of this section, we focus on the construction of *statistical* NIZKs. We briefly discuss the steps necessary for obtaining the computational variant (with adaptive soundness) at the end of the section.

Towards implementing the first of the above two steps, we first build the following two ingredients:

- A lossy public key encryption scheme with an additional property that we refer to as *low-depth decryption*, from DDH. Roughly speaking, this property requires that there exists a TC^0 circuit Dec that takes as input any ciphertext ct and a secret key sk, and outputs the correct plaintext.
- A trapdoor sigma protocol for $\mathcal{NP}$ with bad challenge function in TC^0 from the above lossy public key encrytion scheme. We also require the trapdoor sigma protocol to satisfy an additional "knowledge extraction" property, which can be viewed as an analogue of special soundness for trapdoor sigma protocols. Looking ahead, we use this property to construct NIWIs with argument of knowledge property, which in turn is required for our FLS variant for constructing NIZKs.

Lossy Public Key Encryption. The lossy public key encryption we use is essentially the same as in [105–107]. We start by briefly describing the scheme.

A public key $\mathsf{pk} = \begin{bmatrix} g^1 & g^b \\ g^a & g^c \end{bmatrix}$ is a matrix of elements in a group $\mathbb{G}$. When the matrix $\begin{bmatrix} 1 & b \\ a & c \end{bmatrix}$ is singular (i.e., $c = ab$), then the public key is in the "injective mode" and the

resulting NIZK only achieves *single-theorem* ZK; hence an additional step is still required to obtain multi-theorem NIZKs.

secret key is $\mathsf{sk} = a$; when the matrix is non-singular (i.e., $c \neq ab$), then the public key is in the "lossy mode." The encryption algorithm is described as follows:

$$\mathsf{Enc}\left(\mathsf{pk}, m \in \{0,1\}; r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}\right) = \begin{bmatrix} (g^1)^{r_1} \cdot (g^b)^{r_2} \\ (g^a)^{r_1} \cdot (g^c)^{r_2} \cdot g^m \end{bmatrix} = g^{\begin{bmatrix} 1 & b \\ a & c \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} + \begin{bmatrix} 0 \\ m \end{bmatrix}}.$$

Let us now argue the low-depth decryption property. Let $[c_1, c_2]^T$ denote the ciphertext obtained by encrypting a message m using an injective mode public key pk with secret key $\mathsf{sk} = a$. To decrypt the ciphertext, we can compute $c_1^{-a} \cdot c_2 = g^m$ and then comparing with $1_{\mathbb{G}}$ to recover m. However, it is not known whether c_1^{-a} can be computed in TC^0 (recall that a depends on the security parameter).

In the following, we assume the DDH group is a subgroup of $\mathbb{Z}_q^*$, for some positive integer q. For the instantiation from Elliptic curves over $\mathbb{F}_p$ with a prime $p > 3$, see Appendix 5.7 for more details.

Towards achieving the low-depth decryption property, we use the following observation. Let $a_0, a_1, \ldots a_\lambda$ be the binary representation of a. Then, we have that

$$\left(c_1^{-2^0}\right)^{a_0} \cdot \left(c_1^{-2^1}\right)^{a_1} \cdot \left(c_1^{-2^2}\right)^{a_2} \cdot \ldots \cdot \left(c_1^{-2^\lambda}\right)^{a_\lambda} \cdot c_2 = g^m.$$

Note that given $[c_1, c_2]^T$, one can "precompute" $c_1^{-2^0}, c_1^{-2^1}, \ldots, c_1^{-2^\lambda}$ without using the secret key sk. In our application to NIZKs and Zaps, such pre-computation can be performed by the prover and the verifier.

We leverage this observation to slightly modify the definition of low-depth decryption to allow for a *deterministic* polynomial-time "pre-computation" algorithm $\mathsf{PreComp}$. Specifically, we require that the output of $\mathsf{Dec}(\mathsf{PreComp}(1^\lambda, \mathsf{ct}), \mathsf{sk})$ is the correct plaintext m. We set $\mathsf{PreComp}(1^\lambda, c) = (c_1^{-2^0}, c_1^{-2^1}, \ldots, c_1^{-2^\lambda}, c_2)$, and allow the circuit Dec to receive $c_1^{-2^0}, c_1^{-2^1}, \ldots, c_1^{-2^\lambda}, c_2$ and $a_0, a_1, \ldots, a_\lambda$ as input. The decryption circuit Dec proceeds in the following steps:

- For each $i = 0, 1, \ldots, \lambda$, it chooses g_i to be either $1_{\mathbb{G}}$ or $c_1^{-2^i}$, such that $g_i = (c_1^{-2^i})^{a_i}$. This computation can be done in constant depth, and is hence in TC^0.

- Multiply the values $g_0, g_2, \ldots, g_\lambda$ and c_2. From [108], this iterative multiplication can be computed in TC^0 when we instantiate $\mathbb{G}$ as a subgroup of $\mathbb{Z}_q^*$.

- Compare the resulting value with $1_{\mathbb{G}}$. If they are equal, then output 0. Otherwise output 1.

Since each of the above steps can be computed in TC^0, we have that Dec is also in TC^0.

Trapdoor Sigma Protocol for $\mathcal{NP}$. Recently, Brakerski et al. [51] constructed a "commit-and-open" style trapdoor sigma protocol where the only cryptographic primitive used is a commitment scheme. Crucially, the bad challenge function for their protocol involves the following two computations: extraction from the commitment, and a post-extraction verification using 3-CNF. By exploiting the specific form of their bad challenge function, we construct a trapdoor sigma protocol for $\mathcal{NP}$ with our desired properties by simply instantiating the commitment scheme in their protocol with the above lossy encryption scheme.

Let us analyze the bad challenge function of the resulting trapdoor sigma protocol. Since our lossy public key encryption satisfies the low-depth decryption property, the first step of the bad challenge computation can be done in TC^0. Next, note that the second step of the bad challenge computation is also in TC^0 since it involves evaluation of 3-CNF which can be computed in AC^0. Thus, the bad challenge function is in TC^0.

We observe that our protocol also satisfies a *knowledge extraction* property which requires that one can efficiently extract a witness from a *single* accepting transcript (α, β, γ) by using a trapdoor (namely, the secret key of the lossy public key encryption), if β does not equal to the output of the bad challenge function evaluated on α. We use this property to construct NIWIs with argument of knowledge property.

NIWI from Fiat-Shamir via CIH. We construct NIWI arguments in the CRS model by using CIH to collapse the rounds of our trapdoor sigma protocol repeated λ

times in parallel. The CRS of the resulting construction contains a public-key of lossy public key encryption scheme from above and a CIH key. When the public key is in lossy mode, the NIWI achieves statistical WI property and non-adaptive argument of knowledge property.

To prove the argument of knowledge property, we observe that for any accepting transcript $(\{\alpha_i\}_{i\in[\lambda]}, \{\beta_i\}_{i\in[\lambda]}, \{\gamma_i\}_{i\in[\lambda]})$, it follows from correlation intractability of the CIH that $\{\beta_i\}_{i\in[\lambda]}$ is *not* equal to the outputs of the bad challenge function evaluated on $\{\alpha_i\}_{i\in[\lambda]}$. Hence, there exists at least one index i^* such that β_{i^*} is not equal to the output of the bad challenge function on α_{i^*}. We can now extract a witness by relying on the knowledge extraction property of the i^*-th parallel execution of the trapdoor sigma protocol.

From NIWI to Multi-theorem NIZK. The FLS "OR-trick" [46] is a standard methodology to transform NIWIs (or single-theorem NIZKs) into multi-theorem NIZKs. Roughly speaking, the trick involves supplementing the CRS with an instance (say) y of a hard-on-average decision problem and requiring the prover to prove that either the "original" instance (say) x *or* y is true. This methodology involves switching the CRS either in the proof of soundness or zero-knowledge, which can potentially result in a degradation of security. E.g., in the former case, one may end up with non-adaptive (computational) soundness while in the latter case, one may end up with computational ZK even if the underlying scheme achieves statistical privacy. The instance y also needs to be chosen carefully depending on the desired security and whether one wants the resulting CRS to be a reference string or a random string.

We consider a variant of the "OR-trick" that does not require CRS switching and preserves the distribution of the CRS of the underlying scheme. We supplement the CRS with an instance of hard-on-average *search* problem, where the instance is subjected to the *uniform* distribution, and can be sampled *together with* a witness. For our purposes, the discrete logarithm problem suffices. To sample the instance

uniformly at random together with a witness, we firstly sample a secret exponent, and then set the instance as the exponent raised to a group generator. The ZK simulator simply uses the secret exponent of the discrete-log instance in the CRS to simulate the proof. On the other hand, soundness can be argued by relying on the computational hardness of the discrete-log problem. One caveat of this transformation is that the proof of soundness requires the underlying NIWI to satisfy argument of knowledge property. We, note, however, that this property is usually easy to achieve (in the CRS model). Using this approach, we obtain statistical multi-theorem NIZK arguments in the common *random* string model from sub-exponential DDH. Previously, group-based statistical NIZKs were known only in the common reference string model [49].

We remark that the above idea can be easily generalized to other settings. For example, starting from LWE-based single-theorem statistical NIZKs [50], one can embed the Shortest Integer Solution (SIS) problem in the CRS to build *multi-theorem* statistical NIZKs in the common random string model. This settles an open question stated in the work of [50].

Computational NIZKs with Adaptive Soundness. Using essentially the same approach as described above, we can also construct computational NIZKs for $\mathcal{NP}$ with adaptive soundness. The main difference is that instead of using lossy public-key encryption scheme in the construction of trapdoor sigma protocols, we use ElGamal encryption scheme [99]. Using the same ideas as for our lossy public-key encryption scheme, we observe that the ElGamal encryption scheme also satisfies *low-depth decryption* property. This allows us to follow the same sequence of steps as described above to obtain a computational NIZK for $\mathcal{NP}$ with adaptive soundness in the common *random* string model.[4]

[4]We note that one could obtain computational NIZKs with adaptive soundness by simply "switching the CRS" in our construction of statistical NIZKs. However, the resulting scheme in this case is in the common *reference* string model.

5.1.4 Constructing Zaps

At a high-level, we follow a similar recipe as in the recent works of [57, 58] who construct statistical Zap arguments from quasi-polynomial LWE.

The main idea in these works is to replace the (non-interactive) commitment scheme in a trapdoor sigma protocol with a *two-round* statistical-hiding commitment scheme in the plain model and then collapse the rounds of the resulting protocol using CIH, as in the case of NIZKs. Crucially, unlike the non-interactive commitment scheme that only allows for extraction in the CRS model, the two-round commitment scheme must support extraction in the plain model. The key idea for achieving such an extraction property (in conjunction with statistical-hiding property) is to allow for successful extraction with only negligible but still much larger than sub-exponential probability (for example, $2^{-\log^2 \lambda}$) [109]. By carefully using complexity leveraging, one can prove soundness of the resulting argument system.

Statistical-Hiding Commitment with Low-depth Extraction. We implement this approach by replacing the lossy public-key encryption scheme in our NIWI construction (from earlier) with a two-round statistical hiding commitment scheme. Since we need the bad challenge function of the sigma protocol to be in TC^0, we require the commitment scheme to satisfy an additional *low-depth extraction* property.

To construct such a scheme, we first observe that the construction of (public-coin) statistical-hiding extractable commitments in [57, 58, 109, 110] only makes black-box use of a two-round oblivious transfer (OT) scheme. We instantiate this generic construction via the Naor-Pinkas OT scheme based on DDH [81]. By exploiting the specific structure of the generic construction as well as the fact that Naor-Pinkas OT decryption can be computed in TC^0, we are able to show that the extraction process can also be performed in TC^0. We refer the reader to Section 5.6 for more details.

5.2 Interactive Trapdoor Hashing Protocols

In this section, we define interactive trapdoor hashing protocols (ITDH). At a high-level, ITDH is a generalization of trapdoor hash functions – which can be viewed as two-round two-party protocols with specific structural and communication efficiency properties – to multi-round protocols.

More specifically, an interactive trapdoor hashing protocol involves two parties – a sender and a receiver. The sender has an input x, while the receiver has a circuit f. The two parties jointly compute $f(x)$ over several rounds of interaction. We structure the protocols in multiple *levels*, where a level consists of the following two successive rounds:

- The receiver generates a key k and a trapdoor td using a key generation algorithm KGen, which takes as input the circuit f, the level number, and some additional internal state of the receiver. Then it sends k to the sender.
- Upon receiving a key k, the sender computes a hash value h and an encoding e using the algorithm $\mathsf{Hash\&Enc}$, which takes as input x, the key k, the level number, and the previous level encoding. Then it sends the hash h to the receiver, and keeps e as an internal state.

Finally, there is a decoding algorithm Dec that takes the internal state of the receiver after the last level as input, and outputs a decoding value d. Ideally, we want the output $f(x)$ to be $\mathsf{e} \oplus \mathsf{d}$.

In the following, we proceed to formally define this notion and its properties.

Per-level Security Parameter. In our formal definition of ITDH, we allow the security parameter to be *different* for every level. This formulation is guided by our main application, namely, constructing correlation-intractable hash functions (see Section 5.4). Nevertheless, we note that ITDH could also be meaningfully defined

w.r.t. a single security parameter for the entire protocol.

5.2.1 Definition

Let $\mathcal{C} = \{\mathcal{C}_{n,u}\}_{n,u}$ be a family of circuits, where each circuit $f \in \mathcal{C}_{n,u}$ is a circuit of input length n and output length u. An L-level interactive trapdoor hashing protocol for the circuit family $\mathcal{C}$ is a tuple of algorithms $\mathsf{ITDH} = (\mathsf{KGen}, \mathsf{Hash\&Enc}, \mathsf{Dec})$ that are described below.

We use $\lambda_1, \ldots, \lambda_L$ to denote the security parameters for different levels. Throughout this work, these parameters are set so that they are polynomially related. That is, there exists a λ such that $\lambda_1, \ldots, \lambda_L$ are polynomials in λ.

- $\mathsf{KGen}(1^{\lambda_\ell}, \ell, f, \mathsf{h}_{\ell-1}, \mathsf{td}_{\ell-1})$: The key generation algorithm takes as input a security parameter λ_ℓ (that varies with the level number), a level number ℓ, a circuit $f \in \mathcal{C}_{n,u}$, a level $(\ell-1)$ hash value $\mathsf{h}_{\ell-1}$ and trapdoor $\mathsf{td}_{\ell-1}$ (for $\ell = 1$, $\mathsf{h}_{\ell-1} = \mathsf{td}_{\ell-1} = \perp$). It outputs an ℓ^{th} level key k_ℓ and a trapdoor td_ℓ.

- $\mathsf{Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1})$: The hash-and-encode algorithm takes as input a level ℓ hash key k_ℓ, an input x, and a level $(\ell-1)$ encoding $\mathsf{e}_{\ell-1}$. It outputs an ℓ^{th} level hash value h_ℓ and an encoding $\mathsf{e}_\ell \in \{0,1\}^u$. When $\ell = 1$, we let $\mathsf{e}_{\ell-1} = \perp$.

- $\mathsf{Dec}(\mathsf{td}_L, \mathsf{h}_L)$: The decoding algorithm takes as input a level L trapdoor td_L and hash value h_L, and outputs a value $\mathsf{d} \in \{0,1\}^u$.

We require ITDH to satisfy the following properties:

- **Compactness:** For each level $\ell \in [L]$, the bit length of h_ℓ is at most λ_ℓ.

- (Δ, ϵ)**-Approximate Correctness:** For any $n, u \in \mathbb{N}$, any circuit $f \in \mathcal{C}_{n,u}$ and any sequence of security parameters $(\lambda_1, \ldots, \lambda_L)$, we have

$$\Pr_{r_1, r_2, \ldots, r_L}\left[\forall x \in \{0,1\}^n, \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f(x)) < \Delta(u)\right] > 1 - \epsilon(u, \lambda_1, \ldots, \lambda_L),$$

where e, d are obtained by the following procedure: Let $\mathsf{h}_0 = \mathsf{td}_0 = \mathsf{e}_0 = \perp$. For $\ell = 1, 2, \ldots, L$,

- Compute $(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{KGen}(1^{\lambda_\ell}, \ell, f, \mathsf{h}_{\ell-1}, \mathsf{td}_{\ell-1}; r_\ell)$ using random coins r_ℓ.
- Hash and encode the input x: $(\mathsf{h}_\ell, \mathsf{e}_\ell) \leftarrow \mathsf{Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1})$.

Finally, let $\mathsf{e} = \mathsf{e}_L$ be the encoding at the final level, and $\mathsf{d} = \mathsf{Dec}(\mathsf{td}_L, \mathsf{h}_L)$.

- **Leveled Function Privacy:** There exist a simulator Sim and a negligible function $\nu(\cdot)$ such that for any level $\ell \in [L]$, any polynomials $n(\cdot)$ and $u(\cdot)$ in the security parameter, any circuit $f \in \mathcal{C}_{n,u}$, any trapdoor $\mathsf{td}' \in \{0,1\}^{|\mathsf{td}_{\ell-1}|}$, any hash value $\mathsf{h}' \in \{0,1\}^{|\mathsf{h}_{\ell-1}|}$, and any n.u. PPT distinguisher $\mathcal{D}$,

$$\Big| \Pr\left[(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{KGen}(1^{\lambda_\ell}, \ell, f, \mathsf{h}', \mathsf{td}') : \mathcal{D}(1^{\lambda_\ell}, \mathsf{k}_\ell) = 1\right] - \Pr\left[\widetilde{\mathsf{k}}_\ell \leftarrow \mathsf{Sim}(1^{\lambda_\ell}, 1^n, 1^u, \ell) : \mathcal{D}(1^{\lambda_\ell}, \widetilde{\mathsf{k}}_\ell) = 1\right] \Big| \leq \nu(\lambda_\ell).$$

We say that the ITDH satisfies sub-exponential leveled function privacy, if there exists a constant $0 < c < 1$ such that for any n.u. PPT distinguisher, $\nu(\lambda_\ell)$ is bounded by $2^{-\lambda_\ell^c}$ for any sufficiently large λ_ℓ.

Note that since the security parameters for different levels are polynomially related, $n(\cdot)$ and $u(\cdot)$ are polynomials in λ_ℓ iff they are polynomials in λ.

Relationship with Trapdoor Hash Functions. A 1-level ITDH is essentially the same as TDH, except that in TDH, there are two kinds of keys: a hash key and an encoding key(see Section 2.4.2) . In particular, a hash value is computed using the hash key and can be reused with different encoding keys for different functions. In 1-level ITDH, however, the receiver's message only consists of one key that is used by the sender for computing both the hash value and the encoding. Therefore, the hash value is not reusable for different functions.

We choose the above formulation of ITDH for the sake of a simpler and cleaner definition that suffices for our applications. If we consider multi-bit output functions, then the above difference disappears, since we can combine multiple functions into one multi-bit output function and encode it using one key.

5.3 Construction of ITDH

In this section, we construct an interactive trapdoor hashing protocol (ITDH) for TC^0 circuits. We refer the reader to Section 5.1 for a high-level overview of our approach. The remainder of this section is organized as follows:

- **Depth-1 Circuits:** In Section 5.3.1, we first construct a 2-level ITDH protocol for $\mathcal{T}^{\oplus}$ – roughly speaking, a family of depth-1 Xor-*then*-Not-Threshold circuits (see below for the precise definition of $\mathcal{T}^{\oplus}$).
- **Sequential Composition:** Next, in Section 5.3.4, we present a sequential composition theorem for ITDH where we show how to compose L instances of a 2-level ITDH for some circuit family to obtain a $2L$-level ITDH for a related circuit family.
- **Construction for TC^0:** Finally, in Section 5.3.7, we put these two constructions together to obtain an ITDH for TC^0.

5.3.1 ITDH for $\mathcal{T}^{\oplus}$

We start by introducing some notation and definitions.

XOR-*then*-Compute Circuits. Let $\mathcal{C} = \{\mathcal{C}_{n,u}\}_{n,u}$ be a circuit family, where for any n and u, $\mathcal{C}_{n,u}$ contains circuits with n-bit inputs and u-bit outputs. For any $\mathcal{C}$, we define an Xor-*then*-Compute circuit family $\mathcal{C}^{\oplus} = \{\mathcal{C}^{\oplus}_{n,u}\}_{n,u}$ consisting of circuits that *first* compute a bit-wise xor operation on the input with a fixed string and *then* compute a circuit in $\mathcal{C}$ on the resulting value.

Specifically, $\mathcal{C}^{\oplus}_{n,u}$ contains all the circuit $\mathrm{C}^{\oplus y} : \{0,1\}^n \to \{0,1\}^u$, where $y \in \{0,1\}^n$ and there exists a $C \in \mathcal{C}_{n,u}$ such that for every $x \in \{0,1\}^n$,

$$\mathrm{C}^{\oplus y}(x) = C(x \oplus y).$$

Circuit Families $\mathcal{T}$ and $\mathcal{T}^{\oplus}$. We define a circuit family $\mathcal{T} = \{\mathcal{T}_{n,u}\}_{n,u}$ consisting of depth-1 not-threshold circuits, i.e., a single layer of not-threshold gates (see Section 2). Specifically, $\mathcal{T}_{n,u}$ contains all circuits $\mathrm{T}_{\vec{t},\vec{I}} : \{0,1\}^n \to \{0,1\}^u$ where $\vec{t} = \{t_1, \ldots, t_u\}$ is a set of positive integers, and $\vec{I} = \{I_1, \ldots, I_u\}$ is a collection of sets $I_j \subseteq [n]$ s.t. for any $x \in \{0,1\}^n$,

$$\mathrm{T}_{\vec{t},\vec{I}}(x) = \left(\overline{\mathsf{Th}}_{t_1}(x[I_1]), \ldots, \overline{\mathsf{Th}}_{t_u}(x[I_u])\right),$$

where for any index set $I_j = \{i_1, i_2, \ldots, i_w\} \subseteq [n]$, we denote $x[I_j] = (x_{i_1}, x_{i_2}, \ldots, x_{i_w})$ as the projection of string x to the set I_j.

The function family $\mathcal{T}^{\oplus} = \{\mathcal{T}^{\oplus}_{n,u}\}_{n,u}$ is defined as the Xor-*then*-Compute family corresponding to $\mathcal{T}$. We denote the circuits in $\mathcal{T}^{\oplus}_{n,u}$ as $\mathrm{T}^{\oplus y}_{\vec{t},\vec{I}}$, where $\vec{t}$, $\vec{I}$ and y are as defined above.

For a high-level overview of our construction, see Section 5.1.2. We now proceed to give a formal description of our construction.

Construction of ITDH for $\mathcal{T}^{\oplus}$. We construct a 2-level interactive trapdoor hashing protocol $\mathsf{ITDH} = (\mathsf{KGen}, \mathsf{Hash\&Enc}, \mathsf{Dec})$ for the circuit family $\mathcal{T}^{\oplus}$ as defined above. Our construction relies on the following ingredient: a trapdoor hash function $\mathsf{TDH} = (\mathsf{TDH.HKGen}, \mathsf{TDH.EKGen}, \mathsf{TDH.Hash}, \mathsf{TDH.Enc}, \mathsf{TDH.Dec})$ for the linear function family $\mathcal{F} = \{\mathcal{F}_{n,R}\}_{n,R}$ (see Definition 2.4.3) that achieves τ-enhanced correctness and function privacy.

For ease of exposition, we describe the algorithms of ITDH *separately* for each level. The first level algorithms of ITDH internally use TDH to evaluate a circuit (defined below) with input length $n_1 = n$ and modulus $R_1 = n+1$. The second level algorithms

of ITDH internally use TDH to evaluate another circuit (defined below) with input length $n_2 = R_1 \cdot u$ and modulus $R_2 = 2$. We use λ_1 and λ_2 to denote the security parameters input to the first and second level algorithms, respectively.

- **Level 1** $\mathsf{KGen}(1^{\lambda_1}, 1, \mathrm{T}^{\oplus y}_{\vec{t},\vec{I}}, \mathsf{h}_0 = \bot, \mathsf{td}_0 = \bot)$**:**
  - Sample a hash key of TDH w.r.t. security parameter λ_1, input length $n_1 = n$ and modulus $R_1 = n + 1$

    $$\mathsf{hk}_1 \leftarrow \mathsf{TDH.HKGen}(1^{\lambda_1}, 1^{n_1=n}, 1^{R_1=n+1})$$

  - Parse $\vec{I} = \{I_1, \ldots, I_u\}$. For every $i \in [u]$, sample an encoding key:

    $$(\mathsf{ek}_{1,i}, \mathsf{td}_{1,i}) \leftarrow \mathsf{TDH.EKGen}(\mathsf{hk}_1, \mathsf{XorSum}_{I_i,y})$$

    where for any set $I \subseteq [n]$, $\mathsf{XorSum}_{I,y}$ is the linear function described in Figure 5-2.
  - Output $(\mathsf{k}_1, \mathsf{td}_1)$ where $\mathsf{k}_1 = (1, \mathsf{hk}_1, \{\mathsf{ek}_{1,i}\}_{i\in[u]})$ and $\mathsf{td}_1 = \{\mathsf{td}_{1,i}\}_{i\in[u]}$.
- **Level 1** $\mathsf{Hash\&Enc}(\mathsf{k}_1, x, \mathsf{e}_0 = \bot)$**:**
  - Parse $\mathsf{k}_1 = (1, \mathsf{hk}_1, \{\mathsf{ek}_{1,i}\}_{i\in[u]})$.
  - Compute "first level" hash over x: $\mathsf{h}_1 \leftarrow \mathsf{TDH.Hash}(\mathsf{hk}_1, x)$
  - For every $i \in [u]$, compute a "first level" encoding: $\mathsf{e}_{1,i} \leftarrow \mathsf{TDH.Enc}(\mathsf{ek}_{1,i}, x)$
  - Output $(\mathsf{h}_1, \mathsf{e}_1)$, where $\mathsf{e}_1 = \{\mathsf{e}_{1,i}\}_{i\in[u]}$.
- **Level 2** $\mathsf{KGen}(1^{\lambda_2}, 2, \mathrm{T}^{\oplus y}_{\vec{t},\vec{I}}, \mathsf{h}_1, \mathsf{td}_1)$**:**
  - Parse $\mathsf{td}_1 = \{\mathsf{td}_{1,i}\}_{i\in[u]}$. For every $i \in [u]$, decode h_1: $\mathsf{d}_{1,i} \leftarrow \mathsf{TDH.Dec}(\mathsf{td}_{1,i}, \mathsf{h}_1)$
  - Sample a new hash key of TDH w.r.t. security parameter λ_2, input length $n_2 = R_1 \cdot u$ and modulus $R_2 = 2$,

    $$\mathsf{hk}_2 \leftarrow \mathsf{TDH.HKGen}(1^{\lambda_2}, 1^{n_2=R_1\cdot u}, 1^{R_2=2}).$$

– Parse $\vec{t} = \{t_1, \ldots, t_u\}$. For each $i \in [u]$, sample a new encoding key

$$(\mathsf{ek}_{2,i}, \mathsf{td}_{2,i}) \leftarrow \mathsf{TDH.EKGen}(\mathsf{hk}_2, \mathsf{AddTh}_{i,t_i,\mathsf{d}_{1,i}}),$$

where for any index $i \in [u]$, positive integer t and value $\mathsf{d} \in \mathbb{Z}_{R_1}$, $\mathsf{AddTh}_{i,t,\mathsf{d}}$ is the linear function defined in the Figure 5-3.

– Output $(\mathsf{k}_2, \mathsf{td}_2)$, where $\mathsf{k}_2 = (2, \mathsf{hk}_2, \{\mathsf{ek}_{2,i}\}_{i\in[u]})$ and $\mathsf{td}_2 = \{\mathsf{td}_{2,i}\}_{i\in[u]}$.

- **Level 2** $\mathsf{Hash\&Enc}(\mathsf{k}_2, x, \mathsf{e}_1)$**:**

  – Parse $\mathsf{k}_2 = (2, \mathsf{hk}_2, \{\mathsf{ek}_{2,i}\}_{i\in[u]})$, and $\mathsf{e}_1 = \{\mathsf{e}_{1,i}\}_{i\in[u]}$.

  – Compute "second level" hash over $\{\mathbb{1}_{\mathsf{e}_{1,i}}\}_{i\in[u]}$, where $\mathbb{1}_{\mathsf{e}_{1,i}}$ is the indicator vector for $\mathsf{e}_{1,i}$.

$$\mathsf{h}_2 \leftarrow \mathsf{TDH.Hash}(\mathsf{hk}_2, \{\mathbb{1}_{\mathsf{e}_{1,i}}\}_{i\in[u]})$$

  – For any $i \in [u]$, compute "second level" encoding: $\mathsf{e}_{2,i} \leftarrow \mathsf{TDH.Enc}(\mathsf{ek}_{2,i}, \{\mathbb{1}_{\mathsf{e}_{1,j}}\}_{j\in[u]})$.

  – Output $(\mathsf{h}_2, \mathsf{e}_2)$, where $\mathsf{e}_2 = \{\mathsf{e}_{2,i}\}_{i\in[u]}$.

- **Decoding** $\mathsf{Dec}(\mathsf{td}_2, \mathsf{h}_2)$**:**

  – Parse $\mathsf{td}_2 = \{\mathsf{td}_{2,i}\}_{i\in[u]}$. For every $i \in [u]$, decode h_2: $\mathsf{d}_{2,i} \leftarrow \mathsf{TDH.Dec}(\mathsf{td}_{2,i}, \mathsf{h}_2)$.

  – Output $\mathsf{d} = \{\mathsf{d}_{2,i}\}_{i\in[u]}$.

This completes the description of ITDH. We prove that it achieves approximate correctness and leveled function privacy in Lemmas 5.3.1 and 5.3.2, respectively.

5.3.2 Proof of Approximate Correctness

Lemma 5.3.1 (Approximate Correctness)**.** *For any function $\Delta(u)$, the proposed protocol* ITDH *satisfies (Δ, ϵ)-approximate correctness, where*

$$\epsilon = 2\left(2e \cdot \max\{\tau(\lambda_1), \tau(\lambda_2)\} \cdot u/\Delta\right)^{\Delta/2} \cdot 2^{\lambda_1+\lambda_2}$$

Linear Function $\mathsf{XorSum}_{I,y}(x_1, \ldots, x_n)$ over $\mathbb{Z}_{R_1}$

- Let $y = (y_1, y_2, \ldots, y_n)$.
- Compute and output $\sum_{i \in I} x_i \cdot (1 - y_i) + (1 - x_i) \cdot y_i$.

Figure 5-2. Description of the linear function $\mathsf{XorSum}_{I,y}$. This function computes the sum over $\mathbb{Z}_{R_1}$ of I values obtained by bit-wise XOR of $y[I]$ and $x[I]$, where $x = (x_1, \ldots, x_n)$.

Linear Function $\mathsf{AddTh}_{i,t,\mathsf{d}}(\vec{\mathbb{e}})$ over $\mathbb{Z}_2$

- Let $\vec{\mathbb{e}} = (\mathbb{e}_1, \ldots, \mathbb{e}_u)$, where $\mathbb{e}_j \in \{0,1\}^{R_1}$ for every $j \in [u]$.
- Compute and output the inner product: $\langle \mathbb{e}_i, \mathbb{f} \rangle \bmod 2$, where $\mathbb{f} = \sum_{j=0}^{t-1} \mathbb{1}_{(j-\mathsf{d}) \bmod R_1}$ is the sum of indicator vectors for $(j - \mathsf{d}) \bmod R_1$, for $0 \leq j < t$.

Figure 5-3. Description of the linear function $\mathsf{AddTh}_{i,t,\mathsf{d}}$. For any $\mathsf{e}_1, \mathsf{e}_2, \ldots, \mathsf{e}_u \in \mathbb{Z}_{R_1}$, this function computes whether $(\mathsf{e}_i + \mathsf{d}) \bmod R_1$ is less than the threshold t. The actual input $\vec{\mathbb{e}}$ to the function is such that $\mathbb{e}_i$ is the indicator vector for e_i.

u is the output length of the circuit, and e is the base for natural logarithms.

Proof. We first establish some notation that we shall use throughout the proof. Whenever necessary, we augment a variable with $*$ in the superscript to denote the "ideal" value of the variable, whereas the "real" – and possibly *erroneous* – value is denoted without any emphasis.

Level 1. For each $i \in [u]$, let $\mathsf{sum}_i = (\mathsf{e}_{1,i} + \mathsf{d}_{1,i}) \bmod R_1$. By the τ-enhanced correctness of TDH, for any fixed hk_1, fixed h_1 and index i, we have

$$\Pr\Bigg[(\mathsf{ek}_{1,i}, \mathsf{td}_{1,i}) \leftarrow \mathsf{TDH.EKGen}(\mathsf{hk}_1, \mathsf{XorSum}_{I_i,y}) : \forall x : \mathsf{h}_1 = \mathsf{TDH.Hash}(\mathsf{hk}_1, x), \mathsf{sum}_i = \mathsf{XorSum}_{I_i,y}(x)\Bigg] > 1 - \tau(\lambda_1)$$

Denote $\mathsf{sum}_i^* = \mathsf{XorSum}_{I_i,y}(x)$. Then, for any fixed hk_1, and fixed h_1, since the encoding keys $\{\mathsf{ek}_{1,i}\}_{i \in [u]}$ are sampled independently, for any $\Delta' \in [u]$, we have

$$\Pr_{\{\mathsf{ek}_{1,i}\}_{i\in[u]}}\left[\exists x : \mathsf{h}_1 = \mathsf{TDH.Hash}(\mathsf{hk}_1, x), \mathsf{Ham}(\{\mathsf{sum}_i\}_{i\in[u]}, \{\mathsf{sum}_i^*\}_{i\in[u]}) > \Delta'\right]$$
$$< \tau(\lambda_1)^{\Delta'}\binom{u}{\Delta'} \leq \left(\frac{e\cdot\tau(\lambda_1)\cdot u}{\Delta'}\right)^{\Delta'},$$

where the second inequality follows from the upper bound for the combinatorial coefficients $\binom{u}{\Delta'} < (e\cdot u/\Delta')^{\Delta'}$.

Next, by taking the union bound on the choice of h_1, we have that for any fixed hk_1,

$$\Pr_{\{\mathsf{ek}_{1,i}\}_{i\in[u]}}\left[\exists x, \mathsf{Ham}\left(\{\mathsf{sum}_i\}_{i\in[u]}, \{\mathsf{sum}_i^*\}_{i\in[u]}\right) > \Delta'\right] < \left(\frac{e\cdot\tau(\lambda_1)\cdot u}{\Delta'}\right)^{\Delta'}\cdot 2^{\lambda_1}.$$

Finally, by averaging over all possible choices of hk_1, the above bound still holds when hk_1 is sampled from HKGen.

Level 2. Similarly, in the second level, let's consider two *arbitrary* encoding $\{\mathsf{e}'_{1,i}\}_{i\in[u]}$ and decoding $\{\mathsf{d}'_{1,i}\}_{i\in[u]}$. We will wire them with $\{\mathsf{e}_{1,i}\}_{i\in[u]}$ and $\{\mathsf{d}_{1,i}\}_{i\in[u]}$ in the first level later. Then we have

$$\Pr_{\mathsf{hk}_2,\{\mathsf{ek}_{2,i}\}_{i\in[u]}}\left[\exists\{\mathsf{e}'_{1,i}\}_{i\in[u]}, \mathsf{Ham}\left(\{\mathsf{out}_i\}_{i\in[u]}, \{\mathsf{out}_i^*\}_{i\in[u]}\right) > \Delta'\right] < \left(\frac{e\cdot\tau(\lambda_2)\cdot u}{\Delta'}\right)^{\Delta'}\cdot 2^{\lambda_2}, \tag{5.1}$$

where $\mathsf{out}_i^* = \mathsf{AddTh}_{i,t_i,\mathsf{d}'_{1,i}}(\{\mathbb{1}_{\mathsf{e}'_{1,j}}\}_{j\in[u]})$, and out_i is obtained by the following procedure.

- $\forall i \in [u], (\mathsf{ek}_{2,i}, \mathsf{td}_{2,i}) \leftarrow \mathsf{TDH.EKGen}(\mathsf{hk}_2, \mathsf{AddTh}_{i,t_i,\mathsf{d}'_{1,i}})$.
- $\mathsf{h}_2 \leftarrow \mathsf{TDH.Hash}(\mathsf{hk}_2, \{\mathbb{1}_{\mathsf{e}'_{1,i}}\}_{i\in[u]})$, $\forall i \in [u], \mathsf{e}_{2,i} \leftarrow \mathsf{TDH.Enc}(\mathsf{ek}_{2,i}, \{\mathbb{1}_{\mathsf{e}'_{1,j}}\}_{j\in[u]})$.
- $\forall i \in [u], \mathsf{d}_{2,i} \leftarrow \mathsf{TDH.Dec}(\mathsf{td}_{2,i}, \mathsf{h}_2)$.
- $\forall i \in [u], \mathsf{out}_i = (\mathsf{e}_{2,i} + \mathsf{d}_{2,i}) \bmod 2$.

Now, we fix the random coins r_1 used by the first level key generation algorithm. Let hk_1 and $\{\mathsf{ek}_{1,i}, \mathsf{td}_{1,i}\}_{i\in[u]}$ be the values generated by the first level key generation algorithm using randomness r_1. Let $\mathsf{e}_{1,i}$ be the first level encodings computed by the

sender using the keys $\mathsf{ek}_{1,i}$ and input x. Let out_i and out_i^* be as defined above, except that they are computed w.r.t. $\mathsf{e}_{1,i}$.

Then from Equation 5.1, for any fixed r_1 and fixed d_1', if we let $\{\mathsf{e}_{1,i}'\}_{i\in[u]} = \{\mathsf{e}_{1,i}\}_{i\in[u]}$, we have

$$\Pr_{\mathsf{hk}_2,\{\mathsf{ek}_{2,i}\}_{i\in[u]}}\left[\exists x : \mathsf{d}_1' = \{\mathsf{d}_{1,i}\}_{i\in[u]}, \mathsf{Ham}\left(\{\mathsf{out}_i\}_{i\in[u]}, \{\mathsf{out}_i^*\}_{i\in[u]}\right) > \Delta'\right] < \left(\frac{e\cdot\tau(\lambda_2)\cdot u}{\Delta'}\right)^{\Delta'}\cdot 2^{\lambda_2},$$

where $\mathsf{d}_{1,i} = \mathsf{TDH.Dec}(\mathsf{td}_{1,i}, \mathsf{h}_1)$, and we only consider every x such that $\{\mathsf{d}_{1,i}\}_{i\in[u]}$ derived from r_1 and x is equal to d_1'. To further remove such a constraint on x, we need to take an union bound on all possible choices of $\{\mathsf{d}_{1,i}\}_{i\in[u]}$.

Since $\mathsf{td}_{1,i}$ is fixed by r_1, the total possibilities of $\{\mathsf{d}_{1,i}\}_{i\in[u]}$ is bounded by the number of possible choices of h_1, which is at most 2^{λ_1}. Hence, applying the union bound on d_1', we derive that for any fixed r_1,

$$\Pr_{\mathsf{hk}_2,\{\mathsf{ek}_{2,i}\}_{i\in[u]}}\left[\exists x, \mathsf{Ham}\left(\{\mathsf{out}_i\}_{i\in[u]}, \{\mathsf{out}_i^*\}_{i\in[u]}\right) > \Delta'\right] < \left(\frac{e\cdot\tau(\lambda_2)\cdot u}{\Delta'}\right)^{\Delta'}\cdot 2^{\lambda_1+\lambda_2}$$

By averaging over all possible choices of r_1, this bound still holds when r_1 is sampled uniformly at random.

Putting it all together. From the above, except $(e\max\{\tau(\lambda_1),\tau(\lambda_2)\}\cdot u/\Delta')^{\Delta'}\cdot(2^{\lambda_1}+2^{\lambda_1+\lambda_2})$ fraction of the random coins, we have

$$\forall x, \mathsf{Ham}\left(\{\mathsf{sum}_i\}_{i\in[u]}, \{\mathsf{sum}_i^*\}_{i\in[u]}\right) \le \Delta' \text{ and } \mathsf{Ham}\left(\{\mathsf{out}_i\}_{i\in[u]}, \{\mathsf{out}_i^*\}_{i\in[u]}\right) \le \Delta'.$$

From the triangle inequality of Hamming distance, we have

$$\mathsf{Ham}\left(\{\mathsf{out}_i\}_{i\in[u]}, f(x)\right) \le \mathsf{Ham}\left(\{\mathsf{out}_i\}_{i\in[u]}, \{\mathsf{out}_i^*\}_{i\in[u]}\right) + \mathsf{Ham}\left(\{\mathsf{out}_i^*\}_{i\in[u]}, f(x)\right)$$

On the right hand side, the first term is bounded by Δ'. For the second term, out_i^* in fact only depends on sum_i. This is because $\mathsf{out}_i^* = \mathsf{AddTh}_{i,t_i,\mathsf{d}_{1,i}}(\{\mathbb{1}_{\mathsf{e}_{1,j}}\}_{j\in[u]})$.

Then by construction, AddTh outputs 1 if and only if $(\mathsf{e}_{1,i} + \mathsf{d}_{1,i}) \bmod R_1 < t_i$, which is equivalent to $\mathsf{sum}_i < t_i$.

Since $\mathsf{Ham}(\{\mathsf{sum}_i\}_{i\in[u]}, \{\mathsf{sum}_i^*\}_{i\in[u]}) \leq \Delta'$, we know that there are at most Δ' Hamming errors in $\{\mathsf{sum}_i\}_{i\in[u]}$, and these errors lead to at most Δ' Hamming errors in $\{\mathsf{out}_i^*\}_{i\in[u]}$. Therefore, we obtain that $\mathsf{Ham}\left(\{\mathsf{out}_i^*\}_{i\in[u]}, f(x)\right) \leq \Delta'$.

Hence, we have

$$\begin{aligned}
&\Pr_{r_1,\mathsf{hk}_2,\{\mathsf{ek}_{2,i}\}_{i\in[u]}}[\exists x, \mathsf{Ham}(\{\mathsf{out}_i\}_{i\in[u]}, f(x)) > 2\Delta'] \\
&< (e \cdot \max\{\tau(\lambda_1), \tau(\lambda_2)\} \cdot u/\Delta')^{\Delta'} \cdot (2^{\lambda_1} + 2^{\lambda_1+\lambda_2}) \\
&< 2\,(e \cdot \max\{\tau(\lambda_1), \tau(\lambda_2)\} \cdot u/\Delta')^{\Delta'} \cdot 2^{\lambda_1+\lambda_2}
\end{aligned}$$

By letting $\Delta' = \Delta(u)/2$, we finish the proof. □

5.3.3 Proof of Leveled Function Privacy

Lemma 5.3.2 (Leveled Function Privacy)**.** *The proposed protocol* ITDH *satisfies leveled function privacy property.*

Proof. We build the simulator $\mathsf{Sim}(1^{\lambda_\ell}, 1^n, 1^u, \ell)$ in Figure 5-4.

Simulator $\mathsf{Sim}(1^{\lambda_\ell}, 1^n, 1^u, \ell)$

- If $\ell = 1$, let $n_\ell = n, R_\ell = R_1 = n$. Otherwise let $n_\ell = R_1 \cdot u = n \cdot u, R_\ell = 2$.
- Sample a hash key: $\mathsf{hk}_\ell \leftarrow \mathsf{TDH.HKGen}(1^{\lambda_\ell}, 1^{n_\ell}, 1^{R_\ell})$
- For each $i \in [u]$, compute a simulated encoding key:

$$\mathsf{ek}_{\ell,i} \leftarrow \mathsf{TDH.Sim}(1^{\lambda_\ell}, 1^{n_\ell}, 1^{R_\ell})$$

- Output $\mathsf{k}_\ell = (\ell, \mathsf{hk}_\ell, \{\mathsf{ek}_{\ell,i}\}_{i\in[u]})$.

Figure 5-4. Simulator $\mathsf{Sim}(1^\lambda, 1^n, 1^u, \ell)$

We construct a series of hybrids to prove that the output of Sim is indistinguishable

from an honestly sampled key for any circuit $\mathrm{T}^{\oplus y}_{\vec{t},\vec{I}} \in \mathcal{T}^{\oplus}_{n,u}$. We only prove indistinguishability for the first level, or $\ell = 1$. The proof for the second level ($\ell = 2$) follows similarly.

Hyb_0: This is the "real world", where the keys are computed using the key generation algorithm $\mathsf{KGen}(1^{\lambda_1}, 1, \mathrm{T}^{\oplus y}_{\vec{t},\vec{I}}, h_0 = \bot, \mathsf{td}_0 = \bot)$.

$\mathsf{Hyb}^{i^*}_1$: In this hybrid, for every $i < i^*$, the encoding key $\mathsf{ek}_{1,i}$ is computed using the simulator $\mathsf{TDH.Sim}$. For every $i \geq i^*$, the encoding key $\mathsf{ek}_{1,i}$ is computed honestly using $\mathsf{TDH.EKGen}$.

- For each $i < i^*$, let $\mathsf{ek}_{1,i} \leftarrow \mathsf{TDH.Sim}(1^{\lambda_1}, 1^{n_1}, 1^{R_1})$.
- For each $i \geq i^*$, let $(\mathsf{ek}_{1,i}, \mathsf{td}_{1,i}) \leftarrow \mathsf{TDH.EKGen}(\mathsf{hk}_1, \mathsf{XorSum}_{I_i,y})$.

Hyb_2: This hybrid is the same as the simulator $\mathsf{Sim}(1^{\lambda_1}, 1^n, 1^u, 1)$.

From the description of the hybrids, it follows that Hyb_0 is identical to Hyb^1_1, and Hyb_2 is identical to Hyb^{u+1}_1. Hence, it suffices to show that $\mathsf{Hyb}^{i^*}_1$ and $\mathsf{Hyb}^{i^*+1}_1$ are indistinguishable. For $i^* \in [u]$, suppose that there exists a n.u. PPT distinguisher $\mathcal{D}$ that distinguishes between $\mathsf{Hyb}^{i^*}_1$ and $\mathsf{Hyb}^{i^*+1}_1$ with non-negligible advantage $\delta(\lambda_1)$. We build a distinguisher $\mathcal{D}'$ who breaks the function privacy of TDH.

The distinguisher $\mathcal{D}'(1^{\lambda_1}, (\mathsf{hk}, \mathsf{ek}))$ takes as input the security parameter λ_1, a hash key hk and an encoding key ek. For each $i < i^*$, it runs the simulator $\mathsf{TDH.Sim}(1^{\lambda_1}, 1^{n_1}, 1^{R_1})$ to generate an encoding key $\mathsf{ek}_{1,i}$. For $i = i^*$, it sets $\mathsf{ek}_{1,i} = \mathsf{ek}$ as the encoding key. For each $i > i^*$, it computes an encoding key $(\mathsf{ek}_{1,i}, \mathsf{td}_{1,i}) \leftarrow \mathsf{TDH.EKGen}(\mathsf{hk}, \mathsf{XorSum}_{I_i,y})$ using the key generation algorithm. Finally, $\mathcal{D}'$ runs distinguisher $\mathcal{D}$ with input $(1, \mathsf{hk}, \{\mathsf{ek}_{1,i}\}_{i\in[u]})$, and returns the output of $\mathcal{D}$.

Now, if ek in $(\mathsf{hk}, \mathsf{ek})$ is generated using $\mathsf{TDH.EKGen}$, then $\mathcal{D}'$ successfully simulates the hybrid $\mathsf{Hyb}^{i^*}_1$ for the distinguisher $\mathcal{D}'$. Hence,

$$\Pr\left[\mathcal{D}(1^{\lambda_1}, \mathsf{Hyb}^{i^*}_1) = 1\right] = \Pr\left[\begin{smallmatrix}\mathsf{hk}\leftarrow\mathsf{TDH.HKGen}(1^{\lambda_1},1^{n_1},1^{R_1}),\\ (\mathsf{ek},\mathsf{td})\leftarrow\mathsf{TDH.EKGen}(\mathsf{hk},\mathsf{XorSum}_{I_{i^*},y})\end{smallmatrix} : \mathcal{D}'(1^{\lambda_1}, (\mathsf{hk}, \mathsf{ek})) = 1\right]$$

On the other hand, if ek is generated using $\mathsf{TDH.Sim}$, then $\mathcal{D}'$ successfully simulates the hybrid $\mathsf{Hyb}_1^{i^*+1}$ for the distinguisher $\mathcal{D}'$. Hence,

$$\Pr\left[\mathcal{D}(1^{\lambda_1}, \mathsf{Hyb}_1^{i^*+1}) = 1\right] = \Pr\left[\begin{smallmatrix}\mathsf{hk}\leftarrow\mathsf{TDH.HKGen}(1^{\lambda_1},1^{n_1},1^{R_1}),\\ \mathsf{ek}\leftarrow\mathsf{TDH.Sim}(1^{\lambda_1},1^{n_1},1^{R_1})\end{smallmatrix} : \mathcal{D}'(1^{\lambda_1}, (\mathsf{hk}, \mathsf{ek})) = 1\right]$$

Since TDH achieves function privacy, the difference of the probabilities on the right hand side of the above two equations is bounded by a negligible function. Hence, there exists a negligible function $\nu(\cdot)$ such that $|\Pr[\mathcal{D}'(1^{\lambda_1}, \mathsf{Hyb}_1^{i^*}) = 1] - \Pr[\mathcal{D}'(1^{\lambda_1}, \mathsf{Hyb}_1^{i^*+1}) = 1]| \leq \nu(\lambda_1)$.

Since we have u hybrids, and u is polynomial in λ, we conclude that the construction satisfies leveled function privacy. □

Remark 5.3.3. *If the underlying* TDH *satisfies sub-exponential leveled function privacy, then the proposed construction of* ITDH *also satisfies the sub-exponential leveled function privacy.*

5.3.4 ITDH Composition

In this section, we establish a sequential composition theorem for ITDH. Roughly speaking, we show how a 2-level ITDH for an "Xor-*then*-Compute" circuit family can be executed sequentially L times to obtain an ITDH for a related circuit family (the exact transformation is more nuanced; see below). The main benefit of sequential composition is that it can be used to increase the depth of circuits that can be computed by ITDH.

We start by introducing some notation and terminology for circuit composition that we shall use in the sequel.

Parallel Composition. Let w be a positive integer. Informally, a w-parallel composition circuit f is a structured circuit that computes w circuits $f'_1, f'_2, \ldots, f'_w$ in parallel. More formally, for any circuit family $\mathcal{C}$, we define a corresponding parallel-composition circuit family as follows:

Definition 5.3.4 (Parallel Composition)**.** *For any circuit family $\mathcal{C}$ and any polynomial $w = w(n)$, we say that $\mathcal{C}[\overrightarrow{w}] = \{\mathcal{C}[\overrightarrow{w}]_{n,u}\}_{n,u}$ is a family of w-parallel composition circuits if for every $f \in \mathcal{C}[\overrightarrow{w}]_{n,u}$, there exists a sequence of circuits $f'_1, f'_2, \ldots, f'_w \in \mathcal{C}_{n',u'}$ such that $n = n' \cdot w(n)$ and $u = u' \cdot w(n)$, and for any input $x = (x_1, x_2, \ldots, x_w) \in \{0,1\}^{n=n' \cdot w}$ (where every $x_i \in \{0,1\}^{n'}$), we have*

$$f(x_1, x_2, \ldots, x_w) = (f'_1(x_1), f'_2(x_2), \ldots, f'_w(x_w)).$$

Parallel-and-Sequential-Composition. For any circuit family $\mathcal{C}$, we now define another circuit family obtained via parallel *and* sequential composition of circuits in $\mathcal{C}$.

Informally speaking, for any polynomials $w(n)$ and $L(n)$ and an integer s, a w-parallel-and-L-sequential-composition of a circuit family $\mathcal{C}$ is a new circuit family $\mathcal{C}[\overrightarrow{\substack{w \\ \downarrow L}}] = \{\mathcal{C}[\overrightarrow{\substack{w \\ \downarrow L}}]_{n,s}\}_{n,s}$, where each circuit $f \in \mathcal{C}[\overrightarrow{\substack{w \\ \downarrow L}}]_{n,s}$ is computed by a sequence of circuits $f_1, f_2, \ldots, f_L$. For any input x, to compute $f(x)$, we firstly evaluate f_1 on input x, then use the output $f_1(x)$ as the input to the circuit f_2, and so on, such that the output of f_L is the output of f. Furthermore, we require that for every $\ell \in [L]$, f_ℓ is an m-parallel composition of some sequence of circuits $f'_{\ell,1}, f'_{\ell,2}, \ldots, f'_{\ell,w} \in \mathcal{C}$. For the ease of presentation, we fix the output length of the circuit f_ℓ for every $\ell < L$ as s, and the output length of f as w.

Definition 5.3.5 (Parallel-and-Sequential-Composition)**.** *Let $\mathcal{C} = \{\mathcal{C}_{n,u}\}_{n,u}$ be a circuit family, where each circuit in $\mathcal{C}_{n,u}$ has input length n and output length u. For any polynomials $w = w(n), L = L(n)$, and integer s, we say that $\mathcal{C}[\overrightarrow{\substack{w \\ \downarrow L}}] = \{\mathcal{C}[\overrightarrow{\substack{w \\ \downarrow L}}]_{n,s}\}_{n,s}$ is a family of* w-parallel-and-L-sequential-composition circuits *if every circuit $f \in \mathcal{C}[\overrightarrow{\substack{w \\ \downarrow L}}]_{n,s}$ is of the form*

$$f = f_L \circ f_{L-1} \circ \ldots \circ f_1$$

where for every $\ell \in [L]$, $f_\ell : \{0,1\}^{n_\ell} \to \{0,1\}^{n_{\ell+1}}$ satisfies $n_1 = n, n_2 = n_3 = \ldots = n_{L-1} = s, n_L = w$. Furthermore, there exists a sequence of integers $\{n'_\ell\}_\ell$ and circuits

$\{f'_{\ell,j}\}_{\ell\in[L],j\in[w]}$, *where* $f'_{\ell,j} \in \mathcal{C}_{n'_\ell,n'_{\ell+1}}$, *and* $n_\ell = n'_\ell \cdot w$,

$$f_\ell(x_1,\ldots,x_w) = \Big(f'_{\ell,1}(x_1), f'_{\ell,2}(x_2),\ldots,f'_{\ell,w}(x_w)\Big)$$

for every $x = (x_1,\ldots,x_w) \in \{0,1\}^{n'_\ell\cdot w}$, *where* $x_i \in \{0,1\}^{n'_\ell}$ *for every* $i \in [w]$.

Construction of ITDH for $\mathcal{C}[\overrightarrow{w}_{\downarrow L}]$. Let $\mathcal{C} = \{\mathcal{C}_{n,u}\}_{n,u}$ be any circuit family, and let $\mathcal{C}[\overrightarrow{w}]$ be the corresponding w-parallel composition circuit family. Let $\mathcal{C}[\overrightarrow{w}]^\oplus = \{\mathcal{C}[\overrightarrow{w}]^\oplus_{n,u}\}_{n,u}$ be the "Xor-*then*-Compute" circuit family defined w.r.t. $\mathcal{C}[\overrightarrow{w}]$. Let $\mathsf{ITDH} = (\mathsf{ITDH.KGen}, \mathsf{ITDH.Hash\&Enc}, \mathsf{ITDH.Dec})$ be a 2-level interactive trapdoor hashing protocol for $\mathcal{C}[\overrightarrow{w}]^\oplus = \{\mathcal{C}[\overrightarrow{w}]^\oplus_{n,u}\}_{n,u}$ with (Δ,ϵ)-approximate correctness and leveled function privacy.

Given ITDH, we construct a $2L$-level interactive trapdoor hashing protocol $\mathsf{ITDH}' = (\mathsf{KGen}, \mathsf{Hash\&Enc}, \mathsf{Dec})$ for the circuit family $\mathcal{C}[\overrightarrow{w}_{\downarrow L}]$ as defined above. For ease of exposition, we describe the algorithms of ITDH' for "odd" and "even" levels separately.

- **Level $\ell' = 2\ell - 1$, $\mathsf{KGen}(1^{\lambda_{\ell'}}, \ell', f, \mathsf{h}_{\ell'-1}, \mathsf{td}_{\ell'-1})$:**
  - If $\ell = 1$, set d_0 to be an all zero string of length n.
  - If $\ell \geq 2$, decode $\mathsf{h}_{\ell'-1}$: $\mathsf{d}_{\ell-1} \leftarrow \mathsf{ITDH.Dec}(\mathsf{td}_{\ell'-1}, \mathsf{h}_{\ell'-1})$
  - Let $f_1,\ldots,f_L$ be such that $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ (as defined above), where f_ℓ has input length n_ℓ and output length $n_{\ell+1}$.
  - Compute a key w.r.t. security parameter $\lambda_{\ell'}$ and the "Xor-*then*-Compute" circuit $f_\ell^{\oplus \mathsf{d}_{\ell-1}} \in \mathcal{C}[\overrightarrow{w}]^\oplus_{n_\ell,n_{\ell+1}}$

    $$(\mathsf{k}_{\ell,1}, \mathsf{td}_{\ell,1}) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_{\ell'}}, 1, f_\ell^{\oplus \mathsf{d}_{\ell-1}}, \perp, \perp).$$

  - Output $(\mathsf{k}_{\ell'}, \mathsf{td}_{\ell'})$ where $\mathsf{k}_{\ell'} = (\ell', \mathsf{k}_{\ell,1})$ and $\mathsf{td}_{\ell'} = \mathsf{td}_{\ell,1}$.
- **Level $\ell' = 2\ell - 1$, $\mathsf{Hash\&Enc}(\mathsf{k}_{\ell'}, x, \mathsf{e}_{\ell'-1})$:**

– If $\ell = 1$, let $x_\ell = x$, otherwise, let $x_\ell = \mathsf{e}_{\ell'-1}$. Execute

$$(\mathsf{h}_{\ell,1}, \mathsf{e}_{\ell,1}) \leftarrow \mathsf{ITDH.Hash\&Enc}(\mathsf{k}_{\ell,1}, x, \perp)$$

– Output $(\mathsf{h}_\ell = \mathsf{h}_{\ell,1}, \mathsf{e}_\ell = (x_\ell, \mathsf{e}_{\ell,1}))$.

- **Level $\ell' = 2\ell$, $\mathsf{KGen}(1^{\lambda_{\ell'}}, \ell', f, \mathsf{h}_{\ell'-1}, \mathsf{td}_{\ell'-1})$:**

  – Parse $\mathsf{h}_{\ell'-1} = \mathsf{h}_{\ell,1}$, and $\mathsf{td}_{\ell'-1} = \mathsf{td}_{\ell,1}$.

  $$(\mathsf{k}_{\ell,2}, \mathsf{td}_{\ell,2}) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_{\ell'}}, 2, f_\ell^{\oplus \mathsf{d}_{\ell-1}}, \mathsf{h}_{\ell,1}, \mathsf{td}_{\ell,1})$$

  – Output $(\mathsf{k}_{\ell'}, \mathsf{td}_{\ell'})$, where $\mathsf{k}_{\ell'} = (\ell', \mathsf{k}_{\ell,2})$, and $\mathsf{td}_{\ell'} = \mathsf{td}_{\ell,2}$.

- **Level $\ell' = 2\ell$, $\mathsf{Hash\&Enc}(\mathsf{k}_{\ell'}, x, \mathsf{e}_{\ell'-1})$:**

  – Parse $\mathsf{e}_{\ell'-1} = (x_\ell, \mathsf{e}_{\ell,1}), \mathsf{k}_{\ell'} = \mathsf{k}_{\ell,2}$.

  – Output $(\mathsf{h}_{\ell'}, \mathsf{e}_{\ell'}) \leftarrow \mathsf{Hash\&Enc}(\mathsf{k}_{\ell,2}, x_\ell, \mathsf{e}_{\ell,1})$.

- **Decoding $\mathsf{Dec}(\mathsf{td}_{2L}, \mathsf{h}_{2L})$:**

  – Output $\mathsf{d} \leftarrow \mathsf{ITDH.Dec}(\mathsf{td}_{2L}, \mathsf{h}_{2L})$.

This completes the description of ITDH'.

5.3.5 Proof of Approximate Correctness

Lemma 5.3.6 (Approximate Correctness)**.** *For any circuit $f \in \mathcal{C}[\vec{w}]_{n,s}$, ITDH' satisfies (Δ', ϵ')-approximate correctness, where*

$$\Delta' = \sum_{\ell \in [L]} \Delta_\ell(n_{\ell+1}), \qquad \epsilon' = \sum_{\ell \in [L]} \epsilon_\ell(n_{\ell+1}, \lambda_{2\ell-1}, \lambda_{2\ell}) \cdot 2^{\lambda_1 + \lambda_2 + \ldots + \lambda_{2\ell-2}}$$

Proof. We start by bounding the error at each level ℓ.

Bounding error at ℓ-th level. For each $\ell \in [L]$, let $r_{2\ell-1}$ and $r_{2\ell}$ be the random coins used for the KGen in the $(2\ell-1)^{\text{th}}$ level and $2\ell^{\text{th}}$ level, respectively. For each

$\ell \in [L]$, let $\ell' = 2\ell - 1$ be the starting level number for f_ℓ. Since the underlying ITDH satisfies $(\Delta_\ell(u), \epsilon_\ell(u, \lambda_1, \lambda_2))$-approximate correctness, for any fixed $\ell \in [L]$, and any fixed $\mathsf{d}'_{\ell-1}$ we have

$$\Pr_{r_{\ell'}, r_{\ell'+1}}\left[\exists x'_\ell : \mathsf{Ham}(\mathsf{e}_{2\ell} \oplus \mathsf{d}_\ell, f_\ell^{\oplus \mathsf{d}'_{\ell-1}}(x'_\ell)) > \Delta_\ell(n_{\ell+1})\right] < \epsilon_\ell(n_{\ell+1}, \lambda_{\ell'}, \lambda_{\ell'+1}),$$

where the $\mathsf{e}_{2\ell}$ and d_ℓ are obtained by executing the ITDH protocol with sender's input x'_ℓ, and receiver's input $f_\ell^{\oplus d'_{\ell-1}}$, and the randomness is over the random coins $r_{\ell'}, r_{\ell'+1}$.

Hence, if we fix the random coins $r_1, r_2, \ldots, r_{\ell'-1}$, and also fix $\mathsf{d}'_{\ell-1}$, then we have

$$\Pr_{r_{\ell'}, r_{\ell'+1}}\left[\exists x : \mathsf{d}_{\ell-1} = \mathsf{d}'_{\ell-1}, \mathsf{Ham}(\mathsf{e}_{2\ell} \oplus \mathsf{d}_\ell, f_\ell^{\oplus \mathsf{d}'_{\ell-1}}(x_\ell)) > \Delta_\ell(n_{\ell+1})\right] < \epsilon_\ell(n_{\ell+1}, \lambda_{\ell'}, \lambda_{\ell'+1}),$$

where the $\mathsf{e}_{2\ell}$ and d_ℓ are obtained by executing the ITDH' protocol to the $(\ell'+1)^{\text{th}}$ level, with sender's input x, receiver's input f, and the random coins $r_1, r_2, \ldots, r_{\ell'-1}, r_{\ell'}, r_{\ell'+1}$. Note that in this probability, we only consider all x such that d_ℓ obtained from the execution equals to the fixed $\mathsf{d}'_{\ell-1}$. To further remove this restriction on x, we need to take an union bound on all possible choice of $\mathsf{d}_{\ell-1}$.

Since we fixed $r_1, r_2, \ldots, r_{\ell'-1}$, the decoding $\mathsf{d}_{\ell-1}$ only depends on $\mathsf{h}_1, \mathsf{h}_2, \ldots, \mathsf{h}_{\ell'-1}$. Hence, the total number of possibilities of $\mathsf{d}_{\ell-1}$ is bounded by $2^{\lambda_1+\lambda_2+\ldots+\lambda_{\ell'-1}}$. By taking an union bound, for any fixed $r_1, r_2, \ldots, r_{\ell'-1}$, we have

$$\Pr_{r_{\ell'}, r_{\ell'+1}}\left[\exists x, \mathsf{Ham}(\mathsf{e}_{2\ell} \oplus \mathsf{d}_\ell, f_\ell^{\oplus \mathsf{d}_{\ell-1}}(x_\ell)) > \Delta_\ell(n_{\ell+1})\right] < \epsilon_\ell(n_{\ell+1}, \lambda_{\ell'}, \lambda_{\ell'+1}) \cdot 2^{\lambda_1+\lambda_2+\ldots+\lambda_{\ell'-1}},$$

By averaging over all possibilities of $r_1, r_2, \ldots, r_{\ell'-1}$, the above inequality still holds when $r_1, r_2, \ldots, r_{\ell'-1}$ are sampled uniformly at random.

Now, except with probability

$$\sum_{\ell \in [L]} \epsilon_\ell(n_{\ell+1}, \lambda_{\ell'}, \lambda_{\ell'+1}) \cdot 2^{\lambda_1+\lambda_2+\ldots+\lambda_{\ell'-1}},$$

we have that for any x, and any $\ell \in [L]$, $\mathsf{Ham}(\mathsf{e}_{2\ell} \oplus \mathsf{d}_\ell, f_\ell^{\oplus \mathsf{d}_{\ell-1}}(x_\ell)) \leq \Delta(n_{\ell+1})$.

Controlling the Error Spread. For each $\ell \in [L]$, denote out^*_ℓ as the ideal (intermediate) outputs, $\mathsf{out}^*_\ell = f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1(x)$. Then we have $\mathsf{out}^*_\ell = f_\ell(\mathsf{out}^*_{\ell-1})$. We

denote the real (intermediate) outputs as $\mathsf{out}_\ell = \mathsf{e}_{2\ell} \oplus \mathsf{d}_\ell$ in the honest execution. Next, instead of bounding the Hamming distance out_ℓ and out^*_ℓ directly, we bound the following Hamming distance Ham_w over a larger alphabet.

For any two strings $a, b \in \{0,1\}^n$, where $n = n' \cdot w$, we firstly "partition" a as $a = (a_1, a_2, \ldots, a_w)$ where $a_i \in \{0,1\}^{n'}, i \in [w]$, and b as $b = (b_1, b_2, \ldots, b_w)$ where $b_i \in \{0,1\}^{n'}, i \in [w]$. We define the Hamming distance Ham_w between a and b as the number of index $i \in [w]$ such that a_i and b_i differ. Then we have

$$\begin{aligned}
\mathsf{Ham}_w(\mathsf{out}_\ell, \mathsf{out}^*_\ell) &\leq \mathsf{Ham}_w(\mathsf{out}_\ell, f_\ell^{\oplus \mathsf{d}_{\ell-1}}(x_\ell)) + \mathsf{Ham}_w(f_\ell^{\oplus \mathsf{d}_{\ell-1}}(x_\ell), \mathsf{out}^*_\ell) && (5.2)\\
&= \mathsf{Ham}_w(\mathsf{e}_{2\ell} \oplus \mathsf{d}_\ell, f_\ell^{\oplus \mathsf{d}_{\ell-1}}(x_\ell)) + \mathsf{Ham}_w(f_\ell^{\oplus \mathsf{d}_{\ell-1}}(x_\ell), f_\ell(\mathsf{out}^*_{\ell-1})) && (5.3)\\
&\leq \Delta(n_{\ell+1}) + \mathsf{Ham}_w(f_\ell(x_\ell \oplus \mathsf{d}_{\ell-1}), f_\ell(\mathsf{out}^*_{\ell-1})) && (5.4)\\
&\leq \Delta(n_{\ell+1}) + \mathsf{Ham}_w(\mathsf{out}_{\ell-1}, \mathsf{out}^*_{\ell-1}) && (5.5)
\end{aligned}$$

The first inequality comes from the triangular inequality of the Hamming distance. The second line follows from the definition of out_ℓ and out^*_ℓ. The third line follows from the bound between $\mathsf{e}_{2\ell} \oplus \mathsf{d}_\ell$ and $f_\ell^{\oplus \mathsf{d}_{\ell-1}}(x_\ell)$, and the definition of the circuit $f_\ell^{\oplus \mathsf{d}_{\ell-1}}$. The fourth line follows from the fact that f_ℓ is a w-parallel composition circuit, hence, the "partitioned" Hamming errors between the output of f_ℓ is bounded by the "partitioned" Hamming errors between the input of f_ℓ. Recursively applying the Equation 5.5, we have

$$\mathsf{Ham}(\mathsf{e}_{2L} \oplus \mathsf{d}, f(x)) = \mathsf{Ham}_w(\mathsf{out}_L, \mathsf{out}^*_L) \leq \sum_{\ell \in [L]} \Delta(n_{\ell+1}).$$

By the definition of approximate correctness, we finish the proof. □

5.3.6 Proof of Leveled Function Privacy

Lemma 5.3.7 (Leveled Function Privacy)**.** *The construction above satisfies leveled function privacy.*

Proof. Since each key in construction ITDH' is also a key of ITDH, the leveled function-privacy follows directly from the leveled function privacy of the underlying protocol ITDH. □

Remark 5.3.8. *If the underlying* ITDH *satisfies sub-exponential leveled function privacy, then the* ITDH' *also satisfies sub-exponential leveled function privacy.*

5.3.7 ITDH for TC^0

We now describe how we can put the above constructions together to obtain an ITDH for TC^0. Recall that, we use the notation TC^0_L to denote the class of L-depth TC^0 circuits.

Let $\mathcal{T}[\vec{w}_{\downarrow L}]$ be the circuit family obtained by w-parallel-and-L-sequential composition of the circuit family $\mathcal{T}$, as per Definition 5.3.5. We first show that any circuit in TC^0_L can be converted to a circuit in $\mathcal{T}[\vec{w}_{\downarrow L}]$.

Lemma 5.3.9. TC^0_L *can be computed in* $\mathcal{T}[\vec{w}_{\downarrow L}]$*. Specifically, for any circuit* $f \in \mathsf{TC}^0_L$ *with* n *bit input and* w *output bits, we convert it in polynomial time to a circuit* $f' \in \mathcal{T}[\vec{w}_{\downarrow L}]$ *such that, for any* $x \in \{0,1\}^n$*,* $f(x) = f'(x, x, \ldots, x)$*.*

Proof. For any circuit $f \in \mathsf{TC}^0_L$ with w output bits, we can always convert it to a layered circuit (of the same depth L). Hence, we obtain a series of depth-1 circuits $f'_1, f'_2, \ldots, f'_L$ such that $f' = f'_L \circ f'_{L-1} \circ \ldots f'_1$. To make it a w-parallel-and-L-sequential-composition circuit, we repeat the input for w times, and for every $j \in [w]$, we compute the j^{th} output bit of $f'_L \circ f'_{L-1} \circ \ldots \circ f'_1$ on the j^{th} repetition of the input. □

Next, we combine the construction of ITDH for the circuit family $\mathcal{T}^{\oplus}$ from Section 5.3.1 together with the sequential composition theorem in section 5.3.4 to obtain an ITDH for the circuit family $\mathcal{T}[\vec{w}_{\downarrow L}]$, and therefore an ITDH for TC^0_L.

Theorem 5.3.10. *If for any inverse polynomial τ in the security parameter, there exists a trapdoor hash function* TDH *for linear function family $\mathcal{F}$ (as defined in Definition 2.4.3) with τ-enhanced correctness and sub-exponential function privacy, then for any constants $L = O(1)$, $\alpha = O(1)$, and any polynomial w in the security parameter, there exists a $2L$-level interactive trapdoor hashing protocol for TC^0_L that achieves (Δ, ϵ)-approximate correctness and sub-exponential function privacy, where $\Delta(w) = \alpha \cdot w$ and for any $\lambda_1 < \lambda_2 < \ldots < \lambda_{2L} < w/2L$, $\epsilon(w, \lambda_1, \ldots, \lambda_L) = 2^{-2w+O(1)}$.*

Proof. By Lemma 5.3.9, since each circuit in TC^0_L can be converted to a circuit in $\mathcal{T}[\vec{w}_{\downarrow L}]$, it suffices to construct ITDH for $\mathcal{T}[\vec{w}_{\downarrow L}]$. Since the circuit family $\mathcal{T}[\vec{w}]^{\oplus}$ is a subset of $\mathcal{T}^{\oplus}$, we combine the ITDH for $\mathcal{T}^{\oplus}$ with the generic composition in section 5.3.4. From Lemma 5.3.1, we have that the interactive trapdoor hashing protocol ITDH for circuit family $\mathcal{T}^{\oplus}$ satisfies $(\Delta, \epsilon = 2(2e \cdot \max\{\tau(\lambda_1), \tau(\lambda_2)\} \cdot u/\Delta)^{\Delta/2} \cdot 2^{\lambda_1+\lambda_2})$-approximate correctness for any Δ. Setting $\Delta_\ell(n_{\ell+1}) = \alpha w/L$, we have

$$\epsilon_\ell(u, \lambda_{2\ell-1}, \lambda_{2\ell}) = 2\left(\frac{2eL \cdot \tau_\ell \cdot u}{\alpha w}\right)^{\frac{\alpha w}{2L}} \cdot 2^{\lambda_{2\ell-1}+\lambda_{2\ell}},$$

where $\tau_\ell = \max\{\tau(\lambda_{2\ell-1}), \tau(\lambda_{2\ell})\}$.

From Lemma 5.3.6, for any security parameters $\lambda_1 < \lambda_2 < \ldots < \lambda_{2L}$, we have that ITDH$'$ satisfies (Δ', ϵ')-approximate correctness, where

$$\begin{aligned}
\Delta'(m) &= \sum_{\ell \in [L]} \Delta_\ell(n_{\ell+1}) \leq L \cdot \alpha w/L \leq \alpha w \\
\epsilon'(m, \lambda_1, \lambda_2, \ldots, \lambda_{2L}) &= \sum_{\ell \in [L]} \epsilon_\ell(n_{\ell+1}, \lambda_{2\ell-1}, \lambda_{2\ell}) \cdot 2^{\lambda_1+\lambda_2+\ldots+\lambda_{2\ell-2}} \\
&\leq 2L \cdot \left(\frac{2eL \cdot \tau' \cdot s}{\alpha w}\right)^{\frac{\alpha w}{2L}} \cdot 2^{2L \cdot \lambda_{2L}},
\end{aligned}$$

where $\tau' = \max\{\tau(\lambda_1), \tau(\lambda_2), \ldots, \tau(\lambda_{2L})\}$, and s is the upper bound for n_ℓ. We set τ such that

$$\tau' < \frac{2^{-6L/\alpha} \alpha w}{2eL \cdot s},$$

which is an inverse polynomial. Hence, there exists a TDH construction with τ-enhanced correctness. Since $2L \cdot \lambda_{2L} < w$, we bound ϵ' by $\epsilon' < 2L \cdot 2^{-3w} \cdot 2^w < 2^{-2w+O(1)}$.

For the function privacy, since we assume sub-exponential leveled function privacy for the TDH, by Remark 5.3.3, the ITDH satisfies sub-exponential function privacy. Then by Remark 5.3.8, the ITDH' construction satisfies sub-exponential function privacy. Since s is a polynomial in λ, we prove the theorem. □

ITDH for P/poly. Since any circuit in P/poly can be converted to a layered circuit as in Lemma 5.3.9, the above construction of ITDH for TC^0 can be naturally extended to obtain a polynomial-level ITDH for P/poly.

5.4 Correlation Intractable Hash Functions for TC^0

In this section, we build correlation intractable hash functions for the circuit family TC^0.

5.4.1 Definition

Correlation intractable hash (CIH) function is a tuple of algorithms $\mathsf{CIH} = (\mathsf{Gen}, \mathsf{Hash})$ described as follows:

- $\mathsf{Gen}(1^\lambda)$: It takes as input a security parameter λ and outputs a key k.
- $\mathsf{Hash}(\mathsf{k}, x)$: It takes as input a hash key k and a string x, and outputs a binary string y of length $w = w(\lambda)$.

We require CIH to satisfy the following property:

- **Correlation Intractability:** Recall that, a binary relation R is a subset of $\{0,1\}^* \times \{0,1\}^*$. We say that CIH is correlation intractable for a class of binary relations $\{\mathcal{R}_\lambda\}_\lambda$ if there exists a negligible function $\nu(\lambda)$ such that, for any $\lambda \in \mathbb{N}$, any n.u. PPT adversary $\mathcal{A}$, and any $R \in \mathcal{R}_\lambda$,

$$\Pr\left[\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda), x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k}) : (x, \mathsf{Hash}(\mathsf{k}, x)) \in R\right] \leq \nu(\lambda)$$

We say that the CIH is sub-exponential correlation intractable, if there exists a constants c such that for any n.u. PPT adversary, its successful probability is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

Definition 5.4.1 (CIH for TC^0)**.** *Let $n(\lambda), w(\lambda)$ be polynomials. Let $L = O(1)$ be a constant. Recall that, we use TC^0_L to denote the class of L-depth threshold circuits. We say that CIH is a CIH for TC^0_L, if CIH is correlation intractable for the class of relations $\{\mathcal{R}_\lambda\}_\lambda$, where $\mathcal{R}_\lambda = \{R_{f,\lambda} \mid f \in \mathsf{TC}^0_L\}$, and*

$$R_{f,\lambda} = \{(x, y) \in \{0,1\}^{n(\lambda)} \times \{0,1\}^{w(\lambda)} \mid y = f(x)\}$$

5.4.2 Our Construction

For any $L = O(1)$, we show a generic transformation from an L-level ITDH for TC^0_L to a CIH for the same circuit family.

CIH for TC^0. Let $\mathsf{ITDH} = (\mathsf{ITDH.KGen}, \mathsf{ITDH.Hash\&Enc}, \mathsf{ITDH.Dec})$ be an L-level interactive trapdoor hashing protocol for the circuit class TC^0_L that satisfies the following properties:

- $(0.01w, 2^{-2w+O(1)})$-approximate correctness.
- Sub-exponential leveled function privacy. Let Sim be the leveled function privacy simulator. Let c be the constant in the sub-exponential security definition.

We construct a correlation intractable hash function $\mathsf{CIH} = (\mathsf{CIH.Gen}, \mathsf{CIH.Hash})$ for TC^0_L in Figure 5-5.

Theorem 5.4.2 (Correlation Intractability)**.** *If $w = \Omega(\lambda)$, the construction in Figure 5-5 is sub-exponential correlation intractable for the circuit class TC^0_L.*

5.4.3 Proof of Correlation Intractability

We prove Theorem 5.4.2 by contradiction. Let $\{f_\lambda\}_\lambda$ be a sequence of circuits in TC^0_L, and let $\mathcal{A}$ be a n.u. PPT adversary breaking correlation intractability with probability

Correlation Intractable Hash CIH

- $\mathsf{Gen}(1^\lambda)$:
  - For each $\ell \in [L]$, set $\lambda_\ell = \lambda^{\frac{1}{2}(\frac{c}{2})^{L-\ell}}$.
  - Compute simulated receiver's messages for ITDH:

$$\forall \ell \in [L], \mathsf{k}_\ell \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_\ell}, 1^n, 1^w, \ell)$$

  - Sample a mask $\mathsf{mask} \leftarrow \{0,1\}^w$ uniformly at random.
  - Output $\mathsf{k} = \Big(\{\mathsf{k}_\ell\}_{\ell\in[L]}, \mathsf{mask}\Big)$.
- $\mathsf{Hash}(\mathsf{k}, x)$:
  - Parse $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell\in[L]}, \mathsf{mask})$.
  - Let $\mathsf{e}_0 = \perp$. Compute hash values and encodings for ITDH:

$$\forall \ell \in [L], (\mathsf{h}_\ell, \mathsf{e}_\ell) \leftarrow \mathsf{ITDH.Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1}).$$

  - Output $\mathsf{e} \oplus \mathsf{mask}$, where $\mathsf{e} = \mathsf{e}_L$.

Figure 5-5. Description of CIH.

$\epsilon(\lambda)$.

We find the contradiction by constructing a series of hybrids.

- Hyb_0: In this hybrid, if the adversary's attack successes, then output 1, otherwise output 0.
  - Sample the CIH key $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$, and run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.
  - If $\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x)$, then output 1, otherwise output 0.
- $\mathsf{Hyb}_1^{\ell^*}$: This hybrid is the same as Hyb_0, except that we additionally guess the hash value h_{ℓ^*-1} by sampling h'_{ℓ^*-1} from uniform distribution.
  - Let $\mathsf{h}_0 = \mathsf{td}_0 = \perp$. For $\ell = 1, 2, \ldots, \ell^* - 1$, if $\ell > 1$, let $\mathsf{h}'_{\ell-1} \leftarrow \{0,1\}^{\lambda_{\ell-1}}$. Let

$$(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}'_{\ell-1}, \mathsf{td}_{\ell-1})$$

- If $\ell^* > 1$, sample $\mathsf{h}'_{\ell^*-1} \leftarrow \{0,1\}^{\lambda_{\ell^*-1}}$ uniformly at random. Let $\mathsf{k}_{\ell^*} \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_{\ell^*}}, 1^n, 1^w, \ell^*)$.
- For $\ell = \ell^* + 1, \ldots, L$, let $\mathsf{k}_\ell \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_\ell}, 1^n, 1^w, \ell)$.
- Sample $\mathsf{mask} \leftarrow \{0,1\}^w$ uniformly at random. Let $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell \in [L]}, \mathsf{mask})$.
- Run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.
- If $\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x)$ and $\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell$, then output 1. Otherwise, output 0.

- $\mathsf{Hyb}_{1.5}^{\ell^*}$: This hybrid is the same as $\mathsf{Hyb}_1^{\ell^*}$, except that we replace the ℓ^{th} level key with a "real key" generated by $\mathsf{ITDH.KGen}$.
  - Let $\mathsf{h}_0 = \mathsf{td}_0 = \bot$. For $\ell = 1, 2, \ldots, \ell^*$, if $\ell > 1$, let $\mathsf{h}'_{\ell-1} \leftarrow \{0,1\}^{\lambda_{\ell-1}}$. Let
  $$(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}'_{\ell-1}, \mathsf{td}_{\ell-1})$$
  - For $\ell = \ell^* + 1, \ldots, L$, let $\mathsf{k}_\ell \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_\ell}, 1^n, 1^w, \ell)$.
  - Sample $\mathsf{mask} \leftarrow \{0,1\}^w$ uniformly at random. Let $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell \in [L]}, \mathsf{mask})$.
  - Run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.
  - If $\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x)$ and $\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell$, then output 1. Otherwise, output 0.

- Hyb_2: This hybrid is the same as Hyb_1^{L+1}.
  - Let $\mathsf{h}_0 = \mathsf{td}_0 = \bot$. For $\ell = 1, 2, \ldots, L$, if $\ell > 1$, let $\mathsf{h}'_{\ell-1} \leftarrow \{0,1\}^{\lambda_{\ell-1}}$. Let
  $$(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}'_{\ell-1}, \mathsf{td}_{\ell-1})$$
  - Sample $\mathsf{h}'_L \leftarrow \{0,1\}^{\lambda_L}$ uniformly at random.
  - Sample $\mathsf{mask} \leftarrow \{0,1\}^w$ uniformly at random. Let $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell \in [L]}, \mathsf{mask})$.
  - Run the adversary $x \leftarrow \mathcal{A}(1^\lambda, \mathsf{k})$.

– If $\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x)$ and $\forall i \in [L], \mathsf{h}'_\ell = \mathsf{h}_\ell$, then output 1. Otherwise, output 0.

Lemma 5.4.3. *For any sufficiently large* λ, $\Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1] \geq \Pr[\mathsf{Hyb}_1^{\ell^*} = 1] - 2^{-\lambda_{\ell^*}^c}$.

Proof. For n.u. PPT adversary $\mathcal{A}$, we build the following distinguisher $\mathcal{D}$ for the sub-exponential function privacy in Figure 5-6. When the challenger computes k_{ℓ^*} from $\mathsf{ITDH.KGen}$, the distinguisher $\mathcal{D}$ simulates the environments for $\mathcal{A}$, and hence

$$\Pr\left[\mathsf{k}_{\ell^*} \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_{\ell^*}}, \ell^*, f_\lambda, \mathsf{h}'_{\ell^*-1}, \mathsf{td}_{\ell^*-1}) : \mathcal{D}(1^{\lambda_{\ell^*}}) = 1\right] = \Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1].$$

Similarly, we also have

$$\Pr\left[\mathsf{k}_{\ell^*} \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_{\ell^*}}, 1^n, 1^w, \ell^*) : \mathcal{D}(1^{\lambda_{\ell^*}}) = 1\right] = \Pr[\mathsf{Hyb}_1^{\ell^*} = 1].$$

Note that, the distinguisher $\mathcal{D}(1^{\lambda_{\ell^*}})$ runs in $\mathsf{poly}(\lambda)(\lambda)$ time, since $\lambda = \mathsf{poly}(\lambda)(\lambda_{\ell^*})$, the distinguisher also runs in $\mathsf{poly}(\lambda)(\lambda_{\ell^*})$ time. If the ITDH satisfies the sub-exponential function privacy property, the probabilities on the left hand sides differ by at most $2^{-\lambda_{\ell^*}{}^c}$, where c is a constant. Hence, we finish proving the lemma.

□

Lemma 5.4.4. $\Pr[\mathsf{Hyb}_1^{\ell^*+1} = 1] \geq \Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1]/2^{\lambda_{\ell^*}}$.

Proof. The difference between $\mathsf{Hyb}_{1.5}^{\ell^*}$ and $\mathsf{Hyb}_1^{\ell^*+1}$ is that, in $\mathsf{Hyb}_1^{\ell^*+1}$, we guess the hash value h'_{ℓ^*}. Hence,

$$\begin{aligned}
\Pr\left[\mathsf{Hyb}_1^{\ell^*+1} = 1\right] &= \Pr_{\mathsf{Hyb}_1^{\ell^*+1}}[\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x) \wedge (\forall i \in [\ell^*], \mathsf{h}'_\ell = \mathsf{h}_\ell)] \\
&= \Pr_{\mathsf{Hyb}_1^{\ell^*+1}}[\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x) \wedge (\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell) \wedge \mathsf{h}'_{\ell^*} = \mathsf{h}_{\ell^*}] \\
&= \Pr_{\mathsf{Hyb}_1^{\ell^*+1}}[\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x) \wedge (\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell)] \Pr[\mathsf{h}'_{\ell^*} = \mathsf{h}_{\ell^*}] \\
&= \Pr_{\mathsf{Hyb}_{1.5}^{\ell^*}}[\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x) \wedge (\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell)]/2^{\lambda_{\ell^*}} \\
&= \Pr\left[\mathsf{Hyb}_{1.5}^{\ell^*} = 1\right]/2^{\lambda_{\ell^*}}.
\end{aligned}$$

Distinguisher $\mathcal{D}(1^{\lambda_{\ell^*}})$

- For each $\ell < \ell^*$, generate k_ℓ using $\mathsf{ITDH.KGen}$,

$$\mathsf{h}'_{\ell-1} \leftarrow \{0,1\}^{\lambda_{\ell-1}}, (\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda, \mathsf{h}'_{\ell-1}, \mathsf{td}_{\ell-1}).$$

- If $\ell^* > 1$, sample $\mathsf{h}'_{\ell^*} \leftarrow \{0,1\}^{\ell^*}$ uniformly at random, query the challenger with h'_{ℓ^*-1} and td_{ℓ^*-1}, and get k_{ℓ^*} from the challenger.
- For each $\ell > \ell^*$, generate k_ℓ using $\mathsf{ITDH.Sim}$,

$$\mathsf{k}_\ell \leftarrow \mathsf{ITDH.Sim}(1^{\lambda_\ell}, 1^n, 1^w, \ell).$$

- Sample $\mathsf{mask} \leftarrow \{0,1\}^w$ uniformly at random, and let $\mathsf{k} = (\{\mathsf{k}_\ell\}_{\ell \in [L]}, \mathsf{mask})$.
- Run the adversary $\mathcal{A}(1^\lambda, \mathsf{k})$.
- If $\mathsf{Hash}(\mathsf{k}, x) = f_\lambda(x)$ and $\forall i \in [\ell^* - 1], \mathsf{h}'_\ell = \mathsf{h}_\ell$, then output 1. Otherwise, output 0.

Figure 5-6. Description of the distinguisher $\mathcal{D}$.

The first line follows from the construction of the hybrid $\mathsf{Hyb}_1^{\ell^*+1}$. The second line is obtained by considering the cases $i \in [\ell^* - 1]$ and $i = \ell^*$ separately. The third line follows from the independence of h'_{ℓ^*} and all other random variables. The fourth line follows from the fact that the bit length of h'_{ℓ^*} is λ_{ℓ^*}. The fifth line follows from the definition of $\mathsf{Hyb}_{1.5}^{\ell^*}$. Hence, we finish proving the lemma. □

Lemma 5.4.5. $\Pr[\mathsf{Hyb}_2 = 1] < 2^{-\Omega(\lambda)}$.

Proof. In Hyb_2, we check if $\forall i \in [L], \mathsf{h}'_\ell = \mathsf{h}_\ell$. Note that if such check passes, then $\mathsf{Hash}(\mathsf{k}, x)$ equals to $\mathsf{e} \oplus \mathsf{mask}$, where e is the encoding in the final level in an honest execution. Hence,

$$\Pr[\mathsf{Hyb}_2 = 1] \leq \Pr_{\mathsf{mask} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L} [\exists x : \mathsf{e} \oplus \mathsf{mask} = f_\lambda(x)],$$

where $r_1, r_2, \ldots, r_L$ are the random coins for the ITDH, and the encoding e is obtained from the following procedure.

Let $\mathsf{h}_0 = \mathsf{td}_0 = \mathsf{e}_0 = \perp$. For $\ell = 1, 2, \ldots, L$,

- Compute $(\mathsf{k}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{ITDH.KGen}(1^{\lambda_\ell}, \ell, f_\lambda \mathsf{h}_{\ell-1}, \mathsf{td}_{\ell-1}; r_\ell)$ with random coins r_ℓ.
- Hash the input x using the hash key $(\mathsf{h}_\ell, \mathsf{e}_\ell) \leftarrow \mathsf{ITDH.Hash\&Enc}(\mathsf{k}_\ell, x, \mathsf{e}_{\ell-1})$

Finally, let $\mathsf{e} = \mathsf{e}_L$ be the encoding at the final level, and also let $\mathsf{d} = \mathsf{ITDH.Dec}(\mathsf{td}_L, \mathsf{h}_L)$.

Since the ITDH satisfies $(0.01w, 2^{-2w+O(1)})$-approximate correctness, we have

$$\Pr_{r_1, r_2, \ldots, r_L}[\exists x, \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f_\lambda(x)) > 0.01w] < 2^{-2w+O(1)}.$$

Hence, except with probability $2^{-2w+O(1)}$, we have that $\forall x, \mathsf{Ham}(\mathsf{e} \oplus \mathsf{d}, f_\lambda(x)) \leq 0.01w$. Denote the Hamming error between $\mathsf{e} \oplus \mathsf{d}$ and $f_\lambda(x)$ as ε. Then we have $f_\lambda(x) = \mathsf{e} \oplus \mathsf{d} \oplus \varepsilon$, and the weight of ε is at most $0.01w$. Now, we have,

$$\begin{aligned}
&\Pr_{\mathsf{mask} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L}[\exists x : \mathsf{e} \oplus \mathsf{mask} = f_\lambda(x)] \\
\leq & 2^{-2m+O(1)} + \Pr_{\mathsf{mask} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L}[\exists x, \varepsilon : \mathsf{e} \oplus \mathsf{mask} = \mathsf{e} \oplus \mathsf{d} \oplus \varepsilon] \\
\leq & 2^{-2m+O(1)} + \Pr_{\mathsf{mask} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L}[\exists x, \varepsilon : \mathsf{mask} = \mathsf{d} \oplus \varepsilon]
\end{aligned}$$

Note that, for any fixed random coins $r_1, r_2, \ldots, r_L$, the decoding value d only depends on $\mathsf{h}_1, \mathsf{h}_2, \ldots, \mathsf{h}_L$. The number of possible choice of d is $2^{\lambda_1 + \lambda_2 + \ldots + \lambda_L} \leq 2^{2\lambda_L} \leq 2^{2\lambda^{1/2}}$. The number of possible values of ε is at most $\binom{w}{0.01w} \leq (100e)^{0.01w} \leq 2^{w/2}$. Hence, we have

$$\Pr_{\mathsf{mask} \leftarrow \{0,1\}^w, r_1, r_2, \ldots, r_L}[\exists x, \varepsilon : \mathsf{mask} = \mathsf{d} \oplus \varepsilon] < 2^{2\lambda^{1/2}} \cdot 2^{w/2} \cdot 2^{-w} = 2^{-(w/2 - 2\lambda^{1/2})} = 2^{-\Omega(\lambda)}$$

□

Completing the Proof. Let $\epsilon(\lambda) = \Pr[\mathsf{Hyb}_0 = 1]$. We first claim that, for each $\ell^* \in [L+1]$, we have

$$\Pr\left[\mathsf{Hyb}_1^{\ell^*} = 1\right] \geq (\epsilon(\lambda) - \ell^* 2^{-\lambda_1^2 + 2\lambda_1})/2^{2\lambda_{\ell^*-1}},$$

where $\lambda_0 = 0$.

We now prove this claim by induction on ℓ^*. For $\ell^* = 1$, since Hyb_1^1 and Hyb_0 are identical, we have $\Pr\left[\mathsf{Hyb}_1^1 = 1\right] = \Pr\left[\mathsf{Hyb}_0 = 1\right] \geq \epsilon(\lambda)$. Hence, then the claim holds for $\ell^* = 1$. Now, we assume the claim holds for ℓ^*, we prove that the claim holds for $\ell^* + 1$ as follows.

From Lemma 5.4.3, we have

$$\Pr\left[\mathsf{Hyb}_{1.5}^{\ell^*} = 1\right] \geq \Pr\left[\mathsf{Hyb}_1^{\ell^*} = 1\right] - 2^{\lambda_{\ell^*}^c} \geq \epsilon(\lambda) - \ell^* 2^{-\lambda_1^2+2\lambda_1}/2^{2\lambda_{\ell^*-1}} - 2^{-\lambda_{\ell^*}^c}.$$

By the choice of the parameters, we have $\lambda_{\ell^*} = (\lambda_{\ell^*-1})^{\frac{2}{c}}$. Hence, the right hand side is bounded by

$$\begin{aligned}\epsilon(\lambda) - \ell^* 2^{-\lambda_1^2+2\lambda_1}/2^{2\lambda_{\ell^*-1}} - 2^{-\lambda_{\ell^*-1}^2} &= (\epsilon(\lambda) - \ell^* 2^{-\lambda_1^2+2\lambda_1} - 2^{-\lambda_{\ell^*-1}^2+\lambda_{\ell^*-1}})/2^{2\lambda_{\ell^*-1}} \\ &\geq (\epsilon(\lambda) - (\ell^*+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*-1}}\end{aligned}$$

Then, by Lemma 5.4.4, we have

$$\begin{aligned}\Pr\left[\mathsf{Hyb}_1^{\ell^*+1} = 1\right] \geq \Pr[\mathsf{Hyb}_{1.5}^{\ell^*} = 1]/2^{\lambda_{\ell^*}} &\geq (\epsilon(\lambda) - (\ell^*+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*-1}+\lambda_{\ell^*}} \\ &> (\epsilon(\lambda) - (\ell^*+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda_{\ell^*}}.\end{aligned}$$

Hence, we finish prove the claim.

By this claim, and the fact that Hyb_2 is identical to Hyb_1^{L+1} we know that

$$\Pr[\mathsf{Hyb}_2 = 1] = \Pr[\mathsf{Hyb}_1^{L+1} = 1] \geq (\epsilon(\lambda) - (L+1)2^{-\lambda_1^2+2\lambda_1})/2^{2\lambda^{1/2}}.$$

From Lemma 5.4.5, we have $\Pr[\mathsf{Hyb}_2 = 1] < 2^{-\Omega(\lambda)}$. Hence, we have

$$\epsilon(\lambda) < 2^{-\Omega(\lambda)+2\lambda^{1/2}} + (L+1)2^{-\lambda_1^2+2\lambda_1}.$$

Since $L = O(1)$ and $\lambda_1 = \lambda^{\Theta(1)}$, we finish the proof.

5.4.4 On the Trade-off between DDH-hardness and the Circuit Class for CIH

In the previous subsections, we constructed CIH for TC^0 based on sub-exponential hardness of DDH against polynomial time adversaries. In this section, we show that

we can in fact trade-off between the hardness assumption on DDH and the depth of the circuit class for CIH.

CIH for $O(\log\log\lambda)$-depth Threshold Circuits. If we assume sub-exponential hardness of DDH against sub-exponential time adversaries, then we can obtain CIH for $O(\log\log\lambda)$-depth threshold circuits.

Theorem 5.4.6 (CIH for $O(\log\log\lambda)$-depth Threshold Circuits.)**.** *If we assume there exists a constant $0 < c < 1$ such that for any non-uniform adversary running in time $2^{O(\lambda^c)}$, the advantage for DDH is bounded by $2^{-\Omega(\lambda^c)}$, then there exists a construction of CIH for any polynomial size circuits with depth $\frac{1-o(1)}{4\log(2/c)}\log\log\lambda$, and output length $\Omega(\lambda)$.*

We can set the security parameters for i^{th} level as $\lambda_i = (\log^2\lambda)^{(\frac{2}{c})^i}$, for $i = 1, 2, \ldots, \frac{1}{4\log(2/c)}\log\log\lambda \cdot (1 - o(1))$. Note that at the last level, $\lambda_L \leq \lambda^{1/2}$. Then the proofs in Section 5.4.3 can be extended to CIH for $\frac{1-o(1)}{4\log(2/c)}\log\log\lambda$-depth threshold circuits. Note that here we crucially rely on the sub-exponential time assumption, since an adversary that runs in time polynomial in λ is an adversary that runs in time sub-exponential in λ_i.

CIH for TC^1. If we assume exponential hardness of DDH against polynomial time adversaries, then we can obtain CIH for TC^1, i.e., log-depth threshold circuits.

Theorem 5.4.7 (CIH for TC^1)**.** *If we assume there exists a constant $A > 1$ such that for any non-uniform adversary running in polynomial time, the advantage for DDH is bounded by $2^{-\Omega(\lambda/A)}$. Then there exists a construction of CIH for any polynomial size threshold circuits with depth $\lfloor\frac{1}{3}\cdot\log_{2A}\lambda\rfloor$ and output length $\Omega(\lambda)$.*

We can set the security parameter for i^{th} level as $\lambda_i = \lambda^{1/3}\cdot(2A)^i$, where $i = 1, 2, \ldots, \lfloor\frac{1}{3}\cdot\log_{2A}\lambda\rfloor$. Note that at the last level, $\lambda_L \leq \lambda^{2/3}$. Then the proofs in Section 5.4.3 also work for TC^1 circuits.

5.5 Non-Interactive (Statistical) Zero-Knowledge Arguments for $\mathcal{NP}$

In this section, we present our constructions of NIZK arguments for $\mathcal{NP}$ in the common random string model. We present two variants: one that achieves statistical zero knowledge and non-adaptive soundness, and another that achieves computational zero knowledge and adaptive soundness. For most of this section, we focus on the first variant, namely, statistical NIZK arguments for $\mathcal{NP}$. We obtain the computational variant via simple modifications to our first construction.

The rest of this section is organized as follows:

- In Section 5.5.1, we construct a lossy public key encryption scheme LPKE with *low-depth decryption property*, which essentially requires that the decryption circuit is in TC^0.
- Using LPKE, we construct a trapdoor sigma protocol for $\mathcal{NP}$ that achieves statistical honest-verifier zero knowledge. This protocol achieves two additional key properties – *low-depth bad challenge function* and *knowledge extraction*. We describe this construction in Section 5.5.2.
- Next, in Section 5.5.3, we construct non-interactive statistical witness indistinguishable (NISWI) arguments for $\mathcal{NP}$ in the common random string model via the correlation-intractability framework. Namely, we use CIH for TC^0 to collapse the rounds of λ parallel-repetitions of the trapdoor sigma protocol from the previous step to obtain NISWIs. We further prove that the resulting scheme achieves (sub-exponential) non-adaptive argument of knowledge property.
- In Section 5.5.4, we describe a variant of the FLS "OR-trick" [46] to transform NISWI for $\mathcal{NP}$ from the previous step to multi-theorem *statistical* NIZK for $\mathcal{NP}$, while preserving the distribution of the CRS.

- Finally, in Section 5.5.5, we sketch our construction of multi-theorem *computational* NIZK arguments for $\mathcal{NP}$ with *adaptive* soundness in the common random string model. This variant is obtained via the same steps as above, except that we replace the lossy public-key encryption scheme (used for constructing trapdoor sigma protocol) with Elgamal encryption [99].

5.5.1 Lossy Public Key Encryption with Low-Depth Decryption

A lossy public key encryption scheme is a tuple of algorithms $(\mathsf{LossyGen}, \mathsf{Gen}, \mathsf{Enc})$, which proceeds as follows.

- $\mathsf{Gen}(1^\lambda)$: The key generation algorithm takes as input a security parameter λ, and it outputs a public key pk and a secret key sk.
- $\mathsf{LossyGen}(1^\lambda)$: The lossy public key generation algorithm takes as input the security parameter, and it outputs a lossy public key $\widetilde{\mathsf{pk}}$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: The encryption algorithm takes as input a public key pk, and a message $m \in \mathbb{Z}_2$, it outputs a ciphertext ct.

We require the algorithms to satisfy the following properties.

- **Key Indistinguishability:** For any n.u. PPT distinguisher $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that for any $\lambda \in \mathbb{N}$,

$$\left| \Pr\left[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) : \mathcal{D}(1^\lambda, \mathsf{pk}) = 1\right] - \Pr\left[\widetilde{\mathsf{pk}} \leftarrow \mathsf{LossyGen}(1^\lambda) : \mathcal{D}(1^\lambda, \widetilde{\mathsf{pk}}) = 1\right] \right| \leq \nu(\lambda).$$

  Furthermore, we say the scheme satisfies sub-exponential key indistinguishability, if there exists a constants c and λ_0 such that for any n.u. PPT distinguisher, the advantage $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

- **Statistical Semantic Security in Lossy Mode:** There exists a negligible function $\nu(\lambda)$ s.t. for any two messages $m_1, m_2 \in \mathbb{Z}_2$,

$$\mathsf{SD}\left(\left(\widetilde{\mathsf{pk}}, \mathsf{Enc}(\widetilde{\mathsf{pk}}, m_1)\right), \left(\widetilde{\mathsf{pk}}, \mathsf{Enc}(\widetilde{\mathsf{pk}}, m_2)\right)\right) \leq \nu(\lambda),$$

  where $\widetilde{\mathsf{pk}} \leftarrow \mathsf{LossyGen}(1^\lambda)$, and the randomness of the distribution is over the randomness of $\mathsf{LossyGen}$ and Enc.

- **Low-Depth Decryption:** There exists a sequence of circuits $\{\mathsf{Dec}_\lambda\}_\lambda$ in TC^0 and a deterministic polynomial-time algorithm $\mathsf{PreComp}$ such that, for any $\lambda \in \mathbb{N}$ and any message $m \in \mathbb{Z}_2$,

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda), \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m), m' \leftarrow \mathsf{Dec}_\lambda(\mathsf{PreComp}(1^\lambda, \mathsf{ct}), \mathsf{sk}) : m = m'\Big] = 1.$$

5.5.1.1 Construction

We describe our lossy public key encryption scheme LPKE in Figure 5-7. Our construction is essentially the same as the dual-mode encryption scheme in [106]. We show that this construction achieves low-depth decryption property.

We instantiate the group $\mathbb{G}$ as the standard prime order subgroup of $\mathbb{Z}_q^*$, with efficient group membership testing algorithm. For instantiation from Elliptic curves, see Appendix 5.7.

We now prove that LPKE achieves the required properties.

Lemma 5.5.1 (Statistical Semantic Security in Lossy Mode)**.** *The proposed scheme* LPKE *satisfies statistical semantic security in lossy mode.*

Proof. For any lossy public key

$$\widetilde{\mathsf{pk}} = \left(\mathbb{G}, p, \begin{bmatrix} g^1 & g^a \\ g^b & g^c \end{bmatrix}\right),$$

Lossy Public Key Encryption LPKE

- $\mathsf{Gen}(1^\lambda)$:
  - Generate a group using $\mathcal{G}$: $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$.
  - Sample $a, b \leftarrow \mathbb{Z}_p$.
  - Output $\left(\mathsf{pk} = \left(\mathbb{G}, p, \begin{bmatrix} g & g^b \\ g^a & g^{ab} \end{bmatrix}\right), \mathsf{sk} = a\right)$.
- $\mathsf{LossyGen}(1^\lambda)$:
  - Generate a group using $\mathcal{G}$: $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$.
  - Sample the elements uniformly at random from $\mathbb{G}$, $g_{12}, g_{21}, g_{22} \leftarrow \mathbb{G}$.
  - Output $\widetilde{\mathsf{pk}} = \left(\mathbb{G}, p, \begin{bmatrix} g & g_{12} \\ g_{21} & g_{22} \end{bmatrix}\right)$.
- $\mathsf{Enc}(\mathsf{pk}, m; r)$:
  - Parse $\mathsf{pk} = \left(\mathbb{G}, p, \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}\right)$, $m \in \{0,1\}$ and $r = (r_1, r_2) \in \mathbb{Z}_p^2$.
  - Output the ciphertext $\mathsf{ct} = \begin{bmatrix} g_{11}^{r_1} \cdot g_{12}^{r_2} \\ g_{21}^{r_1} \cdot g_{22}^{r_2} \cdot g^m \end{bmatrix}$

Figure 5-7. Construction of the lossy public key encryption.

where a, b, c are sampled from uniform distribution over $\mathbb{Z}_p$, with probability $1 - 1/|\mathbb{G}|$, $c \neq ab$. When it happens, the matrix $\begin{bmatrix} 1 & a \\ b & c \end{bmatrix}$ is non-singular and hence the ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\widetilde{\mathsf{pk}}, m)$ is uniformly distributed over $\mathbb{G}^2$ and is independent of m. Hence,

$$\mathsf{SD}((\widetilde{\mathsf{pk}}, \mathsf{Enc}(\widetilde{\mathsf{pk}}, m)), (\widetilde{\mathsf{pk}}, U)) \leq 1/|\mathbb{G}|,$$

where U is the uniform distribution over $\mathbb{G}^2$. By the triangular inequality of statistical distance, $(\widetilde{\mathsf{pk}}, \mathsf{Enc}(\widetilde{\mathsf{pk}}, m_1))$ and $(\widetilde{\mathsf{pk}}, \mathsf{Enc}(\widetilde{\mathsf{pk}}, m_2))$ are statistically indistinguishable, for any messages $m_1, m_2 \in \mathbb{Z}_2$. We finish the proof. □

Lemma 5.5.2 (Key Indistinguishability)**.** *Assuming (sub-exponential) DDH, the proposed scheme* LPKE *satisfies (sub-exponential) key indistinguishability.*

Proof. Since an injective mode public key is a DDH tuple and a lossy public key is subjected to uniformly distribution, the (sub-exponential) key indistinguishability follows from the (sub-exponential) DDH assumption directly. □

Lemma 5.5.3 (Low-Depth Decryption)**.** *The proposed scheme* LPKE *satisfies low-depth decryption property.*

Proof. We construct the following algorithm $\mathsf{PreComp}$ and the circuit family $\{\mathsf{Dec}_\lambda\}_\lambda$. For any $\mathsf{sk} = a$, we denote $(a_0, a_1, \ldots, a_\lambda)$ to be binary representation of a, i.e., $a = a_0 2^0 + a_1 2^1 + \ldots + a_\lambda 2^\lambda$ where $a_i \in \{0, 1\}$. Since Dec_λ is a circuit over boolean value, it takes as the inputs in the binary representation form. Hence, we can assume Dec_λ takes $a_0, a_1, \ldots, a_\lambda$ as input.

- $\mathsf{PreComp}(1^\lambda, c = (c_1, c_2) \in \mathbb{G}^2)$: Output $(c_1^{-2^0}, c_1^{-2^1}, c_1^{-2^2}, \ldots, c_1^{-2^\lambda}, c_2)$.
- $\mathsf{Dec}_\lambda\left((c'_0, c'_1, c'_2, \ldots, c'_\lambda, c_2), \mathsf{sk} = (a_0, a_1, \ldots, a_\lambda)\right)$:
  - For each $i \in 0, 1, \ldots, \lambda$, if $a_i = 0$, let $g_i = 1_\mathbb{G}$. Otherwise, let $g_i = c'_i$.
  - Compute $u = g_0 \cdot g_1 \cdot \ldots \cdot g_\lambda \cdot c_2$, where the iterative multiplication is over $\mathbb{Z}_q^*$.
  - Compare u with $1_\mathbb{G}$. If $u = 1_\mathbb{G}$, output 0. Otherwise output 1.

We first argue the correctness of Dec_λ. It is easy to see that for any ciphertext $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, m; r)$ of the message $m \in \{0, 1\}$ and randomness r using public key pk, we have that $\mathsf{Dec}_\lambda(\mathsf{PreComp}(1^\lambda, c), \mathsf{td}) = c_1^{-a} \cdot c_2 = m$.

Next, we argue the low-depth property. Note that the first step of Dec_λ can be easily computed by a constant depth threshold circuit. From [108], we have that the second step, which involves multiplication of λ inputs mod a prime q, can also be computed in TC^0. For the third step, the comparison between g_0 and $1_\mathbb{G}$ can be computed in TC^0. Hence, by composing these circuits, we obtain a circuit for Dec_λ in TC^0. □

5.5.2 Trapdoor Sigma Protocol

In this section, we construct trapdoor sigma protocols for $\mathcal{NP}$ with low-depth bad challenge functions, and with an additional knowledge extraction property. We start by providing a formal definition.

Definition.A trapdoor sigma protocol for a language $\mathcal{L}$ is a tuple of algorithms $\Sigma = (\mathsf{Gen}_{\mathsf{zk}}, \mathsf{Gen}_{\mathsf{sound}}, \mathsf{P}, \mathsf{V})$ described as follows:

- $\mathsf{crs}_{\mathsf{zk}} \leftarrow \mathsf{Gen}_{\mathsf{zk}}(1^\lambda)$: It takes as input the security parameter and outputs a uniformly distributed "ZK mode" CRS $\mathsf{crs}_{\mathsf{zk}}$.
- $(\mathsf{crs}_{\mathsf{sound}}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\mathsf{sound}}(1^\lambda)$: It takes as input the security parameter and outputs a "soundness mode" CRS $\mathsf{crs}_{\mathsf{sound}}$ and a trapdoor td.
- **Round 1:** At the start of the protocol, the prover P and the verifier V receive a CRS crs and an instance $x \in \mathcal{L}$. The prover P additionally receives a witness ω. It computes a first round message α and sends it to V.
- **Round 2:** The verifier V sends a random challenge $\beta \in \{0,1\}$ to P.
- **Round 3:** Upon receiving β, the prover P computes a third round message γ and sends it to the verifier V.
- **Verification:** Upon receiving γ, V either accepts or rejects the transcript (α, β, γ).

We require Σ to satisfy the following properties:

- **Completeness:** For any crs generated by $\mathsf{Gen}_{\mathsf{zk}}$ or $\mathsf{Gen}_{\mathsf{sound}}$, $x \in \mathcal{L}$ and any witness ω for x,

$$\Pr\left[\mathsf{Out}_{\mathsf{V}}\left(\mathsf{P}\left(\mathsf{crs}, x, \omega\right) \leftrightarrow \mathsf{V}\left(\mathsf{crs}, x\right)\right) = \text{accept}\right] = 1,$$

  where $\mathsf{Out}_{\mathsf{V}}(e)$ is the output of V in a protocol execution e.

- **CRS Indistinguishability:** For any n.u. PPT distinguisher $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that for any $\lambda \in \mathbb{N}$,

$$\Big| \Pr\left[\mathsf{crs}_{\mathsf{zk}} \leftarrow \mathsf{Gen}_{\mathsf{zk}}(1^\lambda) : \mathcal{D}(1^\lambda, \mathsf{crs}_{\mathsf{zk}}) = 1\right] - \Pr\left[\mathsf{crs}_{\mathsf{sound}} \leftarrow \mathsf{LossyGen}(1^\lambda) : \mathcal{D}(1^\lambda, \mathsf{crs}_{\mathsf{sound}}) = 1\right] \Big| \leq \nu(\lambda).$$

  Furthermore, we say the scheme satisfies sub-exponential CRS indistinguishability, if there exists a constant $0 < c < 1$ such that for any n.u. PPT distinguisher, the advantage $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

- **Adaptive Statistical Honest-Verifier Zero Knowledge:** There exists a PPT simulator S and a negligible function $\nu(\lambda)$ such that, for any unbounded adversary $\mathcal{A}$,

$$\left|\Pr\left[\mathsf{Real}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}(1^\lambda) = 1\right]\right| \leq \nu(\lambda),$$

  where the experiments Real and Ideal are described as follows:

$\underline{\mathsf{Real}(1^\lambda):}$	$\underline{\mathsf{Ideal}(1^\lambda):}$
$\mathsf{crs}_{\mathsf{zk}} \leftarrow \mathsf{Gen}_{\mathsf{zk}}(1^\lambda)$.	$\mathsf{crs}_{\mathsf{zk}} \leftarrow \mathsf{Gen}_{\mathsf{zk}}(1^\lambda)$.
$(x, \omega) \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{zk}})$.	$(x, \omega) \leftarrow \mathcal{A}(\mathsf{crs}_{\mathsf{zk}})$.
If $\mathcal{R}(x, \omega) \neq 1$, output 0.	If $\mathcal{R}(x, \omega) \neq 1$ output 0.
$\alpha \leftarrow \mathsf{P}(1^\lambda, \mathsf{crs}_{\mathsf{zk}}, x, \omega)$.	$\beta \leftarrow \{0,1\}$.
$\beta \leftarrow \{0,1\}$, and $\gamma \leftarrow \mathsf{P}(\beta)$.	$(\alpha, \gamma) \leftarrow \mathsf{S}(1^\lambda, \mathsf{crs}_{\mathsf{zk}}, x, \beta)$.
Output $\mathcal{A}(\alpha, \beta, \gamma)$.	Output $\mathcal{A}(\alpha, \beta, \gamma)$.

  We say that the trapdoor sigma protocol satisfies adaptive **computational** honest-verifier zero knowledge, if the above condition holds for any non-uniform PPT adversary.

- **Bad Challenge Function in TC^0:** There exists a sequence of circuits $\{\mathsf{BadC}_\lambda\}_\lambda$ in TC^0, and a *deterministic* polynomial time algorithm $\mathsf{PreComp}(\cdot, \cdot)$, such that

for any $(\mathsf{crs}_{\mathsf{sound}}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\mathsf{sound}}(1^\lambda)$, instance $x \notin \mathcal{L}$ and any accepting transcript (α, β, γ) for x,

$$\beta = \mathsf{BadC}_\lambda(\mathsf{PreComp}(1^\lambda, \alpha), \mathsf{td}).^5$$

- **Knowledge Extraction:** There exists a polynomial time extractor Ext such that, for any soundness mode CRS $(\mathsf{crs}_{\mathsf{sound}}, \mathsf{td}) \leftarrow \mathsf{Gen}_{\mathsf{sound}}(1^\lambda)$, any instance $x \in \{0,1\}^*$, and any accepting transcript (α, β, γ), if $\beta \neq \mathsf{BadC}_\lambda(\mathsf{PreComp}(1^\lambda, \alpha), \mathsf{td})$, then

$$\Pr\left[\omega \leftarrow \mathsf{Ext}(\alpha, \beta, \gamma, \mathsf{td}) : \mathcal{R}(x, \omega) = 1\right] = 1.$$

Remark 5.5.4. *The bad challenge function we define above satisfies the "instance-universality" property [51]. Namely, the bad challenge function does not depend on the instance x. As in [51], we rely on this property to achieve adaptive soundness for our computational NIZK construction.*

5.5.2.1 Construction

We construct our desired trapdoor sigma protocol for $\mathcal{NP}$ by instantiating the commitment scheme in the trapdoor sigma protocol of [51] (Construction 3.1) with the lossy public key encryption from Section 5.5.1.

Protocol from [51], Slightly Modified. Brakerski et al. [51] constructed a so-called "commit-and-open" trapdoor sigma protocol where in the first round, the prover commits to a string bit-by-bit, and then in the third round, the prover opens some positions of the commitments and provides some other information to complete the proof. The crucial property of their construction, that we shall use, is that the bad challenge function only consists of the following two computations: extraction from the commitments sent by the prover, and verification of a 3-CNF. While the first step is common to trapdoor sigma protocols, the second step is due to the use of

[5]Note that this implies that for any false instance $x \notin \mathcal{L}$ and any α, there is a unique β for which there exists an accepting γ.

Cook-Levin theorem in [51] (for reducing the depth of the bad challenge function). Recall that for any polynomial size circuit $C(\cdot)$, from the Cook-Levin theorem, we can convert it efficiently to a 3-CNF $\Phi_C(\cdot,\cdot)$ such that, for any input x, $C(x) = 1$ if and only if $\Phi_C(x,\cdot)$ is satisfiable. Furthermore, for any $C(x) = 1$, we can compute efficiently a witness ω' such that $\Phi_C(x,\omega') = 1$.

We now briefly describe the trapdoor sigma protocol, which is slightly modified from [51].

- **CRS Generation:** The CRS crs contains a commitment key. The commitment key is generated with a trapdoor td that allows one to extract the message from the commitment.

- **First Round:** The prover prepares the first round message α in Blum's protocol [111]. In addition, the prover prepares the third round response γ_1 for the challenge bit $\beta = 1$. Let $C(\alpha,\gamma_1) = \mathsf{V}(\mathsf{crs},\alpha,1,\gamma_1)$ be the verification circuit for the challenge bit $\beta = 1$ in Blum's protocol.[6] The prover applies a Cook-Levin reduction to C and the assigment (α,γ_1), and obtains a 3-CNF $\Phi_C(\cdot,\cdot)$ and a witness ω'. Then the prover sends the bit-by-bit commitments $c_{\gamma_1} \leftarrow \mathsf{Com}(\gamma_1), c_{\omega'} \leftarrow \mathsf{Com}(\omega')$ and α to the verifier.

- **Second Round:** The verifier sends a random challenge $\beta \leftarrow \{0,1\}$ to the prover.

- **Third Round:** Upon receiving the random challenge $\beta \in \{0,1\}$, if $\beta = 0$, then the prover responds with the third round message in the Blum's protocol. If $\beta = 1$, the prover opens the commitments to γ_1 and ω', and sends γ_1,ω' and their openings to the verifier.

[6] Here we assume that the verification circuit does not take the instance x as input for the challenge $\beta = 1$. Recall that Blum's protocol only checks whether the committed graph is a cycle graph for one of the challenge. Hence such a property can be achieved.

- **Verification:** Given the transcript, if $\beta = 0$, the verifier performs the same verification as in Blum's protocol. If $\beta = 1$, the verifier checks if the openings of γ_1, ω' are correct, and checks if $\Phi_C((\alpha, \gamma_1), \omega') = 1$.

- **Bad Challenge Function:** It proceeds in the following two steps:

  - **Extraction:** It uses td to extract the (γ_1, ω') from the commitments.

  - **Verfication:** If $\Phi_C((\alpha, \gamma_1), \omega') = 1$, then output 1, otherwise output 0.

Our Construction. We construct our desired trapdoor sigma protocol Σ for $\mathcal{NP}$ by instantiating the commitment scheme Com in the above protocol with the lossy public key encryption LPKE in Section 5.5.1. A CRS of the resulting protocol contains a public key of LPKE; when the public key is lossy (resp., injective), the CRS is in ZK (resp., soundness) mode.

We now prove that the resulting protocol satisfies our required properties. The CRS indistinguishability follows directly from the key indistinguishability of LPKE. The completeness property follows from the completeness of the underlying Blum's protocol and the Cook-Levin theorem. The construction in [51] is proven to achieve adaptive computational zero-knowledge property. Here, we observe that when we instantiate the commitment scheme with LPKE, the resulting construction achieves adaptive *statistical* zero knowledge property since LPKE satisfies statistical semantic security in lossy mode.

Next, we argue that the construction satisfies the bad challenge function in TC^0 and the knowledge extraction properties.

Bad Challenge Function in TC^0. As described above, the bad challenge function for the trapdoor sigma protocol of [51] consists of two steps: extraction from the commitments, and an evaluation of Φ_C. For our instantiation, the first step can be computed as $\mathsf{LPKE.Dec}_\lambda(\mathsf{LPKE.PreComp}(1^\lambda, \cdot), \mathsf{td})$, where $\{\mathsf{LPKE.Dec}_\lambda\}_\lambda$ is the

sequence of decryption circuits in TC^0 for LPKE and $\mathsf{LPKE.PreComp}$ is the associated deterministic pre-computation algorithm. Furthermore, the second step that involves evaluation of Φ_C can be computed in AC^0.

We therefore define BadC_λ and $\mathsf{PreComp}$ for our trapdoor sigma protocol as follows:

$$\mathsf{PreComp}(1^\lambda, (c_{\gamma_1}, c_{\omega'}, \alpha)) = (\mathsf{LPKE.PreComp}(1^\lambda, c_{\gamma_1}), \mathsf{LPKE.PreComp}(1^\lambda, c_{\omega'}), \alpha),$$

$$\mathsf{BadC}_\lambda((c'_{\gamma_1}, c'_{\omega'}, \alpha), \mathsf{td}) = \Phi_C((\alpha, \mathsf{LPKE.Dec}_\lambda(c'_{\gamma_1}, \mathsf{td})), \mathsf{LPKE.Dec}_\lambda(c'_{\omega'}, \mathsf{td})).$$

From the above, it follows that $\mathsf{BadC}_\lambda \in \mathsf{TC}^0$.

Knowledge Extraction. To argue knowledge extraction, we leverage the special soundness of Blum's protocol. Recall that special soundness guarantees that given two accepting transcripts with different challenges where the first round messages are the same, one can efficiently extract a witness.

For any accepting transcript (α', β, γ) with $\beta \neq \mathsf{BadC}_\lambda(\mathsf{PreComp}(1^\lambda, \alpha'), \mathsf{td})$, where $\alpha' = (c_{\gamma_1}, c_{\omega'}, \alpha)$, we consider two cases.

- **Case 1** $\beta = 0$**:** Then the bad challenge function outputs 1. We first execute the extraction step in the bad challenge function, and obtain (γ_1, ω'). As the bad challenge function outputs 1, γ_1 is an accepting witness for the Blum's protocol. Since γ is the other accepting response for the challenge $\beta = 0$ for Blum's protocol, we obtain two accepting transcripts, and hence can extract a witness via the special soundness property of Blum's protocol.

- **Case 2** $\beta = 1$**:** Then the bad challenge function outputs 0. We will argue that this is impossible. Recall that, γ contains (γ_1, ω') and the openings with respect to their commitments. Since the transcript is accepting, $\Phi_C((\alpha, \gamma_1), \omega') = 1$ and the openings are accepted. Recall that, once we instantiate the commitment scheme with our lossy public key encryption, the opening contains the randomness used in the encryption. Hence, since the openings are accepted, the bad

challenge function also extracts the same (γ_1, ω'). However, the output value of the bad challenge function implies that $\Phi_C((\alpha, \gamma_1), \omega') = 0$, which contradicts $\Phi_C((\alpha, \gamma_1), \omega') = 1$. Hence, this case is impossible.

From the above, we have that only the first case is possible. This concludes the proof of the knowledge extraction property.

5.5.3 Non-Interactive Statistical Witness Indistinguishable Argument for $\mathcal{NP}$

We construct a non-interactive statistical witness indistinguishable (NISWI) argument system $\Pi = (\mathsf{CGen}, \mathsf{P}, \mathsf{V})$ for $\mathcal{NP}$ in the common random string model with adaptive statistical witness indistinguishability and (sub-exponential) non-adaptive argument of knowledge properties.

Our construction relies on the following two ingredients:

- A trapdoor sigma protocol $\Sigma = (\Sigma.\mathsf{Gen}_{\mathsf{zk}}, \Sigma.\mathsf{Gen}_{\mathsf{sound}}, \Sigma.\mathsf{P}, \Sigma.\mathsf{V})$ for $\mathcal{L}$. Let $\{\Sigma.\mathsf{BadC}_\lambda\}_\lambda$ be the family of bad challenge functions in TC^0 and $\Sigma.\mathsf{PreComp}$ be the deterministic "pre-computation" algorithm associated with Σ, and let $\Sigma.\mathsf{Ext}$ be the knowledge extractor associated with Σ.

- A correlation intractable hash function $\mathsf{CIH} = (\mathsf{CIH.Gen}, \mathsf{CIH.Hash})$ for TC^0. Furthermore, we require that the CIH satisfies sub-exponential correlation intractability.

Construction of Π. Our scheme Π is described in Figure 5-8.

Lemma 5.5.5 (Completeness)**.** *The proposed scheme Π satisfies completeness.*

Proof. Let $\pi = (\{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]})$ be any proof generated by an honest prover for an instance $x \in \mathcal{L}$. From the completeness of the sigma protocol Σ, we have that

$$\Pr\left[\mathsf{out}_i \leftarrow \Sigma^{(i)}.\mathsf{V}(\mathsf{crs}_{\mathsf{zk}}, x, (\alpha_i, \beta_i, \gamma_i)) : \mathsf{out}_i = 1\right] = 1.$$

NISWI Argument System Π for $\mathcal{NP}$

CRS Generation $\mathsf{CGen}(1^\lambda)$**:** Sample a CIH key $\mathsf{k} \leftarrow \mathsf{CIH.Gen}(1^\lambda)$, and a CRS $\mathsf{crs_{zk}} \leftarrow \Sigma.\mathsf{Gen_{zk}}(1^\lambda)$. Output $\mathsf{crs} = (\mathsf{k}, \mathsf{crs_{zk}})$.
Prover $\mathsf{P}(\mathsf{crs}, x, \omega)$**:**The prover receives as input a CRS $\mathsf{crs} = (\mathsf{k}, \mathsf{crs_{zk}})$, an instance x and a witness ω where $\mathcal{R}(x, \omega) = 1$.

The prover runs $w = \lambda$ copies of the trapdoor sigma protocol Σ, denoted as $\Sigma^{(1)}, \ldots, \Sigma^{(w)}$ in parallel:

- Compute first round prover messages: $\alpha_i \leftarrow \Sigma^{(i)}.\mathsf{P}(1^\lambda, \mathsf{crs_{zk}}, x, \omega)$.
- Compute verifier challenges:

$$\{\beta_i\}_{i\in[w]} \leftarrow \mathsf{CIH.Hash}(\mathsf{k}, \{\Sigma^{(i)}.\mathsf{PreComp}(1^\lambda, \alpha_i)\}_{i\in[w]}).$$

- Compute third round prover messages: $\gamma_i \leftarrow \Sigma^{(i)}.\mathsf{P}(\beta_i)$.

It outputs the proof $\pi = \left(\{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]}\right)$.
Verifier $\mathsf{V}(\mathsf{crs}, x, \pi)$**:**The verifier takes as input a CRS $\mathsf{crs} = (\mathsf{k}, \mathsf{crs_{zk}})$, an instance x and a proof π. It performs the following steps:

- Parse the proof $\pi = \left(\{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]}\right)$.
- Compute verifier challenges:

$$\{\beta_i\}_{i\in[w]} \leftarrow \mathsf{CIH.Hash}\left(\mathsf{k}, \left\{\Sigma^{(i)}.\mathsf{PreComp}(1^\lambda, \alpha_i)\right\}_{i\in[w]}\right).$$

- For every $i \in [w]$, verify the transcript $(\alpha_i, \beta_i, \gamma_i)$ of $\Sigma^{(i)}$: $\mathsf{out}_i \leftarrow \Sigma^{(i)}.\mathsf{V}\left(\mathsf{crs_{zk}}, x, (\alpha_i, \beta_i, \gamma_i)\right)$. If any $\mathsf{out}_i = 0$, then output reject. Otherwise output accept.

Figure 5-8. NISWI Argument System Π for $\mathcal{NP}$.

Hence, the verifier accepts the proof with probability 1. □

Lemma 5.5.6 (Adaptive SWI)**.** *The proposed scheme* Π *is adaptive statistical witness indistinguishable.*

Proof. We prove the adaptive statistical WI property by constructing a series of hybrids.

- Hyb_0**:** This hybrid is simply the experiment Expr_0 in the definition of adaptive SWI. Let (ω_0, ω_1) be the two witnesses chosen by the adversary.

- $\mathsf{Hyb}_1^{i^*}$**:** This hybrid is the same as Hyb_0, except that for each $i < i^*$, we compute (α_i, γ_i) using ω_1. For each $i \geq i^*$, we compute (α_i, γ_i) using ω_0.

- $\mathsf{Hyb}_2^{i^*}$**:** This hybrid is the same as $\mathsf{Hyb}_1^{i^*}$, except that, before the output, we guess a random $\beta \leftarrow \{0,1\}$. Let $\{\beta_i\}_{i\in[w]} = \mathsf{CIH.Hash}(\mathsf{k}, \{\Sigma^{(i)}.\mathsf{PreComp}(1^\lambda, \alpha_i)\}_{i\in[w]})$. If $\beta = \beta_{i^*}$, then proceed to output. Otherwise, start running the hybrid from the beginning again. If the guessing process fails for λ times, then abort.

- $\mathsf{Hyb}_3^{i^*}$**:** This hybrid is the same as $\mathsf{Hyb}_2^{i^*}$, except that for $i = i^*$, we run the simulator on the guessed challenge β to compute $(\alpha_{i^*}, \gamma_{i^*}) \leftarrow \Sigma.\mathsf{S}(1^\lambda, \mathsf{crs}_{\mathsf{zk}}, x, \beta)$.

- Hyb_4 This hybrid is identical to the experiment Expr_1, where all (α_i, γ_i) tuples are computed using ω_1.

We now show that Hyb_0 and Hyb_4 are statistically indistinguishable.

$\mathsf{Hyb}_0 \equiv \mathsf{Hyb}_1^1$**:** This follows from the definition of these two hybrids.

$\mathsf{Hyb}_1^{i^*} \approx_s \mathsf{Hyb}_2^{i^*}$**:** Since β is sampled uniformly at random, it is independent of β_{i^*}. Hence each guess in $\mathsf{Hyb}_2^{i^*}$ is correct with probability $1/2$, and thus the hybrid aborts with probability $1/2^\lambda$. Conditioned on the event that $\mathsf{Hyb}_2^{i^*}$ doesn't abort, its output distribution is identical to $\mathsf{Hyb}_1^{i^*}$. Hence, the statistical distance between $\mathsf{Hyb}_1^{i^*}$ and $\mathsf{Hyb}_2^{i^*}$ is at most $1/2^\lambda$.

$\mathsf{Hyb}_2^{i^*} \approx_s \mathsf{Hyb}_3^{i^*}$**:** This follows from the adaptive statistical honest verifier zero-knowledge property of Σ.

$\mathsf{Hyb}_3^{i^*} \approx_s \mathsf{Hyb}_1^{i^*+1}$**:** This also follows from the adaptive statistical honest verifier zero-knowledge property of Σ.

$\mathsf{Hyb}_1^{w+1} \equiv \mathsf{Hyb}_4$**:** This follows from the definition of the hybrids above.

□

Lemma 5.5.7 (Non-adaptive Argument of Knowledge)**.** *The proposed scheme* Π *satisfies sub-exponential non-adaptive argument of knowledge.*

Proof. Let Ext be the extractor in the knowledge extraction property of the trapdoor sigma protocol. We build the extractor in Figure 5-9.

Extractor $E_1(1^\lambda)$

- Sample $\mathsf{k} \leftarrow \mathsf{CIH.Gen}(1^\lambda)$, $(\mathsf{crs}_{\mathsf{sound}}, \mathsf{td}) \leftarrow \Sigma.\mathsf{Gen}_{\mathsf{sound}}(1^\lambda)$.
- Output $(\mathsf{crs} = (\mathsf{k}, \mathsf{crs}_{\mathsf{sound}}), \mathsf{td})$.

Extractor $E_2(\mathsf{td}, x, \pi)$

- Parse $\pi = (\{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]})$.
- Let $\{\beta_i\}_{i\in[w]} = \mathsf{CIH.Hash}\left(\mathsf{k}, \left\{\Sigma.\mathsf{PreComp}(1^\lambda, \alpha_i)\right\}_{i\in[w]}\right)$.
- If $\beta_i = \Sigma.\mathsf{BadC}_\lambda(\Sigma.\mathsf{PreComp}(1^\lambda, \alpha_i), \mathsf{td})$ for all $i \in [w]$, then abort.
- Let i^* be the smallest i such that $\beta_i \neq \Sigma.\mathsf{BadC}_\lambda(\Sigma.\mathsf{PreComp}(1^\lambda, \alpha_i), \mathsf{td})$.
- Output $\Sigma.\mathsf{Ext}(\alpha_{i^*}, \beta_{i^*}, \gamma_{i^*}, \mathsf{td})$.

Figure 5-9. Extractor $E = (E_1, E_2)$.

Since the only difference between the CRS generated by CGen and the CRS obtained from E_1 is that, CGen invokes $\Sigma.\mathsf{Gen}_{\mathsf{zk}}$ while E_1 invokes $\Sigma.\mathsf{Gen}_{\mathsf{sound}}$, by the sub-exponential CRS indistinguishability, for any n.u. PPT cheating prover P^* we have

$$\Pr\left[x \leftarrow \mathsf{P}^*(1^\lambda), (\mathsf{crs}, \mathsf{td}) \leftarrow E_1(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) : \mathsf{V}(\mathsf{crs}, x, \pi) = 1\right] \geq$$
$$\Pr\left[x \leftarrow \mathsf{P}^*(1^\lambda), \mathsf{crs} \leftarrow \mathsf{CGen}(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) : \mathsf{V}(\mathsf{crs}, x, \pi) = 1\right] - \nu(\lambda),$$

where $\nu(\lambda)$ is a sub-exponential function.

By the correlation intractability of CIH, there exists a sub-exponential function ν' such that

$$\Pr\left[x \leftarrow \mathsf{P}^*(1^\lambda), (\mathsf{crs}, \mathsf{td}) \leftarrow E_1(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) : E_2(\mathsf{td}, x, \pi) \text{ abort}\right] \leq \nu'(\lambda).$$

Hence, we have

$$\begin{aligned}\Pr[x \leftarrow \mathsf{P}^*(1^\lambda), (\mathsf{crs}, \mathsf{td}) \leftarrow E_1(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) :& \\ \mathsf{V}(\mathsf{crs}, x, \pi) = 1 \wedge E_2(\mathsf{td}, x, \pi) \text{ does not abort}] \geq& \\ \Pr\left[x \leftarrow \mathsf{P}^*(1^\lambda), \mathsf{crs} \leftarrow \mathsf{CGen}(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) : \mathsf{V}(\mathsf{crs}, x, \pi) = 1\right] - \nu(\lambda) - \nu'(\lambda),&\end{aligned}$$

When $\mathsf{V}(\mathsf{crs}, x, \pi) = 1$, we have that $(\alpha_i, \beta_i, \gamma_i)$ is accepted, for any $i \in [w]$. By the knowledge extraction property of the trapdoor sigma protocol, when E_2 does not abort, we have $\mathcal{R}(x, \omega) = 1$. Hence, we finish the proof. □

5.5.4 From NISWI to Multi-Theorem NIZK

We now provide a general transformation from a statistical NISWI argument system for $\mathcal{NP}$ with non-adaptive argument of knowledge property to an adaptive, multi-theorem statistical NIZK argument system for $\mathcal{NP}$ with non-adaptive soundness. Our transformation relies on the hardness of discrete logarithm and uses a slight variant of the "OR-trick" from [46].

Construction. Let $\Pi' = (\Pi'.\mathsf{Gen}, \Pi'.\mathsf{P}, \Pi'.\mathsf{V})$ be a NISWI argument system for an NP-Complete language $\mathcal{L}'$ with (sub-exponential) non-adaptive argument of knowledge property. Let $\mathcal{G}$ be a prime-order group generator for which discrete logarithm is hard.

We build a NISZK argument system $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ for any $\mathcal{NP}$ language $\mathcal{L}$, with adaptive statistical zero-knowledge and (sub-exponential) non-adaptive computational soundness property. The construction is described in Figure 5-10.

Lemma 5.5.8 (Completeness)**.** *The proposed scheme* Π *satisfies completeness.*

NISZK Argument System Π for Language $\mathcal{L}$.

- **CRS Generation** $\mathsf{Gen}(1^\lambda)$**:**
  - Sample a CRS for Π': $\Pi'.\mathsf{crs} \leftarrow \Pi'.\mathsf{Gen}(1^\lambda)$.
  - Generate a prime order group $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$.
  - Sample $h \leftarrow \mathbb{G}$ uniformly at random.
  - Output $\mathsf{crs} = (\Pi'.\mathsf{crs}, (\mathbb{G}, p, g, h))$
- **Prover** $\mathsf{P}(\mathsf{crs}, x, \omega)$**:**
  - Let $\mathcal{L}_{\mathsf{or}}$ be an $\mathcal{NP}$ language, where an instance $(x, (\mathbb{G}, p, g, h)) \in \mathcal{L}_{\mathsf{or}}$ if and only if there exists a witness (ω, a) such that either ω is a witness for $x \in \mathcal{L}$ or $g^a = h$.
  - Let $x_{\mathsf{or}} = (x, (\mathbb{G}, p, g, h))$ and $\omega_{\mathsf{or}} = (\omega, 0)$. Use $\mathcal{NP}$ reduction on $(x_{\mathsf{or}}, \omega_{\mathsf{or}})$ to obtain an instance $x' \in \mathcal{L}'$ and witness ω' for x'.
  - Compute a proof for Π': $\pi \leftarrow \Pi'.\mathsf{P}(\Pi'.\mathsf{crs}, x', \omega')$.
  - Output π.
- **Verifier** $\mathsf{V}(\mathsf{crs}, x, \pi)$**:**
  - Parse $\mathsf{crs} = (\Pi'.\mathsf{crs}, (\mathbb{G}, p, g, h))$.
  - Let $x_{\mathsf{or}} = (x, (\mathbb{G}, p, g, h))$. Use $\mathcal{NP}$ reduction on x_{or} to obtain x'.
  - Output $\mathsf{out} \leftarrow \Pi'.\mathsf{V}(\Pi'.\mathsf{crs}, x', \pi)$.

Figure 5-10. NISZK argument system Π for language $\mathcal{L}$.

Proof. For any instance $x \in \mathcal{L}$, and any witness ω of x, by the definition of $\mathcal{L}_{\mathsf{or}}$, we have that x_{or} constructed in Figure 5-10 is in $\mathcal{L}_{\mathsf{or}}$, and $\omega_{\mathsf{or}} = (\omega, 0)$ is a witness for x_{or}. Hence, the completeness follows from the completeness of the underlying protocol Π'. □

Lemma 5.5.9 (Multi-Theorem Adaptive Statistical Zero-Knowledge)**.** *The proposed scheme* Π *is multi-theorem adaptive statistical zero knowledge.*

Proof. We build the following simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$.

- $\underline{\mathsf{S}_1(1^\lambda)}$:
  - Sample a CRS of Π': $\Pi'.\mathsf{crs} \leftarrow \Pi'.\mathsf{Gen}(1^\lambda)$.
  - Generate a prime order group $(\mathbb{G}, p, g) \leftarrow \mathsf{G}(1^\lambda)$.
  - Sample $t \leftarrow \mathbb{Z}_p$, and let $h = g^t$.
  - Output $\mathsf{crs} = (\Pi'.\mathsf{crs}, (\mathbb{G}, p, g, h))$, and $\mathsf{td} = (t, \mathsf{crs})$.
- $\underline{\mathsf{S}_2(\mathsf{td}, x)}$:
  - Let $x_{\mathsf{or}} = (x, (\mathbb{G}, p, g, h))$, and $\widetilde{\omega}_{\mathsf{or}} = (0, t)$. Use $\mathcal{NP}$ reduction on $(x_{\mathsf{or}}, \widetilde{\omega}_{\mathsf{or}})$ to obtain an instance $x' \in \mathcal{L}'$ and witness $\widetilde{\omega}'$ for x'. Run the prover algorithm of Π' to compute:
    $$\pi \leftarrow \Pi'.\mathsf{P}(\Pi'.\mathsf{crs}, x', \widetilde{\omega}').$$
  - Output π.

Since t is sampled uniformly at random from $\mathbb{Z}_p$ and g is a generator of $\mathbb{G}$, g^t is uniformly distributed over $\mathbb{G}$. Hence, the distribution of crs generated by the simulator is identical to that in the real execution. Then, the adaptive statistical zero knowledge property of Π follows from the adaptive statistical witness indistinguishability of Π'. □

Lemma 5.5.10 (Non-adaptive Computational Soundness)**.** *Assuming (sub-exponential) hardness of discrete logarithm, the proposed scheme* Π *satisfies (sub-exponential) non-adaptive computational soundness.*

Proof. Suppose there exists a n.u. PPT adversary P^*, and a non-negligible function $\epsilon(\lambda)$ such that

$$\Pr\left[x \leftarrow \mathsf{P}^*(1^\lambda), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) : x \notin \mathcal{L} \wedge \mathsf{V}(\mathsf{crs}, x, \pi) = 1\right] > \epsilon(\lambda),$$

for infinite many $\lambda \in \mathbb{N}$. Then there exists infinite many λ such that, there exists a random coin r_λ and $x_\lambda = \mathsf{P}^*(1^\lambda; r_\lambda)$ such that

$$\Pr\left[x_\lambda = \mathsf{P}^*(1^\lambda; r_\lambda), \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \pi \leftarrow \mathsf{P}^*(\mathsf{crs}) : x_\lambda \notin \mathcal{L} \wedge \mathsf{V}(\mathsf{crs}, x_\lambda, \pi) = 1\right] > \epsilon(\lambda),$$

where the probability is only over the randomness of $\mathsf{Gen}(1^\lambda)$ and $\mathsf{P}^*(\mathsf{crs})$. Since the probability is greater than 0, $x_\lambda \notin \mathcal{L}$.

We build the following non-uniform cheating prover P' for Π':

- $\mathsf{P}'(1^\lambda)$ computes $x_\lambda = \mathsf{P}^*(1^\lambda; r_\lambda)$, and generates a prime order group $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$. Then it samples $h \leftarrow \mathbb{G}$ and outputs $x_{\mathsf{or}} = (x_\lambda, (\mathbb{G}, p, g, h))$.
- Upon receiving input $\Pi'.\mathsf{crs}$, P' sets $\mathsf{crs} = (\Pi'.\mathsf{crs}, (\mathbb{G}, p, g, h))$. It computes $\pi \leftarrow \mathsf{P}^*(\mathsf{crs})$ and outputs π.

From the non-adaptive argument of knowledge property of Π', there exists an extractor $E = (E_1, E_2)$ and a negligible function $\nu(\lambda)$ such that

$$\begin{aligned}
&\Pr[x_{\mathsf{or}} \leftarrow \mathsf{P}'(1^\lambda), (\Pi'.\mathsf{crs}, \mathsf{td}) \leftarrow E_1(1^\lambda), \pi \leftarrow \mathsf{P}'(\Pi'.\mathsf{crs}), \omega_{\mathsf{or}} \leftarrow E_2(\mathsf{td}, x_{\mathsf{or}}, \pi) : \\
&\qquad\qquad \mathcal{R}_{\mathsf{or}}(x_{\mathsf{or}}, \omega_{\mathsf{or}}) = 1] \\
&\geq \Pr[x_{\mathsf{or}} \leftarrow \mathsf{P}'(1^\lambda), \Pi'.\mathsf{crs} \leftarrow \Pi'.\mathsf{Gen}(1^\lambda), \pi \leftarrow \mathsf{P}'(\Pi'.\mathsf{crs}) : \Pi'.\mathsf{V}(\Pi'.\mathsf{crs}, x_{\mathsf{or}}, \pi) = 1] - \nu(\lambda) \\
&\geq \epsilon(\lambda) - \nu(\lambda),
\end{aligned}$$

where the relation $\mathcal{R}_{\mathsf{or}}(x_{\mathsf{or}}, \omega_{\mathsf{or}}) = 1$, if and only if ω_{or} is a witness for the instance $x_{\mathsf{or}} \in \mathcal{L}_{\mathsf{or}}$.

Next, we build the following adversary $\mathcal{A}$ for the discrete logarithm problem:

- $\mathcal{A}$ receives as input a security parameter λ, a group $\mathbb{G}$ and its order p, a generator g, and $h \in \mathbb{G}$.
- It computes $x_\lambda = \mathsf{P}^*(1^\lambda; r_\lambda)$, and $x_{\mathsf{or}} = (x_\lambda, (\mathbb{G}, p, g, h))$.

- Next, it computes $(\Pi'.\mathsf{crs}, \mathsf{td}) \leftarrow E_1(1^\lambda)$, and sets $\mathsf{crs} = (\Pi'.\mathsf{crs}, (\mathbb{G}, p, g, h))$.
- Finally it computes $\pi \leftarrow \mathsf{P}^*(\mathsf{crs})$ and $\omega_{\mathsf{or}} \leftarrow E_2(\mathsf{td}, x_{\mathsf{or}}, \pi)$. It parses $\omega_{\mathsf{or}} = (\omega, t)$, and outputs t.

When the input to $\mathcal{A}$ is computed $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$ and $h \leftarrow \mathbb{G}$, the distribution of $\mathsf{crs} = (\Pi'.\mathsf{crs}, (\mathbb{G}, p, g, h))$ is identical to the distribution of the $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$ in the real execution of the protocol Π. Hence, the adversary $\mathcal{A}$ correctly simulates the cheating prover P'. Since $x_\lambda \notin \mathcal{L}$, if $\mathcal{R}_{\mathsf{or}}(x_{\mathsf{or}}, \omega_{\mathsf{or}} = (\omega, t)) = 1$, then $g^t = h$. Therefore,

$$\begin{aligned}\Pr\left[(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda), h \leftarrow \mathbb{G}, t \leftarrow \mathcal{A}(1^\lambda, \mathbb{G}, p, g, h) : g^t = h\right] \\ \geq \Pr[x_{\mathsf{or}} \leftarrow \mathsf{P}'(1^\lambda), (\Pi'.\mathsf{crs}, \mathsf{td}) \leftarrow E_1(1^\lambda), \pi \leftarrow \mathsf{P}'(\Pi'.\mathsf{crs}), \\ \omega_{\mathsf{or}} \leftarrow E_2(\mathsf{td}, x_{\mathsf{or}}, \pi) : \mathcal{R}_{\mathsf{or}}(x_{\mathsf{or}}, \omega_{\mathsf{or}}) = 1] \geq \epsilon(\lambda) - \nu(\lambda),\end{aligned}$$

for infinite many λ, which is a contradiction. □

5.5.5 Computational NIZKs for $\mathcal{NP}$ with Adaptive Soundness

In this section, we describe our construction of computational NIZK for $\mathcal{NP}$ with adaptive soundness in the common random string model. We proceed via the same steps as in the construction of statistical NIZK for $\mathcal{NP}$, except that we replace the lossy public-key encryption scheme in the construction of trapdoor sigma protocol with Elgamal encryption.

We outline each of the steps below.

Lemma 5.5.11. *If we replace the lossy public key encryption in the trapdoor sigma protocol in Section 5.5.2 with ElGamal encryption [99], then the resulting trapdoor sigma protocol $\overline{\Sigma}$ satisfies adaptive computational honest-verifier zero-knowledge and the bad challenge in TC^0 property. Furthermore, the resulting protocol is in the common random string model.*

Proof sketch. The proof of adaptive computational honest-verifier zero-knowledge follows from the computational semantic security of ElGammal encryption scheme.

For the bad challenge in TC^0 property, recall that a ciphertext in ElGamal encryption scheme is of the form $(c_1, c_2) \in \mathbb{G}^2$ and a secret key $\mathsf{sk} = s$ is an element in $\mathbb{Z}_p$. Furthermore, the decryption process simply involves computing $c_1^s \cdot c_2$, which is the same as in the case of lossy public-key encryption scheme in Section 5.5.1. Hence, using the same argument as in Lemma 5.5.3, it follows that the ElGamal encryption scheme also satisfies low-depth decryption property. Then, following the same argument as in Section 5.5.2, we have that $\overline{\Sigma}$ satisfies the bad challenge in TC^0 property.

Finally, we note that since a public key of ElGamal encryption scheme is of the form (g^s, g) where $g \leftarrow \mathbb{G}$, the public key is uniformly distributed over $\mathbb{G}^2$. Hence, $\overline{\Sigma}$ is in the common random string model. □

Theorem 5.5.12. *If we replace the trapdoor sigma protocol in Π with $\overline{\Sigma}$ obtained from Lemma 5.5.11 then the resulting protocol $\overline{\Pi}$ satisfies adaptive computational witness indistinguishability and (sub-exponential) adaptive argument of knowledge in the common random string model.*

Proof sketch. The proof of computational witness indistinguishability relies on the computational honest-verifier zero-knowledge property, and its proof follows the same idea as Lemma 5.5.6. The adaptive argument of knowledge property follows the same argument as in Lemma 5.5.7. □

Theorem 5.5.13. *In Figure 5-10, if we replace the NISWI Π' with the computational NIWI $\overline{\Pi}$ obtained from Theorem 5.5.12 then the resulting protocol satisfies adaptive multi-theorem adaptive computational zero-knowledge and (sub-exponential) adaptive computational soundness in the common random string model.*

The proof follows in the same manner as in Lemma 5.5.9 and Lemma 5.5.10.

5.6 Statistical Zap Arguments for $\mathcal{NP}$

In this section, we describe our construction of statistical Zap arguments for $\mathcal{NP}$. Our construction closely follows the recent works of [57, 58] who constructed statistical Zap arguments from quasi-polyomial hardness of LWE.

We proceed in the following two steps:

- In Section 5.6.1, we construct a two-round (public-coin) statistical-hiding commitment scheme with *low-depth extraction property*, which essentially requires that the extraction circuit is in TC^0.
- Next, in Section 5.6.2, we construct statistical Zap arguments by replacing the lossy public-key encryption scheme used in our construction of NISWI in Section 5.5.3 with the two-round commitment scheme from the previous step.

5.6.1 Statistical Hiding Commitments with Low-Depth Extraction

A (public-coin) statistical hiding commitment scheme with low-depth extraction (ECOM) is an interactive protocol between a receiver and a sender: in the first round, the receiver sends a commitment key to the sender. Using the key, the sender computes a commitment to some value and sends it to the receiver. We require such schemes with a statistical hiding property as well as a low-depth extraction property.

Here we are interested in two-round protocols – in the plain model – that achieve security against malicious receivers. More specifically, in such protocols, statistical hiding property is required to hold with respect to even adversarially chosen commitment keys. At the same time, we require the commitments to be extractable, i.e., it should be possibly to efficiently extract the committed value from the sender's message. The extraction property is only required to hold in a special "extraction mode" which is determined by choice of the commitment key.

The key to achieving these two properties simultaneously is to allow the "extraction mode" to happen with only negligible probability [109]. Following the syntax in [57], we provide an additional input – a random string $\mathbf{b}$ – to the commitment algorithm. The "extraction mode" key generation algorithm also takes a random string $\widetilde{\mathbf{b}}$ as an additional input. Consequently, extraction is only required when the strings $\mathbf{b}$ and $\widetilde{\mathbf{b}}$ are *equal*. As in prior works [57, 58, 109], this weak extraction guarantee suffices for our target application by careful use of complexity leveraging.

For our purposes, we further require that the extraction process can be represented via *low-depth* computation, namely, TC^0 circuits.

Definition. For any security parameter λ, let $\mathbb{G}$ denote a cyclic group of order $p = p(\lambda)$. Let $\mu = \mu(\lambda)$ be a polynomial in λ.

A group-based statistical hiding commitment scheme with low-depth extraction, and with message space $\mathbb{Z}_2$ and key space $\mathcal{K}$, is a tuple of algorithms $\mathsf{ECOM} = (\mathsf{Gen}, \mathsf{ExtGen}, \mathsf{Com})$ described as follows:

- $\mathsf{Gen}(1^\lambda)$: It takes as input a security parameter λ, and outputs a uniformly distributed "normal mode" commitment key K.
- $\mathsf{ExtGen}(1^\lambda, \widetilde{\mathbf{b}})$: It takes as input the security parameter λ, and a string $\widetilde{\mathbf{b}}$ of length μ, and outputs an "extraction mode" commitment key $\widetilde{K}$ and a trapdoor td.
- $\mathsf{Com}(\mathbf{b}, K, m; r)$: It takes as input the security parameter λ, an integer μ, a binary string $\mathbf{b} \in \{0,1\}^\mu$, a message m, and the random coins r, and outputs a commitment c.

We require ECOM to satisfy the following properties.

- **Key Verifiability:** There exists a PPT algorithm KeyVer such that, for any

string K,

$$\Pr\left[\mathsf{KeyVer}(1^\lambda, K) = 1 \iff K \in \mathcal{K}\right] = 1.$$

- **Key Indistinguishability:** For any n.u. PPT distinguisher $\mathcal{D}$, there exists a negligible function $\nu(\cdot)$ such that, for any $\lambda \in \mathbb{N}$, any $\widetilde{\mathbf{b}} \in \{0,1\}^\mu$,

$$\left|\Pr\left[K \leftarrow \mathsf{Gen}(1^\lambda) : \mathcal{D}(1^\lambda, K) = 1\right] - \Pr\left[(\widetilde{K}, \mathsf{td}) \leftarrow \mathsf{ExtGen}(1^\lambda, \widetilde{\mathbf{b}}) : \mathcal{D}(1^\lambda, \widetilde{K}) = 1\right]\right| \leq \nu(\lambda).$$

  We say that the ECOM achieves sub-exponential key indistinguishability, if there exists a constants c such that for any n.u. PPT distinguisher, the advantage $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

- **Statistical Hiding:** For any key $K \in \mathcal{K}$, any $m_1, m_2 \in \mathbb{Z}_2$, there exists a negligible function $\nu(\cdot)$ such that, for any $\lambda \in \mathbb{N}$,

$$\mathsf{SD}\left((\mathbf{b}, \mathsf{Com}(\mathbf{b}, K, m_1)), (\mathbf{b}, \mathsf{Com}(\mathbf{b}, K, m_2))\right) \leq \nu(\lambda).$$

- **Low-Depth Extraction:** We say that ECOM supports extraction in TC^0 if there exists a sequence of circuits $\{\mathsf{Dec}_\lambda(\cdot,\cdot)\}_\lambda$ in TC^0 and a deterministic polynomial-time algorithm $\mathsf{PreComp}(1^\lambda, \cdot)$ such that, for any $\lambda \in \mathbb{N}$, any binary string $\widetilde{\mathbf{b}} \in \{0,1\}^\mu$, and extraction mode key $(\widetilde{K}, \mathsf{td}) \leftarrow \mathsf{ExtGen}(1^\lambda, \widetilde{\mathbf{b}})$, and any message $m \in \mathbb{Z}_2$,

$$\Pr\left[c \leftarrow \mathsf{Com}(\widetilde{\mathbf{b}}, \widetilde{K}, m) : m' \leftarrow \mathsf{Dec}_\lambda(\mathsf{PreComp}(1^\lambda, c), \mathsf{td}) : m = m'\right] = 1.$$

5.6.1.1 Construction

Our construction follows the work of [109] who constructed statistical hiding extractable commitments generically from two-round oblivious transfer protocols. To achieve the low-depth extraction property, we instantiate their construction with the Naor-Pinkas oblivious transfer scheme based on DDH [81].

- $\mathsf{Gen}(1^\lambda)$: Sample and output μ strings uniformly at random, where each string is of the same length as an OT receiver first round message. Let K denote the output.

- $\mathsf{ExtGen}(1^\lambda, \widetilde{\mathbf{b}})$: Parse $\widetilde{\mathbf{b}} = (\widetilde{b}_1, \widetilde{b}_2, \ldots, \widetilde{b}_\mu)$. Generate μ first round OT messages

$$\mathsf{OT}_1(1^\lambda, \widetilde{b}_1), \mathsf{OT}_1(1^\lambda, \widetilde{b}_2), \ldots, \mathsf{OT}_1(1^\lambda, \widetilde{b}_\mu),$$

  and output them as commitment key $\widetilde{K}$. Let td denote the set of random strings used for computing the OT messages.

- $\mathsf{Com}(\mathbf{b}, K, m)$: Parse $\mathbf{b} = (b_1, b_2, \ldots, b_\mu)$ and $K = \{\mathsf{ot}_{1,i}\}_{i\in[\mu]}$.

  - Randomly sample $m_1, m_2, \ldots, m_\mu \in \mathbb{Z}_2$ such that $\sum m_i \bmod 2 = m$.
  - For each $i \in [\mu]$, compute the second round OT message $c_i = \mathsf{OT}_2(\mathsf{ot}_{1,i}, m_{i,0}, m_{i,1})$ with $m_{i,b_i} = m_i$, and sample $m_{i,1-b_i} \leftarrow \mathbb{Z}_2$ uniformly at random.
  - Output the second round messages $\{c_i\}_{i\in[\mu]}$ as the commitment.

The key verifiability follows directly from the underlying oblivious transfer protocol and the group instantiation. The key indistinguishability property follows from the pseudorandomness of receiver's message in Naor-Pinkas OT. The statistical hiding property follows the same argument as in [57, 58, 109]. Intuitively, for any key in the key space, one can use an inefficient extractor to extract $\widetilde{\mathbf{b}}$. Since $\mathbf{b}$ is sampled uniformly at random, with probability $1-2^{-\mu}$, there exists an index i such that $b_i \neq \widetilde{b}_i$. Hence, from the statistical sender-privacy of the OT, m_i is statistically hidden. Since $m_1, m_2, \ldots, m_\mu$ constitute an n-out-of-n secret sharing of m, m is also statistically hidden.

Low-Depth Extraction. When $\mathbf{b} = \widetilde{\mathbf{b}}$, to extract m, we need to proceed in the following two steps.

1. Use OT_3 and td to decrypt $m_1, m_2, \ldots, m_\mu$ from the commitment.

2. Compute $m = m_1 + m_2 + \ldots + m_\mu$ in $\mathbb{Z}_2$.

When we instantiate the OT with Naor-Pinkas OT, the output computation algorithm OT_3 is of the same form as the decryption algorithm of the lossy public key encryption in Section 5.5.1. Hence, it also satisfies the low-depth decryption property, and there exists a TC^0 circuit sequence $\{\mathsf{Dec}_\lambda\}_\lambda$ and a deterministic pre-computation algorithm $\mathsf{PreComp}$ such that $m_i = \mathsf{Dec}_\lambda(\mathsf{PreComp}(1^\lambda, c_i), \mathsf{td}_i)$, where td_i is the receiver's randomness for the i^{th} OT. For the second step, the summation of $m_1, m_2, \ldots, m_\mu$ in $\mathbb{Z}_2$ can be computed in TC^0 [108]. Composing these two steps, we prove the low-depth extraction property.

5.6.2 Construction of Statistical Zap Arguments

We now describe our construction of statistical Zap arguments for $\mathcal{NP}$ with non-adaptive soundness. We rely on the following ingredients:

- A statistical-hiding commitment scheme $\mathsf{ECOM} = (\mathsf{Gen}, \mathsf{ExtGen}, \mathsf{Com})$ with low-depth extraction and sub-exponential key indistinguishability.

  Let $\{\mathsf{ECOM.Dec}_\lambda\}$ be the sequence of low-depth decryption circuits and let $\mathsf{ECOM.PreComp}$ be the deterministic pre-computation algorithm associated with ECOM.

- A correlation intractable hash function CIH for TC^0, with sub-exponential correlation intractability.

- A "commit-and-open" trapdoor sigma protocol Σ for $\mathcal{NP}$ with low-depth bad challenge function, from Section 5.5.2.

Each of these ingredients can be based on sub-exponential DDH. We thus obtain our construction based on the same assumption.

Construction. Our construction and its security proof resembles the recent works of [57, 58]. Below, we sketch the construction and the proof of security. In the following, we set $\mu = \log^2 \lambda$.

- **Verifier:** It generates $K \leftarrow \mathsf{ECOM.Gen}(1^\lambda)$ and a CIH key $\mathsf{k} \leftarrow \mathsf{CIH.Gen}(1^\lambda)$, and sends (K, k) to the prover.

- **Prover:** Upon receiving a message (K, k), it first checks if $K \in \mathcal{K}$ by checking $\mathsf{KeyVer}(1^\lambda, K) = 1$. If the check fails, then abort.

  Next, it randomly samples $\mathbf{b} \leftarrow \{0,1\}^\mu$, and emulates $w = \lambda$ executions of the trapdoor sigma protocol Σ in parallel, as follows.

  - For each $i \in [w]$, run the prover's first round algorithm for Σ to generate the first round message α_i, where the commitment scheme is instantiated as $\mathsf{ECOM.Com}(\mathbf{b}, K, \cdot; \cdot)$.
  - Compute the verfier's challenges by using CIH: $\{\beta_i\}_{i\in[w]} \leftarrow \mathsf{CIH.Hash}(\mathsf{k}, \{\mathsf{ECOM.PreComp}(1^\lambda, \alpha_i)\}_{i\in[w]})$.
  - For each $i \in [w]$, run the prover's third round algorithm for Σ, with challenge bit β, where the commitment is instantiated as $\mathsf{ECOM.Com}(\mathbf{b}, K, \cdot; \cdot)$. Let γ_i be the computed message.

  Finally, the prover sends $\pi = (\mathbf{b}, \{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]})$ to the verifier.

- **Verification:** Given the transcript $((K, \mathsf{k}), \pi)$, the verifier parses $\pi = (\mathbf{b}, \{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]})$, and computes

$$\{\beta_i\}_{i\in[w]} = \mathsf{CIH.Hash}(\mathsf{k}, \{\mathsf{ECOM.PreComp}(1^\lambda, \alpha_i)\}_{i\in[w]}).$$

  For each $i \in [w]$, it verifies if $(\alpha_i, \beta_i, \gamma_i)$ is a accepting transcript for Σ, where the commitment is instantiated as $\mathsf{ECOM.Com}(\mathbf{b}, K, \cdot; \cdot)$.

This completes the description of the protocol. The completeness of the protocol directly follows from the completeness of the underlying sigma protocol Σ. The proof of statistical witness indistinguishability resembles the proof of Lemma 5.5.6. The only differences is that, here, for *any* $K \in \mathcal{K}$, when $\mathbf{b} \leftarrow \{0,1\}^\mu$ is sampled from uniform distribution, the commitment $\mathsf{ECOM.Com}(\mathbf{b}, K, \cdot; \cdot)$ is statistical hiding. Below, we argue that the protocol achieves non-adaptive computational soundness.

Non-adaptive Computational Soundness. For any instance $x \notin \mathcal{L}$, and any cheating prover P^* with success probability $\epsilon(\lambda)$, we build the following hybrids.

- Hyb_0: In this hybrid, the cheating prover tries to break the soundness of Zaps.
  - $K \leftarrow \mathsf{ECOM.Gen}(1^\lambda), \mathsf{k} \leftarrow \mathsf{CIH.Gen}(1^\lambda), \pi \leftarrow \mathsf{P}^*(1^\lambda, (K, \mathsf{k}))$
  - If $\pi = (\mathbf{b}, \{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]})$ is accepted, then output 1. Otherwise output 0.

  Since the cheating prover succeeds with probability $\epsilon(\lambda)$, we have that $\Pr[\mathsf{Hyb}_0 = 1] = \epsilon(\lambda)$.

- Hyb_1: This hybrid is the same as Hyb_0, except that we additionally guess $\mathbf{b}$ by sampling $\widetilde{\mathbf{b}}$ uniformly at random.
  - $\widetilde{\mathbf{b}} \leftarrow \{0,1\}^\mu, K \leftarrow \mathsf{ECOM.Gen}(1^\lambda), \mathsf{k} \leftarrow \mathsf{CIH.Gen}(1^\lambda), \pi \leftarrow \mathsf{P}^*(1^\lambda, (K, \mathsf{k}))$.
  - If $\pi = (\mathbf{b}, \{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]})$ is accepted and $\mathbf{b} = \widetilde{\mathbf{b}}$, then output 1. Otherwise output 0.

  Since the only difference between this hybrid and Hyb_0 is that we additionally guess $\widetilde{\mathbf{b}} \leftarrow \{0,1\}^\mu$, we have

  $$\Pr[\mathsf{Hyb}_1 = 1] \geq \epsilon(\lambda)/2^\mu.$$

- Hyb_2: This hybrid is the same as Hyb_1, except that we switch the K to a extraction key $\widetilde{K}$.

– $\widetilde{\mathbf{b}} \leftarrow \{0,1\}^{\mu}, \widetilde{K} \leftarrow \mathsf{ECOM.ExtGen}(1^{\lambda}, \widetilde{\mathbf{b}}), \mathsf{k} \leftarrow \mathsf{CIH.Gen}(1^{\lambda}), \pi \leftarrow \mathsf{P}^*(1^{\lambda}, (\widetilde{K}, \mathsf{k}))$.

– If $\pi = (\mathbf{b}, \{\alpha_i\}_{i\in[w]}, \{\gamma_i\}_{i\in[w]})$ is accepted and $\mathbf{b} = \widetilde{\mathbf{b}}$, then output 1. Otherwise output 0.

Since the only difference between this hybrid and Hyb_1 is the ECOM key generation, let $0 < c < 1$ be the constant in the sub-exponential key indistinguishability definition, we have

$$\Pr[\mathsf{Hyb}_2 = 1] \geq \Pr[\mathsf{Hyb}_1 = 1] - 2^{-\lambda^c} \geq \epsilon(\lambda)/2^{\mu} - 2^{-\lambda^c}.$$

Now we argue that there exists a constant $0 < c' < 1$ such that $\Pr[\mathsf{Hyb}_2 = 1] \leq 2^{-\lambda^{c'}}$ for any sufficiently λ. When the proof π is accepted and $\mathbf{b} = \widetilde{\mathbf{b}}$, the commitment scheme $\mathsf{ECOM.Com}(\mathbf{b}, K, \cdot; \cdot)$ is in "extraction" mode. By the same argument as in Section 5.5.2, we can prove the trapdoor sigma protocol instantiated with $\mathsf{ECOM.Com}(\mathbf{b}, K, \cdot; \cdot)$ as the commitment scheme satisfies the bad challenge function in TC^0 property. Hence, one can compute $\{\mathsf{PreComp}(1^{\lambda}, \alpha_i)\}_{i\in[w]}$ from any accepting transcript, which constitutes an attack for the correlation intractability of CIH. Since we assume the CIH is sub-exponential correlation intractable, let c' be the constant in the definition, we have $\Pr[\mathsf{Hyb}_2 = 1] \leq 2^{-\lambda^{c'}}$, for any sufficiently large λ.

Combining the two inequalities, we obtain $\epsilon(\lambda)/2^{\mu} - 2^{-\lambda^c} \leq \Pr[\mathsf{Hyb}_2 = 1] \leq 2^{-\lambda^{c'}}$, and thus $\epsilon(\lambda) \leq 2^{\mu} \cdot (2^{-\lambda^c} + 2^{-\lambda^{c'}})$. When we set $\mu = \log^2 \lambda$, $\epsilon(\lambda)$ is sub-exponential.

5.7 Instantiation from Elliptic Curves

In this section, we instantiate the result in Theorem 1.1.1 from any Elliptic curves in the short Weierstrass form over $\mathbb{F}_p$ ($p \neq 2, 3$). In Lemma 5.7.1, we show that to compute an iterative multiplication of λ group elements, we only need a $o(\log \lambda)$-depth threshold circuit. Next, when assuming DDH hardness against sub-exponential time adversaries, by complexity leveraging, we shrink the security parameter of the group to $\log^{O(1)} \lambda$,

while the commitments from such groups remain hiding for polynomial time adversaries. Then the depth of the threshold circuit computing the iterative multiplication becomes $o(\log\log\lambda)$, and hence can be handled by the CIH for $O(\log\log\lambda)$-depth threshold circuit in Theorem 5.4.6 (See Theorem 5.7.3).

Lemma 5.7.1. *Let a, b be two integers, $y^2 = x^3+ax+b$ be a short Weierstrass equation, and p be a prime with $4a^3 + 27b^2 \neq 0 \pmod p$. Then the short Weierstrass equation defines a Elliptic curve $\mathbb{G} = \{(X : Y : Z) \mid X, Y, Z \in \mathbb{F}_p, Y^2Z = X^3 + aXZ^2 + bZ^3\}$ in the finite projective space, where the identity is defined as $(0 : 1 : 0)$. Then*

- **Iterative Multiplication:** *For any integer n and security parameter λ, let the iterative multiplication function be $\mathsf{Mul}_{\lambda,n}(g_1, g_2, \ldots, g_n) = g_1 \cdot g_2 \cdot g_3 \ldots g_n$, where $g_1, g_2, \ldots, g_n \in \mathbb{G}$. Then $\mathsf{Mul}_{\lambda,n}$ can be computed by a series of polynomial-size threshold circuits $\{C_{\lambda,n}\}_{\lambda,n}$ of depth $o(\log n)$.*

- **Identity Testing:** *For any security parameter λ, define the $\mathsf{Identity}_\lambda(g)$ be the function which outputs whether g is the identity element $\mathbb{1}_\mathbb{G}$. Then $\mathsf{Identity}_\lambda$ can be computed in TC^0.*

Proof. We prove the lemma by computing $\mathsf{Mul}_{\lambda,n}$ and $\mathsf{Identity}_\lambda$, as follows.

- **Iterative Multiplication:** Our construction of the circuit computing $\mathsf{Mul}_{\lambda,n}$ is a complete B-ary tree of depth $\log_B n$, where $B = \log^* n$. Each tree node is a gate computing the iterative multiplication of its B children. The leaf nodes correspond to $g_1, g_2, \ldots, g_n$. Since the depth of the tree is $\log_B n = o(\log n)$, it suffices to show the product of B group elements can be computed in TC^0.

  Note that, for any group elements $h = (X_h : Y_h : Z_h), f = (X_f : Y_f : Z_f) \in \mathbb{G}$, if we denote $h \cdot f = u$, where $u = (X_u : Y_u : Z_u)$, then X_u, Y_u and Z_u can be computed by constant-degree polynomials in $\mathbb{Z}_p[X_h, Y_h, Z_h, X_f, Y_f, Z_f]$ using the unified point addition formula [112]. Hence, if we let $v = g_1 \cdot g_2 \cdot \ldots g_B$, where

$v = (X_v : Y_v : Z_v)$, then X_v, Y_v, Z_v are $2^{O(B)}$-degree multivariate polynomials about the coordinates of $g_1, g_2, \ldots, g_B$. Since we choose $B = \log^* n$, we have at most $(O(B))^{2^{O(B)}} = \mathsf{poly}(\lambda)(n)$ monomials in each polynomial. Since iterative multiplication and addition in $\mathbb{Z}_p$ can be computed in TC^0 by [108], we can evaluate each monomial in TC^0 by interative multiplication, and add them in TC^0 by iterative addition. Hence, v can be computed in TC^0.

- **Identity Testing:** Given a group element $g = (X : Y : Z)$, we can test whether $g = 1_{\mathbb{G}}$ by checking if $X = Z = 0$. Since this checking can be done in TC^0, we can compute $\mathsf{Identity}_\lambda$ in TC^0.

□

Lemma 5.7.2. *From any Elliptic curve in Lemma 5.7.1, we can construct a lossy public key encryption scheme whose decryption can be decomposed to a deterministic polynomial time algorithm* $\mathsf{PreComp}$ *and* $o(\log \lambda)$*-depth threshold circuit* Dec_λ*.*

The proof follows the same idea as Lemma 5.5.3, the only difference is that we apply Lemma 5.7.1 for the iterative multiplication and the identity testing.

Theorem 5.7.3. *Assuming DDH over the Elliptic curve in Lemma 5.7.1 is hard for any sub-exponential time adversary, we can obtain NIZKs and Zaps with the same properties in Theorem 1.1.1.*

If we assume any $2^{O(\lambda^c)}$-time adversary for DDH can obtain at most $2^{-\Omega(\lambda^c)}$ advantage, then we can set the security parameter of the lossy public key encryption scheme in the construction of NIZKs with the lossy public key encryption scheme to be $\log^{2/c} \lambda$. By Lemma 5.7.2, the decryption circuit of such lossy public key encryption can be decomposed to a deterministic algorithm $\mathsf{PreComp}$ and $o(\log \log \lambda)$-depth threshold circuit Dec_λ. Applying the CIH in Theorem 5.4.6, we obtain the result.

References

1. Brassard, G., Chaum, D. & Crépeau, C. Minimum Disclosure Proofs of Knowledge. *J. Comput. Syst. Sci.* **37,** 156–189 (1988).
2. Ben-Sasson, E. *et al. Zerocash: Decentralized Anonymous Payments from Bitcoin* in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014* (IEEE Computer Society, 2014), 459–474.
3. Goldwasser, S., Micali, S. & Rackoff, C. *The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract)* in *17th Annual ACM Symposium on Theory of Computing* (ACM Press, Providence, RI, USA, 1985), 291–304.
4. Bellare, M., Micciancio, D. & Warinschi, B. *Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions* in *Advances in Cryptology – EUROCRYPT 2003* (ed Biham, E.) **2656** (Springer, Heidelberg, Germany, Warsaw, Poland, 2003), 614–629.
5. Bender, A., Katz, J. & Morselli, R. *Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles* in *TCC 2006: 3rd Theory of Cryptography Conference* (eds Halevi, S. & Rabin, T.) **3876** (Springer, Heidelberg, Germany, New York, NY, USA, 2006), 60–79.
6. Naor, M. & Yung, M. *Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks* in *22nd Annual ACM Symposium on Theory of Computing* (ACM Press, Baltimore, MD, USA, 1990), 427–437.
7. Dolev, D., Dwork, C. & Naor, M. *Non-Malleable Cryptography (Extended Abstract)* in *23rd Annual ACM Symposium on Theory of Computing* (ACM Press, New Orleans, LA, USA, 1991), 542–552.
8. Ben-Sasson, E. *et al. Zerocash: Decentralized Anonymous Payments from Bitcoin* in *2014 IEEE Symposium on Security and Privacy* (IEEE Computer Society Press, Berkeley, CA, USA, 2014), 459–474.
9. Fortnow, L. *The Complexity of Perfect Zero-Knowledge (Extended Abstract)* in *19th Annual ACM Symposium on Theory of Computing* (ed Aho, A.) (ACM Press, New York City, NY, USA, 1987), 204–209.
10. Žák, S. A Turing machine time hierarchy. *Theoretical Computer Science* **26,** 327–333 (1983).
11. Boneh, D. *The Decision Diffie-Hellman problem* in *Algorithmic Number Theory* (ed Buhler, J. P.) (Springer Berlin Heidelberg, Berlin, Heidelberg, 1998), 48–63.

12. Boneh, D., Boyen, X. & Shacham, H. *Short Group Signatures* in *Advances in Cryptology – CRYPTO 2004* (ed Franklin, M.) **3152** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2004), 41–55.

13. Regev, O. *On lattices, learning with errors, random linear codes, and cryptography* in *37th Annual ACM Symposium on Theory of Computing* (eds Gabow, H. N. & Fagin, R.) (ACM Press, Baltimore, MA, USA, 2005), 84–93.

14. Kilian, J. *Founding Cryptography on Oblivious Transfer* in *20th Annual ACM Symposium on Theory of Computing* (ACM Press, Chicago, IL, USA, 1988), 20–31.

15. Micali, S. *CS Proofs (Extended Abstracts)* in *35th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Santa Fe, NM, USA, 1994), 436–453.

16. Groth, J. *Short Pairing-Based Non-interactive Zero-Knowledge Arguments* in *Advances in Cryptology – ASIACRYPT 2010* (ed Abe, M.) **6477** (Springer, Heidelberg, Germany, Singapore, 2010), 321–340.

17. Lipmaa, H. *Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments* in *TCC 2012: 9th Theory of Cryptography Conference* (ed Cramer, R.) **7194** (Springer, Heidelberg, Germany, Taormina, Sicily, Italy, 2012), 169–189.

18. Damgård, I., Faust, S. & Hazay, C. *Secure Two-Party Computation with Low Communication* in *TCC 2012: 9th Theory of Cryptography Conference* (ed Cramer, R.) **7194** (Springer, Heidelberg, Germany, Taormina, Sicily, Italy, 2012), 54–74.

19. Gennaro, R., Gentry, C., Parno, B. & Raykova, M. *Quadratic Span Programs and Succinct NIZKs without PCPs* in *Advances in Cryptology – EUROCRYPT 2013* (eds Johansson, T. & Nguyen, P. Q.) **7881** (Springer, Heidelberg, Germany, Athens, Greece, 2013), 626–645.

20. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R. & Paneth, O. *Succinct Non-interactive Arguments via Linear Interactive Proofs* in *TCC 2013: 10th Theory of Cryptography Conference* (ed Sahai, A.) **7785** (Springer, Heidelberg, Germany, Tokyo, Japan, 2013), 315–333.

21. Bitansky, N., Canetti, R., Chiesa, A. & Tromer, E. *Recursive composition and bootstrapping for SNARKS and proof-carrying data* in *45th Annual ACM Symposium on Theory of Computing* (eds Boneh, D., Roughgarden, T. & Feigenbaum, J.) (ACM Press, Palo Alto, CA, USA, 2013), 111–120.

22. Bitansky, N. *et al.* The Hunting of the SNARK. *Journal of Cryptology* **30,** 989–1066 (Oct. 2017).

23. Naor, M. *On Cryptographic Assumptions and Challenges (Invited Talk)* in *Advances in Cryptology – CRYPTO 2003* (ed Boneh, D.) **2729** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2003), 96–109.

24. Gentry, C. & Wichs, D. *Separating succinct non-interactive arguments from all falsifiable assumptions* in *43rd Annual ACM Symposium on Theory of Computing* (eds Fortnow, L. & Vadhan, S. P.) (ACM Press, San Jose, CA, USA, 2011), 99–108.

25. Canetti, R., Holmgren, J., Jain, A. & Vaikuntanathan, V. *Succinct Garbling and Indistinguishability Obfuscation for RAM Programs* in *47th Annual ACM Symposium on Theory of Computing* (eds Servedio, R. A. & Rubinfeld, R.) (ACM Press, Portland, OR, USA, 2015), 429–437.

26. Koppula, V., Lewko, A. B. & Waters, B. *Indistinguishability Obfuscation for Turing Machines with Unbounded Memory* in *47th Annual ACM Symposium on Theory of Computing* (eds Servedio, R. A. & Rubinfeld, R.) (ACM Press, Portland, OR, USA, 2015), 419–428.

27. Bitansky, N., Garg, S., Lin, H., Pass, R. & Telang, S. *Succinct Randomized Encodings and their Applications* in *47th Annual ACM Symposium on Theory of Computing* (eds Servedio, R. A. & Rubinfeld, R.) (ACM Press, Portland, OR, USA, 2015), 439–448.

28. Canetti, R. & Holmgren, J. *Fully Succinct Garbled RAM* in *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science* (ed Sudan, M.) (Association for Computing Machinery, Cambridge, MA, USA, 2016), 169–178.

29. Ananth, P., Chen, Y.-C., Chung, K.-M., Lin, H. & Lin, W.-K. *Delegating RAM Computations with Adaptive Soundness and Privacy* in *TCC 2016-B: 14th Theory of Cryptography Conference, Part II* (eds Hirt, M. & Smith, A. D.) **9986** (Springer, Heidelberg, Germany, Beijing, China, 2016), 3–30.

30. Chen, Y.-C. *et al. Cryptography for Parallel RAM from Indistinguishability Obfuscation* in *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science* (ed Sudan, M.) (Association for Computing Machinery, Cambridge, MA, USA, 2016), 179–190.

31. Paneth, O. & Rothblum, G. N. *On Zero-Testable Homomorphic Encryption and Publicly Verifiable Non-interactive Arguments* in *TCC 2017: 15th Theory of Cryptography Conference, Part II* (eds Kalai, Y. & Reyzin, L.) **10678** (Springer, Heidelberg, Germany, Baltimore, MD, USA, 2017), 283–315.

32. Jain, A., Lin, H. & Sahai, A. *Indistinguishability Obfuscation from Well-Founded Assumptions* in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (Association for Computing Machinery, Virtual, Italy, 2021), 60–73.

33. Canetti, R. *et al. Fiat-Shamir: from practice to theory* in *51st Annual ACM Symposium on Theory of Computing* (eds Charikar, M. & Cohen, E.) (ACM Press, Phoenix, AZ, USA, 2019), 1082–1090.

34. Jawale, R., Kalai, Y. T., Khurana, D. & Zhang, R. *SNARGs for Bounded Depth Computations and PPAD Hardness from Sub-Exponential LWE* in *STOC* (ACM, 2021).

35. Goldwasser, S., Kalai, Y. T. & Rothblum, G. N. *One-Time Programs* in *Advances in Cryptology – CRYPTO 2008* (ed Wagner, D.) **5157** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2008), 39–56.

36. Kalai, Y. T., Raz, R. & Rothblum, R. D. *Delegation for bounded space* in *45th Annual ACM Symposium on Theory of Computing* (eds Boneh, D., Roughgarden, T. & Feigenbaum, J.) (ACM Press, Palo Alto, CA, USA, 2013), 565–574.

37. Kalai, Y. T., Raz, R. & Rothblum, R. D. *How to delegate computations: the power of no-signaling proofs* in *46th Annual ACM Symposium on Theory of Computing* (ed Shmoys, D. B.) (ACM Press, New York, NY, USA, 2014), 485–494.

38. Kalai, Y. T. & Paneth, O. *Delegating RAM Computations* in *TCC 2016-B: 14th Theory of Cryptography Conference, Part II* (eds Hirt, M. & Smith, A. D.) **9986** (Springer, Heidelberg, Germany, Beijing, China, 2016), 91–118.

39. Brakerski, Z., Holmgren, J. & Kalai, Y. T. *Non-interactive delegation and batch NP verification from standard computational assumptions* in *49th Annual ACM Symposium on Theory of Computing* (eds Hatami, H., McKenzie, P. & King, V.) (ACM Press, Montreal, QC, Canada, 2017), 474–482.

40. Badrinarayanan, S., Kalai, Y. T., Khurana, D., Sahai, A. & Wichs, D. *Succinct delegation for low-space non-deterministic computation* in *50th Annual ACM Symposium on Theory of Computing* (eds Diakonikolas, I., Kempe, D. & Henzinger, M.) (ACM Press, Los Angeles, CA, USA, 2018), 709–721.

41. Holmgren, J. & Rothblum, R. *Delegating Computations with (Almost) Minimal Time and Space Overhead* in *59th Annual Symposium on Foundations of Computer Science* (ed Thorup, M.) (IEEE Computer Society Press, Paris, France, 2018), 124–135.

42. Brakerski, Z. & Kalai, Y. *Witness Indistinguishability for Any Single-Round Argument with Applications to Access Control* in *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II* (eds Kiayias, A., Kohlweiss, M., Wallden, P. & Zikas, V.) **12111** (Springer, Heidelberg, Germany, Edinburgh, UK, 2020), 97–123.

43. Kalai, Y. T., Paneth, O. & Yang, L. *How to delegate computations publicly* in *51st Annual ACM Symposium on Theory of Computing* (eds Charikar, M. & Cohen, E.) (ACM Press, Phoenix, AZ, USA, 2019), 1115–1124.

44. De Santis, A., Micali, S. & Persiano, G. *Non-Interactive Zero-Knowledge Proof Systems* in *Advances in Cryptology – CRYPTO'87* (ed Pomerance, C.) **293** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 1988), 52–72.

45. Blum, M., Feldman, P. & Micali, S. *Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)* in *20th Annual ACM Symposium on Theory of Computing* (ACM Press, Chicago, IL, USA, 1988), 103–112.

46. Feige, U., Lapidot, D. & Shamir, A. *Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String (Extended Abstract)* in *31st Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, St. Louis, MO, USA, 1990), 308–317.

47. Canetti, R., Halevi, S. & Katz, J. *A Forward-Secure Public-Key Encryption Scheme* in *Advances in Cryptology – EUROCRYPT 2003* (ed Biham, E.) **2656** (Springer, Heidelberg, Germany, Warsaw, Poland, 2003), 255–271.

48. Groth, J., Ostrovsky, R. & Sahai, A. *Perfect Non-interactive Zero Knowledge for NP* in *Advances in Cryptology – EUROCRYPT 2006* (ed Vaudenay, S.) **4004** (Springer, Heidelberg, Germany, St. Petersburg, Russia, 2006), 339–358.

49. Groth, J., Ostrovsky, R. & Sahai, A. *Non-interactive Zaps and New Techniques for NIZK* in *Advances in Cryptology – CRYPTO 2006* (ed Dwork, C.) **4117** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2006), 97–111.

50. Peikert, C. & Shiehian, S. *Noninteractive Zero Knowledge for NP from (Plain) Learning with Errors* in *Advances in Cryptology – CRYPTO 2019, Part I* (eds Boldyreva, A. & Micciancio, D.) **11692** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2019), 89–114.

51. Brakerski, Z., Koppula, V. & Mour, T. *NIZK from LPN and Trapdoor Hash via Correlation Intractability for Approximable Relations* in *Advances in Cryptology – CRYPTO 2020, Part III* (eds Micciancio, D. & Ristenpart, T.) **12172** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2020), 738–767.

52. Lombardi, A., Vaikuntanathan, V. & Wichs, D. *Statistical ZAPR Arguments from Bilinear Maps* in *Advances in Cryptology – EUROCRYPT 2020, Part III* (eds Canteaut, A. & Ishai, Y.) **12107** (Springer, Heidelberg, Germany, Zagreb, Croatia, 2020), 620–641.

53. Pass, R. *Unprovable Security of Perfect NIZK and Non-interactive Non-malleable Commitments* in *TCC 2013: 10th Theory of Cryptography Conference* (ed Sahai, A.) **7785** (Springer, Heidelberg, Germany, Tokyo, Japan, 2013), 334–354.

54. Adleman, L. *A subexponential algorithm for the discrete logarithm problem with applications to cryptography* in *20th Annual Symposium on Foundations of Computer Science* (1979), 55–60.

55. Coppersmith, D., Odlyzko, A. M. & Schroeppel, R. Discrete Logarithms in GF(p). *Algorithmica* **1,** 1–15 (Jan. 1986).

56. Canetti, R., Chen, Y., Reyzin, L. & Rothblum, R. D. *Fiat-Shamir and Correlation Intractability from Strong KDM-Secure Encryption* in *Advances in Cryptology – EUROCRYPT 2018, Part I* (eds Nielsen, J. B. & Rijmen, V.) **10820** (Springer, Heidelberg, Germany, Tel Aviv, Israel, 2018), 91–122.

57. Badrinarayanan, S., Fernando, R., Jain, A., Khurana, D. & Sahai, A. *Statistical ZAP Arguments* in *Advances in Cryptology – EUROCRYPT 2020, Part III* (eds Canteaut, A. & Ishai, Y.) **12107** (Springer, Heidelberg, Germany, Zagreb, Croatia, 2020), 642–667.

58. Goyal, V., Jain, A., Jin, Z. & Malavolta, G. *Statistical Zaps and New Oblivious Transfer Protocols* in *Advances in Cryptology – EUROCRYPT 2020, Part III* (eds Canteaut, A. & Ishai, Y.) **12107** (Springer, Heidelberg, Germany, Zagreb, Croatia, 2020), 668–699.

59. Boneh, D. & Franklin, M. K. *Identity-Based Encryption from the Weil Pairing* in *Advances in Cryptology – CRYPTO 2001* (ed Kilian, J.) **2139** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2001), 213–229.

60. Sahai, A. & Waters, B. R. *Fuzzy Identity-Based Encryption* in *Advances in Cryptology – EUROCRYPT 2005* (ed Cramer, R.) **3494** (Springer, Heidelberg, Germany, Aarhus, Denmark, 2005), 457–473.

61. Goyal, V., Pandey, O., Sahai, A. & Waters, B. *Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data* in *ACM CCS 2006: 13th Conference on Computer and Communications Security* (eds Juels, A., Wright, R. N. & De Capitani di Vimercati, S.) Available as Cryptology ePrint Archive Report 2006/309 (ACM Press, Alexandria, Virginia, USA, 2006), 89–98.

62. Boneh, D., Sahai, A. & Waters, B. *Functional Encryption: Definitions and Challenges* in *TCC 2011: 8th Theory of Cryptography Conference* (ed Ishai, Y.) **6597** (Springer, Heidelberg, Germany, Providence, RI, USA, 2011), 253–273.

63. O'Neill, A. Definitional Issues in Functional Encryption. *IACR Cryptol. ePrint Arch.* **2010,** 556 (2010).

64. Döttling, N. & Garg, S. *Identity-Based Encryption from the Diffie-Hellman Assumption* in *Advances in Cryptology – CRYPTO 2017, Part I* (eds Katz, J. & Shacham, H.) **10401** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2017), 537–569.

65. Couteau, G., Katsumata, S. & Ursu, B. *Non-interactive Zero-Knowledge in Pairing-Free Groups from Weaker Assumptions* in *Advances in Cryptology – EUROCRYPT 2020, Part III* (eds Canteaut, A. & Ishai, Y.) **12107** (Springer, Heidelberg, Germany, Zagreb, Croatia, 2020), 442–471.

66. Fiat, A. & Shamir, A. *How to Prove Yourself: Practical Solutions to Identification and Signature Problems* in *Advances in Cryptology – CRYPTO'86* (ed Odlyzko, A. M.) **263** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, Aug. 1987), 186–194.

67. Holmgren, J. & Lombardi, A. *Cryptographic Hashing from Strong One-Way Functions (Or: One-Way Product Functions and Their Applications)* in *59th Annual Symposium on Foundations of Computer Science* (ed Thorup, M.) (IEEE Computer Society Press, Paris, France, 2018), 850–858.

68. Holmgren, J., Lombardi, A. & Rothblum, R. Fiat-Shamir via List-Recoverable Codes (or: Parallel Repetition of GMW is not Zero-Knowledge). *STOC* (2021).

69. Kilian, J. *A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)* in *24th Annual ACM Symposium on Theory of Computing* (ACM Press, Victoria, BC, Canada, 1992), 723–732.

70. Choudhuri, A. R., Jain, A. & Jin, Z. *Non-interactive Batch Arguments for NP from Standard Assumptions* in *Advances in Cryptology – CRYPTO 2021, Part IV* (eds Malkin, T. & Peikert, C.) **12828** (Springer, Heidelberg, Germany, Virtual Event, 2021), 394–423.

71. Canetti, R., Chen, Y. & Reyzin, L. *On the Correlation Intractability of Obfuscated Pseudorandom Functions* in *TCC 2016-A: 13th Theory of Cryptography Conference, Part I* (eds Kushilevitz, E. & Malkin, T.) **9562** (Springer, Heidelberg, Germany, Tel Aviv, Israel, 2016), 389–415.

72. Kalai, Y. T., Rothblum, G. N. & Rothblum, R. D. *From Obfuscation to the Security of Fiat-Shamir for Proofs* in *Advances in Cryptology – CRYPTO 2017, Part II* (eds Katz, J. & Shacham, H.) **10402** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2017), 224–251.

73. Lombardi, A. & Vaikuntanathan, V. *Fiat-Shamir for Repeated Squaring with Applications to PPAD-Hardness and VDFs* in *Advances in Cryptology – CRYPTO 2020, Part III* (eds Micciancio, D. & Ristenpart, T.) **12172** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2020), 632–651.

74. Boneh, D., Bonneau, J., Bünz, B. & Fisch, B. *Verifiable Delay Functions* in *Advances in Cryptology – CRYPTO 2018, Part I* (eds Shacham, H. & Boldyreva, A.) **10991** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2018), 757–788.

75. Choudhuri, A. R. *et al. Finding a Nash equilibrium is no easier than breaking Fiat-Shamir* in *51st Annual ACM Symposium on Theory of Computing* (eds Charikar, M. & Cohen, E.) (ACM Press, Phoenix, AZ, USA, 2019), 1103–1114.

76. Regev, O. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. *J. ACM* **56** (Sept. 2009).

77. Peikert, C., Regev, O. & Stephens-Davidowitz, N. *Pseudorandomness of ring-LWE for any ring and modulus* in *49th Annual ACM Symposium on Theory of Computing* (eds Hatami, H., McKenzie, P. & King, V.) (ACM Press, Montreal, QC, Canada, 2017), 461–473.

78. Peikert, C. *Public-key cryptosystems from the worst-case shortest vector problem: extended abstract* in *41st Annual ACM Symposium on Theory of Computing* (ed Mitzenmacher, M.) (ACM Press, Bethesda, MD, USA, 2009), 333–342.

79. Brakerski, Z., Langlois, A., Peikert, C., Regev, O. & Stehlé, D. *Classical hardness of learning with errors* in *45th Annual ACM Symposium on Theory of Computing* (eds Boneh, D., Roughgarden, T. & Feigenbaum, J.) (ACM Press, Palo Alto, CA, USA, 2013), 575–584.

80. Dwork, C. & Naor, M. *Zaps and Their Applications* in *41st Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Redondo Beach, CA, USA, 2000), 283–293.

81. Naor, M. & Pinkas, B. *Efficient Oblivious Transfer Protocols* in *12th Annual ACM-SIAM Symposium on Discrete Algorithms* (ed Kosaraju, S. R.) (ACM-SIAM, Washington, DC, USA, 2001), 448–457.

82. Döttling, N. *et al. Trapdoor Hash Functions and Their Applications* in *Advances in Cryptology – CRYPTO 2019, Part III* (eds Boldyreva, A. & Micciancio, D.) **11694** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2019), 3–32.

83. Hubacek, P. & Wichs, D. *On the Communication Complexity of Secure Function Evaluation with Long Output* in *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science* (ed Roughgarden, T.) (Association for Computing Machinery, Rehovot, Israel, 2015), 163–172.

84. González, A. & Zacharakis, A. *Fully-succinct Publicly Verifiable Delegation from Constant-Size Assumptions* Cryptology ePrint Archive, Report 2021/353. https://eprint.iacr.org/2021/353. 2021.

85. Ciampi, M., Parisella, R. & Venturi, D. *On Adaptive Security of Delayed-Input Sigma Protocols and Fiat-Shamir NIZKs* in *SCN 20: 12th International Conference on Security in Communication Networks* (eds Galdi, C. & Kolesnikov, V.) **12238** (Springer, Heidelberg, Germany, Amalfi, Italy, 2020), 670–690.

86. Jain, A. & Jin, Z. *Non-interactive Zero Knowledge from Sub-exponential DDH* in *Advances in Cryptology – EUROCRYPT 2021, Part I* (eds Canteaut, A. & Standaert, F.-X.) **12696** (Springer, Heidelberg, Germany, Zagreb, Croatia, 2021), 3–32.

87. Canetti, R., Goldreich, O. & Halevi, S. The Random Oracle Methodology, Revisited. *J. ACM* **51,** 557–594 (July 2004).

88. Bartusek, J., Bronfman, L., Holmgren, J., Ma, F. & Rothblum, R. D. *On the (In)security of Kilian-Based SNARGs* in *TCC 2019: 17th Theory of Cryptography Conference, Part II* (eds Hofheinz, D. & Rosen, A.) **11892** (Springer, Heidelberg, Germany, Nuremberg, Germany, 2019), 522–551.

89. Barak, B. *How to Go Beyond the Black-Box Simulation Barrier* in *42nd Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Las Vegas, NV, USA, 2001), 106–115.

90. Goldwasser, S. & Kalai, Y. T. *On the (In)security of the Fiat-Shamir Paradigm* in *44th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Cambridge, MA, USA, 2003), 102–115.

91. Goldwasser, S., Kalai, Y. T. & Rothblum, G. N. *Delegating computation: interactive proofs for muggles* in *40th Annual ACM Symposium on Theory of Computing* (eds Ladner, R. E. & Dwork, C.) (ACM Press, Victoria, BC, Canada, 2008), 113–122.

92. Reingold, O., Rothblum, G. N. & Rothblum, R. D. *Constant-round interactive proofs for delegating computation* in *48th Annual ACM Symposium on Theory of Computing* (eds Wichs, D. & Mansour, Y.) (ACM Press, Cambridge, MA, USA, 2016), 49–62.

93. Reingold, O., Rothblum, G. N. & Rothblum, R. D. *Efficient Batch Verification for UP* in *Computational Complexity Conference* **102** (Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018), 22:1–22:23.

94. Rothblum, G. N. & Rothblum, R. D. *Batch Verification and Proofs of Proximity with Polylog Overhead* in *TCC 2020: 18th Theory of Cryptography Conference, Part II* (eds Pass, R. & Pietrzak, K.) **12551** (Springer, Heidelberg, Germany, Durham, NC, USA, 2020), 108–138.

95. Setty, S. *Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup* in *Advances in Cryptology – CRYPTO 2020, Part III* (eds Micciancio, D. & Ristenpart, T.) **12172** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2020), 704–737.

96. Babai, L., Fortnow, L., Levin, L. A. & Szegedy, M. *Checking Computations in Polylogarithmic Time* in *23rd Annual ACM Symposium on Theory of Computing* (ACM Press, New Orleans, LA, USA, 1991), 21–31.

97. Merkle, R. C. *A Digital Signature Based on a Conventional Encryption Function* in *Advances in Cryptology – CRYPTO'87* (ed Pomerance, C.) **293** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 1988), 369–378.

98. Boyle, E., Gilboa, N. & Ishai, Y. *Breaking the Circuit Size Barrier for Secure Computation Under DDH* in *Advances in Cryptology – CRYPTO 2016, Part I* (eds Robshaw, M. & Katz, J.) **9814** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2016), 509–539.

99. Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* **31,** 469–472 (1985).

100. Smolensky, R. *Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity* in *19th Annual ACM Symposium on Theory of Computing* (ed Aho, A.) (ACM Press, New York City, NY, USA, 1987), 77–82.

101. Smolensky, R. *On Representations by Low-Degree Polynomials* in *34th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Palo Alto, CA, USA, 1993), 130–138.

102. Oliveira, I. C., Santhanam, R. & Srinivasan, S. *Parity Helps to Compute Majority* in *34th Computational Complexity Conference (CCC 2019)* (ed Shpilka, A.) **137** (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019), 23:1–23:17.

103. Kopparty, S. AC^0 lower bounds and pseudorandomness. *Lecture notes for 'Topics in Complexity Theory and Pseudorandomness'.* https://sites.math.rutgers.edu/~sk1233/courses/topics-S13/lec4.pdf (2013).

104. Goldreich, O., Micali, S. & Wigderson, A. *How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority* in *19th Annual ACM Symposium on Theory of Computing* (ed Aho, A.) (ACM Press, New York City, NY, USA, 1987), 218–229.

105. Kol, G. & Naor, M. *Cryptography and Game Theory: Designing Protocols for Exchanging Information* in *TCC 2008: 5th Theory of Cryptography Conference* (ed Canetti, R.) **4948** (Springer, Heidelberg, Germany, San Francisco, CA, USA, 2008), 320–339.

106. Peikert, C., Vaikuntanathan, V. & Waters, B. *A Framework for Efficient and Composable Oblivious Transfer* in *Advances in Cryptology – CRYPTO 2008* (ed Wagner, D.) **5157** (Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 2008), 554–571.

107. Bellare, M., Hofheinz, D. & Yilek, S. *Possibility and Impossibility Results for Encryption and Commitment Secure under Selective Opening* in *Advances in Cryptology – EUROCRYPT 2009* (ed Joux, A.) **5479** (Springer, Heidelberg, Germany, Cologne, Germany, 2009), 1–35.

108. Reif, J. H. & Tate, S. R. On threshold circuits and polynomial computation. *SIAM Journal on Computing* **21,** 896–908 (1992).

109. Kalai, Y. T., Khurana, D. & Sahai, A. *Statistical Witness Indistinguishability (and more) in Two Messages* in *Advances in Cryptology – EUROCRYPT 2018, Part III* (eds Nielsen, J. B. & Rijmen, V.) **10822** (Springer, Heidelberg, Germany, Tel Aviv, Israel, 2018), 34–65.

110. Khurana, D. & Sahai, A. *How to Achieve Non-Malleability in One or Two Rounds* in *58th Annual Symposium on Foundations of Computer Science* (ed Umans, C.) (IEEE Computer Society Press, Berkeley, CA, USA, 2017), 564–575.

111. Blum, M. *How to prove a theorem so no one else can claim it* in *In: Proceedings of the International Congress of Mathematicians* (1987), 1444–1451.

112. Brier, E. & Joye, M. *Weierstraß Elliptic Curves and Side-Channel Attacks* in *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography* (eds Naccache, D. & Paillier, P.) **2274** (Springer, Heidelberg, Germany, Paris, France, 2002), 335–345.

Curriculum Vitae

EDUCATION

2017–Present PhD student, Department of Computer Science
Johns Hopkins University

2013–2017 Undergraduate student, Department of Mathematics
Fudan University

PUBLICATIONS

Abhishek Jain and Zhengzhong Jin, *Indistinguishability Obfuscation via Mathematical Proofs of Equivalence* in *63rd Annual Symposium on Foundations of Computer Science (FOCS), 2022.*

Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin and Giulio Malavolta, *Pre-Constrained Encryption* in *13th Innovations in Theoretical Computer Science (ITCS), 2022.*

Arka Rai Choudhuri, Abhishek Jain and Zhengzhong Jin, *SNARGs for $\mathcal{P}$ from LWE* in *62nd Annual Symposium on Foundations of Computer Science (FOCS), 2021.*

Arka Rai Choudhuri, Abhishek Jain and Zhengzhong Jin, *Non-Interactive Batch Arguments for NP from Standard Assumptions* in *41st Annual International Cryptology Conference (CRYPTO), 2021.*

Abhishek Jain and Zhengzhong Jin, *Non-Interactive Zero Knowledge from Sub-exponential DDH* in *40th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), 2021.*

Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin and Giulio Malavolta, *Unbounded Multi-party Computation from Learning with Errors* in *40th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), 2021.*

Kuan Cheng, Alireza Farhadi, MohammadTaghi Hajiaghayi, Zhengzhong Jin, Xin Li, Aviad Rubinstein, Saeed Seddighin and Yu Zheng, *Streaming and Small Space Approximation Algorithms for Edit Distance and Longest Common Subsequence* in *48th International Colloquium on Automata, Languages, and Programming (ICALP), 2021.*

Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin and Giulio Malavolta, *Multi-key Fully-Homomorphic Encryption in the Plain Model* in *18th Theory of Cryptography Conference (TCC), 2020*

Vipul Goyal, Abhishek Jain, Zhengzhong Jin and Giulio Malavolta, *Statistical Zaps and New Oblivious Transfer Protocols* in *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), 2020.*

James Bartusek, Brent Carmer, Abhishek Jain, Zhengzhong Jin, Tancrède Lepoint, Fermi Ma, Tal Malkin, Alex J. Malozemoff and Mariana Raykova *Public-Key Function-Private Hidden Vector Encryption (and More)* in *25th Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), 2020.*

Kuan Cheng, Zhengzhong Jin, Xin Li and Ke Wu, *Block Edit Errors with Transpositions: Deterministic Document Exchange Protocols and Almost Optimal Binary Codes* in *46th International Colloquium on Automata, Languages and Programming (ICALP), 2019.*

Kuan Cheng, Zhengzhong Jin, Xin Li and Ke Wu, *Deterministic Document Exchange Protocols, and Almost Optimal Binary Codes for Edit Errors* in *59th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2018.*

Zhengzhong Jin and Yunlei Zhao, *Generic and Practical Key Establishment from Lattice* in *International Conference on Applied Cryptography and Network Security (ACNS), 2019.*