Abstract of “Enhancing Privacy-Preserving Proof Systems” by Apoorvaa Deshpande, Ph.D., Brown University, February 2020.

In today’s age of blockchain and cloud-computing, we are constantly in search of tools that offer accountability without compromising privacy. A class of predominantly used tools is proof systems that provide soundness and privacy guarantees. Over the years, proof systems with different privacy guarantees such as zero-knowledge (ZK), witness-indistinguishability (WI) or witness-hiding (WH) have been constructed. For a lot of applications, it is desirable that proof systems have enhanced properties such as homomorphism, or efficiency for special-purpose languages. In this thesis, we propose and develop the following enhancements to proof systems:

Fully Homomorphic NIZK and NIWI Proofs. Given non-interactive zero-knowledge (NIZK) or witness-indistinguishable (NIWI) proofs for different statements, we give a framework to form proofs for new inferred statements. The security guarantee along with soundness and zero-knowledge is that of unlinkability; an inferred proof should be indistinguishable from a fresh proof for a combined statement.

Privacy Preserving Verifiable Key Directories. The current implementations of online chat services place a lot of trust in the central server that stores all the usernames and their public keys. We initiate the study of the primitive of *Verifiable Key Directories* (VKD) to formalize the security and privacy of a key verification service. As a key building block, we develop append-only zero-knowledge sets and give highly efficient constructions for the same.

Proofs of Ignorance. We introduce the notion of *proofs of ignorance*; we formalize what it means to provably not know and explore the settings in which one could give a convincing proof of ignorance. We define the notion of proofs of ignorance for all of NP, construct such proofs for a subset of NP languages, and show applications to witness-hiding protocols.

Enhancing Privacy-Preserving Proof Systems

by
Apoorvaa Deshpande

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
February 2020

This dissertation by Apoorvaa Deshpande is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____
Anna Lysyanskaya, Director

Recommended to the Graduate Council

Date _____ _____
Yael Tauman Kalai, Reader

Date _____ _____
Seny Kamara, Reader

Date _____ _____
Maurice Herlihy, Reader

Approved by the Graduate Council

Date _____ _____
Andrew G. Campbell
Dean of the Graduate School

Acknowledgements

PhD has been a wonderful journey with all its ups and downs, the paper acceptances and rejections, student discounts and often lack thereof! This journey would not have been possible without the support of amazing mentors, colleagues, friends and family.

First and foremost, I thank my advisor Anna Lysyanskaya for taking me on as her student. I have learnt not only a lot of cryptographic concepts from her, but also a lens to broadly think about privacy and security. I am very thankful to my committee members Yael Tauman Kalai, Seny Kamara and Maurice Herlihy.

I had the great fortune of working closely with Yael for the last two years of my PhD (as an intern at MSR New England and as a visiting student at CSAIL, MIT). She has been an incredible mentor to me and I have learnt so much from her, not only about research but how to live life most productively.

I thank Seny for his constant encouragement and advice, I have had several illuminating conversations with him. I thank Maurice for sharing with me his insights on the blockchain space, especially on how to separate reality from hype. I had the fortune of working with him briefly and I am always amazed at his research vision and deep insights!

I am grateful to all my other collaborators: Prabhanjan Ananth, Marshall Ball, Elette Boyle, Melissa Chase, Esha Ghosh, Venkata Kopulla, Harjasleen Malvai, Alon Rosen, Vinod Vaikuntanathan, Prashant Vasudevan and Brent Waters. My heartfelt thanks to my officemates Sasha, Marilyn and special thanks to Megumi for the endless conversations and frequent life advice!

I am thankful to the Computer Science Department at Brown; I will cherish technical as well as non-technical conversations with faculty Roberto Tamassia, John Savage, Vaseleios Kemerlis, Paul Valiant, Eli Upfal, Ugur Centinemel and with students Evgenios, Alexandra, Nediyana, Jasper, Thomas, Amy, Erfan, Hannah, Lorenzo, Elizabeth, David. A special mention to Esha who has been a wonderful mentor and friend to me!

The CS department at Brown could not have fostered such a great academic environment without the highly efficient TStaff and AStaff members. My sincere thanks to Lauren Clarke, Jane McIlmail, Jesse Polhemus, Genie DeGouveia, Donald Johwa, and to all other staff members. I am also grateful for the support of NSF grants, Celo Fellowship and the Paris C. Kanellakis Fellowship for this dissertation.

I had the opportunity to do wonderful internships and visits throughout my PhD. In the summer after my first year, I attended the special program on cryptography at Simon's Institute, Berkeley. This played a crucial role in forging relationships with several members of the extended crypto community. For two other summers, I interned at IDC Herzliya, Israel (working with Alon Rosen and Elette Boyle), and MSR

New England (working with Yael Kalai). Both internships enriched me as a researcher and further enhanced my skills. I thank Dan Boneh, Elette Boyle, Melissa Chase, Esha Ghosh, Jens Groth, Alon Rosen, Bjorn Tackmann, for hosting me for short visits and talks during my Ph.D.

I want to thank my friends at Brown and MIT; I whole-heartedly acknowledge their contribution in keeping me (mostly) sane through these years: Ruchi, Yashovardhan, Devendra, Shilpa, Krithika, Divya, Vaibhav, Sagar, Pinkesh, Archita, Saurabh, Prashant, Rajan, Aardra, Saleet, Adam. I feel very fortunate that I have come to share a sibling-like bond with Yashovardhan!

My mother introduced me to music early (right from the womb) and ever since, music has been an integral part of my life. Over the years of PhD, listening to music and my *riyaaz* (musical practice) has been a meditative get-away; it helped me come back to research rejuvinated. I am forever indebted to Gaansaraswati Kishori Amonkar for creating the music that she did; her gift of music has been an inexplicable source of inspiration for me!

I am immensely grateful to my family; I have been blessed with supportive and kind in-laws Urmila and Naina Kamath. I have had several thought-provoking conversations and deep insights from my brother-in-law Sudeep. My co-sister Swarna (who is really like a sister to me) is my confidant and dearest ally.

I can never thank my parents Meenal and Chandrahas Deshpande enough for raising me the way they did; to be curious and to always have a learning mindset. Both of them are doctorates in their respective fields, so that set a lower bar for my education! Finally, I want to thank my husband Pritish Kamath for being everything that I could have wished for in a partner, and more :)

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Privacy-preserving proofs are a fundamental building block in cryptography. Most notably, zero knowledge proofs [GMR89] have seen wide-scale applicability ranging from encryption schemes secure against chosen-ciphertext attacks [NY90], identification protocols [Sch91], electronic voting [DGS03] and strengthening the security of multi-party computation protocols [GMW86].* Essentially, a zero-knowledge proof allows the owner of secret data to convincingly prove that she "knows" the secret without revealing the secret to a verifier. Security requirements are *soundness* which ensures that a prover cannot prove a wrong statement and *zero-knowledge* ensures that the verifier learns nothing beyond the validity of the statement.

Weaker notions of privacy in the form of witness hiding and witness indistinguishable proofs were introduced by Feige and Shamir [FS90]. Intuitively, an interactive proof for an NP language L is said to be *witness hiding* if participating in the protocol does not help the verifier find a witness corresponding to the underlying instance. A protocol is said to be *witness indistinguishable* if given a statement and multiple witnesses for the same, the proofs generated by different witnesses are indistinguishable. These privacy properties can replace zero-knowledge in several applications.

In today's age of blockchain and cloud-computing, we are constantly in search of tools that offer accountability without compromising privacy. For example, suppose a pharmaceutical company wants to publish the results of a new drug trial and wants to convince the public that this drug outperforms the previous ones. Due to privacy reasons, the company cannot disclose details about the users that participated in the trials, but the public cannot be expected to blindly trust the results published by the company either.

Thus, we need tools that ensure correctness of statements while hiding sensitive details. Jumping ahead, we also need the ability to compute and infer on the published data in an efficient way, without compromising privacy. In the example described above, we want to publish findings of the study in a way that validates the accuracy of the results without revealing the specifics of the study. A class of predominantly used tools is proof systems with different privacy guarantees such as zero-knowledge (ZK), witness-indistinguishability (WI) or witness-hiding (WH) described before.

*Zero-knowledge proofs act as a *compiler* to compile any multi-party protocol with semi-honest security into a protocol with malicious security.

For a lot of applications, it is desirable that we enhance properties of proof systems in various dimensions. The first natural direction of enhancement is that of *efficiency*; we are constantly in search of proof systems that improve on the proof size, communication complexity and round complexity over existing ones. For example, since the introduction of non-interactive zero-knowledge proofs by Blum, Feldman and Micali [BDMP91, BFM88], a lot of effort has been spent on reducing their size [Dam92, KP98, Gro10]

Kilian [Kil92] showed that in contrast to zero-knowledge proofs, zero-knowledge arguments can have very low communication complexity. His construction relied on the PCP theorem. Paired with Fiat-Shamir transformation [Mic00], this led to the first construction of succinct non-interactive arguments. In recent times, there has been huge progress in the work on succinct proofs, in the form of SNARGs and SNARKs [BCC$^+$17, PHGR13].

In some cases, proof systems can be designed that are more efficient for special-purpose languages. Schnorr [Sch91] and Guillou-Quisquater [GQ88] gave early examples of practical zero-knowledge arguments for concrete number theoretic problems. Extending Schnorr, there have been many constructions of zero-knowledge arguments based on the discrete logarithm assumption [Gro09]. Recent work by Bootle et al. (Bulletproofs) [BBB$^+$18] gives highly efficient arguments for proving that a committed value lies in a particular range.

Another dimension of enhancement is that of *operability*. Namely, given privacy-preserving proofs, can we operate on them to generate new proofs that are valid? There has been significant work in this context; Most notable is the work on malleable proof systems [CKLM12, CKLM13b], which studies how a proof of an instance can be mauled to generate a valid proof with respect to a transformation of the original instance. The notion of malleability has found many applications, such as verifiable shuffles [CKLM12], delegatable anonymous credentials [BCC$^+$09a, CKLM12] and leakage-resilient proof systems [AGP14]. Re-randomizability [BCC$^+$09a], a special case of malleability, has also been studied in the literature. Following [CKLM12, CKLM13b], [ACJ17] construct *privately* malleable NIZK proof systems, and the works of [AN11, AGM18] study homomorphic proof systems for specific relations.

In this thesis, we propose and develop the following enhancements to proof systems:

Fully Homomorphic NIZK and NIWI Proofs. Given non-interactive zero-knowledge (NIZK) or witness-indistinguishable (NIWI) proofs for different statements, we give a framework to form proofs for new inferred statements. The security guarantee along with soundness and zero-knowledge is that of unlinkability; an inferred proof should be indistinguishable from a fresh proof for a combined statement.

Our first result, under the Decision Linear Assumption (DLIN), is a fully homomorphic NIZK proof system in the common reference string model. Our more surprising second result (under a new decisional assumption on groups with bilinear maps) is a fully homomorphic NIWI proof system that requires no setup.

Privacy Preserving Verifiable Key Directories. The current implementations of online chat services place a lot of trust on the central server that stores all the usernames and their public keys. We initiate the

study of the primitive of *Verifiable Key Directories* (VKD) to formalize the security and privacy of a key verification service.

As a key building block, we introduce append-only zero-knowledge sets, a special-purpose instantiation of zero-knowledge sets, and give highly efficient constructions for the same. (See Section 1.2 for details).

Proofs of Ignorance. We introduce the notion of *proofs of ignorance*. We formalize what it means to not know and how can one prove ignorance. We explore the settings in which one could give a convincing *proof of ignorance*. In the context of NP languages, a proof of ignorance (PoI) for an instance x being in an NP language L should convince the verifier that indeed x is in L and yet that the prover does not know a witness corresponding to x. We focus on the setting where a prover can generate an instance x along with a corresponding proof of ignorance on her own (See Section 1.3 for details).

Using proofs of ignorance, we construct 2-round witness-hiding protocols for NP for the first time. [†] While zero-knowledge protocols are impossible to achieve in two rounds [GO94], no such barrier is known for witness hiding.

We now elaborate on each of these enhancements to privacy-preserving proofs.

1.1 Fully Homomorphic NIZK and NIWI Proofs

Consider a cryptocurrency that records financial transactions in a public ledger. For privacy reasons, suppose that these transactions are committed to, so that the public cannot see how the money moves around in the system. Suppose that a user Alice can provably identify the transactions she participated in.

Alice can use a non-interactive zero-knowledge proof system to prove statements about her volume of transactions or about her assets, without revealing which transactions are hers. In general, individual users have made *confidential* disclosures regarding things like their debts, and have proved the authenticity of these disclosures using NIZKs. Now a regulatory body can infer something about the state of the economy, for example how much debt people have, how many assets, etc.

How can this regulatory body convince the public of its findings? It does not know anything about Alice's (or other users') transactions other than what she was required to disclose, so it does not know the witnesses for her transactions. A regulator can just prove knowledge of the statements and their NIZK proof that lets him infer what he wants to publish. But nesting proofs like that is highly inefficient and the regulator's proof cannot be further used by other regulators. Also a regulator might want to show same or related proofs to different entities without them being able to link any information across proofs. Thus, a different kind of approach is needed here altogether.

We define a framework for evaluating over proofs of different statements to generate proofs for new combined statements. We call these *Fully Homomorphic Proofs*. The security requirement along with soundness

[†] In concurrent work, Bitansky et al. [BKP18b] also construct 2-round witness-hiding protocols for NP assuming Fully Homomorphic Encryption. Also the underlying assumption in our witness-hiding protocol, of strong KDM security with bounded auxiliary input has been shown to be false assuming lockable obfuscation [FKP19].

and zero-knowledge is that of unlinkability; an evaluated proof should be indistinguishable from a fresh proof for the inferred statement. Our first result, under the Decision Linear Assumption (DLIN), is a fully homomorphic NIZK proof system in the common reference string model. Our more surprising second result (under a new decisional assumption on groups with bilinear maps) is a fully homomorphic NIWI proof system that requires no setup.

1.2 Privacy Preserving Verifiable Key Directories

In recent years, the use of online chat services for communication has seen an exponential rise. However, concerns have also arisen about the security of messages sent over these chat services. A number of popular services such as Apple iMessage, WhatsApp and Signal have recently deployed end-to-end encryption [FB14, Sch15] which mitigates some of the previous security threats. But end-to-end encryption most often relies on a Public Key Infrastructure (PKI) and these services still require the service provider to maintain a centralized directory of the public keys of its registered users.

Such a system places a lot of trust in the service provider: a malicious service provider can arbitrarily set and reset users' public keys or implement a man in the middle attack. Without some way of verifying that the service provider is indeed returning the correct keys, end-to-end encryption does not provide any protection against malicious (or coerced) service providers. This problem is made even more challenging by the fact that we must assume that a user can lose her device and along with it all of her secrets.

Privacy may not be very important in traditional PKI, where all the players are usually public entities like businesses. But in the context of private messaging, privacy is important for various reasons. Hiding users' identities may help with preventing spam messaging. Also, users change their keys primarily when they change devices, or detect that their devices/keys have been compromised, either of which may be sensitive information. The ability to track a user's change in public key (which we refer to as tracing attack) can have serious security implications which we want to avoid.

We initiate the study of the primitive of *Verifiable Key Directories* (VKD), analyze its building blocks and give more efficient constructions for the building blocks and in turn, for VKD. We formalize the security and privacy of a key verification service in terms of the primitive Verifiable Key Directories (VKD). The server stores a directory Dir with the names of the users and their corresponding keys and posts periodic updates on a public blockchain. The VKD provides different query interfaces to the users to interact with the server: 1. Adding (username, key) to Dir and updating it, 2. Querying for another user's key 3. Querying for one's own update history.

1.3 Proofs of Ignorance and Applications to Witness-Hiding

Cryptography has always challenged the limits of what we believe is possible. With the elegant work of zero knowledge proofs [GMR89], it is possible to prove that a statement is true without revealing anything except its validity. Furthermore, with zero knowledge proofs-of-knowledge [FFS88, GMR89] it is also possible to prove *knowledge* of some secret without revealing anything about the secret. In this work, we try to answer

the following paradoxical question: *Can one prove lack of knowledge?*

Intuitively, it seems impossible, since one can always *pretend* to be ignorant. We explore the settings in which one could give a convincing *proof of ignorance*. As a thought experiment, suppose that Alice holds a locked box and wants to convince Bob that she *does not know* the contents inside the box. In general, Bob has no reason to believe Alice unless Alice provides some evidence on how she got the box in the first place. Suppose Bob trusts that Charlie gives locked boxes without revealing its contents and suppose that Alice is able to *prove* that she got the box from Charlie, then Bob might be convinced of Alice's assertion.

In the context of NP languages, a proof of ignorance (PoI) for an instance x being in an NP language L should convince the verifier that indeed x is in L and yet that the prover does not know a witness corresponding to x. The main question that we need to tackle is *where does the instance x come from?* Note that if the verifier generates a random instance x for the prover, and the language is hard on average, then a PoI is not needed. Moreover, if the prover and verifier generate x together through a two-party computation protocol, then again a PoI is not needed.

We are interested in the setting where a prover can generate an instance x along with a corresponding proof of ignorance *on her own*. For example, suppose there is a way for Alice to sample an instance x through a "provably random process". Then Alice can convince a verifier that she does not know a witness for x (assuming the language is hard on average). In some sense, a proof of ignorance is a proof that the instance x has been generated correctly with "good" randomness.

Zero-knowledge (ZK) proofs are powerful tools that suffice for almost all applications. However, the best we can do are 3-message zero-knowledge protocols for NP [GO94]. Non-interactive ZK requires an additional assumption of a common random string and in a decentralized setting, such a trust assumption is undesirable. Witness hiding is a natural weakening of the security requirement of zero-knowledge, and can replace zero knowledge (ZK) in several applications such as identification protocols.

Despite the fact that witness-hiding is a weaker requirement than ZK, almost all our candidate constructions of witness hiding protocols for NP are themselves zero-knowledge. It is known that there do not exist 2-message zero-knowledge protocols for NP [GO94], and indeed constructing a 2-message witness hiding protocol for NP remained an important open problem.

We make significant progress in this direction and construct 2-round witness-hiding protocols from the following ingredients: subexponential security of standard assumptions (Decisional Linear assumption over bilinear groups [BBS04]) and encryption schemes with strong security guarantees (key-dependent message security with short auxiliary input [CCRR18]).

1.4 Papers Published

- The third and fourth chapter is based on joint work with Prabhanjan Ananth, Yael Kalai and Anna Lysyanskaya [ADKL19].
- Fifth chapter is based on joint work with Esha Ghosh, Harjasleen Malvai and Melissa Chase [CDGM19].
- Sixth chapter is based on joint work with Yael Kalai [DK18].

Chapter 2

Preliminaries

Notation: We denote the security parameter by λ. We use PPT to denote that an algorithm is probabilistic polynomial time. Suppose A is a probabilistic algorithm, then we denote by $y \leftarrow A(x)$ the event that y is generated by sampling randomness $r \xleftarrow{\$} \{0,1\}^*$ and setting $y = A(x;r)$.

We say that a function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible (sometimes denoted by negl) if for every polynomial p there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$, $\nu(\lambda) < 1/p(\lambda)$. For any language L, we denote by $L = \{L_\lambda\}_{\lambda\in\mathbb{N}}$ where $L_\lambda = L \cap \{0,1\}^\lambda$. We use the notation of $\{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}} \approx_c \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, and $\{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}} \approx_s \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, to denote that the distribution ensembles $\{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally and statistically indistinguishable, respectively.

2.1 Proof Systems

2.1.1 Non-Interactive Proof Systems

Definition 1 (Non-interactive Zero-knowledge Proofs [BDMP91])**.** Let $L \in \mathsf{NP}$ and let R_L be the corresponding NP relation. A triplet of PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is called a *non interactive zero knowledge* (NIZK) proof system for L if it satisfies:

- **Perfect Completeness:** For all security parameters $\lambda \in \mathbb{N}$ and for all $(x,w) \in R_L$,

$$\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ \pi \leftarrow \mathsf{Prove}(\mathsf{CRS}, x, w) : \mathsf{Verify}(\mathsf{CRS}, x, \pi) = 1] = 1$$

- **Adaptive Soundness:** For any all-powerful prover P^*, there exists a negligible function μ such that for all λ,

$$\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (x,\pi) = P^*(\mathsf{CRS})\ :\ \mathsf{Verify}(\mathsf{CRS}, x, \pi) = 1 \ \wedge\ x \notin L] \leq \mu(\lambda)$$

  When this probability is 0, we say it is *perfectly* sound.

- **Adaptive Zero Knowledge:** There exists a PPT simulator $S = (S_1, S_2)$ where $S_1(1^\lambda)$ outputs (CRS_S, τ) and $S_2(\mathsf{CRS}_S, \tau, x)$ outputs π_s such that for all non-uniform PPT adversaries $\mathcal{A}$,

$$\{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda) \ : \ \mathcal{A}^{O_1(\mathsf{CRS},\cdot,\cdot)}(\mathsf{CRS})\} \approx_c$$
$$\{(\mathsf{CRS}_S,\tau) \leftarrow S_1(1^\lambda) \ : \ \mathcal{A}^{O_2(\mathsf{CRS}_S,\tau,\cdot,\cdot)}(\mathsf{CRS}_S)\}$$

where O_1, O_2 on input (x,w) first check that $(x,w) \in R_L$, else output $\perp$. Otherwise O_1 outputs $\mathsf{Prove}(\mathsf{CRS},x,w)$ and O_2 outputs $S_2(\mathsf{CRS}_S,\tau,x)$.

Definition 2 (Non interactive Witness Indistinguishable Proofs [BOV05, DN00])**.** A pair of PPT algorithms $(\mathsf{Prove},\mathsf{Verify})$ is called a *non interactive witness indistinguishable* (NIWI) proof for an NP language L with NP relation R_L if it satisfies:

- **Completeness:** For all security parameters λ and for all $(x,w) \in R_L$,

$$\Pr[\pi \leftarrow \mathsf{Prove}(1^\lambda,x,w) \ : \ \mathsf{Verify}(1^\lambda,x,\pi) = 1] = 1$$

- **Soundness:** For any all-powerful prover P^*, if $P^*(1^\lambda) = (x,\pi)$ and $x \notin L$, then $\mathsf{Verify}(1^\lambda,x,\pi) = 0$.

- **Witness Indistinguishability:** For all non-uniform PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that for every $\lambda \in \mathbb{N}$, probability that $b' = b$ in the following game is at most $1/2 + \nu(\lambda)$:

  1. $(\mathsf{state},x,w_0,w_1) \leftarrow \mathcal{A}(1^\lambda)$.
  2. Choose $b \xleftarrow{\$} \{0,1\}$. If $R_L(x,w_0) \neq 1$ or $R_L(x,w_1) \neq 1$ then output $\perp$. Else, if $b = 0$ then $\pi \leftarrow \mathsf{Prove}(1^\lambda,x,w_0)$, and if $b = 1$ then $\pi \leftarrow \mathsf{Prove}(1^\lambda,x,w_1)$.
  3. $b' \leftarrow \mathcal{A}(\mathsf{state},\pi)$.

We say that a pair of PPT algorithms $(\mathsf{Prove},\mathsf{Verify})$ is called a *non interactive proof system* for an NP language L if it satisfies completeness and adaptive soundness.

For our purposes, we will be using NIWI proofs with respect to parameterized languages of the form $L[\mathsf{pp}]$ where pp denotes some global parameters.

Definition 3 (Non interactive Witness Indistinguishability proofs for Parameterized Languages)**.** Let Setup be a PPT algorithm that takes as input the security parameter and outputs a set of parameters pp. A pair of PPT algorithms $(\mathsf{Prove},\mathsf{Verify})$ is called a NIWI proof for a parameterized NP language $L[\mathsf{pp}]$, with NP relation $R_L[\mathsf{pp}]$ if it satisfies:

- **Completeness:** For all security parameters λ, for all $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$ and for all $(x,w) \in R_L[\mathsf{pp}]$, $\Pr[\pi \leftarrow \mathsf{Prove}(\mathsf{pp},x,w) \ : \ \mathsf{Verify}(\mathsf{pp},x,\pi) = 1] = 1$.

- **Adaptive Soundness:** For any all-powerful prover P^*, there exists a negligible function μ such that for all λ,

$$\Pr[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \ : \ (x,\pi) \leftarrow P^*(\mathsf{pp}) \ : \ \mathsf{Verify}(\mathsf{pp},x,\pi) = 1 \ \wedge \ x \notin L] \leq \mu(\lambda)$$

– **Witness Indistinguishability:** For all non-uniform PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that for every $\lambda \in \mathbb{N}$, probability that $b' = b$ in the following game is at most $1/2 + \nu(\lambda)$:

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. $(\mathsf{state}, x, w_0, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$.
3. Choose $b \xleftarrow{\$} \{0,1\}$. If $R_L[\mathsf{pp}](x,w_0) \neq 1$ or $R_L[\mathsf{pp}](x,w_1) \neq 1$ then output $\perp$. Else if $b = 0$ then $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w_0)$, else if $b = 1$ then $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w_1)$. Send π to $\mathcal{A}$.
4. $b' \leftarrow \mathcal{A}(\mathsf{state}, \pi)$.

Definition 4 (Randomizable NIZK and NIWI Proofs [BCC+09b])**.** A NIZK proof system for an NP language L with NP relation R_L with algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is said to be a randomizable proof system if there exists a PPT algorithm Rand which on input a CRS, an instance x and a proof π, outputs a "randomized" proof π' for x such that for all non-uniform PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that for every $\lambda \in \mathbb{N}$, the probability that $b' = b$ in the following game is at most $1/2 + \nu(\lambda)$:

1. $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. $(\mathsf{state}, x, w, \pi) \leftarrow \mathcal{A}(\mathsf{CRS})$.
3. Choose $b \xleftarrow{\$} \{0,1\}$. If $\mathsf{Verify}(\mathsf{CRS}, x, \pi) \neq 1$ or $R_L(x,w) \neq 1$ then output $\perp$.
4. Else if $b = 0$ then $\pi' \leftarrow \mathsf{Prove}(\mathsf{CRS}, x, w)$, else if $b = 1$ then $\pi' \leftarrow \mathsf{Rand}(\mathsf{CRS}, x, \pi)$.
5. $b' \leftarrow \mathcal{A}(\mathsf{state}, \pi')$.

More generally, a (WI) proof system $(\mathsf{Prove}, \mathsf{Verify})$ is said to be randomizable if there exists a PPT algorithm Rand with the same description and properties as above and where $\mathsf{CRS} = 1^\lambda$.

Definition 5 (Malleable NIWI Proofs for Parameterized Languages [**?**])**.** Let $(\mathsf{Prove}, \mathsf{Verify})$ be a NIWI proof system for a parameterized NP language $L[\mathsf{pp}]$ with NP relation $R_L[\mathsf{pp}]$ where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ (as per Definition 3). Let $T = (T_{\mathsf{inst}}, T_{\mathsf{wit}})$ be a pair PPT transformations such that for every $(x,w) \in R_L$ and for every randomness $\sigma \in \{0,1\}^{\mathsf{poly}(\lambda)}$, $\big(T_{\mathsf{inst}}(\mathsf{pp}, x; \sigma), T_{\mathsf{wit}}(\mathsf{pp}, x, w, \sigma)\big) \in R_L$.

Such a proof system is said to be *malleable* with respect to T, if there exists a randomized PPT algorithm Maul which on input parameters pp, an instance x, randomness σ and proof π, outputs a "mauled" proof π' for $T(\mathsf{pp}, x; \sigma)$ such that the following properties hold:

Malleability For all non-uniform PPT $\mathcal{A}$, for all $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, for all $\lambda \in \mathbb{N}$,

$$\Pr\big[(x,\pi) \leftarrow \mathcal{A}(\mathsf{pp})\ ;\ (\sigma, R) \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}\ ;\ \pi' = \mathsf{Maul}(\mathsf{pp}, x, \sigma, \pi; R)\ :\ \big(\mathsf{Verify}(\mathsf{pp}, x, \pi) = 0\big) \vee \big(\mathsf{Verify}(\mathsf{pp}, T(\mathsf{pp}, x; \sigma), \pi') = 1\big)\big] = 1$$

Perfect Randomizability There exists a poly-time function f_T such that for all $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$ and every $(x,w) \in R_L[\mathsf{pp}]$, for every $R, \sigma \in \{0,1\}^{\mathsf{poly}(\lambda)}$,

$$\mathsf{Maul}(\mathsf{pp}, x, \sigma, \mathsf{Prove}(\mathsf{pp}, x, w; R); R') = \mathsf{Prove}(\mathsf{pp}, T_{\mathsf{inst}}(\mathsf{pp}, x; \sigma), T_{\mathsf{wit}}(\mathsf{pp}, x, w, \sigma); S)$$

where $S = f_T(\mathsf{pp}, w, R, R', \sigma)$. Moreover, if R', σ are uniform, then $f_T(w, R, R', \sigma)$ is uniformly distributed.

2.1.2 Interactive Proofs

Definition 6 (Interactive proofs)**.** An **interactive proof** system for an NP language L, with a corresponding NP-relation R_L, is a protocol $\langle P, V \rangle$, between a PPT prover P and a PPT verifier V, at the end of which V outputs a bit b, and such that the following two properties are satisfied.

- **Completeness.** There exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and every $(x, w) \in R_L$,

$$\Pr[b \leftarrow \langle P(w), V \rangle(1^\lambda, x) \ : \ b = 1] \geq 1 - \mu(\lambda)$$

  where the probability is over the random coin tosses of P and V.

- **Soundness.** For any cheating prover P^* there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and every $x \notin L$,

$$\Pr[b \leftarrow \langle P^*, V \rangle(1^\lambda, x) \ : \ b = 1] \leq \mu(\lambda),$$

  where the probability is over the random coin tosses of V.

  We say that $\langle P, V \rangle$ has **perfect soundness** if the soundness condition holds with $\mu = 0$ (for any P^*).

Definition 7 (Interactive Arguments)**.** An **interactive argument** system for a language L is a protocol $\langle P, V \rangle$ as in Definition 6, where the completeness property is as before, but the soundness property is relaxed as follows.

- **Computational Soundness.** For any poly-size cheating prover P^* there exists a negligible function μ, such that for every $\lambda \in \mathbb{N}$, and every $x \in \{0,1\}^{\mathsf{poly}(\lambda)} \setminus L$,

$$\Pr[b \leftarrow \langle P^*, V \rangle(1^\lambda, x) \ : \ b = 1] \leq \mu(\lambda)$$

  where the probability is over the random coin tosses of V.

A 2-message argument system $\langle P, V \rangle$ for a language L is one that consists of only two messages, the first sent from the verifier to the prover and the second sent from the prover to the verifier. In a 2-message argument system, we often denote the verifier by $V = (V_1, V_2)$, where V_1 generates the message to be sent to the prover, and V_2 checks the validity of the proof given by the prover. If the message of the verifier V_1 does not depend on the instance x (and depends only on the security parameter), then we often denote $(\mathsf{pp}, \mathsf{st}) \leftarrow V_1(1^\lambda)$, where pp is the message sent to the prover and st is the secret state used by V_2 to verify the proof.*

Definition 8 (2-Message Arguments with Adaptive Soundness)**.** *A 2-message argument system $\langle P, V \rangle$ for an NP language L, where the message sent by the verifier is independent of the instance x, is said to have* **adaptive soundness** *if for any poly-size cheating prover P^* there exists a negligible function μ such that*

$$\Pr\left[(x, \mathsf{msg}_P) = P^*(1^\lambda, x, \mathsf{pp}) \ s.t. \ \left(x \notin L\right) \ \wedge \ \left(V_2(1^\lambda, x, \mathsf{pp}, \mathsf{st}, \mathsf{msg}_P) = 1\right)\right] \leq \mu(\lambda)$$

where the probability is over $(\mathsf{pp}, \mathsf{st}) \leftarrow V_1(1^\lambda)$ and over the random coin tosses of V_2.†

*If $\mathsf{st} = \emptyset$ then such an argument system is said to be publicly verifiable.

†Usually, in such argument systems V_2 is deterministic.

Definition 9 (Witness Hiding [FS90])**.** *Let $L \in \mathsf{NP}$ and let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ be a PPT distribution ensemble over instances in L. An interactive proof system $\langle P,V\rangle$ for L is* **witness hiding** *with respect to $\mathcal{D}$ if the following holds:*
For every poly-size V^* there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$,

$$\Pr_{x\leftarrow\mathcal{D}_\lambda}[\langle P(w),V^*\rangle(1^\lambda,x) \in R_L(x)] = \mu(\lambda)$$

Definition 10 (T-secure NIWI)**.** *A NIWI proof system ($\mathsf{Prove},\mathsf{Verify}$) for L as in Definition 2 is said to be T-secure if witness indistinguishability holds for all adversaries of size $\mathsf{poly}(T)$.*

2.2 Other Primitives

2.2.1 Encryption Schemes

Definition 11 (T-secure Encryption)**.** *A bit-wise encryption scheme $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ is said to be T-secure if semantic security holds for all adversaries of size $\mathsf{poly}(T)$. Namely, for every $\mathsf{poly}(T)$-size adversary $\mathcal{D}$, there exists a negligible function ν such that for every $\lambda \in \mathbb{N}$,*

$$\Pr[\mathcal{D}(\mathsf{PK},\mathsf{ct}) = b] \leq 1/2 + \nu(T(\lambda)),$$

where the probability is over $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{Gen}(1^\lambda)$, $b \xleftarrow{\$} \{0,1\}$ and $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{PK}}(b)$.

Remark 1. *For the simplicity of notation, we assume that $|\mathsf{PK}| = \lambda$, for every PK generated by $\mathsf{Gen}(1^\lambda)$, and we assume that $\mathsf{Enc}_{\mathsf{PK}}$ uses λ bits of randomness. We note that one can use a bit-wise encryption scheme to encrypt longer messages. Namely, for any $k = \mathsf{poly}(\lambda)$ one can encrypt any message $m = (m_1,\ldots,m_k) \in \{0,1\}^k$ as follows:*

$$\mathsf{Enc}_{\mathsf{PK}}(m) = \Big(\mathsf{Enc}_{\mathsf{PK}}(m_1),\ldots,\mathsf{Enc}_{\mathsf{PK}}(m_k)\Big).$$

The following lemma follows from a standard hybrid argument.

Lemma 1. *If a bit-wise encryption scheme $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ is T-secure then it is T-secure with respect to messages of length $\mathsf{poly}(T(\lambda))$. More concretely, for every adversary $\mathcal{A}$ of size $\mathsf{poly}(T)$, there exists a negligible function ν such that for every $\lambda \in \mathbb{N}$, the probability that $b' = b$ in the following game is at most $1/2+\nu(T(\lambda))$:*

1. $(\mathsf{state},m_0,m_1) \leftarrow \mathcal{A}(1^\lambda)$
2. *If $|m_0| \neq |m_1|$ then output $\perp$.*
3. *Else, compute $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{Gen}(1^\lambda)$, choose $b \xleftarrow{\$} \{0,1\}$, and let $\vec{\mathsf{ct}} = (\mathsf{ct}_1,\ldots,\mathsf{ct}_{|m_b|})$, where $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{PK}}(m_{b,i})$ for $i \in [|m_b|]$.*
4. $b' \leftarrow \mathcal{A}(\mathsf{state},\mathsf{PK},\mathsf{ct})$

Definition 12 (Rerandomizable Encryption)**.** *A T-secure bit-wise public-key encryption scheme $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$, is said to be* **rerandomizable** *if there exists a PPT algorithm Rand that on input any pair $(\mathsf{PK},\mathsf{ct})$ outputs a ciphertext ct' with the following property: For every $\mathsf{poly}(T)$-size adversary $\mathcal{A}$ there exists a negligible function ν such that for every $\lambda \in \mathbb{N}$, the probability that $b' = b$ in the following game is at most $1/2+\nu(T(\lambda))$:*

1. $\mathcal{A}(1^\lambda) = (\mathsf{state}, \mathsf{PK}, \mathsf{ct}, m, r)$.
2. *If* $\mathsf{ct} \neq \mathsf{Enc}_{pk}(m;r)$ *then output* $\perp$.
3. *Else, choose* $b \xleftarrow{\$} \{0,1\}$. *If* $b = 0$, *choose at random* $r' \leftarrow \{0,1\}^\lambda$ *and output* $\mathsf{ct}' = \mathsf{Enc}_{\mathsf{PK}}(m;r')$. *Else if* $b = 1$, *output* $\mathsf{ct}' = \mathsf{Rand}(\mathsf{PK}, \mathsf{ct})$.
4. $b' \leftarrow \mathcal{A}(\mathsf{state}, \mathsf{ct}')$

Lemma 2. *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a poly-secure rerandomizable bit-wise public-key encryption scheme. For any poly-size distinguisher* $\mathcal{D}$ *and any polynomial* $q = \mathsf{poly}(\lambda)$, *there exists a non-uniform* PPT *distinguisher* $\mathcal{D}^*$, *such that for any* $\lambda \in \mathbb{N}$ *and any public key* PK *such that* $|\mathsf{PK}| = \lambda$, *the following holds: If*

$$\Pr[\mathcal{D}(\mathsf{PK}, \mathsf{Enc}_{\mathsf{PK}}(b)) = b] \geq \frac{1}{2} + \frac{1}{q(\lambda)} \tag{2.1}$$

where the probability is over $b \xleftarrow{\$} \{0,1\}$ *and over the randomness of* $\mathsf{Enc}_{\mathsf{PK}}$, *then for every* $b \in \{0,1\}$ *and every* $r \in \{0,1\}^\lambda$,

$$\Pr[\mathcal{D}^*(\mathsf{PK}, \mathsf{Enc}_{\mathsf{PK}}(b;r)) = b] \geq 1 - 2^{-\lambda} \tag{2.2}$$

where the probability is over the random coin tosses of $\mathcal{D}^*$.

Proof. Fix any poly-size distinguisher $\mathcal{D}$ and any polynomial q. Fix any PK for which Equation (2.1) holds. For every $b \in \{0,1\}$, let

$$p_{\mathsf{PK},b} = \Pr[\mathcal{D}(\mathsf{PK}, \mathsf{Enc}_{\mathsf{PK}}(b)) = b]$$

where the probability is over the randomness of $\mathsf{Enc}_{\mathsf{PK}}$. By Equation (2.1), we have that

$$\frac{p_{\mathsf{PK},0} + p_{\mathsf{PK},1}}{2} \geq \frac{1}{2} + \frac{1}{q(\lambda)} \quad \text{or equivalently,} \quad \frac{p_{\mathsf{PK},0} + p_{\mathsf{PK},1} - 1}{2} \geq \frac{1}{q(\lambda)}$$

We now construct a non-uniform PPT distinguisher $\mathcal{D}^*$, that on input $(\mathsf{PK}, \mathsf{ct})$, where $\mathsf{ct} = \mathsf{Enc}_{\mathsf{PK}}(b^*;r)$, does the following:

1. First estimate $p_{\mathsf{PK},b}$ for each $b \in \{0,1\}$, as follows: Set $M = 100\lambda \cdot q(\lambda)^2$. For each $i \in [M]$, compute fresh ciphertext $\mathsf{ct}_{b,i} \leftarrow \mathsf{Enc}_{\mathsf{PK}}(b)$. For $i \in [M]$ and $b \in \{0,1\}$, define the indicator variables

   $$X_{b,i} = \mathbf{1}_{\{\mathcal{D}(\mathsf{PK}, \mathsf{ct}_{b,i}) = b\}}$$

   Compute

   $$\widehat{p_0} = \frac{\sum_{i=1}^M X_{0,i}}{M} \qquad \text{and} \qquad \widehat{p_1} = \frac{\sum_{i=1}^M X_{1,i}}{M}.$$

2. Choose $N = 9\lambda \cdot q(\lambda)^2$. For each $i \in [N]$, compute $\mathsf{ct}'_i \leftarrow \mathsf{Rand}(\mathsf{PK}, \mathsf{ct})$ and $b_i = \mathcal{D}(\mathsf{PK}, \mathsf{ct}'_i)$.

3. Let $T = (\frac{\widehat{p_1} - \widehat{p_0} + 1}{2})N$. If $\sum_{i=1}^N b_i < T$ then output 0, and else output 1.

Analysis:

Set $\varepsilon = \frac{1}{10q(\lambda)}$. Let E_0 be the event that $|\widehat{p_0} - p_{\mathsf{PK},0}| < \varepsilon$, and similarly let E_1 be the event that $|\widehat{p_1} - p_{\mathsf{PK},1}| < \varepsilon$. By Chernoff bound,[‡]

$$\Pr[\bar{E_0}] = \Pr[\ |\widehat{p_0} - p_{\mathsf{PK},0}| \geq \varepsilon] \leq 2^{-2M\varepsilon^2} \text{ and } \Pr[\bar{E_1}] = \Pr[\ |\widehat{p_1} - p_{\mathsf{PK},1}| \geq \varepsilon] \leq 2^{-2M\varepsilon^2}$$

This, together with the fact that $M = 100\lambda \cdot q(\lambda)^2$, implies that

$$\Pr[\overline{E_0 \wedge E_1}] = \Pr[\bar{E_0} \vee \bar{E_1}] \leq 2 \cdot 2^{-2M\varepsilon^2} \leq \frac{2^{-\lambda}}{2}. \tag{2.3}$$

By the definition of rerandomizable encryption (Definition 12), there exists a negligible function ν_{b^*} such that

$$\Pr[\mathcal{D}(\mathsf{PK}, \mathsf{ct}'_i) = b^*] \geq p_{\mathsf{PK},b^*} - \nu_{b^*}(\lambda) \tag{2.4}$$

where the probability is over $\mathsf{ct}'_i \leftarrow \mathsf{Rand}(\mathsf{PK}, \mathsf{ct})$. Let

$$p'_{b^*} \triangleq p_{\mathsf{PK},b^*} - \nu_{b^*}(\lambda).$$

Fix $b^* = 1$ (Input ciphertext ct is an encryption of 1). Let E be the event that $\mathcal{D}^*(\mathsf{PK}, \mathsf{ct}) = 0$. Namely E is the event that $\sum_{i=1}^N b_i < T$ that is, $\sum_{i=1}^N b_i < (\frac{\widehat{p_1} - \widehat{p_0} + 1}{2})N$.

$$\begin{aligned}
\Pr[E] &= \Pr[E \mid E_0 \wedge E_1] \cdot \Pr[E_0 \wedge E_1] + \Pr[E \mid \overline{E_0 \wedge E_1}] \cdot \Pr[\overline{E_0 \wedge E_1}] \\
&\leq \Pr\Big[\sum_{i=1}^N b_i < \left(\frac{\widehat{p_1} - \widehat{p_0} + 1}{2}\right) N \mid E_0 \wedge E_1\Big] + \frac{2^{-\lambda}}{2} \\
&\leq \Pr\Big[\sum_{i=1}^N b_i < \left(\frac{p_{\mathsf{PK},1} - p_{\mathsf{PK},0} + 1}{2} + \varepsilon\right) N\Big] + \frac{2^{-\lambda}}{2} \\
&\leq \Pr\Big[\sum_{i=1}^N b_i < \left(p_{\mathsf{PK},1} - \nu_b(\lambda) - \left(\frac{p_{\mathsf{PK},0} + p_{\mathsf{PK},1} - 1}{2} - \varepsilon - \nu_b(\lambda)\right)\right) N\Big] + \frac{2^{-\lambda}}{2} \\
&\leq \Pr[\sum_{i=1}^N b_i < (p'_1 - \delta)N] + \frac{2^{-\lambda}}{2} \text{ for } \delta = \frac{1}{3q(\lambda)} \\
&\leq \frac{2^{-\lambda}}{2} + \frac{2^{-\lambda}}{2} \leq 2^{-\lambda}
\end{aligned}$$

where the second inequality follows from Equation 2.3, third inequality follows from definition of events E_0, E_1 and last inequality follows from Chernoff bound.

Now we prove that Equation (2.2) holds for $b^* = 1$.

$$\begin{aligned}
\Pr[\mathcal{D}^*(\mathsf{PK}, \mathsf{ct}) = b^* \mid b^* = 1] &= \Pr\big[\sum_{i=1}^N b_i \geq (\frac{\widehat{p_1} - \widehat{p_0} + 1}{2})N\big] \\
&= 1 - \Pr\big[\sum_{i=1}^N b_i < (\frac{\widehat{p_1} - \widehat{p_0} + 1}{2})N\big] \\
&= 1 - \Pr[E] \\
&\geq 1 - 2^{-\lambda}
\end{aligned}$$

[‡]We use the following Chernoff bound: Let $X_1, \ldots, X_n$ be i.i.d random variables taking values in $\{0,1\}$ such that $\Pr[X_i = 1] = p$. Then $\Pr[|\frac{1}{n}\sum_{i=1}^n X_i - p| \geq \varepsilon] \leq e^{-\frac{n^2\varepsilon^2}{2np(1-p)}} \leq 2^{-2n\varepsilon^2}$.

A similar calculation for $b^* = 0$ shows that

$$\Pr[D^*(\mathsf{PK}, \mathsf{ct}) = b^* \mid b^* = 0] > 1 - 2^{-\lambda}.$$

Hence, we proved that Equation (2.2) holds for every $b^* \in \{0,1\}$ and every $r \in \{0,1\}^\lambda$. □

The following lemma follows from Lemma 2, together with a straightforward union bound.

Lemma 3. *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a a poly-secure rerandomizable bit-wise public-key encryption scheme. For any poly-size distinguisher* $\mathcal{D}$ *and any polynomials* $q, k = \mathsf{poly}(\lambda)$*, there exists a non-uniform* PPT *distinguisher* $\mathcal{D}^*$ *such that for any* $\lambda \in \mathbb{N}$ *and any public key* PK *such that* $|\mathsf{PK}| = \lambda$*, the following holds: If*

$$\Pr[\mathcal{D}(\mathsf{PK}, \mathsf{Enc}_{\mathsf{PK}}(b)) = b] \geq \frac{1}{2} + \frac{1}{q(\lambda)},$$

where the probability is over $b \xleftarrow{\$} \{0,1\}$ *and over the randomness of* $\mathsf{Enc}_{\mathsf{PK}}$*, then for any message* $m = (m_1, \ldots, m_k) \in \{0,1\}^k$ *and any* $r_1, .., r_k \in \{0,1\}^\lambda$*,*

$$\Pr[\mathcal{D}^*(\mathsf{PK}, \vec{\mathsf{ct}}) = m] \geq 1 - \mathsf{negl}(\lambda)$$

where $\vec{\mathsf{ct}} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ *and* $\mathsf{ct}_i = \mathsf{Enc}_{\mathsf{PK}}(m_i; r_i)$ *for* $i \in [k]$*, and where the probability is over the random coin tosses of* $\mathcal{D}^*$*.*

Definition 13 (Strong KDM Security)**.** *A semantically secure public-key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be* **strong KDM secure** *if for every* PPT *distribution* $\mathcal{D}$ *used to (maliciously) sample public keys, if*

$$\left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}(0)\right) \approx_c \left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}(1)\right)$$

where $\mathsf{PK}^* = \mathcal{D}(r^*)$ *for* $r^* \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$*, then for every (not necessarily efficient) function* f *such that* $f(\mathsf{PK}^*) \in \{0,1\}^{\mathsf{poly}(\lambda)}$*,*

$$\left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}(f(r^*))\right) \approx_c \left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}\left(0^{\mathsf{poly}(\lambda)}\right)\right)$$

where $\mathsf{PK}^* = \mathcal{D}(r^*)$ *for* $r^* \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$*.*

Definition 14 (Strong KDM Security with Auxiliary Input)**.** *A semantically secure public-key encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be* **strong KDM secure with auxiliary input** *if for every* PPT *distribution* $\mathcal{D}$ *used to (maliciously) sample public keys and for every auxiliary input* z *of size* $\mathsf{poly}(\lambda)$*, if*

$$\left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}(0), z\right) \approx_c \left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}(1), z\right)$$

where $\mathsf{PK}^* = \mathcal{D}(r^*)$ *for* $r^* \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$*, then for every (not necessarily efficient) function* f *such that* $f(\mathsf{PK}^*) \in \{0,1\}^{\mathsf{poly}(\lambda)}$*,*

$$\left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}(f(r^*)), z\right) \approx_c \left(r^*, \mathsf{Enc}_{\mathsf{PK}^*}\left(0^{\mathsf{poly}(\lambda)}, \right), z\right)$$

where $\mathsf{PK}^* = \mathcal{D}(r^*)$ *for* $r^* \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$*.*

2.2.2 Commitment Schemes

Definition 15 (Statistically Binding Commitments)**.** A statistically binding commitment scheme over a message space $\mathcal{M}$ consists of PPT algorithm Com which on input a message $m \in \mathcal{M}$ and randomness $r \in \{0,1\}^\lambda$ outputs a commitment $\mathsf{Com}(m;r)$.
We require the following properties from the commitment scheme:

Statistically Binding: For every $\lambda \in \mathbb{N}$, every $m_0, m_1 \in \mathcal{M}$ such that $m_0 \neq m_1$, and every $r_0, r_1 \in \{0,1\}^\lambda$,

$$\mathsf{Com}(m_0; r_0) \neq \mathsf{Com}(m_1; r_1)$$

Computationally Hiding: For every non uniform PPT adversary $\mathcal{A}$, there exists a negligible function ν such that for every $\lambda \in \mathbb{N}$, the probability that $b' = b$ in the following game is at most $1/2 + \nu(\lambda)$:

1. $(\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda)$
2. If $|m_0| \neq |m_1|$ or if $m_0, m_1 \notin \mathcal{M}$ then output $\perp$. Choose $b \xleftarrow{\$} \{0,1\}$ and sample $c \leftarrow \mathsf{Com}(m_b)$.
3. $b' \leftarrow \mathcal{A}(\mathsf{state}, c)$

We denote the security parameter by λ. We use PPT to denote that an algorithm is probabilistic polynomial time. We denote by $y \leftarrow A(x)$ if y is the output of a single execution of A on input x. We denote by $y = A(x;r)$ to explicitly mention the randomness used in the execution. We denote $y \in A(x)$ if there exists randomness r such that $y = A(x;r)$.

We use $[n]$ to represent the set $\{1, \ldots, n\}$. Vectors are denoted by $\vec{a}$ where $\vec{a} = (a_1, \ldots, a_n)$ and a_i is the i th element of $\vec{a}$. $|\vec{a}|$ denotes the size of $\vec{a}$. $\vec{a} \circ \vec{b}$ denotes concatenation of the vectors $\vec{a}, \vec{b}$. $\{\mathcal{X}\}_{\lambda \in \mathbb{N}} \approx_c \{\mathcal{Y}\}_{\lambda \in \mathbb{N}}$ will denote that distributions $\{\mathcal{X}\}_{\lambda \in \mathbb{N}}$ and $\{Y\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable.

2.2.3 Bilinear Maps

We will be working with abelian groups $\mathbb{G}, \mathbb{G}_T$ of prime order p equipped with a symmetric bilinear map $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$. We let $\mathcal{G}$ be a *deterministic* polynomial time algorithm that takes as input the security parameter 1^λ and outputs $(p, \mathbb{G}, \mathbb{G}_T, e, g_p)$ such that p is a prime, $\mathbb{G}, \mathbb{G}_T$ are descriptions of groups of order p, g_p is a fixed generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$ is a bilinear map with the following properties:

- (Non-degenerate) For any generator g of $\mathbb{G}$, $g_T = e(g,g)$ has order p in $\mathbb{G}_T$
- (Bilinear) For all $a, b \in \mathbb{G}$, for all $x, y \in \mathbb{Z}_p$, $e(a^x, b^y) = e(a,b)^{xy}$

We require that the group operations and the bilinear operations are computable in polynomial time with respect to security parameter.

Assumption 1 (Decisional Linear Assumption)**.** We say that the Decisional Linear (DLIN) Assumption holds for a bilinear group generator $\mathcal{G}$ if the following distributions are computationally indistinguishable:

$$\{(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda)\ ;\ (x,y) \xleftarrow{\$} \mathbb{Z}_p^*\ :\ (r,s) \xleftarrow{\$} \mathbb{Z}_p\ :\ (p, \mathbb{G}, \mathbb{G}_T, e, g, g^x, g^y, g^{xr}, g^{ys}, g^{r+s})\} \text{ and}$$

$$\{(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^\lambda)\ ;\ (x,y) \xleftarrow{\$} \mathbb{Z}_p^*\ :\ (r,s,d) \xleftarrow{\$} \mathbb{Z}_p\ :\ (p, \mathbb{G}, \mathbb{G}_T, e, g, g^x, g^y, g^{xr}, g^{ys}, g^d)\}$$

Chapter 3

Fully Homomorphic NIZK Proofs

Homomorphism is a desirable feature that enhances the capabilities of many cryptographic systems. Most notably, the concept of fully homomorphic encryption [RAD78, Gen09, BV14] has revolutionized the area of cryptography. Other primitives such as homomorphic signatures [BF11, GVW15] and homomorphic secret sharing [BGI+18] have also found useful cryptographic applications [KW18, BCG+17]. In this work, we study homomorphism in the context of non-interactive proof systems. Our goal is to design homomorphic proof systems with secrecy guarantees; specifically, we focus on the most common secrecy guarantees studied in the literature, namely zero-knowledge [BDMP91] and witness indistinguishability [BOV05, DN00].

Our Work: Fully-Homomorphic NIZK and NIWI Proofs. We introduce the notion of fully-homomorphic non-interactive zero-knowledge (FH-NIZK) and witness-indistinguishable (FH-NIWI) proof systems. In the simplest setting, this proof system allows for combining proofs for the instances A and B into a proof for the instance $A \wedge B$. In the more general setting, this proof system allows for combining proofs for multiple instances $A_1, \ldots, A_n$ using a function f into a single proof for $f(A_1, \ldots, A_n)$.

A naïve attempt to combine proofs for the instances $(A_1, \ldots, A_n)$ using a function f is to simply output the concatenation of the individual proofs on each of the instances $A_1, \ldots, A_n$ together with the function f. However, this combined proof does not resemble an honestly generated proof for the instance $f(A_1, \ldots, A_n)$. Our goal is to combine proofs in a way that is indistinguishable from an honestly generated proof for the instance $f(A_1, \ldots, A_n)$. We call this property *unlinkability*.

There are several reasons why unlinkability is an interesting feature: Firstly, it is often desirable to hide the fact that a proof was obtained by combining multiple proofs. Unlinkability also preserves the privacy of the underlying proof; namely, it ensures that homomorphic evaluation of multiple NIZK (resp., NIWI) proofs still results in a NIZK (resp., NIWI) proof. Moreover, it guarantees that the homomorphic evaluation can be multi-hop, meaning that the proofs can be evaluated upon multiple times. We describe the homomorphic evaluation procedure and unlinkability property below.

We define the notion of a fully-homomorphic proof system for the NP-complete language $L_{\mathcal{U}}$ which consists of instances (C, b), where C is a boolean circuit with single-bit output and b is a bit, such that there exists a witness $\vec{w}$ (a vector of bits) for which $C(\vec{w}) = b$. A non-interactive proof system for proving membership

in this language consists of the algorithms Prove and Verify. A fully homomorphic proof system additionally has the algorithm Eval defined as follows:

Homomorphic Evaluation (Eval): On input k instances $\{z_i = (C_i, b_i)\}_{i\in[k]}$ accompanied with proofs $\{\Pi_i\}_{i\in[k]}$ for the statements $\{z_i \in L_{\mathcal{U}}\}_{i\in[k]}$, and a circuit $D : \{0,1\}^k \to \{0,1\}$, Eval outputs a proof Π^* for the statement $z^* = (C^*, D(b_1,\ldots,b_k)) \in L_{\mathcal{U}}$, where C^* is defined to be the circuit that on input $(\vec{w}_1,\ldots,\vec{w}_k)$ outputs $D(C_1(\vec{w}_1),\ldots,C_k(\vec{w}_k))$.

We define *unlinkability* as follows: A proof Π^* output by Eval on input $\{z_i \in L_{\mathcal{U}}\}_{i\in[k]}$ accompanied with proofs $\{\Pi_i\}_{i\in[k]}$, where Π_i is output by Prove on input z_i and a valid witness $\vec{w}_i$, should be indistinguishable from the output of Prove on input the instance $(C^*, D(b_1,\ldots,b_k))$ and witness $(\vec{w}_1,\ldots,\vec{w}_k)$. As mentioned above, unlinkability guarantees that the evaluation property preserves zero-knowledge (ZK) or witness-indistinguishability (WI) of an evaluated proof, depending on whether the fresh proof is ZK or WI respectively. We refer the reader to Figure 3.1 for an illustrative description of unlinkability, and refer the reader to Section 3.2 for our definition of fully homomorphic proofs.

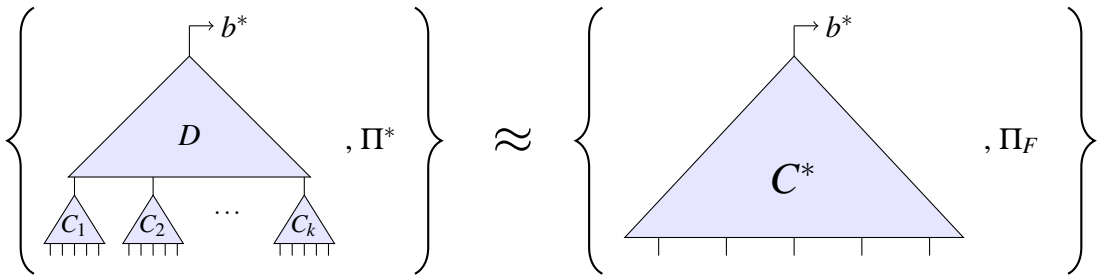

Figure 3.1: *Unlinkability property of Fully Homomorphic Proofs: Let* Π^* *be the output of* Eval *on input* $\{(C_i, b_i) \in L_{\mathcal{U}}\}_{i\in[k]}$ *accompanied with proofs* $\{\Pi_i\}_{i\in[k]}$*, where* Π_i *is output by* Prove *on input* (C_i, b_i) *and a valid witness* $\vec{w}_i$*. Let* C^* *be the circuit that on input* $(\vec{w}_1,\ldots,\vec{w}_k)$*, outputs* $D(C_1(\vec{w}_1),\ldots,C_k(\vec{w}_k))$ *and let* Π_F *be an honestly generated proof for the instance* $(C^*, b^*) \in L_{\mathcal{U}}$*. We require that* Π^* *is computationally indistinguishable from* Π_F*.*

Our Results. We construct both a NIZK and a NIWI fully homomorphic proof system.

Theorem 1 (Informal)**.** *Assuming Decisional Linear Assumption (DLIN), there exists a fully-homomorphic non-interactive zero-knowledge proof system in the common random string model.*

For constructing FH NIWI proofs, we rely on a new decisional assumption on groups with bilinear maps called *DLIN with leakage*, defined in Figure 3.2 (below).
For a more detailed description of the assumption, we refer the reader to Section 3.3.3. We prove that our assumption holds in the bilinear generic group model in Appendix **??**.

Theorem 2 (Informal)**.** *Assuming DLIN with Leakage, there exists a fully-homomorphic non-interactive witness-indistinguishable proof system in the plain model (i.e. without setup).*

Let f, h, g be three random generators in a group $\mathbb{G}$. The assumption states that $\mathcal{D}_0(1^\lambda) \approx_c \mathcal{D}_1(1^\lambda)$, where:

- $\mathcal{D}_0(1^\lambda)$: Choose $R, S, t \leftarrow \mathbb{Z}_p^*$ and output (f, h, g) along with the following matrix:

$$\begin{bmatrix} f^R & h^S & g^{R+S} \\ f^{R^2} & h^{RS-t} & g^{R(R+S+1)-t} \\ f^{RS+t} & h^{S^2} & g^{S(R+S+1)+t} \end{bmatrix}$$

- $\mathcal{D}_1(1^\lambda)$: Choose $R, S, t \leftarrow \mathbb{Z}_p^*$ and output (f, h, g) along with the following matrix:

$$\begin{bmatrix} f^R & h^S & g^{R+S-1} \\ f^{R^2} & h^{RS-t} & g^{R(R+S-1)-t} \\ f^{RS+t} & h^{S^2} & g^{S(R+S-1)+t} \end{bmatrix}$$

Figure 3.2: *Description of the DLIN with leakage, with respect to a group $\mathbb{G}$ of prime order p with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We refer to this as DLIN with leakage assumption since the first row in both the distributions are indistinguishable assuming DLIN, and the second and third rows can be viewed as leakage.*

Related Works. Most relevant to our work is the work on malleable proof systems [CKLM12, CKLM13b], who studied unary transformations, i.e., when Eval receives a *single* instance-proof pair and outputs a mauled instance along with the corresponding proof. The work of [CKLM12] studied malleable proof systems for specific relations, and [CKLM13b] studied malleability for general relations albeit under knowledge assumptions. Moreover, these works consider NIZK proof systems and thus require trusted setup. We note that [CKLM12] satisfies a stronger proof of knowledge property (called controlled-malleable simulation-sound extractability) that we don't achieve in this work.

The notion of malleability, although seemingly limited due to its unary nature, has found many applications, such as verifiable shuffles [CKLM12], delegatable anonymous credentials [BCC$^+$09a, CKLM13a] and leakage-resilient proof systems [AGP14]. Re-randomizability [BCC$^+$09a], a special case of malleability, has also been studied in the literature. Following [CKLM12, CKLM13b], [ACJ17] construct *privately* malleable NIZK proof systems, and the works of [AN11, AGM18] study homomorphic proof systems for specific relations.

It is important to stress that all the prior works, even in the case of unary transformations studied in the context of malleable proofs [CKLM12, CKLM13b], assume trusted setup. Thus, in the context of WI proof systems, our results are especially surprising since it allows for combining proofs that were created completely independently, with no shared setup.

We now describe some applications of fully-homomorphic proofs.

Private Incremental Proofs. Incremental proofs, introduced by Valiant [Val08], allow for merging many computationally sound proofs [Mic00] into one proof which is as short and easily verifiable as the original

proofs. Incremental proofs have been applied in several contexts such as proof-carrying data [BCCT13] and cryptographic image authentication mechanisms [NT16]. It is useful in two types of settings: one where the computation dynamically evolves over a period of time, hence a proof of correctness of the entire computation cannot be computed all at once, and the other where different entities wish to compute a proof for the correctness of computation in a distributed setting.

The focus of prior works on incremental proofs was on succinctness whereas the focus of our work is on privacy. While our work does not achieve succinctness, as we will see later achieving privacy alone turns out to be quite challenging (especially, in the context of fully-homomorphic NIWIs). We hope that our tools can be combined with succinct incremental proofs to yield incremental proofs that enjoy both succinctness and privacy guarantees.

Commit-and-Compute Paradigm. Another application of fully-homomorphic proofs is the commit-and-compute paradigm. At a high level, the commit-and-compute paradigm allows a prover to commit to its sensitive data, and later on, prove statements about the committed data. Proofs from different provers can then be combined to infer arbitrary statements about the committed data. We give two examples that illustrate the applicability of this paradigm: (i) Regulation of cryptocurrency activities and, (ii) Verifiable data analysis.

Regulation of Cryptocurrency Activities. New regulation laws regarding cryptocurrency activities are being formulated and some of them require that financial transactions involving cryptocurrencies be reported [oSBS15]. The regulators can then infer different conclusions about the state of the digital economy; for instance, they can conclude the debt of different entities, and publish the findings for the public. The involved entities may have motives to lie about their finances and this would in turn lead the regulators to arrive at false conclusions. We can address this problem using fully homomorphic NIZK or NIWI proofs: Each entity can now commit to their financial transactions (to maintain privacy) and submit a proof that the financial transactions reported to the regulators are valid. The regulators can collect the data and the proofs from all the entities, publish its conclusions along with a proof that is obtained by homomorphically computing on the individual proofs.

Verifiable Data Analysis. Consulting firms often collect data from different research groups, perform analysis on the joint dataset and then share the analyzed results with different organizations. For instance, there are firms that collect medical data from different research groups and share the analysis on the medical data to pharmaceutical companies. This raises concerns about trusting the research groups and the consulting firms to not lie about their conclusions. We can tackle this concern by using fully homomorphic NIZK or NIWI proofs. The research groups can publish their (committed) data along with a proof that it was collected from valid sources, without revealing the identity of the sources. The consulting firms can then perform analysis on the joint data sets and homomorphically compute a proof that the analysis was performed correctly. Moreover, the homomorphically computed proof will also hide the identities of the research groups involved in sharing the data to the firms.

Commit-and-compute paradigm is formalized by defining the NP language L_{COM}, a modification of $L_{\mathcal{U}}$

so that the instance includes a vector of commitments along with (C,b). The language L_{COM} is as follows:

$$L_{\mathsf{COM}} = \left\{(C,(\mathsf{Com}_1,\ldots,\mathsf{Com}_n),b) \mid \exists\{w_i,r_i\} \text{ such that } C(w_1,\ldots,w_n)=b \wedge \{\mathsf{Com}_i = \mathsf{Commit}(w_i,r_i)\}\right\}$$

The evaluation is defined similarly to that of homomorphic Eval for $L_{\mathcal{U}}$. We show how to instantiate the commit-and-compute paradigm using fully-homomorphic proofs in Section 4.4.

3.1 Technical Overview

Let us start with some intuition. Suppose we want to generate a proof for the satisfiability of $C_1 \wedge C_2$ for some circuits C_1, C_2. Given a proof Π_1 for the satisfiability of C_1 and a proof Π_2 for the satisfiability of C_2, clearly $\Pi = (\Pi_1,\Pi_2)$ is a proof for the satisfiability of $C_1 \wedge C_2$. However, such a proof does not satisfy unlinkability. Moreover, the structure of the proof $\Pi = (\Pi_1,\Pi_2)$ may be different from that of a fresh proof computed for the satisfiability of $C_1 \wedge C_2$.

To achieve homomorphism and unlinkability, a natural candidate is a proof system that works gate-by-gate as follows: Commit to all the wire values of the circuit and prove that each gate is consistent with the committed values. Such a proof structure is a good candidate because structurally, a proof of the composed instance $C_1 \wedge C_2$ will be similar to a fresh proof.

Indeed the beautiful work of Groth, Ostrovsky and Sahai [GOS06a] (henceforth referred to as GOS) has this proof structure and it is the starting point for our FH NIZK construction as well as our FH NIWI construction. GOS constructed NIZK and NIWI proofs under the decisional linear (DLIN) assumption. First in Section 3.1.1, we describe our FH NIZK construction which builds on the GOS NIZK. Then in Section 4.1, we describe our FH NIWI construction which contains the bulk of the technical difficulty in this work.

3.1.1 Overview: Fully Homomorphic NIZK

Recall that an $L_{\mathcal{U}}$ instance is of the form (C,out) where $C : \{0,1\}^t \to \{0,1\}$ and $\mathsf{out} \in \{0,1\}$. Let $\vec{w} = (w_1,\ldots,w_t)$ be a witness such that $C(\vec{w}) = \mathsf{out}$. Let us first recall the GOS NIZK proof for $L_{\mathcal{U}}$.

GOS NIZK. The GOS NIZK proof system is associated with a commitment scheme with public parameters (as we elaborate on later). The CRS consists of the parameters pp for the commitment scheme. The prover on input (C,out) along with witness $\vec{w}$ does the following:

1. Let $w_1,\ldots,w_n$ be the values induced by witness $\vec{w} = (w_1,\ldots,w_t)$ on all the wires of the circuit C. Commit to all the wire values with respect to pp, except the output wire. For every $i \in [n-1]$, denote by $\vec{c}_i$ the commitment to wire value w_i. Denote by $\vec{c}_n = w_n$.

2. For each $i \in [n]$, prove that the commitment $\vec{c}_i$ is a commitment to a boolean value. We refer to such proofs by *Bit Proofs*.

3. For each gate in C, prove that the commitments to the input and the output wires of the gate are consistent with the gate functionality. We refer to such proofs by *Gate Proofs*.

In their construction, GOS use a commitment scheme which has two indistinguishable modes of public parameters: perfectly binding and perfectly hiding. Loosely speaking, the perfectly binding mode is used to argue perfect soundness, and the perfectly hiding mode is used to argue zero-knowledge. In addition, they require the commitment scheme to be additively homomorphic and the additive homomorphism is used in the Gate Proofs.

GOS constructed NIWI proof systems for Bit Proofs and Gate Proofs, and proved that this is sufficient for their NIZK construction. Both Bit and Gate Proofs are computed using the openings of the commitments as the witness. Our FH NIZK construction follows a similar template (our NIZK construction is identical to the GOS NIZK) but in order to achieve unlinkability, we need additional properties from the commitment scheme as well as from the Bit Proofs and Gate Proofs, as we explain below.

Homomorphic Evaluation. Homomorphic evaluation works as follows: On input k instances $\{z_i = (C_i, b_i)\}_{i\in[k]}$ along with proofs $\{\Pi_i\}_{i\in[k]}$ where each Π_i is a proof that $z_i \in L_{\mathcal{U}}$, and a circuit D, we want to output a proof that $(C^*, b^*) \in L_{\mathcal{U}}$ where C^* is the composed circuit and $b^* = D(b_1, \ldots, b_k)$. First, compute a fresh proof for the circuit D with witness $(b_1, \ldots, b_k)$. Note that the fresh proof for (D, b^*) together with the proofs $\{\Pi_i\}_{i\in[k]}$, forms a verifying proof with respect to (C^*, b^*). This follows from the fact that in each proof Π_i, the output wire b_i is given in the clear. However this combined proof is distinguishable from a fresh proof (given the individual proofs $\{\Pi_i\}_{i\in[k]}$). Thus, to achieve unlinkability, we randomize this entire proof.

Randomizing the NIZK Proof. A proof system is said to be randomizable [BCC$^+$09b] if given a proof Π for an instance x, it is possible to randomize the proof Π to obtain a proof Π' for x, such that Π' is indistinguishable from a fresh proof for x. Randomizability of a proof system is sufficient for achieving unlinkability in our construction, as explained above.

At a high level, we randomize the proof Π as follows: Randomize all the commitments in the proof, and then "update" the existing proofs to be with respect to the randomized commitments. Thus, given the original Bit Proofs and Gate Proofs, we need to be able to "maul" them to be with respect to the new randomized commitments in such a way that the updated proofs are distributed as fresh Bit Proofs and Gate Proofs. We refer to such proofs as *malleable proofs.*

Ingredients for our FH NIZK. In summary, for constructing FH NIZK, we use a commitment scheme from GOS, which is also randomizable (we describe the corresponding scheme $(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$ in Definition 18, Section 3.3.1). We also need malleable proof systems for Bit proofs and for Gate proofs (we describe the corresponding proof systems $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.Maul})$ and $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify}, \mathsf{NAND.Maul})$ in Section 3.4.2).

As shown in GOS, both Bit Proofs and Gate Proofs can be reduced to *proofs of linearity* with respect to the NP language L_{Lin}. The language L_{Lin} is parameterized by three random group elements (f, h, g) in some underlying group $\mathbb{G}$ of prime order (which has a bilinear map), and whose instances consists of pairs (A, B), where $A = (f^{a_1}, h^{a_2}, g^{a_3})$ and $B = (f^{b_1}, h^{b_2}, g^{b_3})$, such that $a_1 + a_2 = a_3$ or $b_1 + b_2 = b_3$[*].

[*] If $a_1 + a_2 = a_3$ then A is said to be a linear tuple.

GOS constructed a NIWI proof for L_{Lin}. Recall that for our purposes, we need malleable proof systems for Bit Proofs and Gate Proofs, and as a result we need the underlying NIWI proof for L_{Lin} to be malleable with respect to randomization. Namely given a pair $(\vec{A},\vec{B}) \in L_{\mathsf{Lin}}$ with a NIWI proof Π, it should be possible to maul the proof Π for $(\vec{A},\vec{B})$ into a proof Π' for a randomization $(\vec{A}',\vec{B}')$ of $(\vec{A},\vec{B})$. We show that the GOS proof for L_{Lin} has the desired malleability property, and we refer the reader to Section 3.3.2 for the description of the malleable proof system.

3.2 Fully Homomorphic Proofs: Definition

In this section we define fully homomorphic NIZK and NIWI proofs for the NP-complete language $L_{\mathcal{U}}$ consisting of instances of the form (C,b) where $C : \{0,1\}^k \to \{0,1\}$ is a boolean circuit and $b \in \{0,1\}$. Formally, $L_{\mathcal{U}}$ is defined as:

$$L_{\mathcal{U}} = \{(C,b) \mid \exists\ \vec{w} \text{ such that } C(\vec{w}) = b\}$$

Let $R_{\mathcal{U}}$ be the corresponding NP-relation. We first define the notion of composing multiple instances of $L_{\mathcal{U}}$ to get a new instance in $L_{\mathcal{U}}$:

Composing $L_{\mathcal{U}}$ Instances: On input k instances $\{(C_i,b_i)\}_{i=1}^k$ where $C_i : \{0,1\}^{t_i} \to \{0,1\}$ and $C' : \{0,1\}^k \to \{0,1\}$,

$$\mathsf{Compose}(\{(C_i,b_i)\}_{i=1}^k, C') = (C,b)$$

where $C : \{0,1\}^T \to \{0,1\}$ and $T = \sum_{i=1}^k t_i$ and for all $(\vec{w}_1,\ldots,\vec{w}_k) \in \{0,1\}^{t_1} \times \cdots \times \{0,1\}^{t_k}$,

$$C(\vec{w}_1,\ldots,\vec{w}_k) = C'\big(C_1(\vec{w}_1),\ldots,C_k(\vec{w}_k)\big)\ \wedge\ b = C'(b_1,\ldots,b_k).$$

3.2.1 Definition: Fully Homomorphic NIZK and NIWI Proofs

We now define fully homomorphic NIZK and NIWI proofs for the language $L_{\mathcal{U}}$ defined above.

Definition 16 (Fully Homomorphic NIZK Proofs)**.** A randomizable NIZK proof system $(\mathsf{Setup},\mathsf{Prove},\mathsf{Verify},\mathsf{Rand})$ is a fully homomorphic proof system if there exists a PPT algorithm Eval with the following input-output behavior:

$((C,b),\Pi) \leftarrow \mathsf{Eval}(\mathsf{CRS},\{(C_i,b_i),\Pi_i\}_{i=1}^k,C')$: The Eval algorithm takes as input the CRS, k instances $\{(C_i,b_i)\}_{i=1}^k$ along with their proofs $\{\Pi_i\}_{i=1}^k$, and a circuit $C' : \{0,1\}^k \to \{0,1\}$. It outputs the composed instance $(C,b) = \mathsf{Compose}(\{(C_i,b_i)\}_{i=1}^k,C')$ and a corresponding proof Π such that the following properties hold:

Completeness of Eval: We require that evaluating on valid proofs (proofs that verify), should result in a proof that verifies. More concretely, we require that for all non-uniform PPT $\mathcal{A}$ and for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{c}\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda)\ ;\ (\{(C_i,b_i,\Pi_i)\}_{i=1}^k,C')\leftarrow\mathcal{A}(\mathsf{CRS})\ ;\\ ((C,b),\Pi)\leftarrow\mathsf{Eval}(\mathsf{CRS},\{(C_i,b_i),\Pi_i\}_{i=1}^k,C')\ :\\ \big(\mathsf{Valid}(C')=0\big)\ \vee\ \big(\exists\ i\in[k]\ \text{s.t.}\mathsf{Verify}(\mathsf{CRS},(C_i,b_i),\Pi_i)=0\big)\ \vee\\ \Big((\mathsf{Verify}(\mathsf{CRS},(C,b),\Pi)=1)\ \wedge\ (C,b)=\mathsf{Compose}(\{(C_i,b_i)\}_{i=1}^k,C')\Big)\end{array}\right] = 1$$

where $\mathsf{Valid}(C') = 1$ if and only if $C' : \{0,1\}^k \to \{0,1\}$.

Unlinkability: We require that a proof for $(C,b) \in L_{\mathcal{U}}$ obtained by Eval should be indistinguishable from a fresh proof for the same instance. Namely, for any non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function ν such that for every λ the probability that $\mathsf{bit} = \mathsf{bit}'$ in the following game is at most $1/2 + \nu(\lambda)$:

$\underline{\text{GAME}_{\mathsf{Eval}}}$:

1. $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. $(\mathsf{state}, \{((C_i,b_i),\vec{w}_i,\Pi_i)\}_{i=1}^k, C') \leftarrow \mathcal{A}(\mathsf{CRS})$
3. Choose $\mathsf{bit} \xleftarrow{\$} \{0,1\}$. If for any $i \in [k]$, $\mathsf{Verify}(\mathsf{CRS},(C_i,b_i),\Pi_i) \neq 1$ or $((C_i,b_i),\vec{w}_i) \notin R_{\mathcal{U}}$, output $\perp$.
4. Else if $\mathsf{bit} = 0$ then $((C,b),\Pi) \leftarrow \mathsf{Eval}(\mathsf{CRS}, \{(C_i,b_i),\Pi_i\}_{i=1}^k, C')$. Else if $\mathsf{bit} = 1$ then compute $(C,b) = \mathsf{Compose}(\{(C_i,b_i)\}_{i=1}^k, C')$ and
   $\Pi \leftarrow \mathsf{Prove}(\mathsf{CRS},(C,b),\vec{w})$ where $\vec{w} = \vec{w}_1 \circ \cdots \circ \vec{w}_k$. Send (C,b,Π) to $\mathcal{A}$.
5. $\mathsf{bit}' \leftarrow \mathcal{A}(\mathsf{state},(C,b,\Pi))$.

Definition 17 (Fully Homomorphic NIWI Proofs)**.** A randomizable NIWI proof system $(\mathsf{Prove},\mathsf{Verify},\mathsf{Rand})$ is a fully homomorphic NIWI proof system if there exists a PPT algorithm Eval with the same description and properties as in Definition 16 and where $\mathsf{CRS} = 1^\lambda$.

3.3 Building Blocks for Fully Homomorphic Proofs

In this section we describe the building blocks for our fully homomorphic (FH) NIZK and NIWI constructions. In Section 3.3.1, we define a commitment scheme with additional properties, which we will use in our FH NIZK and NIWI constructions, and we then instantiate it from DLIN.

In Section 3.3.2, we describe a NIWI proof system for the NP language L_{Lin} (defined in Definition 21) based on DLIN. This proof system is the main ingredient in constructing FH NIZK and FH NIWI proofs.

For our FH NIWI construction, we need the NIWI proof for L_{Lin} to have additional properties of malleability and strong WI with respect to specific distributions. We prove that the proof system is malleable and we prove that strong WI holds under a new assumption on bilinear groups: *DLIN with Leakage*. We describe the corresponding bilinear assumption in Section 3.3.3.

3.3.1 Randomizable Commitment Scheme

Definition 18 (Randomizable Commitment Scheme)**.** A Randomizable commitment scheme for message space $\mathcal{M}$ consists of PPT algorithms $\mathsf{COM} = (\mathsf{CS.Setup},\mathsf{CS.Commit},\mathsf{CS.Rand})$ with the following descriptions and properties:

$\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda)$: On input the security parameter, the setup algorithm outputs public parameters params.

$\mathsf{Com} = \mathsf{CS.Commit}(\mathsf{params},b;o)$: Using the public parameters params, the commit algorithm produces commitment Com to message $b \in \{0,1\}$ using randomness $o \leftarrow \{0,1\}^{p(\lambda)}$ for some polynomial p. We will refer to o as "opening" for the commitment Com.

$\mathsf{Com}' = \mathsf{CS.Rand}(\mathsf{params}, \mathsf{Com}; o')$: On input parameters params, commitment Com, randomness o', CS.Rand outputs a randomized commitment Com' to the same value.

We require the following properties from the commitment scheme:

Perfectly Binding: For all $(m_0, m_1) \in \mathcal{M}$ such that $m_0 \neq m_1$ and for all $o_0, o_1 \in \{0,1\}^{\mathsf{poly}(\lambda)}$

$$\Pr[\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda) \ : \ \mathsf{CS.Commit}(\mathsf{params}, m_0; o_0) = \mathsf{CS.Commit}(\mathsf{params}, m_1; o_1)] = 0$$

Computationally Hiding: Let $\mathsf{params} \leftarrow \mathsf{CS.Setup}(1^\lambda)$. For all $(m_0, m_1) \in \mathcal{M}$ and $o_0, o_1 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$,

$$\big(\mathsf{CS.Commit}(\mathsf{params}, m_0; o_0)\big) \approx_c \big(\mathsf{CS.Commit}(\mathsf{params}, m_1; o_1)\big)$$

Perfect Randomizability: Let $\mathsf{params} \leftarrow \mathsf{CS.Setup}(1^\lambda)$. There exists an efficient function f_{Com} such that for any randomness o, the following holds:

- For every $o' \in \{0,1\}^{\mathsf{poly}(\lambda)}$, $\mathsf{CS.Rand}(\mathsf{params}, \mathsf{CS.Commit}(\mathsf{params}, m; o); o') = \mathsf{CS.Commit}(\mathsf{params}, m; s)$ where $s = f_{\mathsf{Com}}(o, o')$.
- If o' is chosen uniformly at random, then $f_{\mathsf{Com}}(o, o')$ is uniformly distributed.

We now describe additional properties that we require from our commitment scheme for our FH NIZK construction:

- **Additive Homomorphism:** We require that if $\vec{c}_1$ and $\vec{c}_2$ are commitments to m_1 and m_2 respectively, then there exists an efficient function f_{add} such that $\vec{c} = f_{\mathsf{add}}(\vec{c}_1, \vec{c}_2)$ is a commitment to $(m_1 + m_2)$.
- **Perfect Equivocation:** There exists a PPT algorithm $\mathsf{CS.Setup}'$ and a polynomial time algorithm CS.Equivocate such that
  - $\mathsf{CS.Setup}'$ on input the security parameter, outputs params', such that
    $$\{\mathsf{params} \leftarrow \mathsf{CS.Setup}(1^\lambda) \ : \ \mathsf{pp}\} \approx_c \{\mathsf{pp}' \leftarrow \mathsf{CS.Setup}'(1^\lambda) \ : \ \mathsf{pp}'\}.$$
  - Fix any $r_{\mathsf{pp}} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, any $m, m' \in \mathcal{M}$ and any randomness $o \in \{0,1\}^{\mathsf{poly}(\lambda)}$. Let $\mathsf{pp}' = \mathsf{CS.Setup}'(1^\lambda; r_{\mathsf{pp}})$ and $\vec{c} = \mathsf{CS.Commit}(\mathsf{params}', m; o)$. Algorithm CS.Equivocate on input $(\mathsf{pp}', r_{\mathsf{pp}}, \vec{c}, o, m')$ outputs o' such that $\vec{c} = \mathsf{CS.Commit}(\mathsf{pp}', m'; o')$. Also, for truly random o, $(\vec{c}, o')$ is distributed identically to $(\vec{c}'', o'')$ where o'' is chosen at random and $\vec{c}'' = \mathsf{CS.Commit}(\mathsf{params}', m'; o'')$.

  Note that the parameters output by $\mathsf{CS.Setup}(1^\lambda)$ are *binding* and the parameters output by $\mathsf{CS.Setup}'(1^\lambda)$ are *hiding*.

We will denote a randomizable commitment which is also additively homomorphic (aH) and equivocable (E) as described above, by a RaHE-commitment scheme.

Remark 2. *We will denote by* ***1*** *and* ***0*** *the canonical commitments to* $1, 0$ *respectively, namely the commitments computed with randomness* $o = 0$*. Given such a commitment it is possible to verify, that the commitment is indeed to* 0 *or* 1*.*

Additional Functionalities for FH NIWI. In our FH NIWI construction, we use a RaHE-commitment scheme which has additional functionalities $(\mathsf{OutParam}, \mathsf{ValidParam}, \mathsf{RParam}, \mathsf{ChangeCom})$ with properties described below:

- **Outputting hiding parameters**: The deterministic algorithm $\mathsf{OutParam}$ takes as input parameters pp^0 and outputs pp^1 such that for all r_{pp}, if $\mathsf{pp}^0 = \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}})$, then $\mathsf{pp}^1 = \mathsf{CS.Setup}'(1^\lambda; r_{\mathsf{pp}})$.
- **Verifying if two parameters are valid**: The algorithm $\mathsf{ValidParam}$ is an efficient predicate that outputs 1 if $\mathsf{pp}^0 \in \mathsf{CS.Setup}(1^\lambda)$ and $\mathsf{pp}^1 = \mathsf{OutParam}(\mathsf{pp}^0)$. It outputs 0 if both parameters are hiding, namely if $\mathsf{pp}^0, \mathsf{pp}^1 \in \mathsf{CS.Setup}'(1^\lambda)$.
- **Randomization of parameters**: The RParam algorithm takes as input parameters pp, randomness r'_{pp}, and outputs new parameters pp' such that for all r_{pp} and for $\mathsf{pp} = \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}})$, the following properties hold:
  - There exists an efficient function f_{pp}: $f_{\mathsf{pp}}(r_{\mathsf{pp}}, r'_{\mathsf{pp}}) = \sigma$ and $\mathsf{pp}' = \mathsf{RParam}(\mathsf{pp}; r'_{\mathsf{pp}}) = \mathsf{CS.Setup}(1^\lambda; \sigma)$.
  - $\mathsf{RParam}(\mathsf{OutParam}(\mathsf{pp}); r'_{\mathsf{pp}}) = \mathsf{OutParam}(\mathsf{RParam}(\mathsf{pp}; r'_{\mathsf{pp}}))$.
- **Transformation of commitments with respect to new parameters**: The $\mathsf{ChangeCom}$ algorithm takes in parameters pp, randomness r'_{pp}, commitment $\vec{c}$, and outputs commitment $\vec{c}'$ to the same value, with respect to the parameters $\mathsf{pp}' = \mathsf{RParam}(\mathsf{pp}; r'_{\mathsf{pp}})$.

Instantiation from DLIN

We will be using the additively homomorphic commitment scheme used in GOS [GOS06a]. We show that the scheme is randomizable. The scheme is as follows:

$\mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}})$: Compute $\mathcal{G}(1^\lambda) = (p, \mathbb{G}, \mathbb{G}_T, e, g_p)$ (as defined in Section 3.2). Parse $r_{\mathsf{pp}} = (x, y, z, R, S)$ for $x, y, z, R, S \in \mathbb{Z}_p^*$. Compute $f = g_p^x, h = g_p^y, g = g_p^z$; $(u, v, w) = (f^R, h^S, g^{R+S+1})$. Output

$$\mathsf{pp} = [p, \mathbb{G}, \mathbb{G}_T, e, g_p, f, h, g, u, v, w]^\dagger$$

$\mathsf{CS.Commit}(\mathsf{pp}, m)$: Choose $r, s \leftarrow \mathbb{Z}_p^*$ and let opening $o = (r, s)$. Output

$$\vec{c} = (c_1, c_2, c_3) = (u^m f^r, v^m h^s, w^m g^{r+s}).$$

$\mathsf{CS.Rand}(\mathsf{pp}, \vec{c})$: Parse $\vec{c} = (c_1, c_2, c_3)$. Choose $r', s' \leftarrow \mathbb{Z}_p^*$ and output

$$\vec{c}' = (c_1 f^{r'}, c_2 h^{s'}, c_3 g^{r'+s'}) = (u^m f^{r+r'}, v^m h^{s+s'}, w^m g^{(r+s)+r'+s'}).$$

Proposition 1. *Assuming DLIN, the commitment scheme described above is an additively homomorphic randomizable commitment scheme as per Definition 18.*

[†] We sometimes write $\mathsf{pp} = [f, h, g, u, v, w]$ and omit $(p, \mathbb{G}, \mathbb{G}_T, e, g_p)$ when it is obvious from the context.

Proof. The fact that this commitment scheme is perfectly binding and computationally hiding was proven in GOS [GOS06a]. Their commitment scheme is also **additively homomorphic**: Let $\vec{c} = (c_1, c_2, c_3)$ and $\vec{c}' = (c'_1, c'_2, c'_3)$ be commitments to (m, m') respectively. Then, $(c_1 c'_1, c_2 c'_2, c_3 c'_3)$ is a commitment to $(m+m')$.

We now prove perfect randomizability: Define $f_{\mathsf{Com}}(o, o') = (r + r', s + s')$, where $o = (r, s)$ and $o' = (r', s')$. We have that if $\vec{c} = (u^m f^r, v^m h^s, w^m g^{r+s})$ and $\vec{c}' = \mathsf{CS.Rand}(\mathsf{pp}, \vec{c}; o')$, then $\vec{c}' = (c'_1, c'_2, c'_3) = (u^m f^{r+r'}, v^m h^{s+s'}, w^m g^{r+r'+s+s'})$. Also for $o, o' \leftarrow (\mathbb{Z}_p^*)^2$, $f_{\mathsf{Com}}(o, o')$ is uniformly distributed.

We now prove perfect equivocation: $\mathsf{CS.Setup}'(1^\lambda)$ is defined exactly as $\mathsf{CS.Setup}(1^\lambda)$ except that it outputs $w = g^{R+S}$, as opposed to g^{R+S+1}. By DLIN, $(f, h, g, f^R, h^S, g^{R+S+1}) \approx_c (f, h, g, f^R, h^S, g^{R+S})$. Also $\mathsf{CS.Equivocate}(\mathsf{pp}, r_{\mathsf{pp}}, \vec{c}, (r, s), m')$ for $r_{\mathsf{pp}} = (x, y, z, R, S)$, outputs $o' = (r - Rm' + Rm, s - Sm' + Sm)$ which is distributed as fresh o.

□

We now instantiate the additional functionalities and observe that they satisfy the desired properties:

- $\mathsf{OutParam}(\mathsf{pp}^0)$: Parse $\mathsf{pp}^0 = [f, h, g, u, v, w]$. Output $\mathsf{pp}^1 = [f, h, g, u, v, w/g]$. Note that if $\mathsf{pp}^0 \in \mathsf{CS.Setup}(1^\lambda)$ then $\mathsf{pp}^1 \in \mathsf{CS.Setup}'(1^\lambda)$.

- $\mathsf{ValidParam}(\mathsf{pp}^0, \mathsf{pp}^1)$: Parse $\mathsf{pp}^b = [f, h, g, u, v, w_b]$ for all $b \in \{0, 1\}$. Output 1 if and only if $w_0 = w_1 \cdot g$.

- $\mathsf{RParam}(\mathsf{pp}, r_{\mathsf{pp}})$: Parse $r_{\mathsf{pp}} = (x', y', z', R', S')$ and parse $\mathsf{pp} = [f, h, g, u, v, w]$ for all $b \in \{0, 1\}$. Compute $f' = f^{x'}, h' = h^{y'}, g' = g^{z'}, (u', v', w') = \left((u f^{R'})^{x'}, (v h^{S'})^{y'}, (w_b g^{R'+S'})^{z'}\right)$. Output $\mathsf{pp}' = [f', h', g', u', v', w']$.

  We now show the function f_{pp} as required by RParam. Let $\sigma = (x, y, z, R, S)$ such that $\mathsf{pp} = \mathsf{CS.Setup}(1^\lambda; \sigma)$. Define

  $$f_{\mathsf{pp}}((x, y, z, R, S), (x', y', z', R', S')) = (xx', yy', zz', R + R', S + S').$$

  Note that $\mathsf{pp}' = \mathsf{CS.Setup}(1^\lambda; \sigma')$ where $\sigma' = (xx', yy', zz', R + R', S + S')$.

- $\mathsf{ChangeCom}(\mathsf{pp}, \vec{c}, r_{\mathsf{pp}})$: Parse $r_{\mathsf{pp}} = (x', y', z', R', S')$ and parse $\vec{c} = (\vec{c}_1, \vec{c}_2, \vec{c}_3)$. Output $\vec{c}' = \left((\vec{c}_1)^{x'}, (\vec{c}_2)^{y'}, (\vec{c}_3)^{z'}\right)$.

Dual Tuples and More Functionalities for FH NIWI

We now define the notion of *Dual Tuples* and *Intermediate Parameter* over a group $\mathbb{G}$. We then describe efficient algorithms $(\mathsf{ValidInter}, \mathsf{InterParam})$ that we will be using in our FH NIWI construction. Specifically, these algorithms are used in the construction of proof system for L_{TC}, to prove that two commitments with respect to two different parameters commit to the same value.

Let $(p, \mathbb{G}, \mathbb{G}_T, e, g_p) = \mathcal{G}(1^\lambda)$ and let $f_1, h_1, g_1, f_2, h_2, g_2 \in \mathbb{G}$.

Definition 19 (Dual Tuples)**.** *Tuples* $(f_1, h_1, g_1, f_1^{a_1}, h_1^{a_2}, g_1^{a_3})$ *and* $(f_2, h_2, g_2, f_2^{b_1}, h_2^{b_2}, g_2^{b_3})$ *are said to be* Dual Tuples *if* $a_i = b_i$ *for all* $i \in [3]$.

Let $\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}^* \in \mathbb{G}^6$ and denote $\mathsf{pp}_i = (f_i, h_i, g_i, u_i, v_i, w_i)$ for all $i \in [2]$.

Definition 20 (Intermediate Parameter)**.** *A tuple* pp^* *is said to be an* Intermediate Parameter *between* $(\mathsf{pp}_1, \mathsf{pp}_2)$ *if* $\mathsf{pp}^* = (f_j, h_j, g_j, u^*, v^*, w^*)$ *for some* $j \in [2]$ *and* $(\mathsf{pp}^*, \mathsf{pp}_{3-j})$ *are dual tuples.*

Remark 3. *If* $\mathsf{pp}_1, \mathsf{pp}_2$ *are such that* $(f_1, h_1, g_1) = (f_2, h_2, g_2)$*, then* pp_1 *is an intermediate parameter between* $(\mathsf{pp}_1, \mathsf{pp}_2)$ *since each tuple is trivially a dual tuple of itself. This is the case for* $\mathsf{pp}_1 \leftarrow \mathsf{CS.Setup}(1^\lambda)$ *and* $\mathsf{pp}_2 = \mathsf{OutParam}(\mathsf{pp}_1)$ *as instantiated in Section 3.3.1.*

We now define efficient algorithms $(\mathsf{ValidInter}, \mathsf{InterParam})$ which we will use in conjunction with the RaHE-commitment scheme:

$\mathsf{bit} = \mathsf{ValidInter}(\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}^*)$: The ValidInter is a efficient predicate which on input $(\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}^*)$ outputs 1 if and only if pp^* is an intermediate parameter between $\mathsf{pp}_1, \mathsf{pp}_2$.

Note that ValidInter can be efficiently checked using the bilinear map: Check that for some $i \in [2]$, $e(u^*, f_i) = e(u_i, f^*)$, $e(v^*, h_i) = e(v_i, h^*)$ and $e(w^*, g_i) = e(w_i, g^*)$.

Similarly, we define the following algorithm that given $\mathsf{pp}_1, \mathsf{pp}_2$, and randomness $r_{\mathsf{pp}} = (x, y, z, R, S)$ that generates pp_1, outputs the intermediate parameter pp^*.

$\mathsf{pp}^* = \mathsf{InterParam}(\mathsf{pp}_1, \mathsf{pp}_2, r_{\mathsf{pp}})$: InterParam is an efficient function which takes as input $(\mathsf{pp}_1, \mathsf{pp}_2, r_{\mathsf{pp}})$ which could be of the following forms:

- For all $i \in [2]$, $\mathsf{pp}_i \in \mathsf{CS.Setup}(1^\lambda)$ or $\mathsf{pp}_i \in \mathsf{CS.Setup}'(1^\lambda)$.
- $\mathsf{pp}_i \in \mathsf{CS.Setup}(1^\lambda)$ or $\mathsf{pp}_i \in \mathsf{CS.Setup}'(1^\lambda)$ for some $i \in [2]$ and where $\mathsf{pp}_{3-i} = \mathsf{RParam}(\mathsf{pp}_i; r_{\mathsf{pp}})$.

In both cases, it outputs pp^* such that $\mathsf{ValidInter}(\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}^*) = 1$.

$\mathsf{InterParam}(\mathsf{pp}_1, \mathsf{pp}_2, r_{\mathsf{pp}})$ can be instantiated as follows: Parse $\mathsf{pp}_i = [f_i, h_i, g_i, u_i, v_i, w_i]$ for all $i \in [2]$ and parse $r_{\mathsf{pp}} = (x, y, z, R, S)$. Suppose that $\mathsf{pp}_1 = \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}})$ (binding case) or $\mathsf{pp}_1 = \mathsf{CS.Setup}'(1^\lambda; r_{\mathsf{pp}})$ (hiding case). Compute $(u^*, v^*, w^*) = (f_2^R, h_2^S, g_2^T)$ where $T = R + S + 1$ for the binding case and $T = R + S$ for the hiding case. Output $\mathsf{pp}^* = [f_2, h_2, g_2, u^*, v^*, w^*]$. The case where $\mathsf{pp}_2 = \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}})$ or $\mathsf{pp}_2 = \mathsf{CS.Setup}'(1^\lambda; r_{\mathsf{pp}})$ is analogous.

Alternatively, if $\mathsf{pp}_{3-j} = \mathsf{RParam}(\mathsf{pp}_j; r_{\mathsf{pp}})$ for some $j \in [2]$, first parse $\mathsf{pp}_i = [f_i, h_i, g_i, u_i, v_i, w_i]$ for all $i \in [2]$ and parse $r_{\mathsf{pp}} = (x, y, z, R, S)$. Output $\mathsf{pp}^* = [f_i^x, h_i^y, g_i^z, u_i^x, v_i^y, w_i^z]$.

3.3.2 Proofs of Linearity.

In this section we describe the main ingredient for our fully homomorphic proofs, which is a NIWI proof system with additional properties for the parameterized language $L_{\mathsf{Lin}}[\mathsf{pp}]$.

Definition 21 (Linear Tuples)**.** *Let* $(p,\mathbb{G},\mathbb{G}_T,e,g_p) = \mathcal{G}(1^\lambda)$ *and let* f,h,g *be any three generators of* $\mathbb{G}$*. A tuple* $\vec{A} = (f^{a_1},h^{a_2},g^{a_3})$ *is said to be* linear *with respect to* (f,h,g) *if* $a_1+a_2=a_3$*.*

Before describing the parameterized language $L_{\mathsf{Lin}}[\mathsf{pp}]$, we describe the corresponding setup algorithm for the parameters of the language, given by Lin.Setup.

$\mathsf{Lin.Setup}(1^\lambda)$: Compute $\mathcal{G}(1^\lambda) = (p,\mathbb{G},\mathbb{G}_T,e,g_p)$. Choose at random $x,y,z \leftarrow \mathbb{Z}_p^*$. Compute $f = g_p^x, h = g_p^y, g = g_p^z$. Output $\mathsf{pp} = [p,\mathbb{G},\mathbb{G}_T,e,g_p,f,h,g]$.

We abuse notation and let pp denote the output of Lin.Setup as well as the output of CS.Setup. Note that $\mathsf{pp} \leftarrow \mathsf{Lin.Setup}(1^\lambda)$ is a subset of $\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda)$.

We now define the language $L_{\mathsf{Lin}}[\mathsf{pp}]$ where $\mathsf{pp} \leftarrow \mathsf{Lin.Setup}(1^\lambda)$. $L_{\mathsf{Lin}}[\mathsf{pp}]$ is the language consisting of a pair of tuples such that one of them is linear. It is defined as follows:

$$L_{\mathsf{Lin}}[\mathsf{pp}] = \left\{ \left(\vec{A},\vec{B}\right) \mid \exists\, (w_1,w_2,w_3)\ \left((w_1+w_2=w_3)\ \wedge\ \left(\vec{A} = (f^{w_1},h^{w_2},g^{w_3})\ \vee\ \vec{B} = (f^{w_1},h^{w_2},g^{w_3})\right)\right)\right\}$$

NIWI Proof from GOS

We first describe the NIWI proof $(\mathsf{Lin.Prove},\mathsf{Lin.Verify})$ for $L_{\mathsf{Lin}}[\mathsf{pp}]$ from GOS [GOS06a]:

$\mathsf{Lin.Prove}(\mathsf{pp},(A_1,A_2,A_3),(B_1,B_2,B_3),(a_1,a_2,a_3))$: Without loss of generality, let (a_1,a_2,a_3) be such that $(A_1,A_2,A_3) = (f^{a_1},h^{a_2},g^{a_3})$ and $a_1+a_2=a_3$. Choose $t \xleftarrow{\$} \mathbb{Z}_p^*$ and output proof Π which consists of the following matrix:

$$\begin{bmatrix} \pi_{11} = B_1^{a_1} & \pi_{12} = B_2^{a_1} h^{-t} & \pi_{13} = B_3^{a_1} g^{-t} \\ \pi_{21} = B_1^{a_2} f^{t} & \pi_{22} = B_2^{a_2} & \pi_{23} = B_3^{a_2} g^{t} \end{bmatrix}$$

$\mathsf{Lin.Verify}(\mathsf{pp},(A_1,A_2,A_3),(B_1,B_2,B_3),\Pi)$:

- Compute $\pi_{31} = \pi_{11}\pi_{21}$ and $\pi_{32} = \pi_{12}\pi_{22}$ and $\pi_{33} = \pi_{13}\pi_{23}$.
- Check $e(A_1,B_1) = e(f,\pi_{11})$, $e(A_2,B_2) = e(h,\pi_{22})$, $e(A_3,B_3) = e(g,\pi_{33})$.
- Finally check $e(A_1,B_2)e(A_2,B_1) = e(f,\pi_{12})e(h,\pi_{21})$, $e(A_2,B_3)e(A_3,B_2) = e(h,\pi_{23})e(g,\pi_{32})$ and $e(A_1,B_3)e(A_3,B_1) = e(f,\pi_{13})e(g,\pi_{31})$.

Proposition 2 ([GOS06a])**.** *Assuming DLIN, the proof system described above is a perfectly sound witness indistinguishable proof system for the language* $L_{\mathsf{Lin}}[\mathsf{pp}]$ *(as per Definition 3).*

Remark 4. *If* $\Pi = [\pi_{11},\ldots,\pi_{33}]$ *is a valid proof for* $((A_1,A_2,A_3),(B_1,B_2,B_3)) \in L_{\mathsf{Lin}}[\mathsf{pp}]$*, then* $\Pi^{-1} = [\pi_{11}^{-1},\ldots,\pi_{33}^{-1}]$ *is a valid proof for* $((A_1^{-1},A_2^{-1},A_3^{-1}),(B_1,B_2,B_3)) \in L_{\mathsf{Lin}}[\mathsf{pp}]$ *and for* $((A_1,A_2,A_3),(B_1^{-1},B_2^{-1},B_3^{-1})) \in L_{\mathsf{Lin}}[\mathsf{pp}]$.

GOS [GOS06a] provided a NIWI proof for $L_{\mathsf{Lin}}[\mathsf{pp}]$ as described above. In our work, we need the NIWI proof system to satisfy two additional properties: The first is malleability with respect to randomization, namely given a tuple $(\vec{A},\vec{B}) \in L_{\mathsf{Lin}}[\mathsf{pp}]$ with NIWI proof Π, it is possible to randomize $(\vec{A},\vec{B})$ to a new tuple $(\vec{A}',\vec{B}') \in L_{\mathsf{Lin}}[\mathsf{pp}]$ and maul the proof Π to be proof Π' with respect to $(\vec{A}',\vec{B}')$. As a second property, we require that the proof system satisfies strong witness indistinguishability with respect to specific distributions (which we describe later in the section).

Malleable Proofs for L_{Lin}

We now show that $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify})$ is malleable with respect to the transformation $\mathsf{Lin.T} = (\mathsf{Lin.Transform}, \mathsf{Lin.WitTrans})$ defined as follows:

$$\mathsf{Lin.Transform}(\mathsf{pp},\vec{A},\vec{B};(r_1,r_2,s_1,s_2)) \triangleq \left((A_1 f^{r_1}, A_2 h^{r_2}, A_3 g^{r_1+r_2}), (B_1 f^{s_1}, B_2 h^{s_2}, B_3 g^{s_1+s_2})\right)$$

where $\mathsf{pp} = [p,\mathbb{G},\mathbb{G}_T,e,g_p,f,h,g]$, $\vec{A} = (A_1,A_2,A_3)$ and $\vec{B} = (B_1,B_2,B_3)$.

$$\mathsf{Lin.WitTrans}(\mathsf{pp},(\vec{A},\vec{B}),(w_1,w_2,w_3);(r_1,r_2,s_1,s_2)) \triangleq (w_1+z_1, w_2+z_2, w_3+z_1+z_2) \text{ where}$$
$$(z_1,z_2) = (r_1,r_2) \text{ if } \vec{A} = (f^{w_1},h^{w_2},g^{w_3}) \text{ else } (z_1,z_2) = (s_1,s_2) \text{ if } \vec{B} = (f^{w_1},h^{w_2},g^{w_3})$$

Mauled proof for $\mathsf{Lin.Transform}(\mathsf{pp},\vec{A},\vec{B},(r_1,r_2,s_1,s_2)) = (A_1 f^{r_1}, A_2 h^{r_2}, A_3 g^{r_3}), (B_1 f^{s_1}, B_2 h^{s_2}, B_3 g^{s_3})$ is given by $\mathsf{Lin.Maul}(\mathsf{pp},(\vec{A},\vec{B}),(r_1,r_2,s_1,s_2),\Pi)$: Choose $t \leftarrow \mathbb{Z}_p^*$, and output a proof Π' consisting of the following matrix:

$$\begin{bmatrix} \pi'_{11} = \pi_{11} A_1^{s_1} B_1^{r_1} f^{r_1 s_1} & \pi'_{12} = \pi_{12} A_2^{s_1} B_2^{r_1} h^{r_1 s_2 - t} & \pi'_{13} = \pi_{13} A_3^{s_1} B_3^{r_1} g^{r_1 s_3 - t} \\ \pi'_{21} = \pi_{21} A_1^{s_2} B_1^{r_2} f^{r_2 s_1 + t} & \pi'_{22} = \pi_{22} A_2^{s_2} B_2^{r_2} h^{r_2 s_2} & \pi'_{23} = \pi_{23} A_3^{s_2} B_3^{r_2} g^{r_2 s_3 + t} \end{bmatrix}$$

Proposition 3. *Assuming DLIN, the proof system* $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify}, \mathsf{Lin.Maul})$ *is a malleable NIWI for* $L_{\mathsf{Lin}}[\mathsf{pp}]$ *as per Definition 5, with respect to transformation* $\mathsf{Lin.T} = (\mathsf{Lin.Transform}, \mathsf{Lin.WitTrans})$.

Proof. Malleability: Fix any PPT adversary $\mathcal{A}$ and fix any $\mathsf{pp} \in \mathsf{Lin.Setup}(1^\lambda)$. Let $(\vec{A},\vec{B},\Pi) \leftarrow \mathcal{A}(\mathsf{pp})$. Let $\Pi' = \mathsf{Lin.Maul}(\mathsf{pp},(\vec{A},\vec{B}),(r_1,r_2,s_1,s_2),\Pi;t')$ for randomly chosen r_1,r_2,s_1,s_2,t'.

We prove that $\mathsf{Lin.Verify}(\mathsf{pp},(\vec{A}',\vec{B}'),\Pi') = 1$ if and only if $\mathsf{Lin.Verify}(\mathsf{pp},(\vec{A},\vec{B}),\Pi) = 1$. Let $g_1 = f, g_2 = h$ and $g_3 = g$. For $i \in \{1,2,3\}$,

$$e(A_i g_i^{r_i}, B_i g_i^{s_i}) = e(A_i,B_i) e(A_i,g_i^{s_i}) e(g_i^{r_i},B_i) e(g_i^{r_i},g_i^{s_i}) = e(g_i,\pi_{i,i}) e(A_i^{r_i} B_i^{s_i} g_i^{r_i s_i}, g_i) = e(g_i,\pi'_{i,i})$$

if and only if $e(A_i,B_i) = e(g_i,\pi_{i,i})$ which are the first three verification check for Π.

For $i \neq j$ and $i,j \in \{1,2,3\}$,

$$\begin{aligned} & e(A_i g_i^{r_i}, B_j g_j^{s_j}).e(A_j g_j^{r_j}, B_i g_i^{s_i}) \\ =& e(A_i,B_j) e(A_j,B_i) e(A_i,g_j^{s_j}) e(g_i^{r_i},B_j) e(A_j,g_i^{s_i}) e(g_j^{r_j},B_i) e(g_i,g_j)^{r_i s_j + r_j s_i} \\ =& e(g_j,\pi_{i,j}) e(g_i,\pi_{j,i}) e(A_i^{s_j} B_i^{r_j} g_i^{r_i s_j + t'}, g_j) e(B_j^{r_i} A_j^{s_i} g_j^{r_j s_i - t'}, g_i) \text{ for } t' = (b_i r_j - b_j r_i) \\ =& e(g_j,\pi'_{i,j}) e(g_i,\pi'_{j,i}) \end{aligned}$$

if and only if $e(A_i, B_j) = e(g_j, \pi_{i,j})e(g_i, \pi_{j,i})$, which are the verification checks for Π.

Perfect Randomizability: Fix any $\mathsf{pp} \in \mathsf{Lin.Setup}(1^\lambda)$ and $(\vec{A}, \vec{B}) \in L_{\mathsf{Lin}}[\mathsf{pp}]$ with witness $\vec{a} = (a_1, a_2, a_3)$. Fix $r_1, r_2, s_1, s_2, t, t'$ such that $\Pi = \mathsf{Lin.Prove}(\mathsf{pp}, \vec{A}, \vec{B}, \vec{a}; t)$, $(\vec{A}', \vec{B}') = \mathsf{Transform}(\mathsf{pp}, (\vec{A}, \vec{B}), (r_1, r_2, s_1, s_2))$ and $\Pi' = \mathsf{Lin.Maul}(\mathsf{pp}, (\vec{A}, \vec{B}), (r_1, r_2, s_1, s_2), \Pi; t')$. Let $r_3 = r_1 + r_2$ and $s_3 = s_1 + s_2$.

Function f_{Lin} is given by: $f_{\mathsf{Lin}}(\mathsf{pp}, \vec{a}, t, t', (r_1, r_2, s_1, s_2)) = t + t' + \sigma$ where $\sigma = a_1 s_2 - a_2 s_1 = a_1 s_3 - s_3 a_1 = a_3 s_2 - a_2 s_3$. Also if t' is uniform, $t'' = f_{\mathsf{Lin}}(\mathsf{pp}, \vec{a}, t, t', (r_1, r_2, s_1, s_2))$ is distributed as uniform. □

Remark 5. *We denote by* $\mathsf{Lin.Transform}(\mathsf{pp}, (\vec{A}, \vec{B}), (r_1, r_2))$ *the transformation given by* $\mathsf{Lin.Transform}(\mathsf{pp}, (\vec{A}, \vec{B}), (r_1, r_2, r_1, r_2))$.

Strong NIWI for L_{Lin}.

For our FH NIWI construction, we require that the NIWI proofs for $(\vec{A}, \vec{B}) \in L_{\mathsf{Lin}}[\mathsf{pp}]$ satisfy strong witness indistinguishability with respect to distributions $\mathcal{D}_0(\mathsf{pp}), \mathcal{D}_1(\mathsf{pp})$ for $\mathsf{pp} \leftarrow \mathsf{Lin.Setup}(1^\lambda)$. For every $b \in \{0, 1\}$, distribution $\mathcal{D}_b(\mathsf{pp})$ is defined as follows:

Parse $\mathsf{pp} = [p, \mathbb{G}, \mathbb{G}_T, e, g_p, f, h, g]$. Choose $a_1, a_2 \leftarrow \mathbb{Z}_p^*$, let $a_3 = a_1 + a_2$. Let $\vec{A}_b = (f^{a_1}, h^{a_2}, g^{a_3 - b})$ and let $\vec{B}_b = (f^{a_1}, h^{a_2}, g^{a_3 - b + 1})$. Output $(\vec{A}_b, \vec{B}_b)$.

Recall that $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify}, \mathsf{Lin.Maul})$ is said to be strong NIWI with respect to distributions $\mathcal{D}_0(\mathsf{pp})$, $\mathcal{D}_1(\mathsf{pp})$ (as per Definition **??**), if the following holds:

$$\{\mathsf{pp}, (\vec{A}_0, \vec{B}_0), \pi_0\} \approx \{\mathsf{pp}, (\vec{A}_1, \vec{B}_1), \pi_1\}$$

where $(\vec{A}_b, \vec{B}_b) \leftarrow \mathcal{D}_b(\mathsf{pp})$ and where $\pi_b \leftarrow \mathsf{Lin.Prove}(\mathsf{pp}, \vec{A}_b, \vec{B}_b, (a_1, a_2, a_3))$.

3.3.3 Assumption: DLIN with Leakage

In this subsection, we state our new assumption on bilinear maps: *DLIN with Leakage*.
Let $\mathsf{pp} \leftarrow \mathsf{Lin.Setup}(1^\lambda)$ and parse $\mathsf{pp} = [p, \mathbb{G}, \mathbb{G}_T, e, f, h, g]$. DLIN with Leakage states that $\mathcal{D}'_0(1^\lambda) \approx_c \mathcal{D}'_1(1^\lambda)$ where $\mathcal{D}'_b(1^\lambda)$ is as follows:

- $\mathcal{D}'_0(1^\lambda)$: Choose $R, S, t \leftarrow \mathbb{Z}_p^*$ and output pp along with the following matrix:

$$\begin{bmatrix} f^R & h^S & g^{R+S} \\ f^{R^2} & h^{RS-t} & g^{R(R+S+1)-t} \\ f^{RS+t} & h^{S^2} & g^{S(R+S+1)+t} \end{bmatrix}$$

- $\mathcal{D}'_1(1^\lambda)$: Choose $R, S, t \leftarrow \mathbb{Z}_p^*$ and output pp along with the following matrix:

$$\begin{bmatrix} f^R & h^S & g^{R+S-1} \\ f^{R^2} & h^{RS-t} & g^{R(R+S-1)-t} \\ f^{RS+t} & h^{S^2} & g^{S(R+S-1)+t} \end{bmatrix}$$

Proposition 4. *The DLIN with Leakage assumption is secure in the generic group model.*

Proof. We defer the proof to Appendix **??**. □

Proposition 5. *Assuming DLIN with Leakage,* $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify})$ *is strong NIWI for* $L_{\mathsf{Lin}}[\mathsf{pp}]$ *with respect to* $\mathcal{D}_0, \mathcal{D}_1$ *(as described in Section 3.3.2).*

3.4 Fully Homomorphic NIZK Proofs

In this section, we construct fully homomorphic NIZK proofs for NP from DLIN. Our construction uses certain NIWI proof systems as ingredients; we describe them in Section 3.4.1. In Section 3.4.2, we present our FH NIZK construction from these ingredients. In Section 3.4.3, we instantiate the ingredients from DLIN.

3.4.1 Ingredients for the FH NIZK Construction

Recall that the generic template for NIZK proofs for $L_{\mathcal{U}}$ from GOS [GOS06b, GOS06a] is as follows:

GOS Template. An $L_{\mathcal{U}}$ instance is of the form (C, out) where $C : \{0,1\}^t \to \{0,1\}$ and $\mathsf{out} \in \{0,1\}$. Denote by n the number of wires in C (including t input wires and excluding the output wire) and denote by m the number of gates. The NIZK proof is computed as follows:

1. Let $w_1, \ldots, w_n$ be the values induced by witness $\vec{w} \in \{0,1\}^t$ on all the wires of the circuit C (except the output wire). Commit to all wire values in the circuit C using an additively homomorphic commitment scheme. Let $\vec{c}_i$ denote the commitment to wire i for $i \in [n]$.

2. For each commitment $\vec{c}_i$ to wire i in C, give a NIWI proof that $\vec{c}_i$ is a commitment to a boolen value. We denote such a proof by π^i_{bit} for $i \in [n]$.

3. For each gate in C, give a NIWI proof that the input and the output values to the gate are consistent with the gate functionality. We denote such a proof by π^j_{gate} for $j \in [m]$.

4. Give a canonical commitment to the output value out, denoted by $\vec{c}_{\mathsf{out}}$.

A NIZK proof for (C, out) is of the form:

$$\Pi = \left[\{\vec{c}_i\}_{i=1}^n, \{\pi^i_{\mathsf{bit}}\}_{i=1}^n, \{\pi^j_{\mathsf{gate}}\}_{j=1}^m, \vec{c}_{\mathsf{out}} \right].$$

Our FH NIZK construction follows a similar template and will use three ingredients: A RaHE-commitment scheme $(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$ as per Definition 18, a NIWI proof system to prove that committed bit is boolean (denoted by *Bit Proofs*), a NIWI proof system to prove that committed bits are consistent with the gate functionality (denoted by *Gate Proofs*).

We note that GOS [GOS06a] defined and constructed NIWI proof systems $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify})$ and $(\mathsf{NAND.Prove},$
$\mathsf{NAND.Verify})$ for bit proofs and gate consistency proofs respectively. This was sufficient for their NIZK

construction. However, to achieve full homomorphism, we require the NIWI proof systems to be malleable. At a high level, we need to be able to randomize the commitments in the proofs and maul the bit proofs and gate consistency proofs with respect to the new commitments. In what follows, we describe the concrete transformations for which malleability is needed.

Malleability of Bit Proofs. Let $(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$ be RaHE-commitment scheme as per Definition 18. For Bit Proofs, we consider the following language parameterized by $\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda)$.

$$L_{\mathsf{Com}}[\mathsf{pp}] = \{\vec{c} \mid \exists\ (b,o) \text{ s.t. } \vec{c} = \mathsf{CS.Commit}(\mathsf{pp}, b; o) \ \wedge\ b \in \{0,1\}\}$$

We require a malleable NIWI proof system $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.Maul})$ for $L_{\mathsf{Com}}[\mathsf{pp}]$ (as per Definition 5), with respect to the transformation: $\mathsf{Bit.T} = (\mathsf{Bit.Transform}, \mathsf{Bit.WitTrans})$ given by

$$\mathsf{Bit.Transform}(\mathsf{pp}, \vec{c}, o') = \mathsf{CS.Rand}(\mathsf{pp}, \vec{c}; o') \text{ and } \mathsf{Bit.WitTrans}(\mathsf{pp}, \vec{c}, (b,o), o') = f_{\mathsf{Com}}(\mathsf{pp}, o, o')$$

where o' is fresh randomness.

Malleability of Gate Consistency Proofs. Without loss of generality, we assume that the circuit is made up of NAND gates. Boolean values b_1, b_2, b_3 satisfy NAND relation that is, $b_3 = b_1 \barwedge b_2$ if and only if $b_1 + b_2 + 2b_3 - 2 \in \{0,1\}$. For Gate Proofs, we consider the following language parameterized by $\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda)$.

$$L_{\mathsf{NAND}}[\mathsf{pp}] = \left\{\{\vec{c}_i\}_{i\in[3]} \mid \exists\ \{b_i, o_i\}_{i\in[3]} \text{ s.t. } \vec{c}_i = \mathsf{CS.Commit}(b_i; o_i) \ \wedge\ (b_3 = b_1 \barwedge b_2) \ \wedge\ \{b_i \in \{0,1\}\}_{i\in[3]}\right\}$$

We require a malleable NIWI proof system $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify}, \mathsf{NAND.Maul})$ for $L_{\mathsf{NAND}}[\mathsf{pp}]$ (as per Definition 5), with respect to the transformation: $\mathsf{NAND.T} = (\mathsf{NAND.Transform}, \mathsf{NAND.WitTrans})$ given by

$$\mathsf{NAND.Transform}(\mathsf{pp}, \{\vec{c}_i\}_{i\in[3]}, \{o'_i\}_{i\in[3]}) = \{\vec{c}'_i\}_{i\in[3]} \text{ and } \mathsf{NAND.WitTrans}(\mathsf{pp}, \{\vec{c}_i, b_i, o_i, o'_i\}_{i\in[3]}) = f_{\mathsf{Com}}(\mathsf{pp}, o, o')$$

where $\vec{c}'_i = \mathsf{CS.Rand}(\mathsf{pp}, \vec{c}_i, o'_i)$ for fresh randomness (o'_1, o'_2, o'_3) and where $o = o_1 + o_2 + 2o_3 - 2$ and $o' = o'_1 + o'_2 + 2o'_3 - 2$.

3.4.2 FH NIZK Construction

We use the following ingredients for our FH NIZK construction:

- Randomizable commitment scheme as per Definition 18, which is additively homomorphic and equivocable, denoted by
$$(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$$
- Malleable NIWI proof system for $L_{\mathsf{Com}}[\mathsf{pp}]$ with respect to transformation $\mathsf{Bit.Transform}$ from Section 3.4.1, denoted by
$$(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.Maul}).$$

– Malleable NIWI proof system for $L_{\mathsf{NAND}}[\mathsf{pp}]$ with respect to transformation NAND.Transform as described in Section 3.4.1, denoted by

$$(\mathsf{NAND.Prove}, \mathsf{NAND.Verify}, \mathsf{NAND.Maul}).$$

We now describe our construction:

$\mathsf{NIZK.Setup}(1^k)$: Output $\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda)$.

$\mathsf{NIZK.Prove}(\mathsf{CRS}, (C, \mathsf{out}), \vec{w})$: Let $C : \{0,1\}^t \to \{0,1\}$ consist of n wires (including input wires and excluding output wire), one output wire and m NAND gates. Let $w_1, \ldots, w_n, w_{\mathsf{out}}$ be the boolean values induced by $\vec{w} \in \{0,1\}^t$ on all (input and internal) the wires of circuit C and where w_{out} is the output wire ($w_{\mathsf{out}} = \mathsf{out}$).

1. For wire i, commit to the value w_i as follows: Choose o_i at random and compute

$$\vec{c}_i = \mathsf{CS.Commit}(w_i; o_i).$$

   For the output wire w_{out}, use canonical commitments so that $\vec{c}_{\mathsf{out}} = \mathbf{1}$ if $\mathsf{out} = 1$ and $\vec{c}_{\mathsf{out}} = \mathbf{0}$ if $\mathsf{out} = 0$.

2. For each wire i (except output), generate a proof that commitment $\vec{c}_i$ commits to a bit. Namely, compute

$$\pi^i_{\mathsf{bit}} = \mathsf{Bit.Prove}(\mathsf{pp}, \vec{c}_i, o_i)$$

   where o_i is the opening for commitment $\vec{c}_i$.

3. For each NAND gate j, let j_1, j_2 be the input wires and j_3 be the output wire with corresponding commitments $\vec{c}_{j_i}$ for $i \in [3]$. Compute

$$\pi^j_{\mathsf{gate}} = \mathsf{NAND.Prove}(\mathsf{pp}, \{\vec{c}_{j_i}\}_{i\in[3]}, \{o_{j_i}\}_{i\in[3]}).$$

Finally output

$$\Pi = \left[\{\vec{c}_i\}_{i=1}^n, \{\pi^i_{\mathsf{bit}}\}_{i=1}^n, \{\pi^j_{\mathsf{gate}}\}_{j=1}^m, \vec{c}_{\mathsf{out}}\right]$$

$\mathsf{NIZK.Verify}(\mathsf{CRS}, (C, \mathsf{out}), \Pi)$: Parse $\Pi = \left[\{\vec{c}_i\}_{i=1}^n, \{\pi^i_{\mathsf{bit}}\}_{i=1}^n, \{\pi^j_{\mathsf{gate}}\}_{j=1}^m, \vec{c}_{\mathsf{out}}\right]$.

1. For each wire $i \in [n]$, check whether $\mathsf{Bit.Verify}(\mathsf{pp}, \vec{c}_i, \pi^i_{\mathsf{bit}}) = 1$. Else output 0.
2. For each NAND gate $j \in [m]$, with input wires j_1, j_2 and output wire j_3 and with corresponding commitments $\vec{c}_{j_i}$, for $i = 1,2,3$. Check that $\mathsf{NAND.Verify}(\mathsf{CRS}, \{\vec{c}_{j_i}\}_{i=1}^3, \pi^j_{\mathsf{gate}}) = 1$. Else output 0.
3. Finally check that $\pi_{\mathsf{out}} = \mathbf{1}$ for $\mathsf{out} = 1$ and $\pi_{\mathsf{out}} = \mathbf{0}$ for $\mathsf{out} = 0$.

$\mathsf{NIZK.Rand}(\mathsf{CRS}, (C, \mathsf{out}), \Pi))$: Parse $\Pi = [\{\vec{c}_i\}_{i=1}^n, \{\pi^i_{\mathsf{bit}}\}_{i=1}^n, \{\pi^j_{\mathsf{gate}}\}_{j=1}^m, \vec{c}_{\mathsf{out}}]$.

1. For each wire i, choose o'_i at random and compute $\vec{c}'_i = \mathsf{CS.Rand}(\mathsf{pp}, \vec{c}_i, o'_i)$.
2. Compute $\pi^{i'}_{\mathsf{bit}} \leftarrow \mathsf{Bit.Maul}\big(\mathsf{pp}, \vec{c}_i, o'_i, \pi^i_{\mathsf{bit}}\big)$.
3. For each NAND gate j, with input wires j_1, j_2 and output wire j_3, compute $\pi^{j'}_{\mathsf{gate}} \leftarrow \mathsf{NAND.Maul}(\mathsf{pp}, \{\{\vec{c}_{j_i}, o'_{j_i}\}_{i\in[3]}, \pi^j_{\mathsf{gate}})$.
4. Finally keep the output proof $\vec{c}_{\mathsf{out}}$ same as before. Output

$$\Pi' = \left[\{\vec{c}'_i\}_{i=1}^n, \{\pi^{i'}_{\mathsf{bit}}\}_{i=1}^n, \{\pi^{j'}_{\mathsf{gate}}\}_{j=1}^m, \vec{c}_{\mathsf{out}}\right]$$

$\mathsf{NIZK.Eval}(\mathsf{CRS}, \{(C_i, b_i, \Pi_i)\}_{i=1}^k, C')$:

1. Compute $(C, \mathsf{out}^*) = \mathsf{Compose}(\{(C_i, b_i, \Pi_i)\}_{i=1}^k, C')$.
2. Let $\pi^i_{\mathsf{out}} \in \Pi'_i$ be the gate consistency proof for the output gate out^i of circuit C_i for $i \in [k]$. Compute $\widehat{\Pi_i}$ as the proof Π'_i without the proof π^i_{out}, namely $\widehat{\Pi_i} = \Pi'_i \setminus \pi^i_{\mathsf{out}}$.
3. Compute a proof for C' with witness $(b_1, \ldots, b_k)$ by computing: $\Pi^* \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, (C', \mathsf{out}^*), (b_1, \ldots, b_k))$ where $\mathsf{out}^* = C'(b_1, \ldots, b_k)$.
4. For each output gate out^i for C_i, $i \in [k]$, let i_1, i_2 be the input wires to the gate and i_3 be the output wire (with value b_i). Let o'_{i_3} be the randomness used in step 2 such that $\vec{c}'_{i_3} \in \Pi'$ and $\vec{c}'_{i_3} = \mathsf{CS.Commit}(\mathsf{pp}, b_i, o'_{i_3})$. Compute $(\pi^i_{\mathsf{out}})' = \mathsf{NAND.Maul}(\mathsf{pp}, \{\{\vec{c}'_{j_i}, o'_{j_i}\}_{i\in[3]}, \pi^i_{\mathsf{out}})$ where $o'_{i_k} = 0$ for $k \in [2]$.
5. Let $\Pi = \big[\widehat{\Pi_1}, \ldots, \widehat{\Pi_k}, \Pi^*, (\pi^1_{\mathsf{out}})', \ldots, (\pi^k_{\mathsf{out}})'\big]$. Compute $\Pi' \leftarrow \mathsf{NIZK.Rand}(\mathsf{CRS}, (C, \mathsf{out}^*), \Pi)$. Finally output $(C, \mathsf{out}^*, \Pi')$.

Theorem 3. *The construction as described above is a fully homomorphic NIZK proof system for $L_{\mathcal{U}}$ as per Definition 16.*

Proof. The algorithms $(\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ are identical to the NIZK construction in Groth-Ostrovsky-Sahai [GOS06a]. Hence completeness, statistical soundness[‡] and perfect zero-knowledge follows. It is left to prove the following claims.

Claim 1. Π_{FHNIZK} *satisfies completeness of evaluation.*

Proof. Completeness of evaluation follows from completeness of randomizability and malleability of Bit proofs and Gate consistency proofs. □

Claim 2. Π_{FHNIZK} *satisfies randomizability.*

Proof. We show that for any instance (C, b) with witness w and any proof Π such that $\mathsf{NIZK.Verify}(\mathsf{CRS}, (C, b), \Pi) = 1$, the following distributions are identical:

$$\big\{(C, b), w, \Pi, \vec{R}, \Pi_f\big\} \text{ and } \big\{(C, b), w, \Pi, \vec{R}, \Pi'\big\}$$

[‡]The GOS NIZK construction achieves statistical soundness in the common random string model.

where Π_f is a fresh proof obtained as $\Pi_f \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS},(C,b),w)$, Π' is a randomized proof obtained as $\Pi' \leftarrow \mathsf{NIZK.Rand}(\mathsf{CRS},(C,b),\Pi)$ and $\vec{R}$ is such that $\Pi = \mathsf{NIZK.Prove}(\mathsf{CRS},(C,b),w;\vec{R})$. Parse

$$\Pi = [\{\vec{c}_i\}_{i=1}^n, \{\pi^i_{\mathsf{bit}}\}_{i=1}^n, \{\pi^j_{\mathsf{gate}}\}_{j=1}^m, \vec{c}_{\mathsf{out}}].$$

We will parse

$$\vec{R} = [o_1,\ldots,o_n,t_1,\ldots,t_n,s_1,\ldots,s_m]$$

where o_i is such that $\vec{c}_i = \mathsf{CS.Commit}(\mathsf{pp},w_i,o_i)$ where w_i is the value on wire i, t_i is such that $\pi^i_{\mathsf{bit}} = \mathsf{Bit.Prove}(\mathsf{pp},\vec{c}_i,o_i;t_i)$ for $i \in [n]$. Finally s_j is such that
$\pi^j_{\mathsf{gate}} = \mathsf{NAND.Prove}(\mathsf{pp},\{\vec{c}_{j_i},o_{j_i}\}_{i\in[3]};s_j)$.

Similarly, let $\Pi' = \mathsf{NIZK.Rand}(\mathsf{CRS},(C,b),\Pi;\vec{R}')$ and parse

$$\vec{R}' = [o'_1,\ldots,o'_n,t'_1,\ldots,t'_n,s'_1,\ldots,s'_m]$$

where o'_i is such that $\vec{c}'_i = \mathsf{CS.Rand}(\mathsf{pp},\vec{c},o'_i)$ for $i \in [n]$, t'_i is such that $\pi^{i'}_{\mathsf{bit}} = \mathsf{Bit.Maul}(\mathsf{pp},\vec{c}_i,o'_i,\pi^i_{\mathsf{bit}};t'_i)$ for $i \in [n]$. Finally s'_j is such that $\pi^{j'}_{\mathsf{gate}} = \mathsf{NAND.Maul}(\mathsf{pp},\{\vec{c}_{j_i},o'_{j_i}\}_{i\in[3]},s'_j,\pi^j_{\mathsf{NAND}})$.

By perfect randomizability of the commitment scheme, there exists f_{Com} such that $o''_i = f_{\mathsf{Com}}(o_i,o'_i)$ and $\vec{c}''_i = \mathsf{CS.Commit}(\mathsf{pp},w_i;o''_i)$ and o''_i is uniformly distributed. By perfect randomizability of bit proofs, there exists f_{bit} such that $f_{\mathsf{bit}}(t_i,o_i,o'_i,t'_i) = t''_i$ such that $\pi^{i'}_{\mathsf{bit}} = \mathsf{Bit.Prove}(\mathsf{pp},\vec{c}'_i,o''_i;t''_i)$ and t''_i is uniformly distributed.

Similarly, perfect randomizability of gate consistency proofs, there exists f_{gate} such that $f_{\mathsf{gate}}(s_j,s''_j,$ $\{o_{j_i},o'_{j_i}\}_{i\in[3]}) = s''_j$ and $\pi^{j'}_{\mathsf{gate}} = \mathsf{NAND.Prove}(\mathsf{pp},\{\vec{c}_{j_i},o''_{j_i}\}_{i\in[3]};s''_j)$ where s''_j is distributed as uniform.

We can now identify $\vec{R}''$ such that $\Pi' = \mathsf{NIZK.Prove}(\mathsf{CRS},(C,b),w;\vec{R}'')$ as follows:

$$\vec{R}'' = [o''_1,\ldots,o''_n,t''_1,\ldots,t''_n,s''_1,\ldots,s''_m]$$

which is distributed as uniform. It follows that $\{(C,b),w,\Pi,\vec{R},\Pi_f\}$, $\{(C,b),w,\Pi,\vec{R},\Pi'\}$ are identical, where $\Pi_f = \mathsf{NIZK.Prove}(\mathsf{CRS},(C,b),w;\vec{S})$ and $\Pi' = \mathsf{NIZK.Prove}(\mathsf{CRS},(C,b),w;\vec{R}'')$ for truly random $\vec{S},\vec{R}''$.

□

Claim 3. Π_{FHNIZK} *satisfies unlinkability.*

Proof. Follows from completeness of underlying primitives and randomizability of the NIZK. □

□

3.4.3 Instantiating the Ingredients

We now give concrete instantiations of the two malleable NIWI proof systems described in Section 3.4.1: bit proofs ($\mathsf{Bit.Prove},\mathsf{Bit.Verify},\mathsf{Bit.Maul}$) and gate consistency proofs ($\mathsf{NAND.Prove},\mathsf{NAND.Verify},\mathsf{NAND.Maul}$) from DLIN.

Bit Proofs. Let $(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$ be the RaHE-commitment scheme from Section 3.3.1. Recall the language,

$$L_{\mathsf{Com}}[\mathsf{pp}] = \{(c_1,c_2,c_3) \mid \exists\ (b,r,s) \text{ s.t. } ((c_1,c_2,c_3) = (u^b f^r, v^b h^s, w^b g^{r+s})) \ \wedge\ b \in \{0,1\}\}$$

Let $\vec{c} = (c_1,c_2,c_3)$, define $\delta_{\mathsf{pp}}(\vec{c}) \triangleq (c_1/u, c_2/v, (c_3/w))$. We drop pp when it is obvious from the context. Observe that if $\vec{c} \in L_{\mathsf{Com}}[\mathsf{pp}]$ then $\vec{c}$ or $\delta(\vec{c})$ is linear. In particular, $\vec{c}$ is linear if $\vec{c}$ commits to 0 and $\delta(\vec{c})$ is linear if $\vec{c}$ commits to 1.

Let $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify}, \mathsf{Lin.Maul})$ be the malleable NIWI proof system for $L_{\mathsf{Lin}}[\mathsf{pp}]$ with respect to transformation $\mathsf{Lin.Transform}$. $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.Maul})$ is as follows:

$\mathsf{Bit.Prove}(\mathsf{pp}, \vec{c}, o)$: Let $\mathsf{pp} = [p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, u, v, w]$. Parse $\vec{c} = (c_1,c_2,c_3)$ and $o = (r,s)$. Compute $\delta_{\mathsf{pp}}(\vec{c}) = (c_1/u, c_2/v, (c_3/w)^{-1})$. Output $\pi_{\mathsf{bit}} \leftarrow \mathsf{Lin.Prove}\big(\mathsf{pp}, \big(\vec{c}, \delta(\vec{c})\big), (r,s,(r+s))\big)$.

$\mathsf{Bit.Verify}(\mathsf{pp}, \vec{c}, \pi_{\mathsf{bit}})$: Output $\mathsf{Lin.Verify}\big(\mathsf{pp}, \big(\vec{c}, \delta(\vec{c})\big), \pi_{\mathsf{bit}}\big)\big)$.

$\mathsf{Bit.Maul}(\mathsf{pp}, \vec{c}, o', \pi_{\mathsf{bit}})$: Parse $o' = (r',s')$. Output $\mathsf{Lin.Maul}\big(\mathsf{pp}, \big(\vec{c}, \delta(\vec{c})\big), (r',s'), \pi_{\mathsf{bit}}\big)$.

Gate Proofs. Recall that boolean values b_1, b_2, b_3 satisfy NAND relation that is, $b_3 = b_1 \barwedge b_2$ if and only if $b_1 + b_2 + 2b_3 - 2 \in \{0,1\}$. We use this observation along with the homomorphic properties of the underlying commitment to prove gate consistency for every NAND gate. This approach was also used in GOS [GOS06a].

Definition 22 (NAND Function)**.** *Function f_{NAND} takes as input three commitments $\{\vec{c}_i\}_{i=1}^3$ where $\vec{c}_i = (c_1^i, c_2^i, c_3^i)$ along with (f,h,g), and outputs a homomorphically computed commitment as follows:*

$$f_{\mathsf{NAND}}\big(\{\vec{c}_i\}_{i=1}^3, f, h, g\big) \triangleq (c_1^1 c_1^2 (c_1^3)^2 f^{-2}, c_2^1 c_2^2 (c_2^3)^2 h^{-2}, c_3^1 c_3^2 (c_3^3)^2 g^{-2})$$

We also define function $\mathcal{R}_{\mathsf{NAND}}$ which on input randomness to commitments $\{\vec{c}_i\}_{i=1}^3$, outputs the randomness corresponding to the commitment $f_{\mathsf{NAND}}\big(\{\vec{c}^i\}_{i=1}^3, f, h, g\big)$.

$$\mathcal{R}_{\mathsf{NAND}}(\{(r_i,s_i)\}_{i=1}^3) \triangleq (r_1 + r_2 + 2r_3 - 2, s_1 + s_2 + 2s_3 - 2)$$

Recall the language, parameterized by $\mathsf{pp} = [p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, u, v, w]$:

$$L_{\mathsf{NAND}}[\mathsf{pp}] = \big\{\{\vec{c}_i\}_{i=1}^3 \mid \exists\ (b_1,b_2,b_3) \text{ and } \{(r_i,s_i)\}_{i=1}^3 \text{ s.t.}$$

$$((c_1^i, c_2^i, c_3^i) = (u^{b_i} f^{r_i}, v^{b_i} h^{s_i}, w^{b_i} g^{r_i+s_i})) \ \wedge\ (b_3 = b_1 \barwedge b_2)\big\}$$

The NIWI proof construction for $L_{\mathsf{NAND}}[\mathsf{pp}]$ is as follows:

$\mathsf{NAND.Prove}(\mathsf{pp}, \{\vec{c}_i\}_{i=1}^3, \{b_i, o_i\}_{i=1}^3)$: Compute a commitment to $b = (b_1 + b_2 + 2b_3 - 2)$ homomorphically. Namely, compute $\vec{d} = f_{\mathsf{NAND}}\big(\{(c_1^i, c_2^i, c_3^i)\}_{i=1}^3, f, h, g\big)$. Note that $(d_1, d_2, d_3) \in L_{\mathsf{Com}}$ with witness $b = (b_1 + b_2 + 2b_3 - 2)$ and $(r, s) = \mathcal{R}_{\mathsf{NAND}}(\{(r_i, s_i)\}_{i=1}^3)$ where $o_i = (r_i, s_i)$. Output L_{Com} proof given by:

$$\Pi_{\mathsf{NAND}} = \mathsf{Bit.Prove}\big(\mathsf{pp}, \vec{d}, \delta(\vec{d}), (r, s, (r+s))\big).$$

$\mathsf{NAND.Verify}(\mathsf{pp}, \{\vec{c}_i\}_{i=1}^3, \Pi_{\mathsf{NAND}})$: Compute $\vec{d} = f_{\mathsf{NAND}}(\{\vec{c}_i\}_{i=1}^3, f, h, g)$ and $\vec{d}' = \delta(\vec{d})$. Output $\mathsf{Bit.Verify}(\mathsf{pp}, \vec{d}, \vec{d}', \Pi_{\mathsf{NAND}})$.

$\mathsf{NAND.Maul}(\mathsf{pp}, \{\vec{c}_i, o_i\}_{i=1}^3, \pi_{\mathsf{NAND}})$: Parse $o_i' = (r_i', s_i')$ for $i \in [3]$ and compute $(r', s') = \mathcal{R}_{\mathsf{NAND}}(\{(r_i', s_i')\}_{i=1}^3)$. Output $\mathsf{Lin.Maul}\big(\mathsf{pp}, (\vec{d}, \vec{d}'), (r', s'), \pi_{\mathsf{NAND}})\big)$.

We again note that $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify})$ and $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify})$ are taken verbatim from GOS [GOS06a]. In this work, we show that both the proof systems are malleable with respect to $\mathsf{Bit.T}$ and $\mathsf{NAND.T}$ respectively. For both the proof systems, malleability follows from malleability of $L_{\mathsf{Lin}}[\mathsf{pp}]$ as per Proposition 3.

Chapter 4

Fully Homomorphic NIWI Proofs

In this chapter, we construct a fully homomorphic (FH) NIWI proof system. We start with an overview of the construction in Section 4.1. In Section 4.2, we describe our main ingredient: a malleable proof system (with additional properties) for proving that two commitments with respect to different parameters commit to the same value. We defer the construction of this malleable proof system to Section 4.3.1. In Section 4.3, we describe our construction for FH NIWI and prove security.

4.1 Overview: Fully Homomorphic NIWI

We now focus on our construction of a FH NIWI proof system for $L_{\mathcal{U}}$. As we will see, this is a significantly harder task compared to the FH NIZK, since NIWI is constructed in the plain model without a CRS.

The GOS NIWI Construction. We will first describe the GOS NIWI proof system. Recall that in the GOS NIZK construction, the CRS consists of the parameters pp of the commitment scheme. In a NIWI construction, there is no CRS. In the GOS NIWI, the prover chooses two parameters $(\mathsf{pp}^0, \mathsf{pp}^1)$ such that it is possible to publicly verify that one of them is binding. The NIWI proof for $(C, \mathsf{out}) \in L_{\mathcal{U}}$ is of the form $(\mathsf{pp}^0, \Pi^0, \mathsf{pp}^1, \Pi^1)$ where Π^b is the NIZK proof with respect to pp^b for each $b \in \{0,1\}$.

Towards Homomorphic Evaluation and Unlinkability. It is not clear how to use the GOS NIWI construction to construct an FH NIWI. In particular, achieving unlinkability here is significantly harder. Intuitively, the difficulty stems from the fact that even though the GOS NIWI appears to be gate-by-gate, there is an overarching pair of parameters associated with the entire proof, and this pair is different for different proofs.

In more detail, a fresh GOS NIWI proof as described above has two parameters $(\mathsf{pp}^0, \mathsf{pp}^1)$ associated with it. Thus, if we use an approach similar to the FH NIZK construction for composing proofs, namely if we prove that $(D(C_1, \ldots, C_k), b^*) \in L_{\mathcal{U}}$, given k instances $\{z_i = (C_i, b_i)\}_{i \in [k]}$ along with corresponding proofs $\{\Pi_i\}_{i \in [k]}$, where $b^* = D(b_1, \ldots, b_k)$, then the resulting composed proof will have $2k$ parameters associated with it. It is unclear how to randomize such a composed proof to look like a fresh proof which has only two

parameters associated with it.

In order to achieve unlinkability in our construction, we diverge from the GOS construction. Rather than choosing a pair of parameters per proof, we choose a fresh pair of parameters $(\mathsf{pp}_j^0, \mathsf{pp}_j^1)$ for each gate of the circuit. As in the GOS construction, the honest prover chooses one of them to be binding and the other hiding such that one can publicly verify that indeed one of the parameters is binding. Recall that in the GOS NIWI construction, the prover committed to each wire value with respect to two parameters $(\mathsf{pp}^0, \mathsf{pp}^1)$. Now that we are choosing fresh parameters per gate, the question is which parameters do we use to commit to a wire value?

We associate four parameters $\mathsf{pp}_i^0, \mathsf{pp}_i^1, \mathsf{pp}_j^0, \mathsf{pp}_j^1$ with an internal wire between the i^{th} and the j^{th} gate in the circuit. In our construction, we commit to the wire value with respect to all of these parameters and thus, have four commitments $\vec{c}_i^0, \vec{c}_i^1, \vec{c}_j^0, \vec{c}_j^1$ per wire. We compute Bit Proofs with respect to each of the four commitments, and compute Gate Proofs for every gate with respect to both parameters associated with that gate.

Ensuring Soundness. Recall that the GOS NIWI consists of two independent NIZK proofs Π^0, Π^1 with respect to parameters $\mathsf{pp}^0, \mathsf{pp}^1$ respectively. Thus, the commitments, Bit Proofs and Gate Proofs with respect to both the parameters are independent of each other, and Π^0, Π^1 are verified separately. This is not the case in our setting.

Our proof contains a pair of parameters per gate, and has four commitments per wire. Thus, we need to prove that the multiple commitments per wire commit to the same value. In particular for soundness, it is sufficient to prove that among the four commitments per wire, the two commitments corresponding to the two binding parameters commit to the same value.

However the verifier does not know which of the four parameters $\mathsf{pp}_i^0, \mathsf{pp}_i^1, \mathsf{pp}_j^0, \mathsf{pp}_j^1$ are binding. All we are guaranteed is that for every gate j, one of $(\mathsf{pp}_j^0, \mathsf{pp}_j^1)$ is binding. So in our construction, we give four pairwise proofs that *each* commitment with respect to gate i commits to the same value as *each* commitment with respect to gate j. Namely, for all $b_1, b_2 \in \{0,1\}$, the commitments $(\vec{c}_i^{b_1}, \vec{c}_j^{b_2})$ with respect to $\mathsf{pp}_i^{b_1}, \mathsf{pp}_j^{b_2}$ commit to the same value. This ensures consistency with respect to the two binding commitments across gates i, j. This, along with the Bit and Gate proofs will ensure that there is a consistent boolean assignment $w_1, \ldots, w_n$ induced by the witness $\vec{w}$ across all the wires of the circuit, such that $C(\vec{w}) = \mathsf{out}$.

We emphasize that we do not provide consistency proofs between the two commitments $(\vec{c}_i^0, \vec{c}_i^1)$ for a gate i, and in fact this is crucial for achieving witness indistinguishability, as we explain later. Towards constructing such pairwise proofs, we define the language L_{TC} [*] which consists of instances of the form $(\vec{c}_i, \vec{c}_j, \mathsf{pp}_i, \mathsf{pp}_j)$ where commitment $\vec{c}_i$ with respect to parameters pp_i and $\vec{c}_j$ with respect to pp_j commit to the same bit. See Section 4.2 for a detailed description of the language.

Arguing Witness Indistinguishability

The main challenge is to prove that the final construction is witness indistinguishable even given the additional L_{TC} proofs for instances of the form $(\vec{c}_i, \vec{c}_j, \mathsf{pp}_i, \mathsf{pp}_j)$. We note that even if the proof system for L_{TC} satisfies

[*] TC stands for the language of Two Commitments.

WI, we do not know how to argue that the final construction is WI. Intuitively, the issue is that an L_{TC} statement may have a unique witness, in which case WI offers no secrecy. As we explain below, we need our L_{TC} proof system to have a secrecy guarantee of the flavor of strong NIWI (with respect to specific distributions).

To argue WI of our final FH NIWI construction, we prove that a proof Π_0 for $(C,\mathsf{out}) \in L_{\mathcal{U}}$ with respect to witness wit_0 is indistinguishable from a proof Π_1 with respect to witness wit_1. Let us zoom in on a wire k between gates i,j whose value changes from 0 (for wit_0) to 1 (for wit_1). Both Π_0,Π_1 will contain four commitments to the wire k with respect to parameters $\mathsf{pp}_i^0,\mathsf{pp}_i^1,\mathsf{pp}_j^0,\mathsf{pp}_j^1$, along with the four L_{TC} proofs (see Figure 3).

Denote by $\mathsf{PP} = (\mathsf{pp}_i^0,\mathsf{pp}_i^1,\mathsf{pp}_j^0,\mathsf{pp}_j^1)$. Denote by $\mathsf{W}(b)$ the four commitments to bit b on wire k, that is $\mathsf{W}(b) = (\vec{c}_i^0,\vec{c}_i^1,\vec{c}_j^0,\vec{c}_j^1)$ where all the four commitments are to the bit b. Denote by $\tilde{\Pi}(b) = (\pi^{00},\pi^{01},\pi^{10},\pi^{11})$ where for all $b_1,b_2 \in \{0,1\}$, $\pi^{b_1b_2}$ is a proof for $(\vec{c}_i^{b_1},\vec{c}_j^{b_2},\mathsf{pp}_i^{b_1},\mathsf{pp}_j^{b_2}) \in L_{\mathsf{TC}}$.

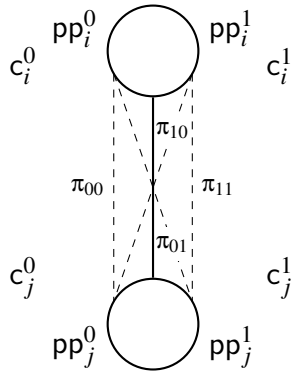

Figure 4.1: *Zooming in on wire k of circuit C with parameters* $\mathsf{PP} = (\mathsf{pp}_i^0,\mathsf{pp}_i^1,\mathsf{pp}_j^0,\mathsf{pp}_j^1)$*, commitments* $\mathsf{W} = (\vec{c}_i^0,\vec{c}_i^1,\vec{c}_j^0,\vec{c}_j^1)$ *and* L_{TC} *proofs* $\tilde{\Pi} = (\pi^{00},\pi^{01},\pi^{10},\pi^{11})$.

To prove WI of the final construction, in particular the following should hold:

$$\left(\mathsf{PP},\mathsf{W}(0),\tilde{\Pi}(0)\right) \approx \left(\mathsf{PP},\mathsf{W}(1),\tilde{\Pi}(1)\right) \tag{4.1}$$

This indistinguishability requirement already implies a strong NIWI for L_{TC}, with respect to distributions $\mathcal{D}_0$ and $\mathcal{D}_1$, where $\mathcal{D}_b$ samples L_{TC} instances $(\vec{c}_i,\vec{c}_j,\mathsf{pp}_i,\mathsf{pp}_j)$ such that $\vec{c}_i,\vec{c}_j$ commit to the bit b.

For our analysis, Equation (4.1) is insufficient since we need Equation (4.1) to hold even given the rest of the proof for $(C,\mathsf{out}) \in L_{\mathcal{U}}$. In other words, we need Equation (4.1) to hold given some auxiliary information aux, where given aux it should be possible to efficiently compute the rest of the proof from it. One possible aux is the openings of all the four commitments so that it is then possible to compute Bit and Gate Proofs for the rest of the proof. But if we give the openings with respect to 0 and 1 respectively, then the two distributions in Equation (4.1) are clearly distinguishable.

So the question is, what aux can we give? Our key insight is that we can give equivocated openings for the commitments with respect to the two hiding parameters and honest openings with respect to the binding

parameters, so that in both the distributions in Equation (4.1), two of the openings are to 0 and two of them are to 1. Without loss of generality, we think of $\mathsf{pp}_i^0, \mathsf{pp}_j^0$ as the binding parameters and $\mathsf{pp}_i^1, \mathsf{pp}_j^1$ as the hiding parameters. We strengthen the requirement in Equation (4.1) as follows:

$$\big(\mathsf{PP}(0), \mathsf{W}(0), \tilde{\Pi}(0), \mathsf{O}(0)\big) \approx \big(\mathsf{PP}(1), \mathsf{W}(1), \tilde{\Pi}(1), \mathsf{O}(1)\big) \tag{4.2}$$

where $\mathsf{PP}(b) = (\mathsf{pp}_i^b, \mathsf{pp}_i^{1-b}, \mathsf{pp}_j^b, \mathsf{pp}_j^{1-b})$, and $\mathsf{W}(b), \tilde{\Pi}(b)$ are as before, and where in both the distributions, $\mathsf{O}(b)$ contains openings for the commitments $\mathsf{W}(b)$ to $(0,1,0,1)$ respectively. This is the case since in the left-hand-side parameters $\mathsf{PP}(0)$, the second and fourth parameters are hiding, and we equivocate $\vec{c}_i^1, \vec{c}_j^1$ to open to 1, whereas in the right-hand-side parameters $\mathsf{PP}(1)$, the first and third parameters are hiding, and we equivocate $\vec{c}_i^1, \vec{c}_j^1$ to open to 0. Note that the L_{TC} proofs in $\tilde{\Pi}(b)$ are still computed using the (honest) openings to b.

This is still not sufficient for our WI analysis. In order to argue WI of the final construction, we need to invoke Equation (4.2) for every wire k in the circuit for which the value of wit_0 on wire k is different from value of wit_1 on wire k. These invocations are not completely independent since two different wires may be associated with the same gate, and in particular the two wires may be associated with an overlapping set of parameters. Thus, we need to further strengthen Equation (4.2) to as follows:

$$\big(\mathsf{PP}(0), \mathsf{W}(0), \tilde{\Pi}(0), \mathsf{O}(0), \mathsf{W}(1), \tilde{\Pi}(1), \mathsf{O}(1)\big) \approx \big(\mathsf{PP}(1), \mathsf{W}(1), \tilde{\Pi}(1), \mathsf{O}(1), \mathsf{W}(0), \tilde{\Pi}(0), \mathsf{O}(0)\big) \tag{4.3}$$

where $\mathsf{PP}(b), \mathsf{W}(b), \tilde{\Pi}(b)$ and $\mathsf{O}(b)$ are as described above. We note that in the left-hand-side, $\mathsf{W}(1)$ are four commitments to 1 with respect to $\mathsf{PP}(0)$, $\tilde{\Pi}(1)$ are the corresponding L_{TC} proofs computed using the honest openings to 1, and $\mathsf{O}(1)$ are the openings to $(1,0,1,0)$ respectively. Similarly, in the right-hand-side, $\mathsf{W}(0)$ are four commitments to 0 with respect to $\mathsf{PP}(1)$, $\tilde{\Pi}(0)$ are the corresponding L_{TC} proofs, and again $\mathsf{O}(0)$ are the openings to $(1,0,1,0)$ respectively. We refer to the property from Equation (4.3) as *Strong Secrecy* of L_{TC} and describe it in detail in Section 4.2. The Strong Secrecy requirement of L_{TC} as in Equation (4.3) is sufficient for our WI analysis. Before explaining our WI analysis, we describe the ingredients for our FH NIWI Construction.

Recall that our NIWI proof for $(C, \mathsf{out}) \in L_{\mathcal{U}}$ is computed as follows: Choose a fresh pair of parameters per gate, commit to all the wire values with respect to all the associated parameters (2 commitments per input wire, 4 commitments per connecting wire), compute Bit Proofs (one per commitment), compute Gate Proofs (two per gate) and compute L_{TC} proofs (four per connecting wire). In order to randomize our NIWI proof, we randomize all the parameters, correspondingly update the commitments and update the proofs to be with respect to the randomized parameters and commitments. Specifically, we need the following ingredients for our final FH NIWI Construction.

Ingredients for our FH NIWI.

- A Commitment Scheme as required in the FH NIZK construction, but with the additional feature that allows for randomizing the parameters and updating the commitments to be with respect to the randomized parameters, so that the randomized parameters and commitments are distributed like fresh commitments.

– Bit Proofs and Gate Proofs as required in the FH NIZK construction, but with the following (modified) malleability property: Given a proof for commitments with respect to some pp, it is possible to efficiently randomize the parameters, correspondingly update the commitments and update the proofs to be with respect to the new parameters and commitments, such that they are all distributed like fresh ones. As in the FH NIZK, we require the Bit and Gate Proofs to satisfy WI.

– A proof system for L_{TC} with the same malleability property as Bit and Gate Proofs, and with the Strong Secrecy property as described in Equation (4.3).

We show (in Section 3.3.1) that the GOS commitment scheme $(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$ satisfies the additional feature that we require. The malleability of Bit Proofs and Gate Proofs can be reduced to the malleability of the NP language L_{Lin} described previously (similar to the FH NIZK construction). We describe the corresponding proof systems $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.GenMaul})$ and $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify}, \mathsf{NAND.GenMaul})$ in Section 4.2.

Jumping ahead, we construct the proof system for L_{TC} also using the proof system for L_{Lin}, and the malleability of L_{TC} follows from the malleability of L_{Lin}. We then argue that the Strong Secrecy follows from our new *DLIN with Leakage* assumption (see Section 4.1 for an overview and Section 4.3.1 for the details).

WI Analysis. To explain our WI analysis, we describe an algorithm $\mathsf{ProofGen}$ that on input a sample from the left-hand-side distribution in Equation (4.3), generates an entire proof Π for $(C, \mathsf{out}) \in L_{\mathcal{U}}$ which is indistinguishable from an honest proof generated using wit_0, and on input a sample from the right-hand-side distribution, $\mathsf{ProofGen}$ generates a proof Π which is indistinguishable from an honest proof generated using wit_1.

$\mathsf{ProofGen}$ *Algorithm.* Without loss of generality, we assume that every circuit is layered; that is, all the gates of the circuit can be arranged in t layers so that for all $i \in [t]$, all the output wires of gates from layer i are input wires to gates in layer $i+1$. Fix any two witnesses wit_0 and wit_1 for $(C, \mathsf{out}) \in L_{\mathcal{U}}$.

On input $\big(\mathsf{PP}(b), \mathsf{W}(b), \tilde{\Pi}(b), \mathsf{O}(b), \mathsf{W}(1-b), \tilde{\Pi}(1-b), \mathsf{O}(1-b)\big)$, $\mathsf{ProofGen}$ does the following:

1. Recall that $\mathsf{PP}(b) = (\mathsf{pp}_i^b, \mathsf{pp}_i^{1-b}, \mathsf{pp}_j^b, \mathsf{pp}_j^{1-b})$. Assign parameters $(\mathsf{pp}_i^b, \mathsf{pp}_i^{1-b})$ to all the odd layer gates of the circuit and $(\mathsf{pp}_j^b, \mathsf{pp}_j^{1-b})$ to all the even layer gates of the circuit. We will refer to $\{\mathsf{pp}_i^b, \mathsf{pp}_j^b\}$ as the *Left Parameters* and $\{\mathsf{pp}_i^{1-b}, \mathsf{pp}_j^{1-b}\}$ as the *Right Parameters*.

2. For all the input wires of the circuit C, commit to wit_0 with respect to pp_i^b (Left Parameter) and commit to wit_1 with respect to pp_i^{1-b} (Right Parameter).

3. For every wire k, produce the 4 commitments and 4 L_{TC} proofs for the wire as follows: Denote by $w_{k,0}$ the value induced by wit_0 on wire k, and denote by $w_{k,1}$ the value induced by wit_1 on wire k in the circuit.

   – If $w_{k,0} = w_{k,1}$ then compute the commitments and L_{TC} proofs honestly.

- If $w_{k,0} = 0$ and $w_{k,1} = 1$ then use $\mathsf{W}(b)$ as the commitments and $\tilde{\Pi}(b)$ as the L_{TC} proofs.
- If $w_{k,0} = 1$ and $w_{k,1} = 0$ then use $\mathsf{W}(1-b)$ as the commitments and $\tilde{\Pi}(1-b)$ as the L_{TC} proofs.

4. Compute the Bit Proofs and Gate Proofs honestly: We have the openings for all the commitments to the input bits (from Step 2). We also have the openings for the commitments to every non-input wire k, namely $\mathsf{O}(b)$ for $\mathsf{W}(b)$ when $w_{k,0} = 0$ and $w_{k,1} = 1$, or $\mathsf{O}(1-b)$ for $\mathsf{W}(1-b)$ when $w_{k,0} = 1$ and $w_{k,1} = 0$, or since we generated the commitments honestly when $w_{k,0} = w_{k,1}$. Note that the openings with respect to the Left Parameters always correspond to wit_0 and the openings with respect to the Right Parameters always correspond to wit_1.
   - Bit Proofs can be computed honestly since all the openings are to 0 or 1.
   - Gate Proofs can be computed honestly since all the openings with respect to the Left Parameters are consistent with wit_0 and all the openings with respect to the Right Parameters are consistent with wit_1.
5. Randomize the entire proof as follows:
   - For every gate, randomize the pair of parameters for that gate.
   - Update all the commitments (2 commitments per input wire, 4 commitments per connecting wire) to be with respect to the randomized parameters.
   - Maul all the Bit Proofs (one per commitment), all the Gate Proofs (two per gate) and all the L_{TC} proofs (four for every connecting wire) to be with respect to the updated parameters and commitments.

   Finally output this randomized proof.

So far, we described the $\mathsf{ProofGen}$ algorithm that given a sample from the distributions in Equation (4.3), generates an entire proof for $(C, \mathsf{out}) \in L_{\mathcal{U}}$. Let Π^0_{Gen} be a proof output by $\mathsf{ProofGen}$ on input a sample from the left-hand-side of Equation (4.3) and let Π^1_{Gen} be a proof output by $\mathsf{ProofGen}$ on input a sample from the right-hand-side of Equation (4.3).

From Equation (4.3), it follows that $\Pi^0_{\mathsf{Gen}} \approx \Pi^1_{\mathsf{Gen}}$. All that remains is to argue that $\Pi_0 \approx \Pi^0_{\mathsf{Gen}}$ and $\Pi_1 \approx \Pi^1_{\mathsf{Gen}}$, where Π_b is an honestly computed proof for $(C, \mathsf{out}) \in L_{\mathcal{U}}$ using witness wit_b. Note that Π_0 and Π^0_{Gen} are identical except that Π^0_{Gen} uses equivocated openings to wit_1 on the Right Parameters to compute the Bit and Gate Proofs. Hence, $\Pi_0 \approx \Pi^0_{\mathsf{Gen}}$ follows from WI of the Bit and Gate Proofs, and in addition follows by the randomizability of the commitment scheme and the malleability of the underlying proofs. By a similar argument, $\Pi_1 \approx \Pi^1_{\mathsf{Gen}}$. Thus, WI of the final construction follows form the Strong Secrecy of L_{TC}.

Constructing the L_{TC} Proof System

We construct a proof system for L_{TC} with the following properties:

1. Strong Secrecy: As defined in Equation (4.3).

2. Malleability: Given a proof π for $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \in L_{\mathsf{TC}}$, one can efficiently randomize the parameters to obtain $\mathsf{pp}'_1, \mathsf{pp}'_2$, update the commitments to obtain $\vec{c}'_1, \vec{c}'_2$ which are with respect to $\mathsf{pp}'_1, \mathsf{pp}'_2$, and then maul π to a proof π' for $(\vec{c}'_1, \vec{c}'_2, \mathsf{pp}'_1, \mathsf{pp}'_2) \in L_{\mathsf{TC}}$ such that $(\vec{c}'_1, \vec{c}'_2, \mathsf{pp}'_1, \mathsf{pp}'_2)$ looks like a fresh instance and π' is distributed like a fresh proof.

3. Soundness: We require that soundness holds for all instances $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2)$ where both $\mathsf{pp}_1, \mathsf{pp}_2$ are binding. As noted above, this is sufficient for the soundness of the final construction.

We construct such a proof system using the malleable NIWI proof system for L_{Lin} described before. Recall that L_{Lin} is a parameterized language with parameters $\mathsf{pp} = (f, h, g)$ where f, h, g are generators of a group $\mathbb{G}$, and it consists of a pair of tuples $(\vec{A}, \vec{B})$ such that one of them is of the form $(f^{a_1}, h^{a_2}, g^{a_3})$ where $a_3 = a_1 + a_2$.

We reduce proving that $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \in L_{\mathsf{TC}}$ to proving that $(\vec{A}, \vec{B}) \in L_{\mathsf{Lin}}$ for some $(\vec{A}, \vec{B})$. However, we only know how to do this reduction for L_{TC} instances $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2)$ for which $\mathsf{pp}_1 = \mathsf{pp}_2$. Therefore, we consider an NP-relation for L_{TC} with an additional witness which lets us convert an instance $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2)$ into an instance $(\vec{c}_*, \vec{c}_2, \mathsf{pp}_2, \mathsf{pp}_2)$. The additional witness for $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2)$ is a hard-to-compute function of the parameters $\mathsf{pp}_1, \mathsf{pp}_2$, and we refer to it as an "intermediate parameter" pp_* of $\mathsf{pp}_1, \mathsf{pp}_2$. Using the intermediate parameter pp_* we can convert the commitment $\vec{c}_1$ with respect to pp_1 into a commitment $\vec{c}_*$ with respect to pp_2.

More specifically in our proof, pp_* helps in converting the commitment $\vec{c}_1$ with respect to parameters pp_1, into a commitment $\vec{c}_*$ (to the same value) with respect to pp_2. Then, we can reduce the instance $(\vec{c}_*, \vec{c}_2, \mathsf{pp}_2, \mathsf{pp}_2) \in L_{\mathsf{TC}}$ to a pair of tuples $(\vec{A}, \vec{B}) \in L_{\mathsf{Lin}}$. The soundness and malleability of the L_{TC} proof system follows from the corresponding properties of L_{Lin} proof system. We refer to Section 4.3.1 for a detailed description of the construction.

Strong Secrecy from DLIN with Leakage. All that remains is to show that the strong secrecy of L_{TC} follows from our new assumption of DLIN with Leakage. We first prove that Strong Secrecy of L_{TC} follows from the fact that the NIWI for L_{Lin} is strong WI with respect to the following distributions $\mathcal{D}_0$ and $\mathcal{D}_1$.

– $\mathcal{D}_0$ generates $(\vec{A}, \vec{B})$ where $\vec{A} = (f^{a_1}, h^{a_2}, g^{a_3})$ for random a_1, a_2, a_3 such that $a_1 + a_2 = a_3$, and $\vec{B} = (f^{a_1}, h^{a_2}, g^{a_3+1})$.

– $\mathcal{D}_1$ generates $(\vec{A}, \vec{B})$ where $\vec{A} = (f^{a_1}, h^{a_2}, g^{a_3-1})$ for random a_1, a_2, a_3 such that $a_1 + a_2 = a_3$, and $\vec{B} = (f^{a_1}, h^{a_2}, g^{a_3})$.

We then prove that the proof system for L_{Lin} is strong WI with respect to $\mathcal{D}_0$ and $\mathcal{D}_1$ under DLIN with Leakage assumption. We refer to Section 4.3.1 for a detailed description of the reduction.

4.2 Ingredients for the FH NIWI Construction

Our* <u>first</u> *ingredient is $(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$, a RaHE-commitment scheme with the additional functionalities $(\mathsf{OutParam}, \mathsf{ValidParam}, \mathsf{RParam}, \mathsf{ChangeCom}, \mathsf{ValidInter}, \mathsf{InterParam})$ as defined in Section 3.3.1.

Our* second *ingredient is a malleable proof system $(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul})$ for the language L_{TC} defined as follows:

$$L_{\mathsf{TC}} = \Big\{ (\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \mid \exists\, (b, \mathsf{pp}_*, o_1, o_2) \text{ s.t.} \\ \{\vec{c}_i = \mathsf{CS.Commit}(\mathsf{pp}_i, b; o_i)\}_{i\in[2]} \ \wedge\ \big(\mathsf{ValidInter}(\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}_*) = 1\big) \Big\}$$

Recall that pp_* is the intermediate parameter between $\mathsf{pp}_1, \mathsf{pp}_2$. It is a hard-to-compute function of the parameters which we require as an additional witness for the language.

The malleability is with respect to the transformation $\mathsf{TC.T} = (\mathsf{TC.Transform}, \mathsf{TC.WitTrans})$. $\mathsf{TC.Transform}$ takes as input an instance $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2)$, randomness $(r^1_{\mathsf{pp}}, r^2_{\mathsf{pp}}, o_1, o_2)$ and outputs transformed instance $(\vec{c}'_1, \vec{c}'_2, \mathsf{pp}'_1, \mathsf{pp}'_2)$.

In detail, $\mathsf{TC.Transform}$ on input $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2)$, does the following:

- Randomize the parameters as follows: For all $i \in [2]$, compute $\mathsf{pp}'_i = \mathsf{RParam}(\mathsf{pp}_i; r^i_{\mathsf{pp}})$.
- Change the commitment $\vec{c}_i$ to be with respect to the new parameters pp'_i, by computing $z_i = \mathsf{ChangeCom}(\mathsf{pp}_i, \vec{c}_i; r^i_{\mathsf{pp}})$ for all $i \in [2]$.
- Randomize the commitments as follows: For all $i \in [2]$, compute $\vec{c}'_i = \mathsf{CS.Rand}(\mathsf{pp}'_i, z_i; o_i)$. Output $(\vec{c}'_1, \vec{c}'_2, \mathsf{pp}'_1, \mathsf{pp}'_2)$.

Correspondingly,

$$\mathsf{TC.WitTrans}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), (b, \mathsf{pp}_*, o_1, o_2), (r^1_{\mathsf{pp}}, r^2_{\mathsf{pp}}, o'_1, o'_2)\big) = (b, \widehat{\mathsf{pp}}, r_1, r_2)$$

where $\widehat{\mathsf{pp}} = \mathsf{InterParam}(\mathsf{pp}_1, \mathsf{pp}_2, r^1_{\mathsf{pp}})$ and where for every $i \in [2]$, $r_i = f_{\mathsf{Com}}(o_i, o'_i)$. Recall that $\mathsf{InterParam}$ and f_{Com} are as per the definition of the RaHE-commitment scheme described in Section 3.3.1.

Let us look at the soundness and secrecy requirements from this proof system. We weaken the soundness requirement of our NIWI proof system and require a stronger secrecy property from the proof system. We now describe both of these properties:

1. *Weak Soundness:* Rather than requiring soundness to hold for every $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \in L_{\mathsf{TC}}$, we only require soundness to hold for all instances for which $\mathsf{pp}_1, \mathsf{pp}_2 \in \mathsf{CS.Setup}(1^\lambda)$ (when both parameters are binding).

   Note that our construction for NIWI proof of L_{TC} achieves standard soundness, however for the FH NIWI construction it suffices for the proof system to have weak soundness.

2. *Strong Secrecy:* We require that the distributions $\mathcal{D}_{\mathsf{Bind}}$ and $\mathcal{D}_{\mathsf{Hide}}$ (described below) are computationally indistinguishable. Both the distributions output two parameters $\mathsf{pp}, \mathsf{pp}'$, four commitments

$\vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1$ where $\vec{c}_0, \vec{c}_1$ are with respect to pp and $\vec{c}'_0, \vec{c}'_1$ are with respect to pp'. The distributions also output openings to the four commitments and two L_{TC} proofs. We now explain in detail.

In the output of $\mathcal{D}_{\mathsf{Bind}}$, pp is the binding parameter and pp' is hiding. The commitments $\vec{c}_0, \vec{c}'_0$ commit to 0 and $\vec{c}_1, \vec{c}'_1$ commit to 1. Honest L_{TC} proofs $\Pi^0_{\mathsf{TC}}, \Pi^1_{\mathsf{TC}}$ are computed with respect to $(\vec{c}_0, \vec{c}'_0, \mathsf{pp}, \mathsf{pp}')$ and $(\vec{c}_1, \vec{c}'_1, \mathsf{pp}, \mathsf{pp}')$ respectively. Finally, the commitments $\vec{c}'_0, \vec{c}'_1$ (with respect to the hiding parameter pp') are equivocated to obtain openings to the compliment bit and $\mathcal{D}_{\mathsf{Bind}}$ outputs honest openings o_0, o_1 for $\vec{c}_0, \vec{c}_1$ (openings to 0, 1 respectively) along with equivocated openings o'_0, o'_1 for $\vec{c}'_0, \vec{c}'_1$ (openings to 1, 0 respectively).

In the output of $\mathcal{D}_{\mathsf{Hide}}$, pp is the hiding parameter and pp' is binding. The commitments $\vec{c}_0, \vec{c}'_0$ now commit to 1 and $\vec{c}_1, \vec{c}'_1$ commit to 0. Honest L_{TC} proofs $\Pi^0_{\mathsf{TC}}, \Pi^1_{\mathsf{TC}}$ are computed with respect to $(\vec{c}_0, \vec{c}'_0, \mathsf{pp}, \mathsf{pp}')$ and $(\vec{c}_1, \vec{c}'_1, \mathsf{pp}, \mathsf{pp}')$ respectively. Finally, the commitments $\vec{c}_0, \vec{c}_1$ (with respect to the hiding parameter pp) are equivocated to obtain openings to the compliment bit and $\mathcal{D}_{\mathsf{Hide}}$ outputs equivocated openings o_0, o_1 for $\vec{c}_0, \vec{c}_1$ (openings to 0, 1 respectively) and honest openings o'_0, o'_1 for $\vec{c}'_0, \vec{c}'_1$ (openings to 1, 0 respectively).

Note that in both the distributions, the openings o_0, o'_0, o_1, o'_1 are with respect to the values $0, 1, 1, 0$ respectively. Formally, the distributions are as follows:

- $\mathcal{D}_{\mathsf{Bind}}(1^\lambda)$: Choose r_{pp} at random and compute $\mathsf{pp} = \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}})$. Compute $\mathsf{pp}' = \mathsf{OutParam}(\mathsf{pp})$. For every $d \in \{0,1\}$, do the following:
  - Choose o_d, o''_d at random and compute $\vec{c}_d = \mathsf{CS.Commit}(\mathsf{pp}, d\,; o_d)$, $\vec{c}'_d = \mathsf{CS.Commit}(\mathsf{pp}', d; o''_d)$.
  - Compute $\Pi^d_{\mathsf{TC}} \leftarrow \mathsf{TC.Prove}((\vec{c}_d, \vec{c}'_d, \mathsf{pp}, \mathsf{pp}'), (d, \mathsf{pp}, o_d, o''_d))$.[†]
  - Compute $o'_d = \mathsf{CS.Equivocate}(\mathsf{pp}', r_{\mathsf{pp}}, \vec{c}'_d, o''_d, 1-d)$.

  Output $\left(\mathsf{pp}, \mathsf{pp}', \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi^0_{\mathsf{TC}}, \Pi^1_{\mathsf{TC}}\right)$.

- $\mathcal{D}_{\mathsf{Hide}}(1^\lambda)$: Choose r_{pp} at random and compute $\mathsf{pp} = \mathsf{CS.Setup}'(1^\lambda; r_{\mathsf{pp}})$. Compute $\mathsf{pp}' = \mathsf{OutParam}(\mathsf{pp})$. For every $d \in \{0,1\}$, do the following:
  - Choose o'_d, o''_d at random. Compute $\vec{c}_d = \mathsf{CS.Commit}(\mathsf{pp}, 1-d\,; o''_d)$ and compute $\vec{c}'_d = \mathsf{CS.Commit}(\mathsf{pp}', 1-d; o'_d)$.
  - Compute $\Pi^d_{\mathsf{TC}} \leftarrow \mathsf{TC.Prove}((\vec{c}_d, \vec{c}'_d, \mathsf{pp}, \mathsf{pp}'), (1-d, \mathsf{pp}, o''_d, o'_d))$.
  - Compute $o_d = \mathsf{CS.Equivocate}(\mathsf{pp}, r_{\mathsf{pp}}, \vec{c}_d, o''_d, d)$.

  Output $\left(\mathsf{pp}, \mathsf{pp}', \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi^0_{\mathsf{TC}}, \Pi^1_{\mathsf{TC}}\right)$.

Remark 6. Note that strong secrecy implies plain witness indistinguishability for L_{TC} instances with more than one witnesses. In particular, it implies that the following two distributions are indistinguishable for any $d \in \{0,1\}$:

[†]Recall that for parameters $\mathsf{pp}, \mathsf{pp}'$ such that $\mathsf{pp}' = \mathsf{OutParam}(\mathsf{pp})$, pp itself is an intermediate parameter between $\mathsf{pp}, \mathsf{pp}'$; see Remark 2 (Section 3.3.1).

- E_0^d: Choose r_1, r_2 at random and for all $i \in [2]$, compute $\mathsf{pp}_i = \mathsf{CS.Setup}'(1^\lambda; r_i)$. Compute $\mathsf{pp}_* = \mathsf{InterParam}(\mathsf{pp}_1, \mathsf{pp}_2, r_1)$. Choose o_1, o_2 at random and for all $i \in [2]$, compute $\vec{c}_i = \mathsf{CS.Commit}(\mathsf{pp}_i, d; o_i)$. Finally compute $\pi^0 \leftarrow \mathsf{TC.Prove}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), (d, \mathsf{pp}_*, o_1, o_2)\big)$. Output $\big(\mathsf{pp}_1, \mathsf{pp}_2, \vec{c}_1, \vec{c}_2, \pi^0\big)$.
- E_1^d: Choose r_1, r_2 at random and for all $i \in [2]$, compute $\mathsf{pp}_i = \mathsf{CS.Setup}'(1^\lambda; r_i)$. Compute $\mathsf{pp}_* = \mathsf{InterParam}(\mathsf{pp}_1, \mathsf{pp}_2, r_1)$. Choose o_1, o_2 at random and for all $i \in [2]$, compute $\vec{c}_i = \mathsf{CS.Commit}(\mathsf{pp}_i, d; o_i)$. For all $i \in [2]$, compute $s_i = \mathsf{CS.Equivocate}(\mathsf{pp}_i, r_i, \vec{c}_i, o_i, 1-d)$. Finally compute $\pi^1 \leftarrow \mathsf{TC.Prove}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), (1-d, \mathsf{pp}_*, s_1, s_2)\big)$. Output $\big(\mathsf{pp}_1, \mathsf{pp}_2, \vec{c}_1, \vec{c}_2, \pi^1\big)$.

Additional Procedures. We now describe procedures $(\mathsf{OutCom}, \mathsf{VerCom}, \mathsf{RCom})$ that will help describe the FH NIWI construction succinctly. These procedures use the RaHE-commitment scheme and the algorithms $(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul})$ as subroutines.

Notation: We denote by $(\mathsf{pp}_j^0, \mathsf{pp}_j^1)$ the public parameters associated with gate j where pp_j^0 denotes the binding parameters and pp_j^1 are the hiding parameters. For any $b \in \{0,1\}$, we denote by $(\mathsf{pp}_j^b)'$ the randomized parameters corresponding to pp_j^b.

$\big(\sigma_c, \sigma_\pi, \mathsf{st}\big) \leftarrow \mathsf{OutCom}(\mathsf{pp}_1^0, \mathsf{pp}_2^0, r_{\mathsf{pp}}^1, b)$: The OutCom algorithm takes as input two pairs of parameters $\mathsf{pp}_1^0, \mathsf{pp}_2^0 \in \mathsf{CS.Setup}(1^\lambda)$, randomness r_{pp}^1 such that $\mathsf{pp}_1^0 = \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}}^1)$ and a bit b, and does the following:

- For all $i \in [2]$, compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$. For all $i \in [2]$, $d \in \{0,1\}$, choose at random o_i^d and compute $\vec{c}_i^d = \mathsf{CS.Commit}(\mathsf{pp}_i^d, b; o_i^d)$. Denote by $\sigma_c = (\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$ and $\mathsf{st} = (o_1^0, o_2^0, o_1^1, o_2^1)$.
- Compute $\mathsf{pp}_*^0 = \mathsf{InterParam}(\mathsf{pp}_1^0, \mathsf{pp}_2^0, r_{\mathsf{pp}}^1)$ and $\mathsf{pp}_*^1 = \mathsf{InterParam}(\mathsf{pp}_1^1, \mathsf{pp}_2^1, r_{\mathsf{pp}}^1)$. For all $b_1, b_2 \in \{0,1\}$, compute

$$\pi^{b_1 b_2} \leftarrow \mathsf{TC.Prove}\big((\vec{c}_1^{b_1}, \vec{c}_2^{b_2}, \mathsf{pp}_1^{b_1}, \mathsf{pp}_2^{b_2}), (b, \mathsf{pp}_*^{b_1}, o_1^{b_1}, o_2^{b_2})\big).$$

  Denote by $\sigma_\pi = (\pi^{00}, \pi^{01}, \pi^{10}, \pi^{11})$. Output $(\sigma_c, \sigma_\pi, \mathsf{st})$.

$\{0,1\} \leftarrow \mathsf{VerCom}\big(\mathsf{pp}_1^0, \mathsf{pp}_2^0, \sigma_c, \sigma_\pi\big)$: The verification algorithm takes as input two pairs of parameters, four commitments σ_c and four proofs σ_π, and outputs 1 if and only if all the following checks are successful:

For every $i \in [2]$, compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$. Parse $\sigma_c = (\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$ and $\sigma_\pi = (\pi^{00}, \pi^{01}, \pi^{10}, \pi^{11})$. For all $b_1, b_2 \in \{0,1\}$, check that

$$\mathsf{TC.Verify}\big((\vec{c}_1^{b_1}, \vec{c}_2^{b_2}, \mathsf{pp}_1^{b_1}, \mathsf{pp}_2^{b_2}), \pi^{b_1 b_2}\big) = 1.$$

$\big(\sigma_c', \sigma_\pi', \mathsf{st}'\big) \leftarrow \mathsf{RCom}\big(\{\mathsf{pp}_i^0, (\mathsf{pp}_i^0)', r_{\mathsf{pp}}^i\}_{i\in[2]}, \sigma_c, \sigma_\pi\big)$: The RCom algorithm takes as input two parameters $\mathsf{pp}_1^0, \mathsf{pp}_2^0$, randomized parameters $(\mathsf{pp}_1^0)', (\mathsf{pp}_2^0)'$, randomness $r_{\mathsf{pp}}^1, r_{\mathsf{pp}}^2$ used in randomizing the parameters $\mathsf{pp}_1^0, \mathsf{pp}_2^0$, commitments σ_c and proofs σ_π, and does the following:

- For all $i \in [2]$, check that $(\mathsf{pp}_i^0)' = \mathsf{RParam}(\mathsf{pp}_i^0; r_{\mathsf{pp}}^i)$ and compute $(\mathsf{pp}_i^1)' = \mathsf{OutParam}(\ (\mathsf{pp}_i^0)'\ )$.

- Parse $\sigma_c = (\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$. For all $i \in [2]$ and $d \in \{0,1\}$, compute $z_i^d = \mathsf{ChangeCom}(\mathsf{pp}_i^d, \vec{c}_i^d, r_{\mathsf{pp}}^i)$, choose randomness o_i^d and compute $(\vec{c}_i^d)' = \mathsf{CS.Rand}(\mathsf{pp}_i^{d'}, z_i^d; o_i^d)$. Denote by $\sigma_c' = \big((\vec{c}_1^0)', (\vec{c}_2^0)', (\vec{c}_1^1)', (\vec{c}_2^1)'\big)$ and $\mathsf{st}' = (o_1^0, o_2^0, o_1^1, o_2^1)$.
- Parse $\sigma_\pi = (\pi^{00}, \pi^{01}, \pi^{10}, \pi^{11})$. For all $b_1, b_2 \in \{0,1\}$, compute

$$(\pi^{b_1 b_2})' \leftarrow \mathsf{TC.Maul}\big((\vec{c}_1^{b_1}, \vec{c}_2^{b_2}, \mathsf{pp}_1^{b_1}, \mathsf{pp}_2^{b_2}), (r_{\mathsf{pp}}^1, r_{\mathsf{pp}}^2, o_1^{b_1}, o_2^{b_2}), \pi^{b_1 b_2}\big).$$

  Denote by $\sigma_\pi' = \big((\pi^{00})', (\pi^{01})', (\pi^{10})', (\pi^{11})'\big)$. Output $(\sigma_c', \sigma_\pi', \mathsf{st}')$.

In Section 4.3.1, we construct the proof system $(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul})$, prove weak soundness, and prove the strong secrecy property assuming *DLIN with Leakage* as described in Section 3.3.3.

The <u>third</u> ingredient in our FH NIWI construction is a malleable NIWI proof system $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.GenMaul})$ for $L_{\mathsf{Com}}[\mathsf{pp}]$. Recall that $L_{\mathsf{Com}}[\mathsf{pp}]$ is parameterized by $\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda)$ and is defined as follows:

$$L_{\mathsf{Com}}[\mathsf{pp}] = \{\vec{c} \mid \exists\ (b, o) \text{ s.t. } \vec{c} = \mathsf{CS.Commit}(\mathsf{pp}, b; o)\ \wedge\ b \in \{0,1\}\}$$

In Section 3.4.1, we described a malleable NIWI $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.Maul})$ for $L_{\mathsf{Com}}[\mathsf{pp}]$ with respect to transformation $\mathsf{Bit.T}$. For the FH NIWI construction, we need the malleability to be with respect to a more general transformation $\mathsf{Bit.GenT} = (\mathsf{Bit.GenTrans}, \mathsf{Bit.GWitTrans})$. The transformation $\mathsf{Bit.GenTrans}$ randomizes and transforms a commitment $\vec{c} \in L_{\mathsf{Com}}[\mathsf{pp}]$ to a new commitment $\vec{c}' \in L_{\mathsf{Com}}[\mathsf{pp}']$. Formally, $\mathsf{Bit.GenTrans}(\vec{c}; o, r_{\mathsf{pp}})$ works as follows:

1. Compute $\mathsf{pp}' = \mathsf{RParam}(\mathsf{pp}, r_{\mathsf{pp}})$ and output $z = \mathsf{ChangeCom}(\mathsf{pp}', \vec{c}, r_{\mathsf{pp}})$.
2. Compute $\vec{c}' = \mathsf{CS.Rand}(\mathsf{pp}, z; o')$. Output $\vec{c}'$.

Recall that the syntax of the associated mauling algorithm is as follows:

$$\pi_{\mathsf{bit}}' \leftarrow \mathsf{Bit.GenMaul}(\mathsf{pp}, \vec{c}, o', r_{\mathsf{pp}}, \pi_{\mathsf{bit}})$$

The <u>fourth</u> ingredient in our FH NIWI construction is the NIWI proof system $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify}, \mathsf{NAND.GenMaul})$ for $L_{\mathsf{NAND}}[\mathsf{pp}]$. Recall that $L_{\mathsf{NAND}}[\mathsf{pp}]$ is parameterized by $\mathsf{pp} \leftarrow \mathsf{CS.Setup}(1^\lambda)$ and is defined as follows:

$$L_{\mathsf{NAND}}[\mathsf{pp}] = \big\{\{\vec{c}_i\}_{i\in[3]} \mid \exists\ \{b_i, o_i\}_{i\in[3]} \text{ s.t. } \vec{c}_i = \mathsf{CS.Commit}(b_i; o_i)\ \wedge\ (b_3 = b_1\ \bar{\wedge}\ b_2)\ \wedge\ \{b_i \in \{0,1\}\}_{i\in[3]}\big\}$$

In Section 3.4.1, we described a malleable NIWI $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify}, \mathsf{NAND.Maul})$ for $L_{\mathsf{NAND}}[\mathsf{pp}]$ with respect to transformation $\mathsf{NAND.T}$. For the FH NIWI construction, we need the malleability to be with respect to a more general transformation $\mathsf{NAND.GenT} = (\mathsf{NAND.GenTrans}, \mathsf{NAND.GWitTrans})$. The transformation $\mathsf{NAND.GenTrans}(\mathsf{pp}, \{\vec{c}_i\}_{i\in[3]},$
$\{o_i'\}_{i\in[3]}, r_{\mathsf{pp}})$ works as follows:

1. Compute $\mathsf{pp}' = \mathsf{RParam}(\mathsf{pp}, r_{\mathsf{pp}})$.
2. For every $i \in [3]$, output $z_i = \mathsf{ChangeCom}(\mathsf{pp}', \vec{c}_i, r_{\mathsf{pp}})$.
3. Compute $\vec{c}'_i = \mathsf{CS.Rand}(z_i, o'_i)$ for $i \in [3]$. Output $(\vec{c}'_1, \vec{c}'_2, \vec{c}'_3)$

Recall that the syntax of the associated mauling algorithm is as follows:

$$\pi'_{\mathsf{NAND}} \leftarrow \mathsf{NAND.GenMaul}(\mathsf{pp}, \{\vec{c}_i\}_{i\in[3]}, \{o'_i\}_{i\in[3]}, r_{\mathsf{pp}}, \pi_{\mathsf{NAND}})$$

Note that the third and fourth ingredients can be instantiated similar to the instantiations described in Section 3.4.3, again using the malleable NIWI proof system $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify}, \mathsf{Lin.Maul})$ for $L_{\mathsf{Lin}}[\mathsf{pp}]$ with respect to transformation $\mathsf{Lin.T}$.

4.3 FH NIWI Construction

In this section we construct a FH NIWI for the language $L_{\mathcal{U}}$. Recall that

$$L_{\mathcal{U}} = \{(C, \mathsf{out}) \mid \exists\, \vec{w} \text{ such that } C(\vec{w}) = \mathsf{out}\}.$$

We start by defining a *connecting wire* for a circuit C.

Definition 23 (Connecting Wire)**.** A wire k in a circuit C is said to be a *connecting wire* for a pair of NAND gates (i, j) if it is an output wire of gate i and an input wire to gate j.

Without loss of generality, we assume that every circuit C in an $L_{\mathcal{U}}$ instance (C, out) is a layered circuit; namely, the circuit consists of t layers of gates such that any output wire from a gate at layer $i \in [t]$ is an input wire to a gate in layer $i+1$.[‡] We also assume without loss of generality that the circuit consists of NAND gates where each gate has fan-in 2 and fan-out at most 2.

We will use the following ingredients in our FH NIWI construction:

– A RaHE-commitment scheme $(\mathsf{CS.Setup}, \mathsf{CS.Commit}, \mathsf{CS.Rand})$ with the additional functionalities

$$(\mathsf{OutParam}, \mathsf{ValidParam}, \mathsf{RParam}, \mathsf{ChangeCom}, \mathsf{ValidInter}, \mathsf{InterParam})$$

as defined in Section 3.3.1.

– Malleable proof system for L_{TC} with weak soundness and strong secrecy, with respect to the transformation $\mathsf{TC.T} = (\mathsf{TC.Transform}, \mathsf{TC.WitTrans})$ as described in Section 4.2, denoted by

$$(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul}).$$

[‡]Any circuit can be converted into a layered circuit by adding dummy gates.

– Malleable NIWI proof system for $L_{\mathsf{Com}}[\mathsf{pp}]$ with respect to the transformation $\mathsf{Bit.GenT} = (\mathsf{Bit.GenTrans}, \mathsf{Bit.GWitTrans})$ as described in Section 4.2, denoted by

$$(\mathsf{Bit.Prove}, \mathsf{Bit.Verify}, \mathsf{Bit.GenMaul}).$$

– Malleable NIWI proof system for $L_{\mathsf{NAND}}[\mathsf{pp}]$ with respect to the transformation $\mathsf{NAND.GenT} = (\mathsf{NAND.GenTrans}, \mathsf{NAND.GWitTrans})$ as described in Section 4.2, denoted by

$$(\mathsf{NAND.Prove}, \mathsf{NAND.Verify}, \mathsf{NAND.GenMaul}).$$

Theorem 4. *Assuming the existence of the ingredients as described above, the following construction* Π_{FHNIWI} *is a Fully Homomorphic NIWI proof system as per Definition 17.*

We instantiate the first, third and fourth ingredients from DLIN and instantiate the second ingredient from DLIN with Leakage as we describe in Section 4.3.1. This gives the following corollary:

Corollary 1. *Assuming DLIN with Leakage, the following construction* Π_{FHNIWI} *is a Fully Homomorphic NIWI proof system as per Definition 17.*

Construction: We now describe our construction of fully homomorphic NIWI proofs for $L_{\mathcal{U}}$.

$\mathsf{NIWI..Prove}((C, \mathsf{out}), \vec{w})$: For $C : \{0,1\}^n \to \{0,1\}$, denote by m the number of NAND gates, and by ℓ the number of connecting wires. Let $\vec{w} = w_1, \ldots, w_{n+\ell}$ be the values induced by $\vec{w}$ on all the wires excluding the output wire (but including the input wires).

1. For each gate $j \in [m]$, choose randomness r^j_{pp} and compute $\mathsf{pp}^0_j = \mathsf{CS.Setup}(1^\lambda; r^j_{\mathsf{pp}})$ and $\mathsf{pp}^1_j = \mathsf{OutParam}(\mathsf{pp}^1_j)$. Denote by $\overrightarrow{\mathsf{pp}}_j = (\mathsf{pp}^0_j, \mathsf{pp}^1_j)$.

2. For each input wire $k \in [n]$, denote by j the gate for which wire k is an input. For every $b \in \{0,1\}$, choose at random $o^b_{k,j}$ and compute $\vec{c}^b_{k,j} = \mathsf{CS.Commit}(\mathsf{pp}^b_j, w_k; o^b_{k,j})$. Let $\vec{c}_k = (\vec{c}^0_{k,j}, \vec{c}^1_{k,j})$.

   For the output wire wout (which is the output wire of gate m) and for every $b \in \{0,1\}$, let $\vec{c}^b_{\mathsf{wout},m} = \mathbf{1}$ for $\mathsf{out} = 1$ and let $\vec{c}^b_{\mathsf{wout},m} = \mathbf{0}$ for $\mathsf{out} = 0$. Recall that $\mathbf{1}, \mathbf{0}$ are the canonical commitments to 1 and 0 respectively (see Remark 2). Let $\vec{c}_{\mathsf{wout}} = (\vec{c}^0_{\mathsf{wout},m}, \vec{c}^1_{\mathsf{wout},m})$.

3. For each connecting wire $k \in \{n+1, \ldots, n+\ell\}$ that connects gates $i, j \in [m]$ $(i < j)$, compute

   $$\left(\sigma^k_c, \sigma^k_\pi, \mathsf{st}^k\right) \leftarrow \mathsf{OutCom}(\mathsf{pp}^0_i, \mathsf{pp}^0_j, r^j_{\mathsf{pp}}, w_k)$$

   where $\sigma^k_c = (\vec{c}^0_{k,i}, \vec{c}^0_{k,j}, \vec{c}^1_{k,i}, \vec{c}^1_{k,j})$, $\sigma^k_\pi = (\pi^{00}_k, \pi^{01}_k, \pi^{10}_k, \pi^{11}_k)$ and where $\mathsf{st}^k = (o^0_{k,i}, o^0_{k,j}, o^1_{k,i}, o^1_{k,j})$. We will denote $(\sigma^k_c, \sigma^k_\pi)$ by Φ_k.

   Note that a commitment $\vec{c}^b_{k,j}$ commits to w_k with respect to parameters pp^b_j, and where wire k is either an input or output wire for gate j. Also note that there are two commitments for each input wire and four commitments for each connecting wire.

4. Denote by S the set of all pairs (k,j) for $k \in [n+\ell], j \in [m]$, such that wire k is an input or output to gate j. For all $(k,j) \in S$ and for every $b \in \{0,1\}$, generate a proof that the commitment $\vec{c}^b_{k,j}$ commits to a bit. Namely, compute

$$\pi_{\mathsf{bit}}[k,j]^b \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}^b_j, \vec{c}^b_{k,j}, o^b_{k,j})$$

where $o^b_{k,j}$ is the opening for commitment $\vec{c}^b_{k,j}$ as computed in step 2 (for input wires) or as part of st^k output by OutCom (for connecting wires) in step 3. Let $\pi_{\mathsf{bit}}[k,j] = \left(\pi_{\mathsf{bit}}[k,j]^0, \pi_{\mathsf{bit}}[k,j]^1\right)$.

5. For each gate $j \in [m]$, denote by k_1, k_2 the input wires of the gate j and by k_3, k_4 the output wires to gate j. For each $t \in \{3,4\}$ and $b \in \{0,1\}$, compute a gate consistency proof as follows:

$$\pi^{j,b}_{\mathsf{gate}}[t] \leftarrow \mathsf{NAND.Prove}\left(\mathsf{pp}^b_j, \{\vec{c}^b_{k_i,j}\}_{i\in\{1,2,t\}}, \{w^b_{k_i}, o^b_{k_i,j}\}_{i\in\{1,2,t\}}\right)$$

Let $\pi^j_{\mathsf{gate}} = \left(\pi^{j,0}_{\mathsf{gate}}[3], \pi^{j,1}_{\mathsf{gate}}[3], \pi^{j,0}_{\mathsf{gate}}[4], \pi^{j,1}_{\mathsf{gate}}[4]\right)$.

6. Finally output

$$\Pi_{\mathsf{NIWI.}} = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S}, \{\pi^j_{\mathsf{gate}}\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right].$$

$\mathsf{NIWI..Verify}((C, \mathsf{out}), \Pi)$: Parse

$$\Pi_{\mathsf{NIWI.}} = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S}, \{\pi^j_{\mathsf{gate}}\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$$

1. For each $j \in [m]$, parse $\overrightarrow{\mathsf{pp}}_j = (\mathsf{pp}^0_j, \mathsf{pp}^1_j)$ and check that $\mathsf{ValidParam}(\mathsf{pp}^0_j, \mathsf{pp}^1_j) = 1$.
2. For each connecting wire k that connects gates $i, j \in [m]$, check that $\mathsf{VerCom}(\mathsf{pp}^0_i, \mathsf{pp}^0_j, \Phi_k) = 1$.
3. For each $(k,j) \in S$, parse $\pi_{\mathsf{bit}}[k,j] = \left(\pi_{\mathsf{bit}}[k,j]^0, \pi_{\mathsf{bit}}[k,j]^1\right)$ and for every $b \in \{0,1\}$, check that $\mathsf{Bit.Verify}(\mathsf{pp}^b_j, \vec{c}^b_{k,j}, \pi_{\mathsf{bit}}[k,j]^b) = 1$.
4. For each gate $j \in [m]$, parse $\pi^j_{\mathsf{gate}} = \left(\pi^{j,0}_{\mathsf{gate}}[3], \pi^{j,1}_{\mathsf{gate}}[3], \pi^{j,0}_{\mathsf{gate}}[4], \pi^{j,1}_{\mathsf{gate}}[4]\right)$. Denote by k_1, k_2 the input wires to gate j and by k_3, k_4 the output wires of gate j. For each $t \in \{3,4\}$ and $b \in \{0,1\}$, check that $\mathsf{NAND.Verify}(\mathsf{pp}^b_j, \{\vec{c}^b_{k_i,j}\}_{i\in\{1,2,t\}}, \pi^{j,b}_{\mathsf{gate}}[t]) = 1$ where $\pi^j_{\mathsf{gate}} = \left(\pi^{j,0}_{\mathsf{gate}}[3], \pi^{j,1}_{\mathsf{gate}}[3], \pi^{j,0}_{\mathsf{gate}}[4], \pi^{j,1}_{\mathsf{gate}}[4]\right)$.
5. Parse $\vec{c}_{\mathsf{wout}} = (\vec{c}^0_{\mathsf{wout},m}, \vec{c}^1_{\mathsf{wout},m})$ and check that for all $b \in \{0,1\}$, $\vec{c}^b_{\mathsf{wout},m} = \mathbf{1}$ if $\mathsf{out} = 1$ and $\vec{c}^b_{\mathsf{wout},m} = \mathbf{0}$ if $\mathsf{out} = 0$.

$\mathsf{NIWI.Rand}((C,\mathsf{out}),\Pi)$: Parse

$$\Pi_{\mathsf{NIWI.}} = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S}, \{\pi^j_{\mathsf{gate}}\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$$

Randomize the proof Π as follows:

1. For each $j \in [m]$, parse $\overrightarrow{\mathsf{pp}}_j = (\mathsf{pp}^0_j, \mathsf{pp}^1_j)$, choose randomness r^j_{pp} and compute $(\mathsf{pp}^0_j)' \leftarrow \mathsf{RParam}(\mathsf{pp}^0_j; r^j_{\mathsf{pp}})$ and $(\mathsf{pp}^1_j)' = \mathsf{OutParam}(\mathsf{pp}^{0'}_j)$.

2. For each $k \in [n]$, denote by j the gate for which wire k is an input. For each $b \in \{0,1\}$,

   - Compute $z^b_{k,j} = \mathsf{ChangeCom}((\mathsf{pp}^b_j)', \vec{c}^b_{k,j}, r^j_{\mathsf{pp}})$.
   - Choose fresh randomness $o^b_{k,j}$ and compute $(\vec{c}^b_{k,j})' = \mathsf{CS.Rand}(\mathsf{pp}^b_j, z^b_{k,j}; o^b_{k,j})$.

   Let $\vec{c}'_k = \left((\vec{c}^0_{k,j})', (\vec{c}^1_{k,j})'\right)$.

3. For each connecting wire k between gates $i, j \in [m]$, compute

   $$\left((\sigma^k_c)', (\sigma^k_\pi)', \mathsf{st}'_k\right) \leftarrow \mathsf{RCom}(\mathsf{pp}^0_i, (\mathsf{pp}^0_i)', r^i_{\mathsf{pp}}, \mathsf{pp}^0_j, (\mathsf{pp}^0_j)', r^j_{\mathsf{pp}}, \sigma^k_c, \sigma^k_\pi)$$

   Let $\mathsf{st}'_k = (o^0_{k,i}, o^0_{k,j}, o^1_{k,i}, o^1_{k,j})$. Denote by $\phi'_k = \left((\sigma^k_c)', (\sigma^k_\pi)'\right)$.

4. For all $(k,j) \in S$, for every $b \in \{0,1\}$, compute

   $$(\pi_{\mathsf{bit}}[k,j]^b)' \leftarrow \mathsf{Bit.GenMaul}(\mathsf{pp}^b_j, \vec{c}^b_{k,j}, o^b_{k,j}, r^j_{\mathsf{pp}}, \pi_{\mathsf{bit}}[k,j]^b)$$

   where $o^b_{k,j}$ is the randomizing factor for commitment $\vec{c}^b_i$ as computed in the step 2 (for input wires) or as part of st'_k output by RCom (for connecting wires) computed in step 3. Let $\pi_{\mathsf{bit}}[k,j]' = \left[(\pi_{\mathsf{bit}}[k,j]^0)', (\pi_{\mathsf{bit}}[k,j]^1)'\right]$.

5. For each gate $j \in [m]$, let k_1, k_2 be the input wires to gate j, and k_3, k_4 be the output wires. For each $t \in \{3,4\}$ and for every $b \in \{0,1\}$, compute

   $$(\pi^{j,b}_{\mathsf{gate}}[t])' \leftarrow \mathsf{NAND.GenMaul}(\mathsf{pp}^b_j, \{\vec{c}^b_{k_i,j}\}_{i\in\{1,2,t\}}, \{o^b_{k_i,j}\}_{i\in\{1,2,t\}}, r^j_{\mathsf{pp}}, \pi^{j,b}_{\mathsf{gate}}[t])$$

   and where $o^b_{k_i,j}$ is the randomizing factor for commitment $\vec{c}^b_{k_i,j}$ as computed in the step 2 (for input wires) or as part of st'_k output by RCom (for connecting wires) computed in step 3. Let $(\pi^j_{\mathsf{gate}})' = \left((\pi^{j,0}_{\mathsf{gate}}[3])', (\pi^{j,1}_{\mathsf{gate}}[3])', (\pi^{j,0}_{\mathsf{gate}}[4])', (\pi^{j,1}_{\mathsf{gate}}[4])'\right)$

Finally output randomized proof as:

$$\Pi'_{\mathsf{NIWI.}} = \left[\{\overrightarrow{\mathsf{pp}}'_j\}_{j\in[m]}, \{\vec{c}'_i\}_{i\in[n]}, \{\Phi'_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[k,j]'\right\}_{(k,j)\in S}, \left\{(\pi^j_{\mathsf{gate}})'\right\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right].$$

$\mathsf{NIWI..Eval}(\{(C_q, b_q, \Pi_q)\}_{q=1}^{K}, C')$:

1. Check that $\mathsf{Valid}(C') = 1$, else output $\perp$. Recall that $\mathsf{Valid}(C') = 1$ if and only if $C' : \{0,1\}^k \to \{0,1\}$. Compute $(C, \mathsf{out}') = \mathsf{Compose}(\{(C_q, b_q, \Pi_q)\}_{i=q}^{K}, C')$.

2. For each $q \in [K]$, let $\pi^{\mathsf{out}}_{\mathsf{gate},q} \in \Pi_q$ be the gate consistency proofs of the output gate q_{out} of circuit C_q and let $\vec{c}^{\,q}_{\mathsf{wout}} \in \Pi_q$ be the canonical output commitments to output b_q of circuit C_q. Let $\widehat{\Pi_q} = \Pi_q \setminus \{\pi^{\mathsf{out}}_{\mathsf{gate},q}, \vec{c}^{\,q}_{\mathsf{wout}}\}$.

3. Compute a proof $\widehat{\Pi'}$ for C' with witness $(b_1, \ldots, b_K)$ by evaluating $\mathsf{NIWI..Prove}((C', \mathsf{out}'), (b_1, \ldots, b_K))$ (where out' is from Step 1) with the following modification:

   For each $q \in [K]$, denote by q_{out} the output gate of C_q, by W_q the output wire and by q_{in} the gate to which wire W_q is an input in the composed circuit C.

   - For every $q \in [K]$, choose randomness $r_{q_{\mathsf{in}}}$ and compute $\mathsf{pp}^0_{q_{\mathsf{in}}} = \mathsf{CS.Setup}(1^\lambda; r_{q_{\mathsf{in}}})$ and $\mathsf{pp}^1_{q_{\mathsf{in}}} = \mathsf{OutParam}(\mathsf{pp}^0_{q_{\mathsf{in}}})$.
   - Instead of fresh commitment for the wires $W_q \in [K]$, compute

   $$\left(\sigma^q_{\vec{c}}, \sigma^q_{\pi}, \mathsf{st}^q\right) \leftarrow \mathsf{OutCom}(\mathsf{pp}^0_{q_{\mathsf{out}}}, \mathsf{pp}^0_{q_{\mathsf{in}}}, r_{q_{\mathsf{in}}}, b_q)$$

4. For each output gate q_{out} for C_q and for $b \in \{0,1\}$, let $\vec{c}^{\,b}_{1,q_{\mathsf{out}}}, \vec{c}^{\,b}_{2,q_{\mathsf{out}}}$ be the commitments to the input wire values to the gate q_{out}. Let o^b_q be the randomness used to commit to b_q with respect to $\mathsf{pp}^b_{q_{\mathsf{out}}}$, computed as part of st^q in step 3. Recall $\pi^{\mathsf{out}}_{\mathsf{gate},q} = (\pi^{\mathsf{out},0}_{\mathsf{gate},q}, \pi^{\mathsf{out},1}_{\mathsf{gate},q})$ and $\vec{c}^{\,q}_{\mathsf{wout}} = (\vec{c}^{\,0}_{W_q,q_{\mathsf{out}}}, \vec{c}^{\,1}_{W_q,q_{\mathsf{out}}})$. For every $b \in \{0,1\}$, compute

   $$(\pi^{\mathsf{out},b}_{\mathsf{gate},q})' \leftarrow \mathsf{NAND.GenMaul}(\mathsf{pp}^b_{q_{\mathsf{out}}}, (\vec{c}^{\,b}_{1,q_{\mathsf{out}}}, \vec{c}^{\,b}_{2,q_{\mathsf{out}}}, \vec{c}^{\,b}_{W_q,q_{\mathsf{out}}}), (1,1,o^b_q), 1, \pi^{\mathsf{out},b}_{\mathsf{gate},q})$$

5. Note that $\Pi = \left[\widehat{\Pi_1}, \ldots, \widehat{\Pi_n}, \widehat{\Pi'}, \left\{(\pi^{\mathsf{out},b}_{\mathsf{gate},q})'\right\}_{i\in[K], b\in\{0,1\}}\right]$ is a complete proof for (C, out). Randomize the proof by computing $\Pi' \leftarrow \mathsf{NIWI..Rand}((C, \mathsf{out}'), \Pi)$. Finally output (C, out', Π').

Proof of Theorem 4: Completeness follows from the completeness of underlying primitives.

Claim 4. Π_{FHNIWI} *satisfies perfect soundness.*

Proof. Suppose for contradiction, there exists P^* and an infinite set $\Lambda \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$,

$$\Pr[((C, \mathsf{out}), \Pi) \leftarrow P^*(1^\lambda) \ : \ \mathsf{NIWI..Verify}((C, \mathsf{out}), \Pi) = 1 \ \wedge \ (C, \mathsf{out}) \notin L_{\mathcal{U}}] > 0. \tag{4.4}$$

For $C : \{0,1\}^n \to \{0,1\}$, denote by m the number of NAND gates, by ℓ the number of connecting wires, and by S the set of all pairs (k, j) for $k \in [n+\ell]$, $j \in [m]$, such that wire k is an input or output to gate j. Parse $\Pi = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S}, \{\pi^j_{\mathsf{gate}}\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$.

Let E_1 be the event that $\mathsf{NIWI..Verify}((C,\mathsf{out}),\Pi)=1$ where $((C,\mathsf{out}),\Pi)\leftarrow P^*(1^\lambda)$. Let E_2 be the event that $(C,\mathsf{out})\notin L_{\mathcal{U}}$ where $((C,\mathsf{out}),\Pi)\leftarrow P^*(1^\lambda)$.

- E_1 implies that for all $j\in[m]$, $\mathsf{ValidParam}(\overrightarrow{\mathsf{pp}}_j)=1$. Namely, for all $j\in[m]$, there exists $b_j\in\{0,1\}$ such that $\mathsf{pp}_j^{b_j}\in\mathsf{CS.Setup}(1^\lambda)$. Let $\{\widehat{\mathsf{pp}}_j\}_{j\in[m]}$ be the set of all binding parameters such that for each $j\in[m]$, $\widehat{\mathsf{pp}}_j=\mathsf{pp}_j^{b_j}$.

- Let $\vec{w}=(w_1,\ldots,w_{n+\ell})$ be the values committed with respect to $\{\widehat{\mathsf{pp}}_j\}_{j\in[m]}$ on all the wires excluding the output wire (but including the input wires). E_1 implies that for all $(k,j)\in S$, $\mathsf{Bit.Verify}(\widehat{\mathsf{pp}}_j,\vec{c}_{k,j},\pi_{\mathsf{bit}}[k,j])=1$ where $\vec{c}_{k,j}$ is the commitment to wire $k\in[n+\ell]$ with respect to $\widehat{\mathsf{pp}}_j$ and $\pi_{\mathsf{bit}}[k,j]$ is the corresponding $L_{\mathsf{Com}}[\widehat{\mathsf{pp}}_j]$ proof. This along with perfect soundness of $(\mathsf{Bit.Prove},\mathsf{Bit.Verify})$, implies that for all $k\in[n+\ell]$, $w_k\in\{0,1\}$.

- E_1 implies that for any connecting wire $k\in[\ell]$ between gates i,j, $\mathsf{VerCom}\left(\widehat{\mathsf{pp}}_i,\widehat{\mathsf{pp}}_j,\Phi_k\right)=1$. This in turn implies that $\mathsf{TC.Verify}((\widehat{\mathsf{pp}}_i,\widehat{\mathsf{pp}}_j,\vec{c}_{k,i},\vec{c}_{k,j}),\pi_{\mathsf{TC}})$ where $\vec{c}_{k,i},\vec{c}_{k,j}$ are commitments to wire value k under $\widehat{\mathsf{pp}}_i,\widehat{\mathsf{pp}}_j$ respectively and π_{TC} is the corresponding L_{TC} proof. This along with the soundness of $(\mathsf{TC.Prove},\mathsf{TC.Verify})$ implies that $\vec{c}_{k,i},\vec{c}_{k,j}$ commit to the same value w_k.

- For any gate $j\in[m]$, let k_{j_1},k_{j_2} be the input wires and k_{j_3} be the output wire. E_1 implies that for all $j\in[m]$, $\mathsf{NAND.Verify}(\widehat{\mathsf{pp}}_j,\widehat{\mathsf{pp}}_j,\{\vec{c}_{k_{j_i},j}\}_{i\in[3]},\pi^j_{\mathsf{gate}})=1)=1$ where $\{\vec{c}_{k_{j_i},j}\}_{i\in[3]}$ are the commitment to wires k_{j_1},k_{j_2},k_{j_3} with respect to $\widehat{\mathsf{pp}}_j$ and π^j_{gate} is the corresponding $L_{\mathsf{NAND}}[\widehat{\mathsf{pp}}_j]$ proof. This along with perfect soundness of $(\mathsf{NAND.Prove},\mathsf{NAND.Verify})$, implies that for all $j\in[m]$, $w_{k_{j_1}}\barwedge w_{k_{j_2}}=w_{k_{j_3}}$. In particular, $w_{n+\ell-1}\barwedge w_{n+\ell}=\mathsf{out}$ where $w_{n+\ell-1},w_{n+\ell}$ are the values of the two input wires to the output gate of C.

Thus, E_1 implies that $\vec{w}=(w_1,\ldots,w_{n+\ell})$ defines a consistent boolean assignment across the entire circuit C such that $C(\vec{w})=\mathsf{out}$. However E_2 implies that that $C(\vec{w})\neq\mathsf{out}$. Hence we contradict Equation (4.4) which says that $\Pr[E_1\wedge E_2]>0$. □

Claim 5. Π_{FHNIWI} *is a randomizable non-interactive proof system as per Definition 4.*

Proof. We show that for any instance $(C,b)\in L_{\mathcal{U}}$ with witness $w\in\{0,1\}^n$ and any proof Π such that $\mathsf{NIWI..Verify}((C,b),\Pi)=1$, the following distributions are identical:

$$\left\{(C,b),w,\Pi,\vec{R},\Pi_f\right\}\text{ and }\left\{(C,b),w,\Pi,\vec{R},\Pi'\right\}$$

where Π_f is a fresh proof obtained by $\Pi_f\leftarrow\mathsf{NIWI..Prove}((C,b),w)$, Π' is a randomized proof obtained by $\Pi'\leftarrow\mathsf{NIWI..Rand}((C,b),\Pi)$ and $\vec{R}$ is randomness such that $\Pi=\mathsf{NIWI..Prove}((C,b),w;\vec{R})$. Let $\vec{w}=w_1,\ldots,w_{n+\ell}$ be the values induced by $\vec{w}$ on all the wires excluding the output wire (but including the input wires). Parse

$$\Pi=\left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]},\{\vec{c}_k\}_{k\in[n]},\{\Phi_k\}_{k\in[\ell]},\left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S},\{\pi^j_{\mathsf{gate}}\}_{j\in[m]},\vec{c}_{\mathsf{wout}}\right]$$

and parse

$$\vec{R}=\left[\{r^j_{\mathsf{pp}},s_j\}_{j\in[m]},\{S_k\}_{k\in[\ell]},\{o_i,t_i\}_{i\in[n+2\ell]}\right]$$

where $\mathsf{pp}_j^0 = \mathsf{CS.Setup}(1^\lambda; r_{\mathsf{pp}}^j)$, $\mathsf{pp}_j^1 = \mathsf{OutParam}(\mathsf{pp}_j^1)$, $(\phi_k, \mathsf{st}_k) \leftarrow \mathsf{OutCom}(\mathsf{pp}_i^0, \mathsf{pp}_j^0, r_{\mathsf{pp}}^j, w_k; S_k)$ for $k \in \{n+1, \ldots, n+\ell\}$, where $\vec{c}_i = \mathsf{CS.Commit}(\mathsf{pp}_i, w_i; o_i)$ for $i \in [n]$, $\pi_{\mathsf{bit}}^i = \mathsf{Bit.Prove}(\mathsf{pp}, \vec{c}_i, o_i; t_i)$ for $i \in [n+2\ell]$. Finally $\pi_{\mathsf{gate}}^j = \mathsf{NAND.Prove}(\mathsf{pp}_j, \{\vec{c}_{j_i}, o_{j_i}\}_{i\in[3]}; s_j)$ for $j \in [m]$.

Similarly, let $\Pi' = \mathsf{NIWI..Rand}((C,b), \Pi; \vec{R}')$ and parse

$$\vec{R}' = \left[\{r_{\mathsf{pp}}^{j'}, s_j'\}_{j\in[m]}, \{S_k'\}_{k\in[\ell]}, \{o_i', t_i'\}_{i\in[n+2\ell]}\right]$$

where $(\mathsf{pp}_j^0)' = \mathsf{RParam}(\mathsf{pp}_j^0; r_{\mathsf{pp}}^{j'})$, $(\phi_k', \mathsf{st}_k') \leftarrow \mathsf{RCom}(\phi_k; S_k')$, $\vec{c}_i' = \mathsf{CS.Rand}(\mathsf{pp}_i, \vec{c}_i; o_i')$ for $i \in [n+2\ell]$, $\pi_{\mathsf{bit}}^{i'} = \mathsf{Bit.Maul}(\mathsf{pp}, \vec{c}_i, o_i', \pi_{\mathsf{bit}}^i; t_i')$. Finally, $\pi_{\mathsf{gate}}^{j'} = \mathsf{NAND.Maul}(\mathsf{pp}, \{\vec{c}_{j_i}, o'_{j_i}\}_{i\in[3]}, \pi_{\mathsf{NAND}}^j; s_j')$.

By perfect randomizability of $\mathsf{TC.Maul}$, there exists function f_σ given by $(R_i', R_j', S_k'') = f_\sigma(r_{\mathsf{pp}}^i, r_{\mathsf{pp}}^j, r_{\mathsf{pp}}^{i'}, r_{\mathsf{pp}}^{j'}, \mathsf{st}_k')$ such that $(\mathsf{pp}_q^0)' \leftarrow \mathsf{CS.Setup}(1^\lambda; R_q')$ for $q \in \{i, j\}$ and $(\phi_k', \mathsf{st}_k') \leftarrow \mathsf{OutCom}(\mathsf{pp}_i, \mathsf{pp}_i, R_i, b; S_k'')$.

By perfect randomizability of commitment scheme and of underlying proof systems, there exists $o_i'' = f_{\mathsf{Com}}(o_i, o_i')$ such that $\vec{c}_i' = \mathsf{CS.Commit}(\mathsf{pp}, w_i; o_i'')$ and o_i'' is distributed as uniform, there exists $f_{\mathsf{bit}}(t_i', t_i, o_i, o_i') = t_i''$ such that $\pi_{\mathsf{bit}}^{i'} = \mathsf{Bit.Prove}(\mathsf{pp}, \vec{c}_i', o_i''; t_i'')$ and t_i'' is distributed as uniform.

Similarly, $f_{\mathsf{gate}}(s_j', s_j, \{o_{j_i}, o'_{j_i}\}_{i\in[3]}) = s_j'$ and $\pi_{\mathsf{gate}}^{j'} = \mathsf{NAND.Prove}(\mathsf{pp}, \{\vec{c}_{j_i}, o''_{j_i}\}_{i\in[3]}; s_j'')$ where s_j'' is distributed as uniform. We can now identify $\vec{R}''$ such that $\Pi' = \mathsf{NIWI..Prove}((C,b), w; \vec{R}'')$ as follows:

$$\vec{R}'' = \left[\{R_{j'}, s_j''\}_{j\in[m]}, \{S_k''\}_{k\in[\ell]}, \{o_i'', t_i''\}_{i\in[n+2\ell]}\right]$$

which is distributed as uniform. It follows that $\left\{(C,b), w, \Pi, \vec{R}, \Pi_f\right\}$ and $\left\{(C,b), w, \Pi, \vec{R}, \Pi'\right\}$ are identical distributions, where $\Pi_f = \mathsf{NIWI..Prove}((C,b), w; \vec{S})$ and $\Pi' = \mathsf{NIWI..Prove}((C,b), w; \vec{R}'')$ for truly random $\vec{S}, \vec{R}''$.

□

Claim 6. Π_{FHNIWI} *satisfies unlinkability.*

Proof. Follows from the completeness of the underlying primitives and randomizability of the NIWI. □

Claim 7. Π_{FHNIWI} *satisfies witness indistinguishabiilty.*

Proof. Fix any $(C, \mathsf{out}), \mathsf{wit}_0, \mathsf{wit}_1$ such that $((C, \mathsf{out}), \mathsf{wit}_0) \in R_{\mathcal{U}}$ and $((C, \mathsf{out}), \mathsf{wit}_1) \in R_{\mathcal{U}}$. We will prove that $\{\Pi_0\} \approx \{\Pi_1\}$ where $\Pi_b \leftarrow \mathsf{NIWI..Prove}((C, \mathsf{out}), \mathsf{wit}_b)$ for $b \in \{0,1\}$.

For $C : \{0,1\}^n \to \{0,1\}$, denote by m the number of NAND gates, by ℓ the number of connecting wires, and by S the set of all pairs (k, j) for $k \in [n+\ell]$, $j \in [m]$, such that wire k is an input or output to gate j. Let $\vec{w}^b = (w_1^b, \ldots, w_{n+\ell}^b)$ be the values induced by wit_b on all the wires excluding the output wire (but including the input wires). We will proceed through the following hybrids:

Hyb_0**:** Compute $\Pi \leftarrow \mathsf{NIWI..Prove}((C, \mathsf{out}), \mathsf{wit}_0)$. Output proof Π.

Hyb_1: This is exactly as Hyb_0 with the following changes: Instead of choosing fresh $\mathsf{pp}_j \leftarrow \mathsf{CS.Setup}(1^\lambda)$ for each gate $j \in [m]$, compute only two parameters $\mathsf{pp}_1, \mathsf{pp}_2$ by running $\mathsf{CS.Setup}(1^\lambda)$ twice independently. Recall that C is a layered circuit. Use parameters pp_1 for all the gates on odd layers of C and pp_2 for all the gates on even layers of C. Compute the rest of the proof honestly and at the end, randomize the resulting proof. In more detail, Hyb_1 is as follows:

1. Denote by $\mathsf{layer}_1, \ldots, \mathsf{layer}_t$ the t layers of gates in C. Choose at random $r_1, r_2 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
   - For each $i \in [2]$, compute $\mathsf{pp}_i^0 = \mathsf{CS.Setup}(1^\lambda; r_i)$ and compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$.
   - For all $j \in [t]$ and for each gate $v_j \in \mathsf{layer}_j$, let $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_1^0, \mathsf{pp}_1^1)$ if j is odd, else $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_2^0, \mathsf{pp}_2^1)$ if j is even.
2. For each input wire $k \in [n]$, denote by j the gate for which wire k is an input. For every $b \in \{0,1\}$, choose at random $o_{k,j}^b$ and compute $\vec{c}_{k,j}^b = \mathsf{CS.Commit}(\mathsf{pp}_j^b, w_k^0; o_{k,j}^b)$. Let $\vec{c}_k = (\vec{c}_{k,j}^0, \vec{c}_{k,j}^1)$.

   For the output wire wout and for every $b \in \{0,1\}$, if $\mathsf{out} = 1$, $\vec{c}_{\mathsf{wout},m}^b = \mathbf{1}$ and if $\mathsf{out} = 0$, $\vec{c}_{\mathsf{wout},m}^b = \mathbf{0}$. Let $\vec{c}_{\mathsf{wout}} = (\vec{c}_{\mathsf{wout},m}^0, \vec{c}_{\mathsf{wout},m}^1)$.
3. For each connecting wire $k \in \{n+1, \ldots, n+\ell\}$ that connects gates $i, j \in [m]$, compute
$$\left(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k\right) \leftarrow \mathsf{OutCom}(\mathsf{pp}_i^0, \mathsf{pp}_j^0, r_j, w_k^0)$$
   where $\sigma_c^k = (\vec{c}_{k,i}^0, \vec{c}_{k,j}^0, \vec{c}_{k,i}^1, \vec{c}_{k,j}^1)$, $\sigma_\pi^k = (\pi_k^{00}, \pi_k^{01}, \pi_k^{10}, \pi_k^{11})$ and where $\mathsf{st}^k = (o_{k,i}^0, o_{k,j}^0, o_{k,i}^1, o_{k,j}^1)$. We will denote $(\sigma_c^k, \sigma_\pi^k)$ by Φ_k.
4. For all $(k, j) \in S$ and for every $b \in \{0,1\}$, generate a proof that the commitment $\vec{c}_{k,j}^b$ commits to a bit. Namely, compute
$$\pi_{\mathsf{bit}}[k,j]^b \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}_j^b, \vec{c}_{k,j}^b, o_{k,j}^b)$$
   where $o_{k,j}^b$ is the opening for commitment $\vec{c}_{k,j}^b$ as computed in step 2 (for input wires) or as part of st^k output by OutCom (for connecting wires) in step 3. Let $\pi_{\mathsf{bit}}[k,j] = \left(\pi_{\mathsf{bit}}[k,j]^0, \pi_{\mathsf{bit}}[k,j]^1\right)$.
5. For each gate $j \in [m]$, denote by k_1, k_2 the input wires of the gate j and by k_3, k_4 the output wires of the gate j. For each $t \in \{3,4\}$ and $b \in \{0,1\}$, compute a gate consistency proof as follows:
$$\pi_{\mathsf{gate}}^{j,b}[t] \leftarrow \mathsf{NAND.Prove}\left(\mathsf{pp}_j^b, \{\vec{c}_{k_i,j}^b\}_{i\in\{1,2,t\}}, \{w_{k_i}^0, o_{k_i,j}^b\}_{i\in\{1,2,t\}}\right)$$
   Let $\pi_{\mathsf{gate}}^j = \left(\pi_{\mathsf{gate}}^{j,0}[3], \pi_{\mathsf{gate}}^{j,1}[3], \pi_{\mathsf{gate}}^{j,0}[4], \pi_{\mathsf{gate}}^{j,1}[4]\right)$.
6. Let $\Pi = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S}, \{\pi_{\mathsf{gate}}^j\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$.
   Finally compute $\Pi' \leftarrow \mathsf{NIWI..Rand}((C, \mathsf{out}), \Pi)$. Output Π'.

Hyb_2: This is exactly as Hyb_1 with the following changes: Recall that for each $j \in [m]$ and each $\overrightarrow{\mathsf{pp}}_j \in \Pi$ where $\overrightarrow{\mathsf{pp}}_j = (\mathsf{pp}_j^0, \mathsf{pp}_j^1)$, pp_j^0 is the binding parameter and pp_j^1 is the hiding parameter. We now equivocate the commitments with respect to the hiding parameters to obtain the openings with respect to wit_1. We then compute the bit consistency and gate consistency proofs for the hiding parameters using the equivocated openings with respect to wit_1. Note that the L_{TC} proofs output by OutCom (step 3) are still with respect to wit_0. In more detail, Hyb_2 is as follows:

1. Denote by $\mathsf{layer}_1, \ldots, \mathsf{layer}_t$ the t layers of gates in C. Choose at random $r_1, r_2 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
   - For each $i \in [2]$, compute $\mathsf{pp}_i^0 = \mathsf{CS.Setup}(1^\lambda; r_i)$ and compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$.
   - For all $j \in [t]$ and for each gate $v_j \in \mathsf{layer}_j$, let $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_1^0, \mathsf{pp}_1^1)$ if j is odd, else $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_2^0, \mathsf{pp}_2^1)$ if j is even.
2. For each input wire $k \in [n]$, denote by j the gate for which wire k is an input. For every $b \in \{0,1\}$, choose at random $o_{k,j}^b$ and compute $\vec{c}_{k,j}^b = \mathsf{CS.Commit}(\mathsf{pp}_j^b, w_k^0; o_{k,j}^b)$. Let $\vec{c}_k = (\vec{c}_{k,j}^0, \vec{c}_{k,j}^1)$.

   For the output wire wout and for every $b \in \{0,1\}$, if $\mathsf{out} = 1$, $\vec{c}_{\mathsf{wout},m}^b = \mathbf{1}$ and if $\mathsf{out} = 0$, $\vec{c}_{\mathsf{wout},m}^b = \mathbf{0}$. Let $\vec{c}_{\mathsf{wout}} = (\vec{c}_{\mathsf{wout},m}^0, \vec{c}_{\mathsf{wout},m}^1)$.
3. For each connecting wire $k \in \{n+1, \ldots, n+\ell\}$ that connects gates $i, j \in [m]$, compute

   $$\left(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k\right) \leftarrow \mathsf{OutCom}(\mathsf{pp}_i^0, \mathsf{pp}_j^0, r_j, w_k^0)$$

   where $\sigma_c^k = (\vec{c}_{k,i}^0, \vec{c}_{k,j}^0, \vec{c}_{k,i}^1, \vec{c}_{k,j}^1)$, $\sigma_\pi^k = (\pi_k^{00}, \pi_k^{01}, \pi_k^{10}, \pi_k^{11})$ and where $\mathsf{st}^k = (o_{k,i}^0, o_{k,j}^0, o_{k,i}^1, o_{k,j}^1)$. We will denote $(\sigma_c^k, \sigma_\pi^k)$ by Φ_k.
4. For all $(k,j) \in S$, first compute $s_{k,j}^1 = \mathsf{CS.Equivocate}(\mathsf{pp}_j^1, r_j, \vec{c}_{k,j}^1, o_{k,j}^1, w_k^1)$ where r_j is the randomness used to generate pp_j^0 in step 1. Note that if $w_k^0 = w_k^1$, then $s_{k,j}^1 = o_{k,j}^1$ since equivocation is to the same bit as the committed bit.

   Compute $\pi_{\mathsf{bit}}[k,j]^0 \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}_j^0, \vec{c}_{k,j}^0, o_{k,j}^0)$ as before where $o_{k,j}^0$ is the opening for commitment $\vec{c}_{k,j}^0$ as computed in step 2 (for input wires) or as part of st^k output by OutCom (for connecting wires) in step 3. Compute $\pi_{\mathsf{bit}}[k,j]^1 \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}_j^1, \vec{c}_{k,j}^1, s_{k,j}^1)$ where $s_{k,j}^1$ is the opening of $\vec{c}_{k,j}^1$ with respect to wit_1 as computed before. Let $\pi_{\mathsf{bit}}[k,j] = \left(\pi_{\mathsf{bit}}[k,j]^0, \pi_{\mathsf{bit}}[k,j]^1\right)$.
5. For each gate $j \in [m]$, denote by k_1, k_2 the input wires of the gate j and by k_3, k_4 the output wires to gate j. For each $t \in \{3,4\}$, compute gate consistency proofs as follows:

   $$\pi_{\mathsf{gate}}^{j,0}[t] \leftarrow \mathsf{NAND.Prove}\left(\mathsf{pp}_j^0, \{\vec{c}_{k_i,j}^0\}_{i\in\{1,2,t\}}, \{w_{k_i}^0, o_{k_i,j}^0\}_{i\in\{1,2,t\}}\right)$$

   as before and $\pi_{\mathsf{gate}}^{j,1}[t] \leftarrow \mathsf{NAND.Prove}\left(\mathsf{pp}_j^1, \{\vec{c}_{k_i,j}^1\}_{i\in\{1,2,t\}}, \{w_{k_i}^1, s_{k_i,j}^1\}_{i\in\{1,2,t\}}\right)$.
   Let $\pi_{\mathsf{gate}}^j = \left(\pi_{\mathsf{gate}}^{j,0}[3], \pi_{\mathsf{gate}}^{j,1}[3], \pi_{\mathsf{gate}}^{j,0}[4], \pi_{\mathsf{gate}}^{j,1}[4]\right)$.
6. Let $\Pi = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \{\pi_{\mathsf{bit}}[i,j]\}_{(i,j)\in S}, \{\pi_{\mathsf{gate}}^j\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$. Finally compute $\Pi' \leftarrow \mathsf{NIWI..Rand}((C, \mathsf{out}), \Pi)$. Output $\left\{(C, \mathsf{out}), \mathsf{wit}_0, \mathsf{wit}_1, \Pi'\right\}$.

Hyb$_3$**:** This is exactly as hybrid 2 and the only change is in step 3 where we use $\mathsf{OutCom}_{\mathsf{Bind}}$ instead of using

OutCom. Recall that OutCom outputs four commitments $(\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$ with respect to $(\mathsf{pp}_1^0, \mathsf{pp}_2^0, \mathsf{pp}_1^1, \mathsf{pp}_2^1)$ respectively, four L_{TC} proofs and openings for the four commitments. $\mathsf{OutCom}_{\mathsf{Bind}}$ is same as OutCom except that it computes the L_{TC} proof for $(\vec{c}_1^1, \vec{c}_2^1, \mathsf{pp}_1^1, \mathsf{pp}_2^1)$ differently. In more detail,

$\big(\sigma_c, \sigma_\pi, \mathsf{st}\big) \leftarrow \mathsf{OutCom}_{\mathsf{Bind}}(\mathsf{pp}_1^0, \mathsf{pp}_2^0, r_1, r_2, \mathsf{bit})$: The $\mathsf{OutCom}_{\mathsf{Bind}}$ algorithm takes as input two pairs of parameters $\mathsf{pp}_1^0, \mathsf{pp}_2^0 \in \mathsf{CS.Setup}(1^\lambda)$, randomness r_1, r_2 such that for all $i \in [2]$, $\mathsf{pp}_i^0 = \mathsf{CS.Setup}(1^\lambda; r_i)$ and a bit, and does the following:

- For all $i \in [2]$, compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$.
- For all $i \in [2]$, for all $d \in \{0,1\}$, choose at random o_i^d and compute $\vec{c}_i^d = \mathsf{CS.Commit}(\mathsf{pp}_i^d, \mathsf{bit}; o_i^d)$. Denote by $\sigma_c = (\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$.
- Compute $\mathsf{pp}_*^0 = \mathsf{InterParam}(\mathsf{pp}_1^0, \mathsf{pp}_2^0, r_1)$ and $\mathsf{pp}_*^1 = \mathsf{InterParam}(\mathsf{pp}_1^1, \mathsf{pp}_2^1, r_1)$. For all $b_1, b_2 \in \{0,1\}$ except for $b_1 = b_2 = 1$, compute

$$\pi^{b_1 b_2} \leftarrow \mathsf{TC.Prove}\big((\vec{c}_1^{b_1}, \vec{c}_2^{b_2}, \mathsf{pp}_1^{b_1}, \mathsf{pp}_2^{b_2}), (\mathsf{bit}, \mathsf{pp}_*^{b_1}, o_1^{b_1}, o_2^{b_2})\big).$$

  For all $i \in [2]$, compute $s_i^1 = \mathsf{CS.Equivocate}(\mathsf{pp}_i^1, r_i, \vec{c}_i^1, o_i^1, 1 - \mathsf{bit})$. Denote by $\mathsf{st} = (o_1^0, o_2^0, s_1^1, s_2^1)$.
  Compute $\pi^{11} \leftarrow \mathsf{TC.Prove}\big((\vec{c}_1^1, \vec{c}_2^1, \mathsf{pp}_1^1, \mathsf{pp}_2^1), (1 - \mathsf{bit}, \mathsf{pp}_*^1, s_1^1, s_2^1)\big)$.
  Denote by $\sigma_\pi = (\pi^{00}, \pi^{01}, \pi^{10}, \pi^{11})$. Output $(\sigma_c, \sigma_\pi, \mathsf{st})$.

Concretely, the changed step 3 in Hyb_3 will be as follows:

For each connecting wire $k \in \{n+1, \ldots, n+\ell\}$ that connects gates $i, j \in [m]$, if $w_k^0 \neq w_k^1$ then compute $\big(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k\big) \leftarrow \mathsf{OutCom}_{\mathsf{Bind}}(\mathsf{pp}_i^0, \mathsf{pp}_j^0, r_i, r_j, w_k^0)$ else compute $\big(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k\big) \leftarrow \mathsf{OutCom}(\mathsf{pp}_i^0, \mathsf{pp}_j^0, r_i, w_k^0)$.

Hyb_4**:** In this hybrid, compute $\mathsf{pp}_1^0 \leftarrow \mathsf{CS.Setup}'(1^\lambda)$ and $\mathsf{pp}_2^0 \leftarrow \mathsf{CS.Setup}'(1^\lambda)$. As before, compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$ for all $i \in [2]$. Note that $\mathsf{pp}_1^0, \mathsf{pp}_2^0$ are now the hiding parameters and $\mathsf{pp}_1^1, \mathsf{pp}_2^1$ are the binding parameters.

In addition, all the commitments are now with respect to wit_1 but the bit consistency and gate consistency proofs for the hiding parameters, are with respect to wit_0. These are computed by using the equivocations to wit_0, with respect to hiding parameters (similar to Hyb_2, Hyb_3).

Also in step 3, use $\mathsf{OutCom}_{\mathsf{Hide}}$ instead of using $\mathsf{OutCom}_{\mathsf{Bind}}$. $\mathsf{OutCom}_{\mathsf{Hide}}$ is similar to OutCom except that it computes the L_{TC} proof for $(\vec{c}_1^0, \vec{c}_2^0, \mathsf{pp}_1^0, \mathsf{pp}_2^0)$ differently. In more detail,

$\big(\sigma_c, \sigma_\pi, \mathsf{st}\big) \leftarrow \mathsf{OutCom}_{\mathsf{Hide}}(\mathsf{pp}_1^0, \mathsf{pp}_2^0, r_1, r_2, \mathsf{bit})$: The $\mathsf{OutCom}_{\mathsf{Hide}}$ algorithm takes as input two pairs of parameters $\mathsf{pp}_1^0, \mathsf{pp}_2^0 \in \mathsf{CS.Setup}'(1^\lambda)$, randomness r_1, r_2 such that for all $i \in [2]$, $\mathsf{pp}_i^0 = \mathsf{CS.Setup}'(1^\lambda; r_i)$ and a bit, and does the following:

- For all $i \in [2]$, compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$.
- For all $i \in [2]$, for all $d \in \{0,1\}$, choose at random o_i^d and compute $\vec{c}_i^d = \mathsf{CS.Commit}(\mathsf{pp}_i^d, \mathsf{bit}; o_i^d)$. Denote by $\sigma_c = (\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$.

– Compute $\mathsf{pp}^0_* = \mathsf{InterParam}(\mathsf{pp}^0_1, \mathsf{pp}^0_2, r_1)$ and $\mathsf{pp}^1_* = \mathsf{InterParam}(\mathsf{pp}^1_1, \mathsf{pp}^1_2, r_1)$. For all $b_1, b_2 \in \{0,1\}$ except for $b_1 = b_2 = 0$, compute

$$\pi^{b_1 b_2} \leftarrow \mathsf{TC.Prove}\big((\vec{c}_1^{\,b_1}, \vec{c}_2^{\,b_2}, \mathsf{pp}_1^{b_1}, \mathsf{pp}_2^{b_2}), (\mathsf{bit}, \mathsf{pp}_*^{b_1}, o_1^{b_1}, o_2^{b_2})\big).$$

For all $i \in [2]$, compute $s_i^0 = \mathsf{CS.Equivocate}(\mathsf{pp}_i^0, r_i, \vec{c}_i^{\,1}, o_i^0, 1-\mathsf{bit})$. Denote by $\mathsf{st} = (s_1^0, s_2^0, o_1^1, o_2^1)$.

Compute $\pi^{00} \leftarrow \mathsf{TC.Prove}\big((\vec{c}_1^{\,0}, \vec{c}_2^{\,0}, \mathsf{pp}_1^0, \mathsf{pp}_2^0), (1-\mathsf{bit}, \mathsf{pp}_*^1, s_1^0, s_2^0)\big)$.

Denote by $\sigma_\pi = (\pi^{00}, \pi^{01}, \pi^{10}, \pi^{11})$. Output $(\sigma_c, \sigma_\pi, \mathsf{st})$.

We now describe the hybrid in detail:

1. Denote by $\mathsf{layer}_1, \ldots, \mathsf{layer}_t$ the t layers of gates in C. Choose at random $r_1, r_2 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
   – For each $i \in [2]$, compute $\mathsf{pp}_i^0 = \mathsf{CS.Setup}'(1^\lambda; r_i)$ and compute $\mathsf{pp}_i^1 = \mathsf{OutParam}(\mathsf{pp}_i^0)$.
   – For all $j \in [t]$ and for each gate $v_j \in \mathsf{layer}_j$, let $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_1^0, \mathsf{pp}_1^1)$ if j is odd, else $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_2^0, \mathsf{pp}_2^1)$ if j is even.
2. For each input wire $k \in [n]$, denote by j the gate for which wire k is an input. For every $b \in \{0,1\}$, choose at random $o_{k,j}^b$ and compute $\vec{c}_{k,j}^{\,b} = \mathsf{CS.Commit}(\mathsf{pp}_j^b, w_k^1; o_{k,j}^b)$. Let $\vec{c}_k = (\vec{c}_{k,j}^{\,0}, \vec{c}_{k,j}^{\,1})$.
   For the output wire wout and for every $b \in \{0,1\}$, if $\mathsf{out} = 1$, $\vec{c}_{\mathsf{wout},m}^{\,b} = \mathbf{1}$ and if $\mathsf{out} = 0$, $\vec{c}_{\mathsf{wout},m}^{\,b} = \mathbf{0}$. Let $\vec{c}_{\mathsf{wout}} = (\vec{c}_{\mathsf{wout},m}^{\,0}, \vec{c}_{\mathsf{wout},m}^{\,1})$.
3. For each connecting wire $k \in \{n+1, \ldots, n+\ell\}$ that connects gates $i, j \in [m]$, if $w_k^0 \neq w_k^1$ then compute $\big(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k\big) \leftarrow \mathsf{OutCom}_{\mathsf{Hide}}(\mathsf{pp}_i^0, \mathsf{pp}_j^0, r_i, r_j, w_k^1)$ else compute $\big(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k\big) \leftarrow \mathsf{OutCom}(\mathsf{pp}_i^0, \mathsf{pp}_j^0, r_i, w_k^1)$ where $\sigma_c^k = (\vec{c}_{k,i}^{\,0}, \vec{c}_{k,j}^{\,0}, \vec{c}_{k,i}^{\,1}, \vec{c}_{k,j}^{\,1})$, $\sigma_\pi^k = (\pi_k^{00}, \pi_k^{01}, \pi_k^{10}, \pi_k^{11})$ and where $\mathsf{st}^k = (o_{k,i}^0, o_{k,j}^0, o_{k,i}^1, o_{k,j}^1)$. We will denote $(\sigma_c^k, \sigma_\pi^k)$ by Φ_k.
4. For all $(k,j) \in S$, first compute $s_{k,j}^0 = \mathsf{CS.Equivocate}(\mathsf{pp}_j^0, r_j, \vec{c}_{k,j}^{\,0}, o_{k,j}^0, w_k^0)$ where r_j is the randomness used to generate pp_j^0 in step 1. Compute $\pi_{\mathsf{bit}}[k,j]^0 \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}_j^0, \vec{c}_{k,j}^{\,0}, s_{k,j}^0)$.

   For all $(k,j) \in S$, compute $\pi_{\mathsf{bit}}[k,j]^1 \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}_j^1, \vec{c}_{k,j}^{\,1}, o_{k,j}^1)$, where $o_{k,j}^1$ is the opening for commitment $\vec{c}_{k,j}^{\,0}$ as computed in step 2 (for input wires) or as part of st^k output by OutCom (for connecting wires) in step 3. Let $\pi_{\mathsf{bit}}[k,j] = \big(\pi_{\mathsf{bit}}[k,j]^0, \pi_{\mathsf{bit}}[k,j]^1\big)$.
5. For each gate $j \in [m]$, denote by k_1, k_2 the input wires of the gate j and by k_3, k_4 the output wires to gate j. For each $t \in \{3,4\}$, compute gate consistency proofs as follows:

$$\pi_{\mathsf{gate}}^{j,0}[t] \leftarrow \mathsf{NAND.Prove}\big(\mathsf{pp}_j^0, \{\vec{c}_{k_i,j}^{\,0}\}_{i\in\{1,2,t\}}, \{w_{k_i}^0, s_{k_i,j}^0\}_{i\in\{1,2,t\}}\big)$$

and

$$\pi_{\mathsf{gate}}^{j,1}[t] \leftarrow \mathsf{NAND.Prove}\big(\mathsf{pp}_j^1, \{\vec{c}_{k_i,j}^{\,1}\}_{i\in\{1,2,t\}}, \{w_{k_i}^1, o_{k_i,j}^1\}_{i\in\{1,2,t\}}\big)$$

Let $\pi_{\mathsf{gate}}^j = \big(\pi_{\mathsf{gate}}^{j,0}[3], \pi_{\mathsf{gate}}^{j,1}[3], \pi_{\mathsf{gate}}^{j,0}[4], \pi_{\mathsf{gate}}^{j,1}[4]\big)$.

6. Let $\Pi = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \{\pi_{\mathsf{bit}}[i,j]\}_{(i,j)\in S}, \{\pi^j_{\mathsf{gate}}\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$. Finally compute $\Pi' \leftarrow \mathsf{NIWI..Rand}((C,\mathsf{out}),\Pi)$. Output $\left\{(C,\mathsf{out}), \mathsf{wit}_0, \mathsf{wit}_1, \Pi'\right\}$.

Hyb$_5$**:** This hybrid is same as *Hyb*$_4$ except that in step 3 it uses OutCom instead of $\mathsf{OutCom}_{\mathsf{Hide}}$. More specifically step 3 is as follows:

For each connecting wire $k \in \{n+1,\ldots,n+\ell\}$ that connects gates $i,j \in [m]$, compute

$$\left(\sigma^k_c, \sigma^k_\pi, \mathsf{st}^k\right) \leftarrow \mathsf{OutCom}(\mathsf{pp}^0_i, \mathsf{pp}^0_j, r_j, w^0_k)$$

Hyb$_6$**:** This hybrid is same as *Hyb*$_5$ except that it uses wit_1 for all the bit consistency and gate consistency proofs in steps 4,5. In detail, the hybrid is as follows:

1. Denote by $\mathsf{layer}_1,\ldots,\mathsf{layer}_t$ the t layers of gates in C. Choose at random $r_1, r_2 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
   - For each $i \in [2]$, compute $\mathsf{pp}^0_i = \mathsf{CS.Setup}(1^\lambda; r_i)$ and compute $\mathsf{pp}^1_i = \mathsf{OutParam}(\mathsf{pp}^0_i)$.
   - For all $j \in [t]$ and for each gate $v_j \in \mathsf{layer}_j$, let $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}^0_1, \mathsf{pp}^1_1)$ if j is odd, else $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}^0_2, \mathsf{pp}^1_2)$ if j is even.
2. For each input wire $k \in [n]$, denote by j the gate for which wire k is an input. For every $b \in \{0,1\}$, choose at random $o^b_{k,j}$ and compute $\vec{c}^b_{k,j} = \mathsf{CS.Commit}(\mathsf{pp}^b_j, w^1_k; o^b_{k,j})$. Let $\vec{c}_k = (\vec{c}^0_{k,j}, \vec{c}^1_{k,j})$.

   For the output wire wout and for every $b \in \{0,1\}$, if $\mathsf{out} = 1$, $\vec{c}^b_{\mathsf{wout},m} = \mathbf{1}$ and if $\mathsf{out} = 0$, $\vec{c}^b_{\mathsf{wout},m} = \mathbf{0}$. Let $\vec{c}_{\mathsf{wout}} = (\vec{c}^0_{\mathsf{wout},m}, \vec{c}^1_{\mathsf{wout},m})$.
3. For each connecting wire $k \in \{n+1,\ldots,n+\ell\}$ that connects gates $i,j \in [m]$, compute

   $$\left(\sigma^k_c, \sigma^k_\pi, \mathsf{st}^k\right) \leftarrow \mathsf{OutCom}(\mathsf{pp}^0_i, \mathsf{pp}^0_j, r_j, w^1_k)$$

   where $\sigma^k_c = (\vec{c}^0_{k,i}, \vec{c}^0_{k,j}, \vec{c}^1_{k,i}, \vec{c}^1_{k,j})$, $\sigma^k_\pi = (\pi^{00}_k, \pi^{01}_k, \pi^{10}_k, \pi^{11}_k)$ and where $\mathsf{st}^k = (o^0_{k,i}, o^0_{k,j}, o^1_{k,i}, o^1_{k,j})$. We will denote $(\sigma^k_c, \sigma^k_\pi)$ by Φ_k.
4. For all $(k,j) \in S$ and $b \in \{0,1\}$, compute

   $$\underline{\pi_{\mathsf{bit}}[k,j]^b \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}^b_j, \vec{c}^b_{k,j}, o^b_{k,j})}$$

   where $o^b_{k,j}$ is the opening for commitment $\vec{c}^b_{k,j}$ as computed in step 2 (for input wires) or as part of st^k output by OutCom (for connecting wires) in step 3. Let $\pi_{\mathsf{bit}}[k,j] = \left(\pi_{\mathsf{bit}}[k,j]^0, \pi_{\mathsf{bit}}[k,j]^1\right)$.
5. For each gate $j \in [m]$, denote by k_1, k_2 the input wires of the gate j and by k_3, k_4 the output wires to gate j. For each $t \in \{3,4\}$ and $b \in \{0,1\}$, compute a gate consistency proof as follows:

   $$\underline{\pi^{j,b}_{\mathsf{gate}}[t] \leftarrow \mathsf{NAND.Prove}\left(\mathsf{pp}^b_j, \{\vec{c}^b_{k_i,j}\}_{i\in\{1,2,t\}}, \{w^1_{k_i}, o^b_{k_i,j}\}_{i\in\{1,2,t\}}\right)}$$

   Let $\pi^j_{\mathsf{gate}} = \left(\pi^{j,0}_{\mathsf{gate}}[3], \pi^{j,1}_{\mathsf{gate}}[3], \pi^{j,0}_{\mathsf{gate}}[4], \pi^{j,1}_{\mathsf{gate}}[4]\right)$.

6. Let $\Pi = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S}, \{\pi^j_{\mathsf{gate}}\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$. Finally compute $\Pi' \leftarrow \mathsf{NIWI..Rand}((C,\mathsf{out}),\Pi)$. Output $\left\{(C,\mathsf{out}), \mathsf{wit}_0, \mathsf{wit}_1, \Pi'\right\}$.

Hyb$_7$**:** Exactly as *Hyb*$_6$ except in step 1, compute parameters $\mathsf{pp}^0_1, \mathsf{pp}^0_2$ using $\mathsf{CS.Setup}$ instead of using $\mathsf{CS.Setup}'$ so that they are binding again. In detail, step 1 will be as follows:

Denote by $\mathsf{layer}_1, \ldots, \mathsf{layer}_t$ the t layers of gates in C. Choose at random $r_1, r_2 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.

- For each $i \in [2]$, compute $\mathsf{pp}^0_i = \mathsf{CS.Setup}(1^\lambda; r_i)$ and compute $\mathsf{pp}^1_i = \mathsf{OutParam}(\mathsf{pp}^0_i)$.
- For all $j \in [t]$ and for each gate $v_j \in \mathsf{layer}_j$, let $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}^0_1, \mathsf{pp}^1_1)$ if j is odd, else $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}^0_2, \mathsf{pp}^1_2)$ if j is even.

Hyb$_8$**:** Compute $\Pi \leftarrow \mathsf{NIWI..Prove}((C,\mathsf{out}), \mathsf{wit}_1)$. Output proof Π.

We now prove indistinguishability of all the hybrids. We note that the main challenge is proving that hybrids $3,4$ are indistinguishable (we prove this at the end), the proof of which uses the strong secrecy of L_{TC}.

Proposition 6. *Hyb*$_0 \approx$ *Hyb*$_1$

Proof. Follows directly from randomizability of the proof system (as proved in Claim 5). □

Proposition 7. *Hyb*$_1 \approx$ *Hyb*$_2$

Proof. Follows by witness indistinguishability of $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify})$ and of $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify})$. □

Proposition 8. *Hyb*$_2 \approx$ *Hyb*$_3$

Proof. Follows by witness indistinguishability (WI) of $(\mathsf{TC.Prove}, \mathsf{TC.Verify})$. Recall that strong secrecy of L_{TC} implies plain WI (see Remark 6). □

Proposition 9. *Hyb*$_4 \approx$ *Hyb*$_5$

Proof. Follows by witness indistinguishability of $(\mathsf{TC.Prove}, \mathsf{TC.Verify})$. □

Proposition 10. *Hyb*$_5 \approx$ *Hyb*$_6$

Proof. Follows by witness indistinguishability of $(\mathsf{Bit.Prove}, \mathsf{Bit.Verify})$ and of $(\mathsf{NAND.Prove}, \mathsf{NAND.Verify})$. □

Proposition 11. *Hyb*$_6 \approx$ *Hyb*$_7$

Proof. Follows from the perfect equivocation of the RaHE-commitment scheme. Recall that $\mathsf{CS.Setup}'$ outputs params', such that $\{\mathsf{params} \leftarrow \mathsf{CS.Setup}(1^\lambda) \ : \ \mathsf{pp}\} \approx \{\mathsf{pp}' \leftarrow \mathsf{CS.Setup}'(1^\lambda) \ : \ \mathsf{pp}'\}$. □

Proposition 12. $Hyb_7 \approx Hyb_8$

Proof. Follows directly from randomizability of the proof system. □

Proposition 13. $Hyb_3 \approx Hyb_4$

Proof. We will prove this via intermediate hybrids Hyb'_3 and Hyb'_4. Hyb'_3 is generated using a sample drawn from $\mathcal{D}_{\mathsf{Bind}}$ and its output is distributed identically to Hyb_3. Similarly, Hyb'_4 is generated using a sample drawn from $\mathcal{D}_{\mathsf{Hide}}$ and its output is distributed identically to Hyb_4. The proposition then follows directly from the strong secrecy of L_{TC}.

Recall that by strong secrecy of L_{TC}, the following two distributions are computationally indistinguishable.

- $\mathcal{D}_{\mathsf{Bind}}(1^\lambda)$: Choose r at random and compute $\mathsf{pp}_1^0 = \mathsf{CS.Setup}(1^\lambda; r)$. Compute $\mathsf{pp}_1^1 = \mathsf{OutParam}(\mathsf{pp}_1^0)$. For every $d \in \{0,1\}$, do the following:
  - Choose o_d, o''_d at random and compute $\vec{c}_d = \mathsf{CS.Commit}(\mathsf{pp}_1^0, d\,; o_d)$, $\vec{c}'_d = \mathsf{CS.Commit}(\mathsf{pp}_1^1, d; o''_d)$.
  - Compute $\Pi^d \leftarrow \mathsf{TC.Prove}((\vec{c}_d, \vec{c}'_d, \mathsf{pp}_1^0, \mathsf{pp}_1^1), (d, \mathsf{pp}_1^0, o_d, o''_d))$.
  - Compute $o'_d = \mathsf{CS.Equivocate}(\mathsf{pp}_1^1, r, \vec{c}'_d, o''_d, 1-d)$.

  Output $\left(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi^0, \Pi^1\right)$.

- $\mathcal{D}_{\mathsf{Hide}}(1^\lambda)$: Choose r at random and compute $\mathsf{pp}_1^0 = \mathsf{CS.Setup}'(1^\lambda; r)$. Compute $\mathsf{pp}_1^1 = \mathsf{OutParam}(\mathsf{pp})$. For every $d \in \{0,1\}$, do the following:
  - Choose o'_d, o''_d at random. Compute $\vec{c}_d = \mathsf{CS.Commit}(\mathsf{pp}_1^0, 1-d\,; o''_d)$ and compute $\vec{c}'_d = \mathsf{CS.Commit}($ $\mathsf{pp}_1^1, 1-d; o'_d)$.
  - Compute $\Pi^d \leftarrow \mathsf{TC.Prove}((\vec{c}_d, \vec{c}'_d, \mathsf{pp}_1^0, \mathsf{pp}_1^1), (1-d, \mathsf{pp}_1^0, o''_d, o'_d))$.
  - Compute $o_d = \mathsf{CS.Equivocate}(\mathsf{pp}_1^0, r, \vec{c}_d, o''_d, d)$.

  Output $\left(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi^0_{\mathsf{TC}}, \Pi^1_{\mathsf{TC}}\right)$.

Before describing the hybrids Hyb'_3 and Hyb'_4, we describe intermediate procedures $\mathsf{SCom}_0, \mathsf{SCom}_1$ that take as input a sample from $\mathcal{D}_{\mathsf{Bind}}$ or $\mathcal{D}_{\mathsf{Hide}}$ and output four commitments σ_c, four proofs σ_π and state st.

In detail, for every $d \in \{0,1\}$, SCom_d on input $(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi^0, \Pi^1)$ uses only a part of its input as follows:

SCom_d uses $(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_d, \vec{c}'_d, o_d, o'_d, \Pi^d)$ and does the following:

- Choose randomness r' and compute $\mathsf{pp}_2^0 = \mathsf{RParam}(\mathsf{pp}_1^0; r')$. Compute $\mathsf{pp}_2^1 = \mathsf{OutParam}(\mathsf{pp}_2^0)$. For every $b \in \{0,1\}$, compute $\mathsf{pp}_*^b = \mathsf{InterParam}(\mathsf{pp}_1^b, \mathsf{pp}_2^b, r')$.

– Choose randomness o^0, o^1 and compute $\vec{c}_2^0 = \mathsf{CS.Rand}(\mathsf{pp}_1^0, \vec{c}_d; o^0)$, $\vec{c}_2^1 = \mathsf{CS.Rand}(\mathsf{pp}_1^1, \vec{c}'_d; o^1)$. For every $b \in \{0,1\}$, let o_2^b be the new opening of $\vec{c}_2^b$ computed as $o_2^0 = f_{\mathsf{Com}}(o_d, o^0)$ and $o_2^1 = f_{\mathsf{Com}}(o'_d, o^1)$. Denote by $\sigma_c = (\vec{c}_d, \vec{c}_2^0, \vec{c}'_d, \vec{c}_2^1)$ and $\mathsf{st} = (o_d, o_2^0, o'_d, o_2^1)$.

– Compute two fresh L_{TC} proofs as follows:

  – $\pi^{00} \leftarrow \mathsf{TC.Prove}\big((\vec{c}_d, \vec{c}_2^0, \mathsf{pp}_1^0, \mathsf{pp}_2^0), (d, \mathsf{pp}_*^0, o_d, o_2^0)\big)$.
  – $\pi^{11} \leftarrow \mathsf{TC.Prove}\big((\vec{c}'_d, \vec{c}_2^1, \mathsf{pp}_1^1, \mathsf{pp}_2^1), (1-d, \mathsf{pp}_*^1, o'_d, o_2^1)\big)$.

– Compute two mauled L_{TC} proofs as follows:

  – $\pi^{01} \leftarrow \mathsf{TC.Maul}\big((\vec{c}_d, \vec{c}_1^1, \mathsf{pp}_1^0, \mathsf{pp}_1^1), (1, r', 1, o^1), \pi\big)$. Note that mauled proof π^{01} is a proof that $(\vec{c}_d, \vec{c}_2^1, \mathsf{pp}_1^0, \mathsf{pp}_2^1) \in L_{\mathsf{TC}}$.
  – $\pi^{10} \leftarrow \mathsf{TC.Maul}\big((\vec{c}'_d, \vec{c}_1^1, \mathsf{pp}_1^0, \mathsf{pp}_1^1), (r', 1, o^0, 1), \pi\big)$. Note that mauled proof π^{10} is a proof that $(\vec{c}_2^0, \vec{c}'_d, \mathsf{pp}_2^0, \mathsf{pp}_1^1) \in L_{\mathsf{TC}}$.

  Denote by $\sigma_\pi = (\pi^{00}, \pi^{01}, \pi^{10}, \pi^{11})$. Output $(\sigma_c, \sigma_\pi, \mathsf{st})$.

Claim 8. *Let $\Sigma_{\mathsf{Bind}} \leftarrow \mathcal{D}_{\mathsf{Bind}}(1^\lambda)$. Let r_1, r_2 be chosen at random and for all $i \in [2]$, let $\mathsf{pp}_i = \mathsf{CS.Setup}(1^\lambda; r_i)$. Then, for all $d \in \{0,1\}$, the following distributions are identical:*

$$\big(\mathsf{SCom}_d(\Sigma_{\mathsf{Bind}})\big) \text{ and } \big(\mathsf{OutCom}_{\mathsf{Bind}}(\mathsf{pp}_1, \mathsf{pp}_2, r_1, r_2, d)\big)$$

Proof. Let us look at the difference in the two distributions: $\mathsf{SCom}_d(\Sigma_{\mathsf{Bind}})$ and $\mathsf{OutCom}_{\mathsf{Bind}}(\mathsf{pp}_1, \mathsf{pp}_2, r_1, r_2, d)$. Recall that $\mathsf{OutCom}_{\mathsf{Bind}}(\mathsf{pp}_1, \mathsf{pp}_2, r_1, r_2, d)$ computes four fresh commitments with respect to d to obtain $\sigma_c = (\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$. Proofs $\pi^{00}, \pi^{01}, \pi^{10}$ are computed honestly using the openings with respect to d and randomness r_1 such that $\mathsf{pp}_1 = \mathsf{CS.Setup}(1^\lambda; r_1)$. Proof π^{11} is computed using equivocated openings of $\vec{c}_1^1$ and $\vec{c}_2^1$ with respect to $1-d$. Denote $\sigma_\pi = (\pi^{00}, \pi^{01}, \pi^{10}, \pi^{11})$. Finally, $\mathsf{OutCom}_{\mathsf{Bind}}$ outputs $(\sigma_c, \sigma_\pi, \mathsf{st})$ where st consists of openings of $\vec{c}_1^0, \vec{c}_2^0$ wrt. d and openings of $\vec{c}_1^1, \vec{c}_2^1$ wrt. $1-d$.

$\mathsf{SCom}_d(\Sigma_{\mathsf{Bind}})$ outputs $\sigma_c = (\vec{c}_1^0, \vec{c}_2^0, \vec{c}_1^1, \vec{c}_2^1)$ where $\vec{c}_2^0, \vec{c}_2^1$ are obtained by randomizing $\vec{c}_1^0, \vec{c}_1^1$ respectively and, their openings are also computed using openings of $\vec{c}_1^0, \vec{c}_1^1$ and randomization values. Again, st consists of openings of $\vec{c}_1^0, \vec{c}_2^0$ wrt. d and openings of $\vec{c}_1^1, \vec{c}_2^1$ wrt. $1-d$. In this distribution, π^{00}, π^{11} are computed identically as in $\mathsf{OutCom}_{\mathsf{Bind}}$ whereas, proofs π^{01}, π^{10} are both computed by mauling proof π from Σ_{Bind} with respect to different transformations.

Indistinguishability of the distributions follows from the perfect randomizability and equivocability of the commitment scheme, perfect randomizability of RParam and by malleability of the proof system $(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul})$ for L_{TC} with respect to the transformation $\mathsf{TC.T}$. □

Similarly, we have the following claim.

Claim 9. *Let $\Sigma_{\mathsf{Hide}} \leftarrow \mathcal{D}_{\mathsf{Hide}}(1^\lambda)$. Let r_1, r_2 be chosen at random and for all $i \in [2]$, let $\mathsf{pp}_i = \mathsf{CS.Setup}'(1^\lambda; r_i)$. Then, for all $d \in \{0,1\}$, the following distributions are identical:*

$$\big(\mathsf{SCom}_d(\Sigma_{\mathsf{Hide}})\big) \text{ and } \big(\mathsf{OutCom}_{\mathsf{Hide}}(\mathsf{pp}_1, \mathsf{pp}_2, r_1, r_2, d)\big)$$

We are now ready to describe hybrids Hyb'_3, Hyb'_4. Hyb'_3 differs from Hyb_3 in the following ways:

- Parameters $(\mathsf{pp}_2^0, \mathsf{pp}_2^1)$ are computed as randomization of $(\mathsf{pp}_1^0, \mathsf{pp}_1^1)$ rather than as fresh parameters.
- For all connecting wires, SCom_d is used instead of $\mathsf{OutCom}_{\mathsf{Bind}}$. All commitments (including to input wires) with respect to $\mathsf{pp}_1^0, \mathsf{pp}_2^0$ (binding parameters) are with respect to wit_0 and all commitments with respect to $\mathsf{pp}_1^1, \mathsf{pp}_2^1$ (binding parameters) are with respect to wit_1.
- For bit-proofs and gate-proofs, instead of using equivocated openings use the inconsistent openings output by SCom_d. Note that in Hyb'_3, we do not have the randomness used in the generation of $\mathsf{pp}_1, \mathsf{pp}_2$ to equivocate the commitments. But we get the openings (distributed identically as Hyb_3) through the sample Σ_{Bind}.

Concretely, Hyb'_3 does the following:

1. Sample $(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi_{\mathsf{TC}}^0, \Pi_{\mathsf{TC}}^1) \leftarrow \mathcal{D}_{\mathsf{Bind}}(1^\lambda)$.
2. Choose randomness r' and compute $\mathsf{pp}_2^0 = \mathsf{RParam}(\mathsf{pp}_1^0; r')$. Compute $\mathsf{pp}_2^1 = \mathsf{OutParam}(\mathsf{pp}_2^0)$.
3. Denote by $\mathsf{layer}_1, \ldots, \mathsf{layer}_t$ the t layers of gates in C. For all $j \in [t]$ and for each gate $v_j \in \mathsf{layer}_j$, let $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_1^0, \mathsf{pp}_1^1)$ if j is odd, else $\overrightarrow{\mathsf{pp}}_{v_j} = (\mathsf{pp}_2^0, \mathsf{pp}_2^1)$ if j is even.
4. For each input wire $k \in [n]$, denote by j the gate for which wire k is an input. For every $b \in \{0,1\}$, choose at random $o_{k,j}^b$ and compute $\vec{c}_{k,j}^b = \mathsf{CS.Commit}(\mathsf{pp}_j^b, w_k^b; o_{k,j}^b)$. Let $\vec{c}_k = (\vec{c}_{k,j}^0, \vec{c}_{k,j}^1)$.

   For the output wire wout and for every $b \in \{0,1\}$, if $\mathsf{out} = 1$, $\vec{c}_{\mathsf{wout},m}^b = \mathbf{1}$ and if $\mathsf{out} = 0$, $\vec{c}_{\mathsf{wout},m}^b = \mathbf{0}$. Let $\vec{c}_{\mathsf{wout}} = (\vec{c}_{\mathsf{wout},m}^0, \vec{c}_{\mathsf{wout},m}^1)$.
5. For each connecting wire $k \in \{n+1, \ldots, n+\ell\}$ that connects gates $i, j \in [m]$,
   - If $w_k^0 \neq w_k^1$ and $w_k^0 = 0$ then
     $$(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k) \leftarrow \mathsf{SCom}_0(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi_{\mathsf{TC}}^0, \Pi_{\mathsf{TC}}^1; r', \cdot)$$
   - If $w_k^0 \neq w_k^1$ and $w_k^0 = 1$ then
     $$(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k) \leftarrow \mathsf{SCom}_1(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi_{\mathsf{TC}}^0, \Pi_{\mathsf{TC}}^1; r', \cdot)$$
     Note that we use same r' in $\mathsf{SCom}_0, \mathsf{SCom}_1$ as used in step 2 so that the parameters $(\mathsf{pp}_2^0, \mathsf{pp}_2^1)$ used in $\mathsf{SCom}_0, \mathsf{SCom}_1$ are consistent with step 2.
   - Finally if $w_0^k = w_1^k$ compute
     $$(\sigma_c^k, \sigma_\pi^k, \mathsf{st}^k) \leftarrow \mathsf{OutCom}(\mathsf{pp}_i, \mathsf{pp}_j, r', w_0^k)$$
     Note here that r' is the randomization factor between $\mathsf{pp}_i, \mathsf{pp}_j$ and that is sufficient for computing the intermediate parameter pp_* required for L_{TC} proofs output by OutCom. See description of $\mathsf{InterParam}$ in Section 3.3.1.

where $\sigma_c^k = (\vec{c}_{k,i}^0, \vec{c}_{k,j}^0, \vec{c}_{k,i}^1, \vec{c}_{k,j}^1)$, $\sigma_\pi^k = (\pi_k^{00}, \pi_k^{01}, \pi_k^{10}, \pi_k^{11})$ and where $\mathsf{st}^k = (o_{k,i}^0, o_{k,j}^0, o_{k,i}^1, o_{k,j}^1)$. We will denote $(\sigma_c^k, \sigma_\pi^k)$ by Φ_k.

6. For all $(k,j) \in S$ and $b \in \{0,1\}$, compute

$$\underline{\pi_{\mathsf{bit}}[k,j]^b \leftarrow \mathsf{Bit.Prove}(\mathsf{pp}_j^b, \vec{c}_{k,j}^b, o_{k,j}^b)}$$

where $o_{k,j}^b$ is the opening for commitment $\vec{c}_{k,j}^b$ as computed in step 2 (for input wires) or as part of st^k output by OutCom (for connecting wires) in step 3. Let $\pi_{\mathsf{bit}}[k,j] = \left(\pi_{\mathsf{bit}}[k,j]^0, \pi_{\mathsf{bit}}[k,j]^1\right)$.

7. For each gate $j \in [m]$, denote by k_1, k_2 the input wires of the gate j and by k_3, k_4 the output wires to gate j. For each $t \in \{3,4\}$ and $b \in \{0,1\}$, compute a gate consistency proof as follows:

$$\underline{\pi_{\mathsf{gate}}^{j,b}[t] \leftarrow \mathsf{NAND.Prove}\left(\mathsf{pp}_j^b, \{\vec{c}_{k_i,j}^b\}_{i\in\{1,2,t\}}, \{w_{k_i}^b, o_{k_i,j}^b\}_{i\in\{1,2,t\}}\right)}$$

Let $\pi_{\mathsf{gate}}^j = \left(\pi_{\mathsf{gate}}^{j,0}[3], \pi_{\mathsf{gate}}^{j,1}[3], \pi_{\mathsf{gate}}^{j,0}[4], \pi_{\mathsf{gate}}^{j,1}[4]\right)$.

8. Let $\Pi = \left[\{\overrightarrow{\mathsf{pp}}_j\}_{j\in[m]}, \{\vec{c}_k\}_{k\in[n]}, \{\Phi_k\}_{k\in[\ell]}, \left\{\pi_{\mathsf{bit}}[i,j]\right\}_{(i,j)\in S}, \{\pi_{\mathsf{gate}}^j\}_{j\in[m]}, \vec{c}_{\mathsf{wout}}\right]$. Finally compute $\Pi' \leftarrow \mathsf{NIWI..Rand}((C,\mathsf{out}),\Pi)$. Output $\left\{(C,\mathsf{out}), \mathsf{wit}_0, \mathsf{wit}_1, \Pi'\right\}$.

Hyb_4' is exactly the same as Hyb_3' except that in step 1, we sample from $\mathcal{D}_{\mathsf{Hide}}$ instead of $\mathcal{D}_{\mathsf{Bind}}$. Concretely, step 1 will be:

Sample $(\mathsf{pp}_1^0, \mathsf{pp}_1^1, \vec{c}_0, \vec{c}_0', \vec{c}_1, \vec{c}_1', o_0, o_0', o_1, o_1', \Pi_{\mathsf{TC}}^0, \Pi_{\mathsf{TC}}^1) \leftarrow \mathcal{D}_{\mathsf{Hide}}(1^\lambda)$.

Hyb_4' differs from Hyb_4 in the following ways:

– Parameters $(\mathsf{pp}_2^0, \mathsf{pp}_2^1)$ are computed as randomization of $(\mathsf{pp}_1^0, \mathsf{pp}_1^1)$ rather than as fresh parameters.

– For all connecting wires, SCom_d is used instead of $\mathsf{OutCom}_{\mathsf{Bind}}$. All commitments (including to input wires) with respect to $\mathsf{pp}_1^0, \mathsf{pp}_2^0$ (binding parameters) are with respect to wit_0 and all commitments with respect to $\mathsf{pp}_1^1, \mathsf{pp}_2^1$ (binding parameters) are with respect to wit_1.

– For bit-proofs and gate-proofs, instead of using equivocated openings use the inconsistent openings output by SCom_d.

Hyb_3, Hyb_3' are identically distributed by Claim 8, by the hiding property of the commitment and by the perfect randomizability of RParam. Similarly, Hyb_4, Hyb_4' are identically distributed by Claim 9, by the hiding property of the commitment and by the perfect randomizability of RParam. Hence, Proposition 13 follows from strong secrecy of L_{TC}. □

□

4.3.1 Constructing Malleable Proof System for L_{TC}

In this section, we construct the malleable proof system $(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul})$ for L_{TC}, with respect to the transformation $\mathsf{TC.T}$ as described in Section 4.2. We also prove that it satisfies weak soundness and satisfies strong secrecy assuming DLIN with Leakage. Recall that

$$L_{\mathsf{TC}} = \Big\{ (\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \mid \exists\, (b, \mathsf{pp}_*, o_1, o_2) \text{ s.t.} \\ \{\vec{c}_i = \mathsf{CS.Commit}(\mathsf{pp}_i, b; o_i)\}_{i\in[2]} \wedge \big(\mathsf{ValidInter}(\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}_*) = 1\big) \Big\}$$

We now describe the proof system $(\mathsf{TC.Prove}, \mathsf{TC.Verify})$ for L_{TC}. At a high level, a proof for $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \in L_{\mathsf{TC}}$ is computed by first converting the commitment $\vec{c}_1$ with respect to pp_1 to a commitment $\vec{c}_*$ with respect to pp_2 (using the intermediate parameter pp_* which is part of the witness). The next step is to prove that the homomorphically computed commitment $(\vec{c}_2 \cdot \vec{c}_*)$ is a commitment to 0 or 2, which can be reduced to an L_{Lin} statement with respect to pp_2.

Let $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify}, \mathsf{Lin.Transform})$ be the NIWI proof system for $L_{\mathsf{Lin}}[\mathsf{pp}]$ from Section 3.3.2. Concretely, the proof system for L_{TC} is as follows.

$\mathsf{TC.Prove}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), (b, \mathsf{pp}_*, o_1, o_2)\big)$: For $i \in [2]$, parse $\mathsf{pp}_i = [f_i, h_i, g_i, u_i, v_i, w_i]$. Parse $\mathsf{pp}_* = [f_*, h_*, g_*, u_*, v_*, w_*]$. Without loss of generality, let $(f_*, h_*, g_*) = (f_2, h_2, g_2)$. For $i \in [2]$, parse $\vec{c}_i = (c_1^i, c_2^i, c_3^i)$, parse $o_i = (r_i, s_i)$, and compute

$$\vec{c}_* = (u_*^b f_2^{r_1}, v_*^b h_2^{s_1}, w_*^b g_2^{r_1+s_1}).$$

Compute $\vec{A}, \vec{B}$ as follows:

$$\vec{A} = \Big(c_1^* \cdot c_1^2, c_2^* \cdot c_2^2, c_3^* \cdot c_3^2\Big),\ \vec{B} = \Big(\frac{c_1^* \cdot c_1^2}{u_* u_2}, \frac{c_2^* \cdot c_2^2}{v_* v_2}, \frac{c_3^* \cdot c_3^2}{w_* w_2}\Big)$$

Compute $\Pi_{\mathsf{Lin}} = \mathsf{Lin.Prove}\big((f_2, h_2, g_2), (\vec{A}, \vec{B}), (r, s, u)\big)$ and where $(r, s, u) = (r_1 + r_2, s_1 + s_2, (r_1 + s_1 + r_2 + s_2))$.

Finally output $\Pi_{\mathsf{TC}} = [\mathsf{pp}_*, \vec{c}_*, \Pi_{\mathsf{Lin}}]$.

$\mathsf{TC.Verify}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), \Pi_{\mathsf{TC}}\big)$: Parse $\Pi_{\mathsf{TC}} = [\mathsf{pp}_*, \vec{c}_*, \Pi_{\mathsf{Lin}}]$. Make the following checks:

- Check that $\mathsf{ValidInter}(\mathsf{pp}_1, \mathsf{pp}_2, \mathsf{pp}_*) = 1$.
- Check $e(c_1^*, f_1) = e(c_1^1, f_2)$, $e(c_2^*, h_1) = e(c_2^1, h_2)$ and $e(c_3^*, g_1) = e(c_3^1, g_2)$.

Finally check that $\mathsf{Lin.Verify}((f_2, h_2, g_2), \vec{A}, \vec{B}, \Pi_{\mathsf{Lin}}) = 1$ where $\vec{A} = (c_1^* \cdot c_1^2, c_2^* \cdot c_2^2, c_3^* \cdot c_3^2)$ and $\vec{B} = (\frac{c_1^* \cdot c_1^2}{u_* u_2}, \frac{c_2^* \cdot c_2^2}{v_* v_2}, \frac{c_3^* \cdot c_3^2}{w_* w_2})$.

It is easy to see that completeness holds for all parameters. We now prove *weak soundness*, namely that this proof system is sound if both parameters are binding.

Proposition 14. *For $i \in [2]$, let $\mathsf{pp}_i = [f_i, h_i, g_i, u_i, v_i, w_i]$ and let $\vec{c}_i = (c_1^i, c_2^i, c_3^i)$. Let Π_{TC} be a proof for $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \in L_{\mathsf{TC}}$. If,*

$$\mathsf{TC.Verify}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), \Pi_{\mathsf{TC}}\big) = 1 \ \wedge \ \big(\{\mathsf{pp}_i \in \mathsf{CS.Setup}(1^\lambda)\}_{i\in[2]}\big) \text{ then, } (\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \in L_{\mathsf{TC}}$$

Proof. If $\mathsf{TC.Verify}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), \Pi_{\mathsf{TC}}\big) = 1$, bilinear checks ensure that there exists $\eta, R_1, S_1 \in \mathbb{Z}_p^*$ such that $(u_*, v_*, w_*) = (f_2^{R_1}, h_2^{S_1}, g_2^{R_1+S_1+\eta})$ and $(u_1, v_1, w_1) = (f_1^{R_1}, h_1^{S_1}, g_1^{R_1+S_1+\eta})$. Also there exists b, r_1, s_1 such that $\vec{c}_* = (u_*^b f_2^{r_1}, v_*^b h_2^{s_1}, w_*^b g_2^{r_1+s_1})$ and $\vec{c}_1 = (u_1^b f_1^{r_1}, v_1^b h_1^{s_1}, w_1^b g_1^{r_1+s_1})$. By perfect soundness of $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify})$,

$$\Pr\left[\exists\ (a_1, a_2, a_3) \ \text{ s.t. } (a_1 + a_2 = a_3) \ \wedge \ \left(\vec{A} = (f^{a_1}, h^{a_2}, g^{a_3}) \ \vee \ \left(\vec{B} = (f^{a_1}, h^{a_2}, g^{a_3})\right)\right)\right] = 1$$

where $\vec{A} = (c_1^* \cdot c_1^2, c_2^* \cdot c_2^2, c_3^* \cdot c_3^2)$ and $\vec{B} = (\frac{c_1^* \cdot c_1^2}{u_* u_2}, \frac{c_2^* \cdot c_2^2}{v_* v_2}, \frac{c_3^* \cdot c_3^2}{w_* w_2})$.

Also since $\mathsf{pp}_i \in \mathsf{CS.Setup}(1^\lambda)$ for $i \in [2]$, there exists $r_2 = a_1 - r_1, s_2 = a_2 - s_1$ such that, $\vec{c}_2 = (u_2^b f_2^{r_2}, v_2^b h_2^{s_2}, w_2^b g_2^{r_2+s_2})$. Hence for $b = 0$, $\vec{A}$ is linear and for $b = 1$, $\vec{B}$ is linear, and we conclude that the proposition follows.

□

Malleability of the L_{TC} Proof System

Let $(\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2) \in L_{\mathsf{TC}}$ and let Π_{TC} be the corresponding proof as described above. Recall that $\mathsf{TC.Transform}$ outputs a randomized instance $(\vec{c}_1', \vec{c}_2', \mathsf{pp}_1', \mathsf{pp}_2') \in L_{\mathsf{TC}}$ (as described in Section 4.2). The transformation is defined by randomness $\{o_k', r_{\mathsf{pp}}^k\}_{k\in[2]}$.

For $r_{\mathsf{pp}} = (x', y', z', R', S')$, and for any proof for $(\vec{A}, \vec{B}) \in L_{\mathsf{Lin}}[\mathsf{pp}]$ where $\Pi = [\pi_{11}, \ldots, \pi_{23}]$, define

$$\mathsf{ChangeGen}(\Pi, r_{\mathsf{pp}}) \triangleq [\pi_{11}^{x'}, \pi_{12}^{y'}, \pi_{13}^{z'}, \pi_{21}^{x'}, \pi_{22}^{y'}, \pi_{23}^{z'}].$$

$\mathsf{TC.Maul}\big((\vec{c}_1, \vec{c}_2, \mathsf{pp}_1, \mathsf{pp}_2), \{o_k', r_{\mathsf{pp}}^k\}_{k\in[2]}, \Pi_{\mathsf{TC}}\big)$ works as follows:

1. Parse $\Pi_{\mathsf{TC}} = [\mathsf{pp}_*, \vec{c}_*, \Pi_{\mathsf{Lin}}]$. For $k \in [2]$, parse $r_{\mathsf{pp}}^k = (x_k', y_k', z_k', R_k', S_k')$ and parse $o_k' = (q_k, t_k)$.
2. Randomize pp_* by computing $\mathsf{pp}_*'' = (u_*'', v_*'', w_*'') = (u_* f_2^{R_1'}, v_* h_2^{S_1'}, w_* g_2^{R_1'+S_1'})$.
3. Randomize commitment $\vec{c}_*$ by computing $\vec{c}_*'' = (\vec{c}_1^* f_2^{q_1}, \vec{c}_2^* h_2^{t_1}, \vec{c}_3^* g_2^{q_1+t_1})$.
4. Π_{Lin} is a linearity proof for $(\vec{A}, \vec{B})$ where $\vec{A} = (c_1^* \cdot c_1^2, c_2^* \cdot c_2^2, c_3^* \cdot c_3^2)$ and $\vec{B} = (\frac{c_1^* \cdot c_1^2}{u_* u_2}, \frac{c_2^* \cdot c_2^2}{v_* v_2}, \frac{c_3^* \cdot c_3^2}{w_* w_2})$. Let $(\vec{A}', \vec{B}')$ be the transformed L_{Lin} statement with respect to the randomized commitments. Namely, $(\vec{A}', \vec{B}') = \mathsf{Lin.Transform}(\mathsf{pp}, \vec{A}, \vec{B}; (q_1 + q_2, t_1 + t_2, q_1 + q_2 - R_1' - R_2', t_1 + t_2 - S_1' - S_2'))$. Compute
$$\Pi_{\mathsf{Lin}}'' \leftarrow \mathsf{Lin.Maul}(\mathsf{pp}, (\vec{A}, \vec{B}), \vec{r}, \vec{s}, \Pi_{\mathsf{Lin}})$$
where $\vec{r} = (q_1 + q_2, t_1 + t_2)$ and $\vec{s} = (q_1 + q_2 - R_1' - R_2', t_1 + t_2 - S_1' - S_2')$.
5. Compute $\vec{c}_*' = \mathsf{ChangeCom}(\vec{c}_*'', r_{\mathsf{pp}}^2)$, $\Pi_{\mathsf{Lin}}' = \mathsf{ChangeGen}(\Pi_{\mathsf{Lin}}'', r_{\mathsf{pp}}^2)$, and $\mathsf{pp}_*' = \big((u_*'')^{x_2'}, (v_*'')^{y_2'}, (w_*'')^{z_2'}\big)$
6. Finally output $\Pi_{\mathsf{TC}}' = [\mathsf{pp}_*', \vec{c}_*', \Pi_{\mathsf{Lin}}']$.

Proposition 15. *The proof system* $(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul})$ *is a malleable proof system for* L_{TC} *as per Definition 5, with respect to the transformation* $\mathsf{TC.Transform}$.

Proof. Follows from malleability of $(\mathsf{Lin.Prove}, \mathsf{Lin.Verify}, \mathsf{Lin.Maul})$.

□

Strong Secrecy from the DLIN with Leakage Assumption

In Section 4.2, we described the strong secrecy property required from the NIWI proof system $(\mathsf{TC.Prove}, \mathsf{TC.Verify})$. In particular, strong secrecy states that the distributions $\mathcal{D}_{\mathsf{Hide}}, \mathcal{D}_{\mathsf{Bind}}$ are indistinguishable.

We now show that strong secrecy for the proof system $(\mathsf{TC.Prove}, \mathsf{TC.Verify}, \mathsf{TC.Maul})$ constructed above, follows from the Strong NIWI for $L_{\mathsf{Lin}}[\mathsf{pp}]$ with respect to specific distributions as described in Section 3.3.2. Strong NIWI for $L_{\mathsf{Lin}}[\mathsf{pp}]$ in turn, follows from DLIN with Leakage (as per Proposition 5).

Recall that strong NIWI for $L_{\mathsf{Lin}}[\mathsf{pp}]$ states that:

$$\left\{\mathsf{pp}, (\vec{A}_0, \vec{B}_0), \pi_0\right\} \approx \left\{\mathsf{pp}, (\vec{A}_1, \vec{B}_1), \pi_1\right\}$$

where $\vec{A}_b = (f^{a_1}, h^{a_2}, g^{a_3 - b})$ for $a_1, a_2 \leftarrow \mathbb{Z}_p^*$ and $a_3 = a_1 + a_2$, where $\vec{B}_b = (f^{a_1}, h^{a_2}, g^{a_3 - b + 1})$, and where $\pi_b \leftarrow \mathsf{Lin.Prove}(\mathsf{pp}, (\vec{A}_b, \vec{B}_b), (a_1, a_2, a_3))$.

We now describe a reduction S that takes as input $(\mathsf{pp}, \vec{A}, \vec{B}, \pi)$ where $(\vec{A}, \vec{B}, \pi) = (\vec{A}_0, \vec{B}_0, \pi_0)$ or $(\vec{A}, \vec{B}, \pi) = (\vec{A}_1, \vec{B}_1, \pi_1)$ as described above, and does the following:

1. Parse $\mathsf{pp} = [p, \mathbb{G}, \mathbb{G}_T, e, g_p, f, h, g]$. Denote by $(u, v, w') = \vec{A}$ and $(u, v, w) = \vec{B}$. Let $\mathsf{pp}_D = [f, h, g, u, v, w]$ and $\mathsf{pp}'_D = [f, h, g, u, v, w']$.

2. For every $d \in \{0, 1\}$, choose r_d, s_d, r'_d, s'_d at random and compute commitments $\vec{c}_d = (u^d f^{r_d}, v^d h^{s_d}, w^d g^{r_d + s_d})$ and $\vec{c}'_d = (u^{1-d} f^{r'_d}, v^{1-d} h^{s'_d}, (w')^{1-d} g^{r'_d + s'_d})$. Let $o_d = (r_d, s_d)$ and $o'_d = (r'_d, s'_d)$.

3. For every $d \in \{0, 1\}$, compute $\Pi^d_{\mathsf{Lin}} \leftarrow \mathsf{Lin.Maul}(\mathsf{pp}, \vec{A}, \vec{B}, (t_d, z_d), \pi^{-1})$ where $t = (r_d + r'_d), z = (s_d + s'_d)$. Let $\Pi^b_{\mathsf{TC}} = [\mathsf{pp}'_D, \vec{c}'_d, \Pi^d_{\mathsf{Lin}}]$.

Output $\left(\mathsf{pp}_D, \mathsf{pp}'_D, \vec{c}_0, \vec{c}'_0, \vec{c}_1, \vec{c}'_1, o_0, o'_0, o_1, o'_1, \Pi^0_{\mathsf{TC}}, \Pi^1_{\mathsf{TC}}\right)$.

Note that o_d, o'_d are openings with respect to d and $1 - d$ respectively. Let $\vec{c}_d = (c_1^d, c_2^d, c_3^d)$ and $\vec{c}'_d = (c_1^{d'}, c_2^{d'}, c_3^{d'})$. The main observation is that when $\vec{c}_d$ and $\vec{c}'_d$ commit to different bits, then the L_{Lin} statement in Π_{TC} given by

$$\left(c_1^d \cdot c_1^{d'}, c_2^d \cdot c_2^{d'}, c_3^d \cdot c_3^{d'}\right), \left(\frac{c_1^d \cdot c_1^{d'}}{u^2}, \frac{c_2^d \cdot c_2^{d'}}{v^2}, \frac{c_3^d \cdot c_3^{d'}}{ww'}\right)$$

is a transformed instance of $(\vec{A}^{-1}, \vec{B})$ for $d = 1$ and $(\vec{A}, \vec{B}^{-1})$ for $d = 0$ where $\vec{A} = (u, v, w')$ and $\vec{B} = (u, v, w)$, and where the instance is transformed by (t_d, z_d) for $t = (r_d + r'_d), z = (s_d + s'_d)$. Recall from Remark 4 that given linearity proof for $(\vec{A}, \vec{B})$, it is possible to compute linearity proof for $(\vec{A}^{-1}, \vec{B})$ or $(\vec{A}, \vec{B}^{-1})$ by inverting all the terms in proof π to obtain the proof π^{-1}.

It is easy to see that when reduction S gets as input $(\mathsf{pp}, \vec{A}_0, \vec{B}_0, \pi_0)$, it outputs a sample from $\mathcal{D}_{\mathsf{Hide}}$ since $\vec{B}_0 \in \mathsf{CS.Setup}'(1^\lambda)$, and when it gets as input $(\mathsf{pp}, \vec{A}_1, \vec{B}_1, \pi_1)$, it outputs a sample from $\mathcal{D}_{\mathsf{Bind}}$ since $\vec{B}_1 \in \mathsf{CS.Setup}'(1^\lambda)$.

4.4 Commit-and-Compute Paradigm

In this section, we define and instantiate *commit-and-compute* proof systems. Our motivation is the setting where users commit to their private data (say on a public ledger), and then may wish to prove statements about their private committed data. We want the ability to evaluate an arbitrary function (in the form of a circuit) over individual proofs to obtain proofs on inferred statements about the committed data.

We first define a commit-and-compute NIZK and NIWI proof system. We then use the construction ideas in the previous sections to achieve this notion.

4.4.1 Definition of Commit-and-Compute

Definition 24 (Commit-and-Compute NIZK Proofs)**.** A commit-and-compute NIZK proof system consists of the PPT algorithms $(\mathsf{CnC.Setup}, \mathsf{CnC.Commit}, \mathsf{CnC.Prove}, \mathsf{CnC.Verify}, \mathsf{CnC.Eval})$ with the following input-output behavior:

$\mathsf{CRS} \leftarrow \mathsf{CnC.Setup}(1^\lambda)$: The setup algorithm takes as input the security parameter and outputs a common random string CRS.

$\vec{c} \leftarrow \mathsf{CnC.Commit}(\mathsf{CRS}, d; r)$: The commit algorithm takes as input the CRS, bit d, randomness r, and outputs a commitment $\vec{c}$. We denote a vector of commitments $(\vec{c}_1, \ldots, \vec{c}_n)$ by $\mathbf{com}$.

We are now ready to define our language L_{COM}:

$$L_{\mathsf{COM}} = \left\{ (C, \mathbf{com}, b) \mid \exists (\vec{w}, \vec{r}) \text{ s.t.} \big(C(w_1, \ldots, w_n) = b \big) \ \wedge \ \big(\{ \vec{c}_i = \mathsf{CnC.Commit}(\mathsf{CRS}, w_i; r_i) \}_{i \in [n]} \big) \right\}$$

$\Pi \leftarrow \mathsf{CnC.Prove}(\mathsf{CRS}, (C, \mathbf{com}, b), (\vec{w}, \vec{r}))$: The prove algorithm takes as input the CRS, an instance $(C, \mathbf{com}, b)$ along with its witness $(\vec{w}, \vec{r})$ and outputs a proof Π.

$0/1 \leftarrow \mathsf{CnC.Verify}(\mathsf{CRS}, (C, \mathbf{com}, b), \Pi)$: The verification algorithm takes as input the CRS, an instance $(C, \mathbf{com}, b)$ and a proof Π, and outputs a boolean value indicating success or failure.

$((C, \mathbf{com}, b), \Pi) \leftarrow \mathsf{CnC.Eval}(\mathsf{CRS}, \{(C_i, \mathbf{com}_i, b_i), \Pi_i\}_{i=1}^K, C')$: The evaluation algorithm takes as input the CRS, K instances $\{(C_i, \mathbf{com}_i, b_i)\}_{i=1}^K$ of L_{COM} with their proofs $\{\Pi_i\}_{i=1}^K$ and a circuit C', and outputs the composed instance $(C, \mathbf{com}, b)$ and a corresponding proof Π.

Composing for L_{COM} instances: Recall that in Section 3.2, we defined the $\mathsf{Compose}()$ operation that takes as input $\{(C_i, b_i)\}_{i=1}^k$ where $C_i : \{0,1\}^{n_i} \to \{0,1\}$ and a circuit $C' : \{0,1\}^k \to \{0,1\}$, and outputs (C, b) where $C : \{0,1\}^N \to \{0,1\}$ for $N = n_1 + \cdots + n_k$. In this case, all the circuits $\{C_i\}_{i=1}^k$ were with respect to independent inputs.

We now consider the case where different circuits $\{C_i\}_{i=1}^k$ may have overlapping inputs. In particular, given k instances $\{(C_i, \mathbf{com}_i, b_i)\}_{i\in[k]} \in L_{\mathsf{COM}}$, we want to support the case that different $\mathbf{com}_i$ given as input to $\mathsf{CnC.Eval}$ are overlapping; namely, there is a commitment $\vec{c}$ such that $\vec{c}$ is part of $\mathbf{com}_i$ as well as part of $\mathbf{com}_j$ for some $i, j \in [k]$. We define the composed $\mathbf{com}$ as the sequence $(\mathbf{com}_1, \ldots, \mathbf{com}_k)$ where we delete a commitment $\vec{c}$ if it has previously appeared. We formalize the compose operation as follows:

$\mathsf{Compose}(\{(C_i, \mathbf{com}_i, b_i)\}_{i=1}^k, C')$: Let $\mathbf{com}$ be the vector of commitments obtained as follows: Instantiate $\mathbf{com}$ with $\mathbf{com}_1$. For each commitment $\vec{c}$ in subsequent $\mathbf{com}_i$ for $i \in \{2, \ldots, k\}$, append $\vec{c}$ to $\mathbf{com}$ only if the string $\vec{c}$ does not already appear in $\mathbf{com}$. Hence we finally obtain $\mathbf{com}$ such that each commitment in $\mathbf{com}_1, \ldots, \mathbf{com}_k$ appears in $\mathbf{com}$ exactly once. Thus, $\mathbf{com}$ is the union of all the commitments in $\mathbf{com}_1, \ldots, \mathbf{com}_k$. Let $M = |\mathbf{com}|$.

We will now think of each $C_i : \{0,1\}^M \to \{0,1\}$ where C_i might use only a part of the M inputs. The compose algorithm outputs circuit $C : \{0,1\}^M \to \{0,1\}$ such that for all $\vec{w} \in \{0,1\}^M$, we have $C(\vec{w}) = C'(C_1(\vec{w}), \ldots, C_k(\vec{w}))$ and $b = C'(b_1, \ldots, b_k)$.
If $\mathsf{Compose}()$ is given as input witnesses $(\vec{w}_i, \vec{r}_i)$ for the k instances then it outputs the composed witness $(\vec{w}, \vec{r})$ corresponding to commitments in the vector $\mathbf{com}$, where $\mathbf{com}$ is computed as explained before.

We require the following properties from Commit-and-Compute NIZK proof system:

- **NIZK:** $(\mathsf{CnC.Setup}, \mathsf{CnC.Prove}, \mathsf{CnC.Verify})$ is a non-interactive zero-knowledge proof system for L_{COM} as per Definition 1.

- **Completeness of Eval:** We require that for all non-uniform PPT $\mathcal{A}$ and for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{c} \mathsf{CRS} \leftarrow \mathsf{CnC.Setup}(1^\lambda)\ ;\ (\{(C_i, \mathbf{com}_i, b_i, \Pi_i)\}_{i=1}^k, C') \leftarrow \mathcal{A}(\mathsf{CRS})\ ; \\ ((C, \mathbf{com}, b), \Pi) \leftarrow \mathsf{CnC.Eval}(\mathsf{CRS}, \{(C_i, \mathbf{com}_i, b_i), \Pi_i\}_{i=1}^k, C')\ : \\ \Big(\mathsf{Valid}(C') = 0\Big) \vee \Big(\exists\ i \in [k] \text{ s.t.} \mathsf{CnC.Verify}(\mathsf{CRS}, (C_i, \mathbf{com}_i, b_i), \Pi_i) = 0\Big) \vee \\ \Big((\mathsf{CnC.Verify}(\mathsf{CRS}, (C, \mathbf{com}, b), \Pi) = 1) \wedge (C, \mathbf{com}, b) = \mathsf{Compose}(\{(C_i, \mathbf{com}_i, b_i)\}_{i=1}^k, C')\Big) \end{array}\right] = 1$$

  where $\mathsf{Valid}(C') = 1$ if and only if $C' : \{0,1\}^k \to \{0,1\}$.

- **Unlinkability:** For all PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that for all λ, the probability that $\mathsf{bit}' = \mathsf{bit}$ in the following game is at most $1/2 + \nu(\lambda)$:

  $\underline{\text{GAME}_{\mathsf{Eval}}}$:

  1. $\mathsf{CRS} \leftarrow \mathsf{CnC.Setup}(1^\lambda)$.
  2. $(\mathsf{state}, \{((C_i, \mathbf{com}_i, b_i), (\vec{w}_i, \vec{r}_i), \Pi_i)\}_{i=1}^k, C') \leftarrow \mathcal{A}(\mathsf{CRS})$
  3. Choose $\mathsf{bit} \leftarrow \{0,1\}$. If for any $i \in [k]$, $\mathsf{CnC.Verify}(\mathsf{CRS}, (C_i, \mathbf{com}_i, b_i), \Pi_i) \neq 1$ or $\big((C_i, \mathbf{com}_i, b_i), (\vec{w}_i, \vec{r}_i)\big) \notin R_{\mathsf{COM}}$, then output $\perp$.
  4. If $\mathsf{bit} = 0$ then $((C, \mathbf{com}, b), \Pi) \leftarrow \mathsf{CnC.Eval}(\mathsf{CRS}, \{(C_i, \mathbf{com}_i, b_i), \Pi_i\}_{i=1}^k, C')$.
     If $\mathsf{bit} = 1$ then $\Pi \leftarrow \mathsf{CnC.Prove}(\mathsf{CRS}, (C, \mathbf{com}, b), (\vec{w}, \vec{r}))$ where $((C, \mathbf{com}, b), (\vec{w}, \vec{r})) \leftarrow \mathsf{Compose}(\{(C_i, \mathbf{com}_i, b_i), \Pi_i, C', (\vec{w}_i, \vec{r}_i)\}_{i=1}^k)$. Send (C, b, Π) to $\mathcal{A}$.
  5. $\mathsf{bit}' \leftarrow \mathcal{A}(\mathsf{state}, (C, b, \Pi))$.

Definition 25 (Commit-and-Compute NIWI Proofs)**.** $(\mathsf{CnC.Commit}, \mathsf{CnC.Prove}, \mathsf{CnC.Verify}, \mathsf{CnC.Eval})$ is a Commit-and-Compute NIWI if it has the same description as in Definition 24 where $\mathsf{CRS} = 1^\lambda$. Additionally, $(\mathsf{CnC.Prove}, \mathsf{CnC.Verify})$ is a NIWI proof system for L_{COM} as per Definition 2, and it also satisfies the completeness of evaluation and unlinkability properties as in Definition 24.

4.4.2 Construction Overview

Our commit-and-compute (NIZK or NIWI) proof system is very similar to that of a fully homomorphic (NIZK or NIWI) proof system. The main difference is that there is an explicit commitment vector **com** as part of the instance, and the proofs are with respect to the values committed in this specific vector.

Recall that in our constructions, an FH NIZK as well as FH NIWI proof for $(C,b) \in L_{\mathcal{U}}$ contains commitments to all the wire values in C. One idea is to use the commitments in the instance **com** directly in the proof for $(C, \mathbf{com}, b)$. However if we do that, it will make an evaluated proof distinguishable from a fresh proof when circuits share input variables.

For example, let $(C_1, \mathbf{com}_1, b) \in L_{\mathsf{COM}}$ and $(C_2, \mathbf{com}_2, b) \in L_{\mathsf{COM}}$ such that circuits C_1, C_2 share some input variable. Let $\vec{c}$ be the commitment corresponding to the common input such that $\vec{c}$ is part of both $\mathbf{com}_1$ and $\mathbf{com}_2$. An evaluated proof for $C_1 \wedge C_2$ will contain the commitment $\vec{c}$ twice (the proofs for C_1 and C_2 will each contain $\vec{c}$), whereas a fresh proof for $C_1 \wedge C_2$ will contain the commitment $\vec{c}$ only once.

We deal with this issue by keeping the commitments **com** in the instance separate from the commitments in the proof. We compute the FH NIZK or NIWI proof for $(C,b) \in L_{\mathcal{U}}$ as before. We then add proofs of consistency between the values in **com** and the commitments to the input values in the proof.

Commit-and-Compute NIZK Proofs. We want to compute a proof for $(C, \mathbf{com}, b) \in L_{\mathsf{COM}}$. As described above, we first compute a FH NIZK proof Π for $(C,b) \in L_{\mathcal{U}}$ with witness $(\vec{w}, \vec{r})$. We additionally need to prove consistency between commitments $\{\mathsf{Com}_1, \ldots, \mathsf{Com}_t\}$ part of **com**, and the commitments $\{\vec{c}_1, \ldots, \vec{c}_t\}$ to the input values. Namely, for each $i \in [t]$, we need to prove that Com_i and $\vec{c}_i$ commit to the same value.

This is done as follows: Using the homomorphic properties of the commitment, we can prove the statement that the commitment $\mathsf{Com}_i \cdot \vec{c}_i$ is either a commitment to 0 or a commitment to 2. This can be reduced to a statement of $L_{\mathsf{Lin}}[\mathsf{pp}]$ where $\mathsf{pp} = \mathsf{CRS}$, similar to the reduction in Bit Proofs and Gate Proofs of FH NIZK. Our final commit-and-compute NIZK proof for $(C, \mathbf{com}, b)$ will consist of an FH NIZK proof Π for $(C,b) \in L_{\mathcal{U}}$ along with t WI proofs that for each $i \in [t]$, Com_i and $\vec{c}_i$ commit to the same value. The completeness of evaluation and unlinkability follow from the corresponding properties of the underlying FH NIZK and malleability properties of $L_{\mathsf{Lin}}[\mathsf{pp}]$.

Commit-and-Compute NIWI Proofs. We start be explaining how we generate the commitments in this setting. Note that since we have no CRS in a NIWI, we need to slightly modify our commitment scheme to be without any public parameters. Similar to our FH NIWI construction, we choose two parameters $(\mathsf{pp}^0, \mathsf{pp}^1)$ such that one of them is verifiably binding and commit with respect to both the parameters. Thus, our **com** will be of the form $\mathbf{com} = (\mathsf{pp}^0, \mathsf{pp}^1, (\mathsf{Com}_1^0, \mathsf{Com}_1^1), \ldots, (\mathsf{Com}_n^0, \mathsf{Com}_n^1))$.

Again, we first compute FH NIWI proof Π for $(C,b) \in L_{\mathcal{U}}$. We also need to add consistency proofs with respect to **com** and commitments in the FH NIWI proof. Consider an input wire k to circuit C;

the proof Π contains $(\mathsf{pp}_k^0, \mathsf{pp}_k^1, \vec{c}_k^0, \vec{c}_k^1)$ corresponding to wire k. Proving consistency between these and some $(\mathsf{pp}^0, \mathsf{pp}^1, \mathsf{Com}_i^0, \mathsf{Com}_i^1) \in \mathbf{com}$ boils down to proving four L_{TC} statements exactly as in the procedure OutCom (described in Section 4.2). Our final commit-and-compute NIWI proof for $(C, \mathbf{com}, b)$ will consist of an FH NIWI proof Π for $(C, b) \in L_{\mathcal{U}}$ along with L_{TC} proofs for consistency of commitments. The completeness of evaluation and unlinkability follows from the corresponding properties of the underlying FH NIWI and malleability properties of L_{TC}.

Thus, commit-and-compute NIZK and NIWI proofs can be directly instantiated using our FH NIZK and FH NIWI constructions and its building blocks, and we get the following theorems.

Theorem 5. *There exists Commit-and-Compute NIZK proof system as per Definition 24, assuming DLIN.*

Theorem 6. *There exists a commit-and-compute NIWI proof system as per Definition 25, assuming DLIN with Leakage.*

Chapter 5

Privacy Preserving Verifiable Key Directories

A number of popular messaging apps such as iMessage, WhatsApp and Signal have recently deployed end-to-end encryption (E2EE) in an attempt to mitigate some of the serious privacy concerns that arise in these services. E2EE is a system of communication where only encrypted messages leave the sender's device. These messages are then downloaded and decrypted on the recipient's device, ensuring that only the communicating users can read the messages. But E2EE relies on a Public Key Infrastructure (PKI); this in practice requires the provider of the messaging service (such as Apple, Facebook, Microsoft etc.) to maintain a centralized directory of the public keys of its registered users. To be accessible to average users, these systems assume the user will store her secret key on her personal device, and do not assume she has any other way to store long term secrets. When a user loses her device (and thus her secret key), she will need to generate a new (secret key, public key) pair and replace her old public key stored in the PKI with the newly generated public key.

Such a system naturally places a lot of trust in the service provider – a malicious service provider (or one who is compelled to act maliciously, possibly because of a compromise) can arbitrarily set and reset users' public keys. It might, for example replace an honest user's public key with one whose secret key it knows, and thus implement a man-in-the-middle attack without the communicating users ever noticing. Ironically, this defeats the purpose of E2EE. Without some way of verifying that the service provider is indeed returning the correct keys, E2E encryption does not provide any protection against malicious (or coerced) service providers. This problem has been well recognized as an important and challenging open problem [sig16, Key19a, MBB+15].

Some service providers provide a security setting option to notify a sender when a recipient's public key changes. In WhatsApp, the sender can scan a QR code on the recipient's device to verify the authenticity of the new public key. Skype encrypted messaging provides a similar interface for checking fingerprints. This option, however, is turned off by default to provide a seamless user experience. Moreover, the communicating users will be able to verify each other's QR codes only if their devices are physically close to each other, which

is most often not the case. And these features are something few users use, as is evidently uncovered in this Man-in-the-Middle attack [wha17].

To enable E2EE with real security, we need to keep the inherent constraints of the system in mind. To begin with, the primary objective of any E2EE messaging system is to provide a *secure and seamless* communication service between remote users. This problem is made even more challenging by the fact that we must assume that a user can lose her device and along with it all of her secrets. Moreover, in our attempt to reduce trust on the service provider, we must not introduce any new attack surface. For example, if we introduce a mechanism that will enable a user to verify that she is receiving the correct key of the intended recipient, this mechanism should not leak any additional information about the other users and the public keys they have registered with the service provider. Privacy may not be very important in a traditional PKI, where all the players are usually public entities like businesses, but in the context of private messaging, privacy is very important. Hiding usernames may help prevent spam messaging. And the user's update pattern may itself be sensitive information. Users change their keys primarily when they change devices, or detect that their devices/keys have been compromised, either of which may be sensitive. Moreover, if a user rarely changes her key, then compromising her secret key gives the attacker the ability to decrypt large volumes of messages at once, making her a more vulnerable and attractive target. Or if a device is compromised but the user does not update her key, then the attacker knows the compromise has gone undetected.

Keeping these inherent constraints in mind, we design and provide a prototype implementation of SEEMless, *a verifiable key directory service* for end-user key verification and initiate the study of the privacy and security guarantees of such a key directory service formally. In our design, we follow the basic approach introduced by by Melara et al. in [MBB+15], in which the service provider maintains the public key directory as described above, but each user is in charge of verifying that the public key that the service provider is presenting on their behalf is correct. (We call this key monitoring.) To facilitate this, the service provider will at regular intervals (called epochs) publish commitment of the latest directory. When responding to queries (either from Bob looking up Alice's key or from Alice monitoring her own key) it will then prove that the response is correct with respect to the current commitment.

5.0.3 Our Contributions

Here we highlight the significant contributions of this work and contrast them with the existing privacy-preserving key directory services for encrypted messaging [MBB+15, TD17, Bon16].

(A) Formalizing key directory

1. We formalize the security and privacy requirements of a verifiable key directory service, (such CONIKS [MBB+15, TD17], EthIKS [Bon16]), in terms of a new primitive that we call Verifiable Key Directories (VKD) (Section 5.1). To the best of our knowledge, this is the first formal treatment of VKD, which is absent in all the prior works [MBB+15, Bon16, TD17].
2. Proving security of any VKD system (including our instantiation, SEEMless) in our framework requires specifying a precise leakage function. This leads to a better understanding of the privacy guarantee of any VKD system. As a concrete example, we were able to identify a *tracing attack* in existing VKD

systems [MBB+15, Bon16, TD17].

(B) New primitives, modular and principled design

1. We take a modular and principled approach in designing our VKD system, SEEMless (Section 5.3). We first define a new primitive, *Append-Only Zero Knowledge Set* (aZKS) and construct SEEMless using aZKS in a blackbox way. Then we construct aZKS modularly from a *strong accumulator* [CHKO08] and a *verifiable random function* [CL07]. None of the prior works [MBB+15, Bon16, TD17] has a modular approach in designing their systems which makes it hard to reason about those constructions and improve on them. With our modular design, we significantly simplify the presentation and analysis and allow for independent study of each of the building blocks: an improvement in any of them would directly translate to improvement in SEEMless.
2. Our new primitive aZKS (that acts as the building block for SEEMless) generalizes the definition of a traditional static zero-knowledge set [MRK03, CHL+05] by accounting for verifiable updates and could be of independent interest.

(C) Privacy and Efficiency Improvements Compared to the prior work, we significantly improve on the following parameters. For experimental comparison, we implemented a prototype of our system, SEEMless in Java and compared it with CONIKS, which is also implemented in Java.

1. Server storage: CONIKS [MBB+15, TD17], uses a space-inefficient data-structure in order to retain the history of all user key updates, where the size of the data-structure is proportional to the number of server epochs. In contrast, thanks to our Persistent Patricia Trie construction, the size of our data-structure (aZKS) is proportional to the total number of key updates in the system as opposed to the number of epochs. The space overhead in EthIKS [Bon16] is even more since the data-structure at the server is posted on the Ethereum blockchain, thereby replicating it at each network node.

2. Key Monitoring cost: The cost that a user has to pay to monitor her key revision history at the server is an important metric in a VKD system, especially because the users are often on a weak device, like a mobile phone. We significantly improve on the monitoring cost compared to [MBB+15, TD17, Bon16]. CONIKS [MBB+15, TD17] required that each user queries the server *every epoch* to ensure that her key was correct at that time; a user who was offline for a period would have to perform all the checks for that time on coming back online. EthIKS [Bon16] lets a user audit only for the*latest version* of her key. In SEEMless, a user can monitor her key revisions *any time* with cost that depends on the *number of times her key has been updated* rather than the number of epochs that have passed. For example, she can be offline for several server epochs and can verify her entire key history by downloading around 2MB data (even if she is updating her key often) when she comes back online; in contrast, a CONIKS user will need to download about 577MB.

3. Frequency of server updates: In a VKD system, when a user updates her key, the change is not reflected immediately; it is reflected in the server's commitment of the subsequent epoch. Therefore, until the next epoch, either the server can continue to respond with the stale user key, or the server can respond

with a key which is not consistent with the published commitment and update it in the next epoch. CONIKS proposes this latter approach; it is unclear how the user can verify that this new key is in fact correct. In our solution, we propose making the epochs short enough that we can take the first approach without significantly affecting security or usability of the system. Thanks to our space-efficient design, we can afford to have short epochs. This makes SEEMless scale well with frequent server updates, as opposed to CONIKS. For the same update frequency and experimental setup, CONIKS cannot handle more than 4.5M users, owing to the inefficiency of their underlying data structure and hangs with more frequent epochs whereas our system could handle 10M users seamlessly. In EthIKS, the frequency of server updates is at least as slow as the block frequency of Ethereum.

4. Privacy: SEEMless provides provably stronger privacy guarantees as compared to [MBB$^+$15, TD17, Bon16]. In fact, we identify some significant privacy leakage in [MBB$^+$15, Bon16, TD17] through which it might be possible to trace the entire update history of a particular user. This means once Alice queried for Bob's key, she might be able to completely trace when Bob's key changed without ever querying for his key again. Even if Bob has deleted Alice from his contact list (and the server does not give out Alice's key to Bob after he has removed Alice from his contact list), Alice will still be able to trace when Bob's key changed just by looking at the proof of her own key. We call this *tracing attack* and explain its technical details in Appendix **??**.

5. User cost: In our experiments, the cost of lookup query in SEEMless is slightly more expensive than in CONIKS (at most $3\times$) due to our stronger privacy guarantees, but it is still about $10ms$ from request to complete verification, which is reasonably fast. In EthIKS each user lookup requires interaction with the Ethereum blockchain, which which introduces significant overhead compared to CONIKS and SEEMless.

6. Auditors: We introduce the concept of auditors in a VKD — potentially untrusted parties who verify that the server's commitments are well formed and updated in a valid way. This is crucial to our more efficient key monitoring: essentially where in CONIKS the users do a lot of redundant work to verify that each server update is correct with respect to their keys, we only need one honest party to check each server update. By design, this verification is not privacy-sensitive in SEEMless, so we can open this verification up to anyone, including privacy concerned users or privacy advocacy groups. See Section 5.1.1 for more discussion. In EthIKS the auditing is implicitly performed by Etherum smart contracts adding a huge overhead.

5.0.4 Related Work

Our work broadly falls in the category of building provably secure and efficient privacy-preserving key directory service, particularly relevant in the context of end-to-end secure messaging services. In the recent past, this problem has received significant attention in both the academic community and industry [Bon16, TD17, MBB$^+$15, EK15, Pro12, Key14, ANSF16, Nam14]. This line of work is related to transparency logs [LLK13, EMBB17], but here we focus on the works that are most relevant to us.

The work closest to us in the literature is CONIKS [MBB+15], a directory service that lets users of an end-to-end encrypted communication system verify that their keys are being correctly reported to all other users. We have already discussed this construction in Section 5.3. EthIKS [Bon16] implements CONIKS using the Ethereum blockchain for storing the server's authentication data-structures in an Ethereum contract. Not only does this weaken the privacy guarantee of CONIKS, invoking the blockchain for every operation leads to a significantly high overhead and makes this approach quickly infeasible. Catena [TD17] provides a more general infrastructure for managing application-specific logs of *append only* statements using the `OP_RETURN` function of the Bitcoin blockchain and implements CONIKS using Catena. Catena can be used for storing SEEMless digests with the same overhead as for storing CONIKS digests. We discuss how we can integrate SEEMless with Catena in Section 5.1.1.

Recently, Keybase has rolled out an auditable key directory service [Key19c, Key19b] but without any privacy; all the key changes by end-users in Keybase are publicly available. Keybase also differs from SEEMless, CONIKS and EthIKS in the user assumptions, they assume that users have multiple trusted devices and access to long-term cryptographic signature keys with which they sign all their encryption key updates. For an end user, it is incredibly difficult to manage cryptographic keys, so we believe this assumption is not very realistic.

5.1 Verifiable Key Directory (VKD)

In this section, we will define the primitive of a *Verifiable Key Directory* (VKD) and formalize its properties. The goal of a VKD is to capture the functionality and security of a privacy-preserving verifiable key directory system.

A VKD consists of three types of parties: an identity provider or server, clients or users and external auditors. The server stores a directory Dir with the names of the users (which we call labels) and their corresponding public keys (the values corresponding to the labels). For the ease of exposition, let Alice and Bob be two users. VKD provides the following query interface to the users. 1) Alice can add her (username, key), i.e., (label=username, val=key), to Dir. 2) Alice can update her key and request that Dir be updated with the new key value. 3) She can query the server periodically to obtain history of her key updates over time (VKD.KeyHist). 4) Bob can also query for the key corresponding to username Alice (VKD.Query) at the current time.

The functionality VKD.KeyHist warrants further discussion since this is not a functionality that one usually expects from a key directory service. In a privacy-preserving verifiable key directory service, intuitively, we expect the server to be able to prove to Bob that he is seeing Alice's latest key without leaking any additional information about the directory. This is trivial to achieve if we assume that Alice can always sign her new public key with her old secret key. But this is a completely unreasonable assumption from an average user who may lose her device or re-install the software, thereby losing her previous secret key; the user will only have access to her latest secret key which is stored on her latest device. It is crucial to *not assume* that Alice or Bob can remember any cryptographic secret. Under this constraint, we need Alice to monitor her key sufficiently often to make sure her latest key is in the server directory. Only then, we can talk about Bob

getting Alice's latest key in a meaningful way. Alice could of course check every epoch to make sure that her key is being correctly reported, but this becomes costly, particularly when epochs are short. Instead, we allow Alice to query periodically and retrieve a list of all the times her key has changed and the resulting values using the VKD.KeyHist interface.

The server applies updates from its users (of type 1 and 2 described above) in batches, and publishes a commitment to the current state of the database Com and proof Π^{Upd} that a valid update has been performed (VKD.Publish) periodically. The server also publishes a public datastructure which maintains information about all the commitments so far. The auditors in the system keep checking the published update proofs and the public datastructure in order to ensure global consistency (VKD.Audit) of Dir.

The updates should be sufficiently frequent, so that the user keys are not out-of-date for long. The exact interval between these server updates, or epochs has to be chosen as a system parameter. We use time and epoch interchangeably in our descriptions.

The server also produces proofs for VKD.Query and VKD.KeyHist. At a very high level, the users verify the proofs (VKD.QueryVer, VKD.HistVer) to ensure that the server is not returning an incorrect key corresponding to a username or an inconsistent key history for the keys corresponding to a username. VKD also requires the proofs to be privacy-preserving, i.e., the proofs should not leak information about any other key (that has not been queried) in Dir. The auditors are not trusted for privacy and hence, the proofs that the server produces as part of VKD.Publish need to be privacy-preserving as well.

Definition 26. A Verifiable Key Directory is comprised of the algorithms (VKD.Publish,VKD.Query, VKD.QueryVer,VKD.KeyHist,VKD.HistVer,VKD.Audit) and all the algorithms have access to the system parameters. We do not make the system parameters explicit. The algorithms are:

Periodic Publish:

$\triangleright(\mathsf{Com}_t, \Pi_t^{\mathsf{Upd}}, \mathsf{st}_t, \mathsf{Dir}_t) \leftarrow \mathsf{VKD.Publish}(\mathsf{Dir}_{t-1}, \mathsf{st}_{t-1}, S_t)$:

This algorithm takes the previous state of the server and the key directory at previous epoch $t-1$ and also a set S_t of elements to be updated. Whenever a client submits a request to add a new label or update an existing label from epochs $t-1$ to t, the corresponding $(\mathsf{label}, \mathsf{val})$ pair is added to S_t to be added in the VKD at epoch t. The algorithm produces a commitment to the current state of the directory Com_t and a proof of valid update Π^{Upd} all of which it broadcasts at epoch t. It also outputs the updated directory Dir_t and an updated internal state st_t. If this is the first epoch, i.e., no previous epoch exists, then the server initializes an empty directory and its internal state first, then updates them as described above.

Querying for a Label:

$\triangleright\ (\mathsf{val}, \pi) \leftarrow \mathsf{VKD.Query}(\mathsf{st}_t, \mathsf{Dir}_t, \mathsf{label})$: This algorithm takes the current state of the server for epoch t, the directory Dir_t at that epoch and a query label label and returns the corresponding value if it is present in the current directory, $\perp$ if it is not present, a proof of membership or non-membership respectively.

$\triangleright\ 1/0 \leftarrow \mathsf{VKD.QueryVer}(\mathsf{Com}, \mathsf{label}, (\mathsf{val}, \pi))$:

This algorithm takes a commitment with respect to some epoch, a label, value pair and verifies the above proof.

Checking Consistency of Key Updates:

▷ $(\{(\mathsf{val}_i, t_i)\}_{i=1}^{n}, \Pi^{\mathsf{Ver}}) \leftarrow \mathsf{VKD.KeyHist}(\mathsf{st}_t, \mathsf{Dir}_t, \mathsf{label})$: This algorithm takes in the server state, the directory at current time t and a label. It outputs $\{(\mathsf{val}_i, t_i)\}_{i=1}^{n}$ which are all the times at which the value corresponding to label was updated so far, the resulting val's, along with a proof Π^{Ver}.

▷ $1/0 \leftarrow \mathsf{VKD.HistVer}(\mathsf{Com}_t, \mathsf{label}, \{(\mathsf{val}_i, t_i)\}_{i=1}^{n}, \Pi^{\mathsf{Ver}})$: This algorithm takes the commitment published by the server for the current time t, a label, and $\{(\mathsf{val}_i, t_i)\}_{i=1}^{n}$, and verifies Π^{Ver}.

Auditing the VKD:

▷ $1/0 \leftarrow \mathsf{VKD.Audit}(t_1, t_n, \{(\mathsf{Com}_t, \Pi_t^{\mathsf{Upd}})\}_{t=t_1}^{t_n})$: This algorithm takes the epochs t_1 and t_n between which audit is being done, the server's published pub for all the epochs from times t_1 to t_n. It outputs a boolean indicating whether the audit is successful.

Now we discuss the security properties we require from a VKD. We give the informal descriptions here and defer the formal definitions to Appendix **??**. The security properties are the following.

– **Completeness:** We want to say that if a VKD is set up properly and if the server behaves honestly at all epochs, then all the following things should happen for any label updated at $t_1, \ldots, t_n$ with $\mathsf{val}_1, \ldots, \mathsf{val}_n$: 1) their history proof with $\{(\mathsf{val}_i, t_i)\}_{i=1}^{n}$ and Π^{Ver} should verify at t_n 2) the query proof for the label at any $t_j \leq t^* < t_{j+1}$ should verify with respect to the value consistent with the versions proof at t_j which is val_j and 3) the audit from epochs t_1 to t_n should verify.

Note that for KeyHist and HistVer, we consider epochs $t_1, t_2, \ldots, t_n$ when the updates have happened for a label. These will be epochs distributed in the range $[t_1, t_n]$. However for Audit, we consider all possible pairwise epochs between t_1 and t_n. For example, for $t_1 = 3$ to $t_n = 10$, there might be updates at $3, 5, 8, 10$ but for audit we need to consider all of the epochs $3, 4, 5, 6, 7, 8, 9, 10$.

– **Soundness:** VKD soundness guarantees that if Alice has verified the update history of her key till time t_n and if for each t_i from the beginning of time till t_n, there exists at least one honest auditor whose audits have been successful then, whenever Bob queried before t_n, he would have received Alice's key value that is consistent with the key value reported in Alice's history query. Thus soundness is derived from all of VKD.Publish, VKD.QueryVer, VKD.HistVer and VKD.Audit.

Note that the onus is on the user, Alice, to make sure that the server is giving out the most recent and *correct* value for her key. Soundness guarantees that under the circumstances described above, Bob will always see a key consistent with what Alice has audited. But Alice needs to verify that her key as reported in the history query is consistent with the actual key that she chose.

– **Privacy:** The privacy guarantee of a VKD system is that the outputs of Query, HistVer or Audit should not reveal anything beyond the answer and a well defined leakage function on the directory's state. So, the proofs for each of these queries should be simulatable given the output of the leakage function and the query answer.

5.1.1 Model Assumptions

Here we discuss the assumptions we have in our VKD model and discuss why we think these are justified.

What we assume from users. Our goal is to make VKDs usable by average users who cannot be assumed to remember or correctly store long term cryptographic secrets. We do assume that the user's device can store a secret (the user's current key), although that device might be lost/re-imaged, etc in which case the secret would be lost. We assume that the user has some way of authenticating to the service provider (this could be through a second factor text message, or a phone call to answer security questions, etc); this is already a requirement in existing end-to-end encryption systems or other messaging services so we are not increasing the attack surface. We also assume that the server enforces some kind of access control on the public key directory, for example only responding to Bob's query for Alice's key if he is on her contact list. Finally, we assume that the user can recognize the approximate times when she updated her key (enough to be able to identify if extra updates have been included in the list). To help with this we could also have the server store a note from the user about each update, e.g. "bought new iphone" or "re-installed app". VKDs could of course support options for power users to make KeyHist queries whenever they want, to explicitly compare public keys with their friends (as in WhatsApp), or to sign their key updates.

Assumptions on system. We do assume that Alice, Bob, and the server have clocks that are approximately in sync. We also assume some way of ensuring that all parties have consistent views of the current root commitment or at least that they periodically compare these values to make sure that their views have not forked. A simple way of doing this would be for the server to publish the head periodically to a blockchain (as in Catena [TD17]), but we could also implement it with a gossip protocol as in CONIKS [MBB+15]. If we implement this by having the server post the commitment on the blockchain, then this means we assume the client software periodically interact with the blockchain. (Because of the hash chain, even if clients do not check the root on the blockchain every epoch, if two different client's views of the root diverge, they will eventually detect that when they do contact the blockchain.) Similarly, our epoch length need not be limited by the blockchain block time – it is enough if the server periodically posts the current commitment on the blockchain; a misbehaving server will be caught as soon as the next blockchain post happens. This vulnerability window is a tunable parameter. The bandwidth requirements (for interacting with the blockchain) of log auditors of the system and of thin clients (e.g. cell phones) can be significantly reduced by using an efficiently-verifiable blockchain witnessing service, like Catena [TD17].

Cost of auditors. In a VKD system, the audit functionality ensures that the server updates are consistent. The audit proofs are not privacy sensitive, so we do not have to place any restrictions on who can perform audits. Audit can be performed by many auditors in parallel and it is sufficient to have *at least one honest auditor* perform audits over each adjacent server epoch. This means, auditor A could audit for server update between epoch $t, t+1$, while a different auditor B could audit between epoch $t+1, t+2$. It is reasonable to assume that *some privacy conscious* users of a VKD system or *privacy-advocacy groups* or even altruistic nodes[CKC+10], would be willing to periodically audit server updates. Furthermore, in the prototype implementation of our VKD system, SEEMless, we observed that the bandwidth cost for an audit to check server update between two consecutive epochs is less than 5MB and takes less than 0.3s to verify. For scale, 5MB is smaller than the size of an average image taken from a 12MP smartphone camera. Hence, auditing a single

server update is not a significant burden. We discuss the details of the audit experiments in Section **??**.

User Experience. Having an end-user seamlessly interact with the functionality of any VKD is crucial for successful deployment. Designing a user interface that exposes the VKD functionality to end users without drastically changing their experience of using existing messaging software requires usability studies, which we leave for future work. However, here we discuss a blueprint of what we envision the user interface of a VKD system to be.

The high level goal is to have an end-user's software run the queries and verification the background *in a timely manner* and alert an end-user only when some verification fails. When Alice first registers for the service, the client software installed on her device generates keys in the background, and makes a KeyHist query to the server to verify that this is the first key that was issued for her, and a Query to verify that the key is stored correctly. This happens in the background and is invisible to Alice unless verification of one of these fails, in which case she will be alerted of possible misbehavior. Similarly, when Alice requests Bob's key, the client software will run verification of the proof received from the server in the background and will only notify Alice if the verification fails. In addition to that, we want Alice's client to check her key history *sufficiently often*. This entails running KeyHist query in the background periodically and notifying Alice if it is not consistent with the updates she has made.* *keyver* must also be run when Alice changes her device or reinstalls the client software (thereby forcing a key update) in which cases the software would display to Alice a list of the times when her key was updated.

Multiple devices. We have described the system as storing only a single key per user, but in reality Alice might use her account on multiple devices, and the system would be used to store a list of public keys, one for each of her devices. Bob would then encrypt under all of these keys, and the service would determine which to deliver. This works without any modification of the VKD system. The only change in user experience is that when Alice's device runs periodic updates, it might find updates made by other devices: the times for these updates should be displayed to Alice for verification.

Distributing the service. We have described the service in terms of a single server, but in practice, the "server" can be implemented using a distributed network of servers, for reliability and redundancy. Our model captures the server as a *single logical entity*, so it can accommodate a distributed implementation fairly easily. Constructing proofs just requires reading the shared data structures for previous time epochs, so that can easily be done by many processes in parallel. Once all of the updates for a given epoch have been collected, then the authentication data structure needs to be updated. In our VKD system, SEEMless, this can also be parallelized fairly easily because of the tree structure (details in Section **??**). We can support queries even during the data structure update by keeping a snapshot of the last authentication data structure until the update epoch completes. Once the epoch completes, the snapshot can be discarded (so this does not blow up memory).

*Determining how often the client software needs to run KeyHist depends depends how quickly users want to be notified of misbehavior and requires user studies which we defer for future work.

5.2 Append-Only Zero Knowledge Set (aZKS)

In this section we introduce a new primitive, *Append-Only Zero Knowledge Set* (aZKS) which we will use to build SEEMless. Zero Knowledge Set [MRK03, CHL+05] (ZKS) is a primitive that lets a (potentially malicious) prover commit to a static collection of (label,value) pairs (where the labels form a set) such that: 1) the commitment is succinct and does not leak any information about the committed collection 2) the prover can prove statements about membership/non-membership of labels (from the domain of the labels) in the committed collection with respect to the succinct commitment 3) the proofs are efficient and do not leak any information about the rest of the committed collection. Our primitive, *Append-Only Zero Knowledge Set* (aZKS) generalizes the traditional zero-knowledge set primitive by accounting for append-only updates and characterizing the collection with a leakage function.

Here it is worth pointing out that the notion of soundness one would expect from updates in a ZKS is not obvious. For example, if the expectation is that updates leak absolutely no information about the underlying sets or type of updates (inserts/deletes), then there is no reasonable definition of soundness of updates: any set the prover chooses will be the result of some valid set of updates. In [Lis05], Liskov did not define any soundness notion for updates. In our context, we want to be able to define an append-only ZKS, which makes the expectation of update soundness clear: it should ensure for any label, its value never gets modified and in particular, it never gets deleted.

Here we describe the primitive and informally define its security properties.

Definition 27. Append-Only Zero Knowledge Set is comprised of the algorithms (ZKS.CommitDS, ZKS.Query, ZKS.Verify, ZKS.UpdateDS, ZKS.VerifyUpd)[†] described as follows:
$\triangleright$ $(\mathsf{Com}, \mathsf{st}_{\mathsf{Com}}) \leftarrow \mathsf{ZKS.CommitDS}(1^{\lambda}, \mathsf{D})$: This algorithm takes the security parameter and the datastore to commit to as input, and produces a commitment to the data store and an internal state to pass on to the Query algorithm. Datastore D will be a collection of $(\mathsf{label}, \mathsf{val})$ pairs.
$\triangleright$ $(\pi, \mathsf{val}) \leftarrow \mathsf{ZKS.Query}(\mathsf{st}_{\mathsf{Com}}, \mathsf{D}, \mathsf{label})$: This algorithm takes the state output by ZKS.CommitDS, the datastore and a query label and returns its value ($\perp$ if not present) and a proof of (non-)membership.
$\triangleright$ $1/0 \leftarrow \mathsf{ZKS.Verify}(\mathsf{Com}, \mathsf{label}, \mathsf{val}, \pi)$: This algorithm takes a (label, value) pair, its proof and a commitment by ZKS.CommitDS and verifies the above proof.
$\triangleright$ $(\mathsf{Com}', \mathsf{st}'_{\mathsf{Com}}, \mathsf{D}', \pi_S) \leftarrow \mathsf{ZKS.UpdateDS}(\mathsf{st}_{\mathsf{Com}}, \mathsf{D}, S)$: This algorithm takes in the current server state $\mathsf{st}_{\mathsf{Com}}$, the current state of the datastore and a set $S = \{(\mathsf{label}_1, \mathsf{val}_1), \ldots, (\mathsf{label}_k, \mathsf{val}_k)\}$ of new (label, value) pairs for update. It outputs an updated commitment to the datastore, an updated internal state and an updated version of the datastore and proof π_S that the update has been done correctly.
$\triangleright$ $0/1 \leftarrow \mathsf{ZKS.VerifyUpd}(\mathsf{Com}, \mathsf{Com}', \pi_S)$: This algorithm takes in two commitments to the datastore before and after an update and verifies the above proof.

We require the following security properties of an *append-only* ZKS:
Completeness: For all security parameters λ, for all D_0 whose size is polynomial in λ, for all n and for all update sets $(S_1, \ldots, S_n)$ and for every label,

[†]The original ZKS definition also included a setup algorithm run by a trusted party to generate public parameters used in all the algorithms. Our construction does not need such a set up (we show security in the random oracle model), so we omit it here.

$$\Pr[\widehat{\mathsf{pp}} \leftarrow \mathsf{ZKS.Setup}(1^\lambda); (\mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_{\mathsf{Gen}}) \leftarrow \mathsf{ZKS.Gen}(\widehat{\mathsf{pp}}); (\mathsf{Com}_0, \mathsf{st}^0) \leftarrow \mathsf{ZKS.CommitDS}(\mathsf{pp}, \mathsf{st}_{\mathsf{Gen}}, \mathsf{D}_0);$$
$$\{(\mathsf{Com}_i, \mathsf{st}^i, \mathsf{D}_i, \mathsf{Com}_S^i, \pi_S^i) \leftarrow \mathsf{ZKS.UpdateDS}(\mathsf{pp}, \mathsf{st}^{i-1}, \mathsf{D}_{i-1}, S_i)\}_{i=1}^n; \{(\pi_i, \mathsf{val}_i) \leftarrow \mathsf{ZKS.Query}(\mathsf{pp}, \mathsf{st}_{\mathsf{Com}}^i, \mathsf{D}_i, \mathsf{label})\}_{i=1}^n :$$
$$\{\mathsf{ZKS.VerifyUpd}(\mathsf{pp}, \mathsf{Com}_{i-1}, \mathsf{Com}_i, \mathsf{Com}_S^i, \pi_S^i) = 1\}_{i=1}^n \wedge \{\mathsf{ZKS.Verify}(\mathsf{pp}, \mathsf{Com}_i, \mathsf{label}, \mathsf{val}, \pi_i) = 1\}_{i=0}^n] = 1$$

Soundness: For soundness we want to capture two things: First, a malicious prover $\mathcal{A}^*$ algorithm should not be able to produce two verifying proofs for two different values for the same label with respect to a Com. Second, since the aZKS is append-only, a malicious server should be unable to modify an existing label.

We allow $\mathcal{A}^*$ to win the soundness game if it is able to do either of the following: Output $\mathsf{Com}, \mathsf{label}, \mathsf{val}_1, \mathsf{val}_2, \pi_1, \pi_2$ such that both proofs verify for $\mathsf{val}_1 \neq \mathsf{val}_2$. Or output $\mathsf{Com}_1, \mathsf{Com}_2, \mathsf{label}, \mathsf{val}_1, \mathsf{val}_2, \pi_1, \pi_2, S, \pi_S$ such that π_1 verifies for $\mathsf{Com}_1, \mathsf{val}_1$ for $\mathsf{val}_1 \neq \perp$ and π_2 verifies for $\mathsf{Com}_2, \mathsf{val}_2$ for $\mathsf{val}_2 \neq \mathsf{val}_2$ and the update verifies for $\mathsf{Com}_1, \mathsf{Com}_2, S, \pi_S$.
In general, we want that for all PPT $\mathcal{A}^*$ algorithm there exists a negligible function $\nu()$ such that for all n, λ:

$$\begin{aligned}
\Pr[\widehat{\mathsf{pp}} \leftarrow \mathsf{ZKS.Setup}(1^\lambda)\ ;\ (\mathsf{PK}_{\mathsf{Gen}}, j^*, \mathsf{D}_0, \mathsf{Com}_0,\ & \\
\{(\mathsf{Com}_i, \mathsf{D}_i, \mathsf{Com}_S^i, \pi_S^i)\}_{i=1}^n, \mathsf{label}, \mathsf{val}_1, \mathsf{val}_2, \pi_1, \pi_2) & \\
\leftarrow \mathcal{A}^*(1^\lambda, \mathsf{pp}) : & \\
\{\mathsf{ZKS.VerifyUpd}(\mathsf{pp}', \mathsf{Com}_{i-1}, \mathsf{Com}_i, \mathsf{Com}_S^i, \pi_S^i) = 1\}_{i=1}^n & \\
\wedge\ j^* \in [n] \wedge (\mathsf{val}_1 \neq \perp) \wedge (\mathsf{val}_1 \neq \mathsf{val}_2) & \\
\wedge\ \mathsf{ZKS.Verify}(\mathsf{pp}', \mathsf{Com}_{j^*}, \mathsf{label}, \mathsf{val}_1, \pi_1) = 1 & \\
\wedge\ \Big(\mathsf{ZKS.Verify}(\mathsf{pp}', \mathsf{Com}_{j^*}, \mathsf{label}, \mathsf{val}_2, \pi_2) = 1 & \\
\vee\ \mathsf{ZKS.Verify}(\mathsf{pp}', \mathsf{Com}_{j^*+1}, \mathsf{label}, \mathsf{val}_2, \pi_2) = 1\Big)] \leq \nu(\lambda) &
\end{aligned}$$

where $\mathsf{pp} = (\widehat{\mathsf{pp}}, \mathsf{PK}_{\mathsf{Gen}})$ for notational convenience.

Zero-Knowledge with Leakage: We generalize the definition ZKS [MRK03, CHL$^+$05] by introducing leakage functions in the classical definition. The goal of our privacy definition is to capture the following: we want the query proofs and update proofs to leak no information beyond the query answer (which is a $\mathsf{val}/\perp$ in case of query and a bit indicating validity of the update operation). But often, it is reasonable to tolerate a small leakage to gain more efficiency. To capture this sort of leakage formally, we parameterize our definition with a leakage function. If the leakage function is set to null, then our definition reduces to the classical ZKS definition.

In our definition, we have a setup leakage function $L_1()$ and a leakage function on the updates $L_2()$. $L_1()$ captures what we leak about the collection/datastore when initializing the system and $L_2()$ captures what we leak during an update. We will say that a set is zero knowledge with respect to updates if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$ and a leakage function $L = (L_1, L_2)$ such that for any PPT malicious client algorithms $\mathcal{C}^*$, the outputs of the following two experiments are computationally indistinguishable:

Real:

$$\begin{aligned}\mathsf{pp} \leftarrow \mathsf{ZKS.Setup}(1^\lambda); (\mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_{\mathsf{Gen}}) \leftarrow \\ \mathsf{ZKS.Gen}(\mathsf{pp}); (\mathsf{D}, \mathsf{st}_C) \leftarrow \mathcal{C}^*(1^\lambda, \mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}) \ ; \\ (\mathsf{Com}, \mathsf{st}_P) \leftarrow \mathsf{ZKS.CommitDS}(1^\lambda, \mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_{\mathsf{Gen}}, \mathsf{D}) \ : \\ \mathcal{C}^{*O_Q(\mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_P, -), O_U(\mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_{\mathsf{Com}}, \mathsf{D}, -)}(\mathsf{st}_C) = 1\end{aligned}$$

Simulated:

$$\begin{aligned}(\mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_1) \leftarrow \mathsf{Sim}_1(1^\lambda) \ ; \ (\mathsf{D}, \mathsf{st}_C) \leftarrow \\ \mathcal{C}^*(1^\lambda, \mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}); (\mathsf{Com}, \mathsf{st}_2) \leftarrow \mathsf{Sim}_2(1^\lambda, L_1(D), \mathsf{st}_1) \ : \\ \mathcal{C}^{*\mathsf{Sim}_3(\mathsf{st}_2, -), \mathsf{Sim}_4(\mathsf{st}_2, -)}(\mathsf{st}_C) = 1\end{aligned}$$

We have oracle O_Q, the query oracle which on query label_i, will output $(\pi_i, \mathsf{val}_i) \leftarrow \mathsf{ZKS.Query}(\mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_{\mathsf{Com}}, \mathsf{D}, \mathsf{label}_i)$ and O_U is the update oracle which on getting a set of updates $S = \{(\mathsf{label}_i, \mathsf{val}_i)\}$ as input will first check that S is an allowed update set and output $(\mathsf{Com}', \mathsf{st}'_{\mathsf{Com}}, \pi_S) \leftarrow \mathsf{ZKS.UpdateDS}(\mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_{\mathsf{Gen}}, \mathsf{D}, S)$.

For the simulated game, Sim_3 answers to membership queries and outputs $(\pi_i, \mathsf{val}_i) \leftarrow \mathsf{Sim}_3(\mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{label}_i, \mathsf{val}_i, \mathsf{st}_2)$ and Sim_4 emulates the update oracle O_U. Sim_4 gets a leakage on the set to be updated if it is an allowed update set and outputs $(\mathsf{Com}', \mathsf{st}'_S, \pi_S) \leftarrow \mathsf{Sim}_4(\mathsf{pp}, \mathsf{PK}_{\mathsf{Gen}}, \mathsf{st}_2, L_2(S))$

Append-Only Strong Accumulator: Finally, we remark that the primitive *strong accumulator* (SA) [CHKO08] can be extended to a new primitive of *append-only strong accumulator* (aSA) trivially from the definition of aZKS. An aSA is essentially a aZKS with completeness and soundness and *without* the privacy requirement.

5.3 SEEMless Construction

In this section, we will describe SEEMless: our construction of VKD from *append-only* zero-knowledge sets (aZKS). We first review the construction for CONIKS, then give an informal overview of our SEEMless construction and then describe the construction more formally. In our construction, we will be using a *hash chain*. Hash Chain is a classical authentication data structure that chains multiple data elements by successively applying cryptographic hashes, e.g. a hash chain that hashes elements a, b, c is $H(c, H(b, H(a))$.

5.3.1 An overview of the CONIKS construction

In a verifiable directory, as described in the introduction, every epoch the server will publish a commitment to the current state of the directory; this will allow the server to respond to username queries with the corresponding public key and a proof that that is correct according to the committed database. CONIKS is implicitly based on the zero knowledge set we describe in Section **??**, although they do not explictly describe it as such and they do not use the append-only property. Recall that a zero knowledge set, as described in Section **??**, allows the server to give a concise commitment to a set of $(\mathsf{label}, \mathsf{val})$ pairs, and to give proofs

for each query that reveal nothing beyond the query, the response, and the number of pairs in the directory. Each epoch the server will modify the zks with any updates to correspond to the latest directory and publish the resulting zks commitment; it will then respond to each Query with a zks proof. The system should also ensure that if two users at some point compare their root commitments and find that they match, they will be assured that all previous responses were consistent. To allow this, the server will actually publish the head of a hash chain containing all of the zks commitments.

This construction has two issues. First, in terms of privacy, because of the way the the zks is modified, it reveals additional timing information about when Alice's key is updated, leading to the tracing attack described in Appendix **??**. On the efficiency side, the issue with this construction is that in order to audit her key history Alice will have to obtain a zks proof for every epoch, even if she has only made a few updates.

Key History The KeyHist functionality is not provided in CONIKS, as such. Instead, to ensure that the key served by a CONIKS provider for her was always correct, a user `alice` would either need to remain always online and verify the response to the Query request for her key at each epoch, or, retroactively check these responses.

5.3.2 Intuition behind our SEEMless construction

To explain the intuition behind our construction, we take the CONIKS construction as a starting point. However, here we want two additional properties. First, we want to allow a user to check the history of when her key has been updated in time proportional to the number of updates rather than the number of edits. That means we need to do something more sophisticated than having the user query her key in every epoch. The second is that we want that all of the proofs leak as little information as possible about users' update patterns. We build up to our construction gradually. (We will describe our construction generally in terms of $(\mathsf{label}, \mathsf{val})$ pairs, but in our application label will be a username, and val will be an associated public key.)

Attempt 1: To address the efficiency issue, we need to ensure user can see changes to his key even if he doesn't check the update when they occur. For instance, suppose that in the above construction Alice performs a KeyHist query by only verifying ZKS proofs for the epochs in which she changed her key. An adversarial server could change Alice's key for a few epochs, and then change it back to the correct value, and if Alice does not request ZKS proofs for those epochs (because she doesn't expect her key to have changed during that time), then she will never detect it. We prevent this by using an *append only* ZKS as described in Section **??**, which guarantees that all changes will be retained. We use the aZKS to store $((\mathsf{label}||\mathsf{version}), \mathsf{val})$ pairs, and when Bob queries he will get Alice's latest version number and an aZKS proof for this $(\mathsf{label}||\mathsf{version})$. Alice on KeyHist query will get proofs for the values associated with $(\mathsf{label}||1) \ldots (\mathsf{label}||n)$ where n is the number of updates she's made so far.

We note that this also addresses the privacy issue discussed above. Instead of modifying the value for an existing label in the ZKS we add a new label; privacy of the aZKS then says that the update proof and commitment reveal nothing about which label was added.

The issue with this construction is that we have no way to ensure that the server shows Bob the correct latest version. For example a corrupt server could show Bob an out of date key. Or if Alice has performed n updates, it could correctly show Alice the values for $\mathsf{label}||1, \ldots \mathsf{label}||n$, but then show Bob a much higher version.

Attempt 2: To prevent the server from showing Bob an out of date key, we modify the construction so that at every epoch we store 2 aZKS, one $\mathsf{aZKS}_{\mathsf{all}}$ as described above, and a second one $\mathsf{aZKS}_{\mathsf{old}}$ that stores all of the out of date versions. When Bob queries for Alice's key he will receive a proof for the current version with respect to $\mathsf{aZKS}_{\mathsf{all}}$ as above, but he will now also recive a proof for the previous version with respect to $\mathsf{aZKS}_{\mathsf{old}}$. When Alice performs a $\mathsf{KeyHist}$ check, she now additionally checks that the checks that at each update the previous version is added to $\mathsf{aZKS}_{\mathsf{old}}$.

This does not prevent the server from showing Bob a version where version is much higher than Alice's real latest update. One approach to prevent this would be to have the server provide a that $|\mathsf{version}$ is not in the $\mathsf{aZKS}_{\mathsf{all}}$ for any higher version than Alice's current version. However, this potentially has a very high cost for Alice, proportional to the total possible number of updates, i.e. the number of epochs. Instead, we want to achieve the same guarantee but reduce Alice's work. To do this, we have the server add a "marker" node, on every 2^ith update Alice makes. When Bob queries for Alice's key, he will also check that the previous marker node is in the $\mathsf{aZKS}_{\mathsf{all}}$. When Alice performs a $\mathsf{KeyHist}$ check, she will also check that no higher marker nodes are included in the $\mathsf{aZKS}_{\mathsf{all}}$. Because we mark only the 2^ith updates, this cost is now only logarithmic in the number of epochs.

Final construction The previous attempt captures the core of our construction. The one additional aspect is that the append only ZKS requires that someone verify the update proofs to ensure that each update is indeed only adding values to the set. We could have the users all perform this check, but that would again require them to do work linear in the number of epochs. Instead, we observe that the privacy guarantees of the aZKS mean that anyone can be given access to these proofs, so we can open verification up to anyone willing to perform this task; we call these parties auditors. See Section 5.1.1 for more discussion.

5.3.3 SEEMless construction:

In the construction we assume that the server's identity and public key is known to each user and auditor and all the messages from the server are signed under the server's key, so that the server cannot be impersonated.

Along with the steps of our construction, we will provide a running example for expositional clarity.

> Consider a VKD with 2 chat client users, in which the labels are usernames and the values are the corresponding public keys. Suppose at some point in time between server epochs $t-1$ and t, `alice` requested registration with her first ever public key $\mathsf{PK}_{\mathtt{a},1}$ and an existing user `bob` requested to update his public key for the 2nd time to $\mathsf{PK}_{\mathtt{b},3}$. These values will reflect in the VKD at epoch t. Previously, `bob` registered his first key $\mathsf{PK}_{\mathtt{bob},1}$ at server epoch 10 and updated it to $\mathsf{PK}_{\mathtt{bob},2}$ at server epoch 100.

Notation	Description
T	A table containing usernames (labels) and all epochs at which their values were updated until the current epochs.
$D_{\mathsf{all},t}$	A datastore containing all the label-value pairs until epoch t.
$D_{\mathsf{old},t}$	A datastore containing all the *stale* label-value pairs until epoch t.
Dir_t	The ordered pair $(D_{\mathsf{all},t}, D_{\mathsf{old},t})$.
$\mathsf{st}_{\mathsf{all},t}$	The internal state of the aZKS built on $D_{\mathsf{all},t}$.
$\mathsf{st}_{\mathsf{old},t}$	The internal state of the aZKS built on $D_{\mathsf{old},t}$.
st_t	The internal state of the VKD at epoch t.
S_t	The set of elements (updates to existing usernames and new registrations) added between epochs $t-1$ and t.
α_i	The version number of the value corresponding to label_i. Sometimes written as $\alpha_{\mathsf{label}_i}$

Table 5.1: *Notation for our VKD construction.*

▷ $\mathsf{VKD.Publish}(\mathsf{Dir}_{t-1}, \mathsf{st}_{t-1}, S_t)$: At every epoch, the server gets a set S_t of (label, value) pairs that have to be added to the VKD. The server first checks if the label already exists for some version $\alpha - 1$, else sets $\alpha = 1$. It adds a new entry $(\mathsf{label} \mid \alpha, \mathsf{val})$ to the "all" aZKS and also adds $(\mathsf{label} \mid \alpha - 1,)$ to the "old" aZKS if $\alpha > 1$. If the new version $\alpha = 2^i$ for some i, then the server adds a marker entry $(\mathsf{label} \mid \mathsf{mark} \mid i$,"marker") to the "all" aZKS. The server computes commitments to both the aZKS, and adds them to the hash chain to obtain a new head Com_t. It also produces a proof Π^{Upd} consisting of the previous and new pair of aZKS commitments $\mathsf{Com}_{\mathsf{all},t-1}, \mathsf{Com}_{\mathsf{all},t}$ and $\mathsf{Com}_{\mathsf{old},t-1}, \mathsf{Com}_{\mathsf{old},t}$ and the corresponding aZKS update proofs.

Remark 7. *Note that the directory of (username, public-key) pairs is never published. The published commitments are privacy-preserving.*

- For each $\mathsf{label}_i \in S_t$, such that label_i is present in T, append t to the list of epochs corresponding to label_i. For a new label_i, add $(\mathsf{label}_i, \{t\})$ to T. Let β_i be the number of entries in T for label_i (not including the newest entry, t). $\beta_i = 0$ if label_i has no previous entries associated with it. $\alpha_i = \beta_i + 1$ is the version number of the new key. In our example, since the label `alice` is being added to the VKD for the first time, $\alpha_{\texttt{alice}}$ would be 1. For `bob` a third value is being added, so $\alpha_{\texttt{bob}}$ would be 3. Based on the version number, we will create the (label, value) pairs to be added in the "all" aZKS.
- If for any label_i, $\alpha_i = 2^y$ for some $y \geq 0$ add the following (label, value) pair to the marker set M_t: $(\mathsf{label}_i \mid \mathsf{mark} \mid y$,"marker entry 2^y for label_i").
- Compute new update set to update $D_{\mathsf{all},t-1}$ to $D_{\mathsf{all},t}$: $S'_t = \{(\mathsf{label}'_i, \mathsf{val}_i) \mid (\mathsf{label}_i, \mathsf{val}_i) \in S_t \wedge \mathsf{label}'_i = \mathsf{label} \mid \alpha_i\}$. See Table **??** for concrete examples. Compute the update on the "all" aZKS for the set $S'_t \cup M_t$: $(\mathsf{Com}_{\mathsf{all},t}, \mathsf{st}_{\mathsf{all},t}, D_{\mathsf{all},t}, \pi_{S_t}) \leftarrow \mathsf{ZKS.UpdateDS}(\mathsf{st}_{\mathsf{all},t-1}, D_{\mathsf{all},t-1}, S'_t \cup M_t)$.
- Form a new set of (label, value) pairs to be added to the "old" aZKS. Let S_t^{old} be the list of entries in S_t that are updates to existing labels. For each $\mathsf{label}_i \in S_t^{\mathsf{old}}$, concatenate it with its version $\alpha_i - 1$ before the update. Hence, let $S_t^{\mathsf{old}'} = \{(\mathsf{label}'_i, \mathsf{null}) \mid (\mathsf{label}_i, \mathsf{val}_i) \in S_t^{\mathsf{old}} \wedge \mathsf{label}'_i = \mathsf{label}_i \mid \alpha_i - 1\}$. For the "old" tree, compute $(\mathsf{Com}_{\mathsf{old},t}, \mathsf{st}_{\mathsf{old},t}, D_{\mathsf{old},t}, \pi_{S_t^{\mathsf{old}}}) \leftarrow \mathsf{ZKS.UpdateDS}(\mathsf{st}_{\mathsf{old},t-1}, D_{\mathsf{old},t-1}, S_t^{\mathsf{old}'})$.

See examples of these sets in Table **??**.

– Update the hash chain: $\mathsf{Com}_t = H(H(\mathsf{Com}_{\mathsf{all},t}, \mathsf{Com}_{\mathsf{old},t}), \mathsf{Com}_{t-1})$.

Output Com_t, $\mathsf{st}_t = (\mathsf{st}_{\mathsf{all},t}, \mathsf{st}_{\mathsf{old},t}, T, \mathsf{Dir}_t)$ and $\Pi_t^{\mathsf{Upd}} = (\pi_{S_t}, \pi_{S_t^{\mathsf{old}}}, \mathsf{Com}_{\mathsf{all},t}, \mathsf{Com}_{\mathsf{old},t}, \mathsf{Com}_{\mathsf{all},t-1}, \mathsf{Com}_{\mathsf{old},t-1}, \mathsf{Com}_{t-2})$.

▷ $\mathsf{VKD.Query}(\mathsf{st}_t, \mathsf{Dir}_t, \mathsf{label})$: When a client Bob queries for Alice's label, he should get the val corresponding to the latest version α for Alice's label and a proof of correctness. Bob gets three proofs in total: First is the membership proof of $(\mathsf{label} \mid \alpha, \mathsf{val})$ in the "all" aZKS. Second is the membership proof of the most recent marker entry $(\mathsf{label} \mid \mathsf{mark} \mid a)$ for $\alpha \geq 2^a$. And third is non membership proof of $\mathsf{label} \mid \alpha$ in the "old" aZKS. Proof 2 ensures that Bob is not getting a value higher than Alice's current version and proof 3 ensures that Bob is not getting an old version for Alice's label.

Retrieve latest version number α for queried label from table T (by counting the number of epoch entries for label). Let β be the largest power of 2 less than α such that $\beta = 2^b$. Compute the following proofs:

– $(\pi_1, \mathsf{val}_1) \leftarrow \mathsf{ZKS.Query}(\mathsf{st}_{\mathsf{all},t}, D_{\mathsf{all},t}, \mathsf{label} \mid \alpha)$: This gives a proof of membership of the latest version of label in the "all" aZKS and its corresponding value.

– $(\pi_2, \mathsf{val}_2) \leftarrow \mathsf{ZKS.Query}(\mathsf{st}_{\mathsf{all},t}, D_{\mathsf{all},t}, \mathsf{label} \mid \mathsf{mark} \mid b)$: This gives a proof of membership of the marker entry $\beta = 2^b$ which is the marker entry right before the current version α.

– $(\pi_3, \mathsf{val}_3) \leftarrow \mathsf{ZKS.Query}(\mathsf{st}_{\mathsf{old},t}, D_{\mathsf{old},t}, \mathsf{label} \mid \alpha)$: This gives a proof of non membership of the latest version in the "old" aZKS making sure that the claimed "latest" version is not outdated.

Output $\Pi = (\pi_1, \pi_2, \pi_3, \mathsf{Com}_{\mathsf{all},t}, \mathsf{Com}_{\mathsf{old},t}, \mathsf{Com}_{t-1}))$ and $\mathsf{val} = (\mathsf{val}_1, \mathsf{val}_2, \perp)$ and α.

> In our example, if `alice` requested to see `bob`'s public key at epoch t, `alice` would receive proofs for $\mathtt{bob}|3 \in \mathsf{aZKS}_{\mathsf{all},t}$ with value $\mathsf{PK}_{\mathtt{b},3}$ and $\mathtt{bob}|\mathsf{mark}|1 \in \mathsf{aZKS}_{\mathsf{all},t}$ and lastly, $\mathtt{bob}|3 \notin \mathsf{aZKS}_{\mathsf{old},t}$. Additionally, `alice` will receive $\mathsf{Com}_{\mathsf{all},t}, \mathsf{Com}_{\mathsf{old},t}, \mathsf{Com}_{t-1}$.

▷ $\mathsf{VKD.QueryVer}(\mathsf{Com}_t, \mathsf{label}, \mathsf{val}_t, \pi_t, \alpha)$: The client checks each membership or non-membership proof, and the hash chain. Also check that version α as part of proof is less than current epoch t.

▷ $\mathsf{VKD.KeyHist}(\mathsf{st}_t, \mathsf{Dir}_t, \mathsf{label})$: The server first retrieves all the update epochs $t_1, \ldots, t_\alpha$ for label versions $1, \ldots, \alpha$ from T, the corresponding $\mathsf{Com}_{\mathsf{all},t_1-1}, \mathsf{Com}_{\mathsf{all},t_1}, \ldots, \mathsf{Com}_{\mathsf{all},t_\alpha-1}, \mathsf{Com}_{\mathsf{all},t_\alpha}$ and $\mathsf{Com}_{\mathsf{old},t_1}, \ldots, \mathsf{Com}_{\mathsf{old},t_\alpha}$ and the hashes necessary to verify the hash chain: $H(\mathsf{Com}_{\mathsf{all},0}, \mathsf{Com}_{\mathsf{old},0}), \ldots, H(\mathsf{Com}_{\mathsf{all},t}, \mathsf{Com}_{\mathsf{old},t})$. For versions $i = 1$ to n, the server retrieves the val_i for t_i and version i of label from Dir_{t_i}. Let $2^a \leq \alpha < 2^{a+1}$ for some a where α tis the current version of the label. The server generates the following proofs (together called as Π):

1. **Correctness of** Com_{t_i} **and** Com_{t_i-1}: For each i, output Com_{t_i} Com_{t_i-1}. Also output the values necessary to verify the hash chain: $H(\mathsf{Com}_{\mathsf{all},0}, \mathsf{Com}_{\mathsf{old},0}), \ldots, H(\mathsf{Com}_{\mathsf{all},t}, \mathsf{Com}_{\mathsf{old},t})$.
2. **Correct version** i **is set at epoch** t_i**:** For each i: Membership proof for $(\mathsf{label} \mid i)$ with value val_i in the "all" aZKS with respect to Com_{t_i}.

3. **Server couldn't have shown version $i-1$ at or after t_i** : For each i: Membership proof in "old" aZKS with respect to Com_{t_i} for $(\mathsf{label}|\ i-1)$.
4. **Server couldn't have shown version i before epoch t_i**: For each i: Non membership proof for $(\mathsf{label}|\ i)$ in "all" aZKS with respect to Com_{t_i-1}
5. **Server can't show any version from $\alpha+1$ to 2^{a+1} at epoch t or any earlier epoch:** Non membership proofs in the "all" aZKS with respect to Com_t for $(\mathsf{label}|\ i+1),(\mathsf{label}|\ i+2),\ldots,(\mathsf{label}|\ 2^{a+1}-1)$.
6. **Server can't show any version higher than 2^{a+1} at epoch t or any earlier epoch:** Non membership proofs in "all" aZKS with respect to Com_t for marker nodes $(\mathsf{label}|\ \mathsf{mark}|\ a+1)$ up to $(\mathsf{label}|\ \mathsf{mark}|\ \log t)$.

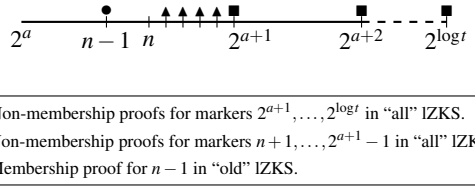

Figure 5.1: *Versions Proofs*

▷ $\mathsf{VKD.HistVer}(\mathsf{Com}_t,\mathsf{label},\{(\mathsf{val}_i,t_i)\}_{i=1}^n,\Pi^{\mathsf{Ver}})$: Verify each of the above proofs.

In our example, if bob queried for his key history at epoch t, he would check the following,

1. $\mathsf{Com}_{10},\mathsf{Com}_9,\mathsf{Com}_{100},\mathsf{Com}_{99},\mathsf{Com}_t$ and Com_{t-1} and the hashes necessary to verify the the hashchain $H(\mathsf{Com}_{\mathsf{all},0},\mathsf{Com}_{\mathsf{old},0}),\ldots,H(\mathsf{Com}_{\mathsf{all},t},\mathsf{Com}_{\mathsf{old},t})$.
2. bob$|1$ exists and has value $\mathsf{PK}_{\mathrm{b},1}$ in $\mathsf{aZKS}_{\mathsf{all},10}$, bob$|2$ exists and has value $\mathsf{PK}_{\mathrm{b},2}$ in $\mathsf{aZKS}_{\mathsf{all},100}$ and bob$|3$ exists and has value $\mathsf{PK}_{\mathrm{b},3}$ in $\mathsf{aZKS}_{\mathsf{all},t}$.
3. bob$|1 \in \mathsf{aZKS}_{\mathsf{old},100}$ and bob$|2 \in \mathsf{aZKS}_{\mathsf{old},t}$.
4. bob$|1 \notin \mathsf{aZKS}_{\mathsf{all},9}$, bob$|2 \notin \mathsf{aZKS}_{\mathsf{all},99}$ and bob$|3 \notin \mathsf{aZKS}_{\mathsf{all},t-1}$.
5. Since bob's version is $3 < 2^2 = 4$, nothing to check here.
6. bob$|\mathsf{mark}|2\ldots$, bob$|\mathsf{mark}|\log t \notin \mathsf{aZKS}_{\mathsf{all},t}$

Remark 8. *The client software runs* VKD.HistVer *to monitor the history of the user keys. This software either downloads the entire proof from the server each epoch it runs* VKD.HistVer *(when the software is re-installed or the user installs it on a new device) or caches parts of the proof from the first run of* VKD.HistVer *to use in the subsequent verifications. We experimentally evaluate the performance for* VKD.HistVer *with and without caching in Section* **??**.

▷ $\mathsf{VKD.Audit}(t_1,t_n,\{(\mathsf{Com}_t,\Pi_t^{\mathsf{Upd}})\}_{t=t_1}^{t_n})$: Auditors will audit the commitments and proofs to make sure that no entries ever get deleted in either aZKS. They do so by verifying the update proofs Π^{Upd} output by the server. They also check that at each epoch both aZKS commitments are added to the hash chain. Note that, while the Audit interface gives a monolithic audit algorithm, our audit is just checking the

updates between each adjacent pair of aZKS commitments, so it can be performed by many auditors in parallel. For security, it is sufficient to have at least one honest auditor perform audits over each adjacent pair.

Remark 9. *In our implementation, the marker entry doubles up as the normal key entry for that version, where the version number is* $\log(k), k$ *being the count of number of updates for a specific user. We use a special symbol* `mark` *for the marker entries. For example, the* 3*rd key for* `bob` *will be saved as* `bob`|3 *and the* 1*st key for* `alice` *will be saved as* `alice`|`mark`|0*, (since* $2^0 = 1$*), hence only two new entries are made for any user update request.*

Remark 10. *We describe our construction with a hash chain to for simplicity of exposition. However, we note that this could instead be replaced with a Merkle tree (we describe this data structure in Section* **??***) built over the list of all commitments to date. This would result in slightly higher update and audit costs (adding a new entry to the end of this list would require up to a logarithmic number of hashes), but would significantly reduce the cost of history queries (from linear in the number of epochs to logarithmic). We discuss this in Section* **??** *(Update, Gethistory and Audit experiments).*

5.3.4 Privacy of SEEMless

We have the following leakage for SEEMless:

Publish: For each label that was updated, if this is the first update since the adversary queried for it via Query then add it to set Q_{Query}, and if it was previously queried to KeyHist then add it to set Q_{KeyHist}. The leakage from this query is the number of new registrations and of key updates in the latest epoch, and the sets $Q_{\mathsf{Query}}, Q_{\mathsf{KeyHist}}$.

Query: The leakage from this query is the version number of the queried key and the epoch at which it was last updated.

KeyHist: There is no additional leakage from this query.

Interpreting this leakage: A party who only acts as an auditor learns only the numbers of keys added and keys updated each epoch. If that party additionally acts as a user (Alice) performing KeyHist queries, the combined leakage may reveal when her keys are updated (even if she does not perform more KeyHist queries), but that is expected to be something Alice knows since she is the one requesting the updates. If Alice additionally queries for Bob's key, the leakage reveals the version number of Bob's current key and the epoch when it was last updated, and may reveal when that key is no longer valid (because Bob performed an update), but will not reveal anything about subsequent or previous updates.

Remark 11. *Note that, while the functionality inherently allows Bob to learn Alice's entire update history if he queries for her key every epoch, we assume as discussed in Section 5.1.1 that the server limits who is allowed to query for Alice's key to e.g. her contacts. Our goal here then is to guarantee that Bob cannot learn about Alice's update history without performing these queries.*

Chapter 6

Proofs of Ignorance and Applications to 2-Message Witness Hiding

We define proofs of ignorance (PoI) with respect to a language $L \in \mathsf{NP}$ and a distribution $\mathcal{D}$ over instances in L. A PoI proof system for $(L, \mathcal{D})$, consists of a triplet of PPT algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify})$, such that Setup generates a common reference string (CRS). Any party can use the algorithm Gen, together with the CRS, to sample an instance x together with a PoI π, such that x is distributed according to $\mathcal{D}$.

The soundness guarantee we want is that if the verification algorithm Verify on input (CRS, x, π) outputs 1 then the prover who generated (x, π) does not know a witness corresponding to x. This is formalized by defining a computationally indistinguishable common reference string CRS' such that *for any* (x, π) if the algorithm Verify accepts (x, π) with respect to CRS', then it must be the case that $x \notin L$. Our intuition here is that since given CRS' one cannot generate a valid PoI together with a witness (since a witness does not exit), and since CRS' is computationally indistinguishable from CRS, it follows that given CRS one cannot generate a valid PoI together with a witness. We refer the reader to Definition 33 for the formal definition.

We also provide an alternative definition, which we refer to as *Trapdoor PoI*. In this definition, the CRS is generated together with a trapdoor td such that given (x, π, td) where π is a valid PoI with respect to x, it is easy to compute a valid witness w corresponding to x. The soundness guarantee is that given only the CRS (without the trapdoor), it is computationally infeasible for an adversary to output (x, π, w) such that π is a valid PoI for x and w is a witness for x. We refer the reader to Definition 34 for the formal definition. Jumping ahead, we note that we use a trapdoor PoI in our 2-message witness hiding protocol.

Constructing Proofs of Ignorance

Let us start by giving a proof of ignorance protocol for the DDH language L_{DDH}. This language consists of elements of the form $x = (g^y, g^z, g^{yz})$, where g is a fixed generator of a primed order group $\mathbb{G}$, and where the corresponding witness is $w = (y, z)$. We construct a proof of ignorance for L_{DDH} with respect to the uniform distribution $\mathcal{U}_{\mathsf{DDH}}$, which samples an element in L_{DDH} by choosing at random $y, z \leftarrow \mathbb{Z}_p$ where $p = |\mathbb{G}|$ and outputting $x = (g^y, g^z, g^{yz})$.

The CRS of our PoI protocol is simply a random element in L_{DDH}; namely, $\mathsf{CRS} = (g^y, g^z, g^{yz})$ for randomly chosen $y, z \leftarrow \mathbb{Z}_p$. To generate a random element in L_{DDH} together with a PoI, simply choose at random $r, s \leftarrow \mathbb{Z}_p$, and let $x = (g^{yr}, g^{zs}, g^{yzrs})$ and the PoI be $\pi = (r, s)$. To check the validity of π simply check that indeed $x = \left((g^y)^r, (g^z)^s, (g^{yz})^{rs}\right)$. It is easy to see that by the discrete log assumption, if a prover given CRS generates an accepting (x, π) then the prover does not know a valid witness corresponding to x.

What enabled us to construct a proof of ignorance protocol for L_{DDH} is the fact that L_{DDH} is a random self-reducible language. More generally, we construct a proof of ignorance protocol for *any random self-reducible language*, where a language $L \in \mathsf{NP}$ is said to be random self-reducible if there exists a PPT algorithm f that converts any $x \in L$ into a uniformly distributed $x' = f(x, r) \in L$.

Theorem 1 (Informal)**.** *If a language L is hard on average and is random self-reducible with respect to a distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ then there exists a proof of ignorance protocol for $(L, \mathcal{D})$.*

We also construct a trapdoor PoI (which is the primitive we use in our 2-message witness hiding protocol) for any random self-reducible language that is *witness preserving*. Loosely speaking, we say that a random self-reducible language $L \in \mathsf{NP}$ is witness preserving, if given a valid witness w for x, and given x' and r such that $x' = f(x, r)$, one can efficiently compute a valid witness for w' for x', and similarly, given x, x', r, w' such that $x' = f(x, r)$ and such that w' is a valid witness for x', one can efficiently compute a valid witness w for x. It is easy to see that L_{DDH} is a witness preserving random self-reducible language with respect to the uniform distribution $\mathcal{U}_{\mathsf{DDH}}$.

Theorem 2 (Informal)**.** *If a language L is hard-to-extract with respect to a distribution $\mathcal{D}$,[*] and is witness-preserving random self-reducible with respect to $\mathcal{D}$, then there exists trapdoor proof of ignorance protocol for $(L, \mathcal{D})$.*

We prove these theorems in Appendix 6.3.

6.0.5 Witness Hiding from Proofs of Ignorance

Witness hiding proofs were introduced by Feige and Shamir [FS90]. Intuitively, an interactive proof for an NP language L is said to be *witness hiding* if participating in the protocol does not help the verifier find a witness corresponding to the underlying instance. Witness hiding is a natural weakening of the security requirement of zero-knowledge, and can replace zero knowledge (ZK) in several applications.

Despite the fact that witness-hiding is a weaker requirement than ZK, almost all our candidate constructions of witness hiding protocols for NP are themselves zero-knowledge (or weak zero-knowledge). In particular, it is known that there do not exist 2-message zero-knowledge protocols for NP [GO94], and indeed constructing a 2-message witness hiding protocol for NP remained an important open problem.

In this work, we construct a 2-message witness hiding protocol for NP, using the following ingredients. Fix any super-polynomial function $T = \lambda^{\omega(1)}$, where λ is the security parameter and fix any $\varepsilon > 0$. The ingredients are:

[*]By hard-to-extract we mean that for every poly-size adversary$\mathcal{A}$, the probability that $\mathcal{A}$ outputs the witness for an instance drawn from $\mathcal{D}$ is negligible.

1. T-secure trapdoor proof of ignorance protocol.
2. T-secure non-interactive witness indistinguishable (NIWI) proofs.[†]
3. T-secure rerandomizable encryption which is strong KDM secure with ε-bounded auxiliary input. Namely, for any PPT distribution $\mathcal{D}$ that (maliciously) samples $(\mathsf{PK}, \mathsf{aux})$ such that $|\mathsf{aux}| \leq |\mathsf{PK}|^{\varepsilon}$, the following holds: If $(\mathsf{PK}, \mathsf{aux}, \mathsf{Enc}(1)) \approx (\mathsf{PK}, \mathsf{aux}, \mathsf{Enc}(0))$ then for any possibly inefficient function f such that $f(\mathsf{PK}, \mathsf{aux}) \in \{0,1\}^{\mathsf{poly}(\lambda)}$, $(\mathsf{PK}, \mathsf{aux}, \mathsf{Enc}(f(\mathsf{PK}, \mathsf{aux}))) \approx (\mathsf{PK}, \mathsf{aux}, \mathsf{Enc}(0^{\mathsf{poly}(\lambda)}))$.[‡]

Theorem 3 (Informal)**.** *Assuming the existence of the ingredients above, there exists a 2-message witness hiding protocol with adaptive soundness for* NP.

Concurrently and independently of our work, Bitansky-Khurana-Paneth [BKP18b] constructed two-message witness hiding arguments for NP assuming super-polynomial hardness of fully homomorphic encryption (and other standard assumptions).

We note that Groth, Ostrovsky and Sahai [**?**] construct a T-secure NIWI from the T security of the DLIN assumption. In this work, we construct a T-secure trapdoor proof-of-ignorance protocol under the same assumption. Thus, we obtain the following corollary.

Corollary 2. *There exists a 2-message witness hiding protocol with adaptive soundness for* NP *assuming that DLIN is T-secure and assuming a T-secure rerandomizable encryption which is strong KDM secure with ε-bounded auxiliary input.*

Remark 12. *We have several constructions of T-secure rerandomizable encrypton schemes from standard assumptions, such as the T-security of DDH, or the T-security of quadratic residuosity. We do not know how to prove that these schemes are strong KDM secure with ε-bounded auxiliary input under standard assumptions, but we do not have any evidence that they are not.*

Strong KDM security with unbounded auxiliary input has been shown to be impossible under learning with errors (LWE) assumption [FKP19]. However the impossibility crucially relies on the fact that the auxiliary input is long, in particular it grows with the size of the public key. For our purposes, we need the auxiliary input to be significantly shorter than the public key ($|\mathsf{aux}| \leq |\mathsf{PK}|^{\varepsilon}$).

Related Work on Witness Hiding Protocols

3-Message Protocols Most witness hiding protocols in the literature are also zero-knowledge. It is known that 3-message ZK protocols with black-box simulation do not exist [GK96]. Several 3-message ZK protocols with non-black-box simulation were constructed: Most known constructions are based on auxiliary-input knowledge assumptions [BP04, HT98, CD09, BCC$^+$17],[§] and very recently Bitanski *et. al.* [BKP18a] gave a construction based on multi-collision-resistant hash functions. 3-message ZK protocols have also been

[†] By NIWI we mean a one message WI proofs *without* a CRS.

[‡] A different flavor of strong KDM security was recently given by Canetti et al. [CCRR18].

[§] These assumptions are believed to be false assuming that indistinguishability obfuscation exists [BCPR16].

constructed under standard assumptions in restricted adversarial models, where either the verifier or the prover is assumed to be uniform [BCPR16, BBK$^+$16].

The only example of a 3-message WH protocol which is not ZK, is by Bitansky and Paneth [BP12]. They rely on the assumption that there exist auxiliary input point-function obfuscators that satisfy a distributive requirement.[¶]

2-Message Protocols It is well known that 2-message zero-knowledge protocols do not exist [GO94]. Indeed, constructing a 2-message witness hiding protocol for all of NP remained an elusive task. However, significant progress on this question has been made.

Feige and Shamir [FS90] observed that if a language has two *independent* witnesses then witness indistinguishability implies witness hiding. Importantly, constructing a 2-message witness indistinguishable (WI) protocol, and even a non-interactive WI protocol, is known for all of NP under various (standard) assumptions [DN00, BOV05, **?**]. Pass [Pas03] used this observation to construct a 2-message ZK protocol with quasi-polynomial simulation for all of NP. Roughly speaking, his protocol follows the following blueprint: The verifier sends $y = f(r)$ where r is a random string and f is one-way function that is invertible in quasi-polynomial time, and such that every element in the range has a pre-image. The prover then sends a commitment c and gives a WI proof that $x \in L$ or that c commits to r' such that $f(r') = y$. Simulation works by inverting y in quasi-polynomial time and using that as a witness in the WI proof.

We note that this protocol is WH for super-polynomial hard languages. More generally, any protocol that is ZK with T-time simulation is WH for T-hard languages. The reason is that if (by contradiction) the resulting protocol is not WH, then one can find a witness in time roughly T, by simulating the prover (in time T) and then extracting a witness from the simulated transcript (in polynomial time). This contradicts the T-hardness of the language.

In this work, we construct a WH protocol for *all of* NP. We follow the blue-print of Pass, where we use our proof of ignorance protocol to construct an independent witness, and as a result avoid putting any restrictions on the hardness of L. We refer to Section 6.1 for details.

Other WH Protocols Jain *et. al.* [JKKR17] construct 2-message WH protocols (and distributional ZK) under standard assumptions, in the *delayed input* setting, where the instance is only determined by the prover in the last round.

There have been several works on witness hiding protocols for languages where each instance has a unique witness. Haitner, Rosen and Shaltiel [HRS09] showed that such languages do not have constant round public-coin witness hiding protocols which are based on standard assumptions via some restricted types of black-box reductions. Deng et al. [DSYC17] showed that for any such language L, and for any distribution $\mathcal{D}$ over L that has an indistinguishable counterpart distribution over a relation with multiple witnesses, it holds that any witness indistinguishable protocol is witness hiding with respect to $\mathcal{D}$. Bellare and Palacio [BP02] showed that the Schnorr and Guillou-Quisquater 3-message identification protocols are witness hiding under the assumptions of one-more Discrete Log and one-more RSA.

[¶]This assumption is believed to be false assuming that Virtual Grey Box obfuscation exists [BCKP17].

6.1 Technical Overview: Witness Hiding Arguments

The main technical contribution of this work is a 2-message witness hiding protocol from proofs of ignorance. This protocol, as well as its analysis, contain a novel non-black-box technique, which is of conceptual interest. Starting with the seminal work of Barak [BGI+01], most non-black-box techniques use the code of the cheating verifier V^* in an "efficient manner" (eg., the simulator commits to the code of V^* and proves that this code satisfies a desired property). To prove that our protocol is witness hiding, we do not use the code of V^* in an efficient manner; rather, we resort to a form of case analysis. We argue that either it is possible to efficiently generate some trapdoor, in which case we can simulate the prover's message in a certain way, or the trapdoor cannot be generated efficiently, in which case we can simulate the prover's message in a different way. However, we do not know in which case we reside. This is what distinguishes our WH proof from a ZK proof.

In what follows we give an overview of our construction and proof of security. At a very high-level, we follow the approach of Pass [Pas03]. Our starting point is the observation of [FS90] that if a language L has two independent witnesses then a WI proof for L is also WH. We use this observation to construct a 2-message WH protocol for any language $L \in \mathsf{NP}$ and use our proof of ignorance protocol to generate an additional independent witness (corresponding to an independent instance).

The basic blueprint of our protocol is the following: The prover will generate an independent instance x' and prove that either $x' \in L$ or $x \in L$, using a 2-message WI proof. This 2-message protocol is definitely witness hiding, but it is not sound, since the prover can cheat and prove that $x \in L$ (even though this is false) by generating $x' \in L$ and using a witness w' for x' to convince the verifier. We overcome this obstacle by using a proof-of-ignorance; the prover will send x' together with a proof of ignorance, and will then prove that either $x \in L$ or $x' \in L$.

Note that we do not have proof of ignorance (PoI) protocol for all of NP. This problem can be bypassed quite easily by choosing some language $L' \in \mathsf{NP}$ that has a PoI protocol, and now the prover will generate $x' \in L'$ together with a PoI, and will add a WI proof that $x \in L$ or $x' \in L'$.

Attempt 1. Our first attempt at constructing a 2-message WH protocol is the following:

- **Verifier's message:** Verifier samples CRS corresponding to the PoI proof system for L', and sends it to the prover.
- **Prover's message:** The prover samples $x' \in L'$ with a proof of ignorance π', and send (x', π') along with a WI proof for the following language:

$$L_{\mathsf{WI}} = \{(x, x') \mid \exists\, w \text{ such that } (x, w) \in R_L \ \vee\ (x', w) \in R_{L'}\}$$

Intuitively, this 2-message protocol seems to be sound, since if $x \notin L$ and if the prover generates a valid PoI for x' then he does not know a witness to x (since one does not exist) nor to x', and thus cannot cheat. The

actual proof is quite subtle, and in particular requires using a trapdoor PoI, and relying on super-polynomial hardness assumptions. Subsequently, we elaborate on these subtleties.

However, a more serious problem here is that adding the PoI seems to damage the WH property. For example, the (cheating) verifier can generate CRS maliciously in a way that if π' is a valid PoI for x' then it must be the case that $x' \notin L'$, and thus we do not have two independent witnesses, and hence the WI property may not protect us at all.

We fix this problem by having the verifier not only send the CRS for the PoI, but also prove that it is "well formed". Namely, he proves that there exists randomness that "explains" this CRS. There are two issues with adding this proof of correctness: First, it seems like we need a PoI with the strong property that *for every* well formed CRS, if the prover is honest then he generates a randomly distributed $x' \in L'$, *independent* of CRS, whereas our PoI have this property only for an honestly generated CRS. Here again, the trapdoor PoI comes to the rescue (as we explain in more detail below).

The other issue is that for soundness it is crucial that the prover does not learn sensitive information about the CRS (in particular, how it was generated). Thus, it seems that to maintain soundness, the verifier will need to use a zero-knowledge proof, or at least a witness-hiding proof, which brings us back to where we started from in the first place!

We solve the latter problem using the same blueprint that we started with. Rather than sending a single CRS, the verifier will send two independent copies CRS_0 and CRS_1 and will give a WI proof that at least one of them is well formed. Since the verifier sends the first message we need to rely on a *non-interactive WI* (NIWI) proof. The prover will then send (x'_0, π'_0) corresponding to CRS_0 and (x'_1, π'_1) corresponding to CRS_1, and will give a NIWI proof that either $x \in L$ or $x'_0 \in L'$ or $x'_1 \in L'$.

Attempt 2. Our second attempt at constructing a 2-message WH protocol is the following:

- **Verifier's message:** Verifier independently samples CRS_0 and CRS_1 corresponding to the PoI proof system for L', and generates a NIWI proof π_{NIWI} that at least one of them is well-formed. He sends $(\mathsf{CRS}_0, \mathsf{CRS}_1, \pi_{\mathsf{NIWI}})$ to the prover.
- **Prover's message:** The prover first checks that the NIWI proof π_{NIWI} is valid, and if not aborts. Otherwise, for every $b \in \{0,1\}$, the prover samples $x'_b \in L'$ with its proof of ignorance π'_b, and sends (x'_0, π'_0), (x'_1, π'_1), along with a NIWI proof that $(x \in L)$ or $(x'_0 \in L')$ or $(x'_1 \in L')$.

Intuitively, the soundness of this protocol follows from the fact that if $x \notin L$ then the only way to cheat is by using either a witness for x'_0 or a witness for x'_1, and the PoI guarantees that a cheating prover does not know such a witness. However, to argue this formally, a NIWI does not suffice and we need a NIWI *proof of knowledge*. This is the case since the PoI only guarantees that it is hard to find a witness, and not that a witness does not exist. We achieve this proof-of-knowledge property by resorting to complexity leveraging. Namely, the prover will also send a (statistically binding) commitment c and will prove that either $x \in L$ or

that there exists $b \in \{0,1\}$ such that c is a commitment to a valid witness of x'_b. Suppose this commitment can be broken in time T, then we can *extract* from the cheating prover a witness w'_b to x'_b, and argue that this breaks the soundness of the underlying PoI, (which asserts that given CRS_b it is hard to generate (x'_b, π'_b, w'_b)). However, to argue this formally, one needs to assume T-security of the PoI protocol, and T-security of the WI protocol. Then we can prove that the following protocol is indeed sound.

Attempt 3. Our third attempt at constructing a 2-message WH protocol is the following:

- **Verifier's message:** Verifier samples independently CRS_0 and CRS_1 corresponding to the PoI proof system for L', and generates a NIWI proof π_{NIWI} that at least one of them is well-formed. He sends $(\mathsf{CRS}_0, \mathsf{CRS}_1, \pi_{\mathsf{NIWI}})$ to the prover.
- **Prover's message:** The prover does the following:
  1. Check that the NIWI proof π_{NIWI} is valid, and if not abort.
  2. For every $b \in \{0,1\}$, use CRS_b to sample $x'_b \in L'$ along with a proof of ignorance π'_b.
  3. Compute c which is a commitment to 0.
  4. Compute a NIWI proof π'_{NIWI} that $x \in L$ or that there exists $b \in \{0,1\}$ for which c is a commitment to a valid witness corresponding to x'_b.
- Send $\big((x'_0, \pi'_0), (x'_1, \pi'_1), c, \pi'_{\mathsf{NIWI}}\big)$.

We can formally argue that this protocol is sound. However, it is still not clear how to argue that it is WH. As mentioned before, the problem is that the CRS could still be maliciously chosen (albeit well-formed). Our first observation is that this protocol is WH against cheating verifiers who "know" a valid trapdoor td_b corresponding to CRS_b (for some $b \in \{0,1\}$). This is true because given td_b one can efficiently compute w'_b from (x'_b, π'_b), and thus simulate the NIWI proof of the prover efficiently.

Thus, restating the problem: What if the cheating verifier managed to construct a valid NIWI proof without knowing a valid trapdoor to CRS_0 or CRS_1? Again, this would have been solved with a NIWI proof-of-knowledge. However, for one-message NIWI PoK we would need complexity leveraging and the use of complexity leveraging here would result in a WH protocol only for T-hard languages.[‖]

Our Non-Black-Box Technique: We overcome the hurdle described above by instructing the verifier to encrypt the trapdoors of each CRS and prove that one of these encryptions indeed encrypts a valid trapdoor. Namely, the verifier does the following: Sample two fresh and independent public keys $(\mathsf{PK}_0, \mathsf{PK}_1)$ corresponding to a semantically secure encryption scheme, and send $\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}$, where $\mathsf{ct}_b \leftarrow \mathsf{Enc}_{\mathsf{PK}_b}(\mathsf{td}_b)$, along with a NIWI proof that there exists $b \in \{0,1\}$ such that ct_b is an encryption of a valid td_b corresponding to CRS_b.

[‖]This approach was used by Pass in [Pas03].

As explained above, we cannot afford to extract a trapdoor from the ciphertexts (since this may take super-polynomial time). Instead, we instruct the prover to give its proof of ignorance π'_b *encrypted* under PK_b. This allows us to prove witness hiding using the following non-black-box approach. We distinguish between the following two cases:

- **Case 1:** The verifier "knows" a trapdoor corresponding to CRS_0 or CRS_1. In this case, one can efficiently simulate the prover's message using this trapdoor, and thus WH holds (as was argued before).
- **Case 2:** The verifier does not know a trapdoor to CRS_0 or to CRS_1. In this case, we argue that the verifier cannot distinguish between the case that the prover encrypts a valid PoI π'_b corresponding to x'_b, or encrypts 0, and thus again one can efficiently simulate the prover's message by encrypting 0, and generating x'_b together with a valid witness w'_b. To argue that indeed the verifier cannot distinguish between $\mathsf{Enc}_{\mathsf{PK}_b}(\pi'_b)$ and $\mathsf{Enc}_{\mathsf{PK}_b}(0)$ we rely on an encryption scheme that is rerandomizable. However, we need something stronger. Namely, we need to argue that the verifier cannot distinguish between $\mathsf{Enc}_{\mathsf{PK}_b}(\pi'_b)$ and $\mathsf{Enc}_{\mathsf{PK}_b}(0)$ even given x. To prove this we need the assumption that this encryption scheme is strong KDM secure with ε-bounded auxiliary input. In our analysis, the auxiliary input is the instance x such that $|x| < |\mathsf{PK}|^{\varepsilon}$. We defer the details of the analysis to Appendix 6.5.

Attempt 4. Our fourth (and almost final) attempt is the following.

- **Verifier's message:** The verifier does the following:
  1. Sample independently two public keys PK_0 and PK_1 (corresponding to a rerandomizable T-secure strong KDM secure encryption with ε-bounded auxiliary input).
  2. Sample independently CRS_0 and CRS_1, together with corresponding trapdoors td_0 and td_1.
  3. Generate $\mathsf{ct}_0 \leftarrow \mathsf{Enc}_{\mathsf{PK}_0}(\mathsf{td}_0)$ and $\mathsf{ct}_1 \leftarrow \mathsf{Enc}_{\mathsf{PK}_1}(\mathsf{td}_1)$.
  4. Generate a NIWI proof π_{NIWI} that there exists $b \in \{0,1\}$ for which ct_b encrypts a valid trapdoor corresponding to CRS_b.

  Send $\Big((\mathsf{CRS}_0, pk_0, \mathsf{ct}_0), (\mathsf{CRS}_1, pk_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\Big)$ to the prover.
- **Prover's message:** The prover does the following:
  1. Check that the NIWI proof π_{NIWI} is valid, and if not abort.
  2. For every $b \in \{0,1\}$, use CRS_b to sample $x'_b \in L'$ along with a proof of ignorance π'_b.
  3. For every $b \in \{0,1\}$, generate $\mathsf{ct}'_b \leftarrow \mathsf{Enc}_{\mathsf{PK}_b}(\pi'_b)$.
  4. Compute c which is a commitment to 0.
  5. Compute a NIWI proof π'_{NIWI} that $x \in L$ or that there exists $b \in \{0,1\}$ for which c is a commitment to a valid witness corresponding to x'_b.
- Send $\Big((x'_0, \mathsf{ct}'_0), (x'_1, \mathsf{ct}'_1), c, \pi'_{\mathsf{NIWI}}\Big)$.

We can indeed argue that this protocol is WH, as argued above. However, to argue soundness recall that we need to extract from the cheating prover a tuple (x'_b, w'_b, π'_b). Previously, π'_b was given in the clear, and w'_b was extracted in time T from the commitment. However, now we also need to extract π'_b, which will take more time, since the encryption is T-secure. Instead, we instruct the prover to generate a commitment $c'_b = \mathsf{Com}(\pi'_b; r'_b)$, in addition to ct'_b, and compute a NIWI proof π'_{NIWI} that $x \in L$ or that there exists $b \in \{0,1\}$ for which c is a commitment to a valid witness corresponding to x'_b and ct'_b is an encryption to a pair (π'_b, r'_b) such that $c'_b = \mathsf{Com}(\pi'_b; r'_b)$.

The formal protocol can be found in Section 6.4 and the analysis in Appendix 6.5.

6.2 Random Self-Reducible Languages

In this section we define random self-reducible (RSR) languages and witness-preserving random self-reducible languages, and provide examples of such languages.

6.2.1 Definitions

Definition 28 (Hard on Average)**.** *A language L is said to be* **hard on average** *with respect to a distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{D}_\lambda$ is over $\{0,1\}^\lambda \cap L$, if there exists a distribution $\bar{\mathcal{D}} = \{\bar{\mathcal{D}}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\bar{\mathcal{D}}_\lambda$ is over $\{0,1\}^\lambda \setminus L$, such that $\mathcal{D} \approx_c \bar{\mathcal{D}}$.*

Definition 29 (Hard-to-Extract on Average)**.** *A language $L \in \mathsf{NP}$, with a corresponding NP relation R_L, is said to be* **hard-to-extract** *with respect to a distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ if for any poly-size $\mathcal{A}$ there exists a negligible function ν such that for all $\lambda \in \mathbb{N}$,*

$$\Pr_{x\leftarrow\mathcal{D}_\lambda}[w = \mathcal{A}(x) \ : \ (x,w) \in R_L] \leq \nu(\lambda)$$

In what follows we define the notion of a random self-reducible language.

Definition 30 (Random Self Reducibility)**.** *An NP language L with a corresponding NP relation R_L is said to be* **random self-reducible** *(RSR) with respect to distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{D}_\lambda$ is over $\{0,1\}^\lambda \cap L$, if there exists a polynomial p and a poly-time computable function family $f = \{f_\lambda\}_{\lambda\in\mathbb{N}}$ such that for every $\lambda \in \mathbb{N}$,*

$$f_\lambda : \{0,1\}^\lambda \times \{0,1\}^{p(\lambda)} \to \{0,1\}^\lambda,$$

and the following two conditions hold.

- *For every $x \in \{0,1\}^\lambda \cap L$ and for $r \xleftarrow{\$} \{0,1\}^{p(\lambda)}$, $f_\lambda(x,r) \approx_s y$, where $y \leftarrow \mathcal{D}_\lambda$.*
- *For every $x \in \{0,1\}^\lambda \setminus L$ and for every $r \in \{0,1\}^{p(\lambda)}$, $f_\lambda(x,r) \notin L$.*

Definition 31 (Witness Preserving Random Self Reducibility)**.** *An NP language L with a corresponding NP relation R_L is said to be* **witness-preserving random self-reducible** *with respect to distribution $\mathcal{D} =$*

$\{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\mathcal{D}_\lambda$ is over $\{0,1\}^\lambda \cap L$, if there exists a polynomial p and a poly-time computable function family $f = \{f_\lambda\}_{\lambda\in\mathbb{N}}$ such that for every $\lambda \in \mathbb{N}$,

$$f_\lambda : \{0,1\}^\lambda \times \{0,1\}^{p(\lambda)} \to \{0,1\}^\lambda$$

and the following two conditions hold.

- *For any $x \in \{0,1\}^\lambda \cap L$ and for $r \xleftarrow{\$} \{0,1\}^{p(\lambda)}$, $f_\lambda(x,r) \approx_s y$, where $y \leftarrow \mathcal{D}_\lambda$.*
- *Let $q(\lambda)$ be the length of w for $(x,w) \in R_{L_\lambda}$. There exist poly-time computable function families $g = \{g_\lambda\}_{\lambda\in\mathbb{N}}$ and $h = \{h_\lambda\}_{\lambda\in\mathbb{N}}$, where for every $\lambda \in \mathbb{N}$*

  $$g_\lambda : \{0,1\}^\lambda \times \{0,1\}^{p(\lambda)} \times \{0,1\}^{q(\lambda)} \to \{0,1\}^{q(\lambda)} \text{ and } h_\lambda : \{0,1\}^\lambda \times \{0,1\}^{p(\lambda)} \times \{0,1\}^{q(\lambda)} \to \{0,1\}^{q(\lambda)},$$

  such that the following holds.

  - *For any $x \in \{0,1\}^\lambda \cap L$, any $r \in \{0,1\}^{p(\lambda)}$ and any $w' \in \{0,1\}^{q(\lambda)}$, if $(f_\lambda(x,r), w') \in R_L$ then $(x, g_\lambda(x,r,w')) \in R_L$.*
  - *For any $x \in \{0,1\}^\lambda \cap L$ and any $w \in \{0,1\}^{q(\lambda)}$ such that $(x,w) \in R_L$, it holds that for every $r \in \{0,1\}^{p(\lambda)}$, $(f_\lambda(x,r), h_\lambda(x,r,w)) \in R_L$.*

Note: We will refer to f, g, h as the *reduction functions*.

Remark 13. *We note that the notions of witness-preserving RSR and RSR are incomparable.*

Definition 32 (Instance-Witness Distribution)**.** *Let $L \in \mathsf{NP}$, let R_L be the corresponding NP relation, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ be a distribution, where $\mathcal{D}_\lambda$ is over L_λ. A distribution $\mathcal{E} = \{\mathcal{E}_\lambda\}_{\lambda\in\mathbb{N}}$ is said to be an instance-witness distribution corresponding to $\mathcal{D}$ if for every (x,w) in the support of $\mathcal{E}_\lambda$ it holds that $(x,w) \in R_{L_\lambda}$, and*

$$\{(x,w) \leftarrow \mathcal{E}_\lambda \ : \ x\}_{\lambda\in\mathbb{N}} \mathsf{Equivocate} \{y \leftarrow \mathcal{D}_\lambda \ : \ y\}_{\lambda\in\mathbb{N}} \tag{6.1}$$

6.2.2 Examples of Random Self-Reducible Languages

Decisional Diffie Hellman

Let $L_{\mathsf{DDH}} = \{L_{\mathsf{DDH},\lambda}\}_{\lambda\in\mathbb{N}}$ be the following language, where for each $\lambda \in \mathbb{N}$ the language $L_{\mathsf{DDH},\lambda}$ is parameterized by a group $\mathbb{G}$ of prime order $p \in [2^{\lambda-1}, 2^\lambda]$ and a generator $g \in \mathbb{G}$:

$$L_{\mathsf{DDH},\lambda} = \{(X,Y,Z) \mid \exists\, x,y \in \mathbb{Z}_p^* \text{ such that } X = g^x \ \wedge \ Y = g^y \ \wedge \ Z = g^{xy}\}$$

Theorem 4. *L_{DDH} is a random self-reducible language (Definition 30) and a witness-preserving random self-reducible language (Definition 31), with respect to the distribution $\mathcal{U}_{\mathsf{DDH}} = \{\mathcal{U}_{\mathsf{DDH},\lambda}\}_{\lambda\in\mathbb{N}}$, where for each $\lambda \in \mathbb{N}$, the distribution $\mathcal{U}_{\mathsf{DDH},\lambda}$ generates (g^x, g^y, g^{xy}) for $x,y \xleftarrow{\$} \mathbb{Z}_p^*$.*

Proof. Consider the poly-time computable function $f_\lambda : L_{\mathsf{DDH},\lambda} \times (\mathbb{Z}_p^*)^2 \to L_{\mathsf{DDH},\lambda}$ defined by

$$f_\lambda((X,Y,Z),(r,s)) = (X^r, Y^s, Z^{rs}).$$

For any $(g^x, g^y, g^{xy}) \in L_{\mathsf{DDH},\lambda}$ and for $r,s \xleftarrow{\$} \mathbb{Z}_p^*$, the tuple $(g^{xr}, g^{ys}, g^{xyrs})$ is distributed according to $\mathcal{U}_{\mathsf{DDH},\lambda}$. Moreover, for every (g^x, g^y, g^z) where $z \neq xy$ it holds that for every $(r,s) \in \mathbb{Z}_p^*$,

$$f((g^x, g^y, g^z),(r,s)) = (g^{xr}, g^{ys}, g^{zrs}) \notin L_{\mathsf{DDH},\lambda}$$

since $zrs \neq xyrs$. This proves that L_{DDH} is random self-reducible.

Also consider poly-time computable functions $g_\lambda, h_\lambda : L_{\mathsf{DDH},\lambda} \times \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \mathbb{Z}_p^*$, defined as follows:

$$g_\lambda((X,Y,Z),(r,s),(a,b)) = (ar^{-1}, bs^{-1}), \quad h_\lambda((X,Y,Z),(r,s),(x,y)) = (xr, ys)$$

For every $(A,B,C) = f_\lambda((X,Y,Z),(r,s)) = (X^r, Y^s, Z^{rs})$, if $((A,B,C),(a,b)) \in R_{L_{\mathsf{DDH}}}$ then $((X,Y,Z),$ $(ar^{-1}, bs^{-1})) \in R_{L_{\mathsf{DDH}}}$. Similarly, for every $((X,Y,Z),(x,y)) \in R_{L_{\mathsf{DDH}}}$ and every $(r,s) \in \mathbb{Z}_p^*$, it holds that $(f_\lambda((X,Y,Z),(r,s)),(xr,ys))) \in R_{L_{\mathsf{DDH}}}$, giving us witness-preserving random self-reducibility, as desired. □

Discrete Log

Let $L_{\mathsf{DL}} = \{L_{\mathsf{DL},\lambda}\}_{\lambda \in \mathbb{N}}$ be the following language, where for each $\lambda \in \mathbb{N}$ the language $L_{\mathsf{DL},\lambda}$ is parameterized by a group $\mathbb{G}$ of prime order $p \in [2^{\lambda-1}, 2^\lambda]$ and a generator $g \in \mathbb{G}$, and is defined by

$$L_{\mathsf{DL},\lambda} = \{X \mid \exists\, x \in \mathbb{Z}_p^* \text{ such that } X = g^x\}$$

Theorem 5. *L_{DL} is a witness-preserving random self-reducible language (as per Definition 31) with respect to the uniform distribution $\mathcal{U}_{\mathsf{DL}} = \{\mathcal{U}_{\mathsf{DL},\lambda}\}_{\lambda \in \mathbb{N}}$, where for each $\lambda \in \mathbb{N}$ the distribution $\mathcal{U}_{\mathsf{DL},\lambda}$ generates g^x for $x \xleftarrow{\$} \mathbb{Z}_p^*$.*

Proof. Consider the poly-time computable functions $f_\lambda : L_{\mathsf{DL},\lambda} \times \mathbb{Z}_p^* \to L_{\mathsf{DL},\lambda}$ and $g_\lambda, h_\lambda : L_{\mathsf{DL},\lambda} \times \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \mathbb{Z}_p^*$, defined as follows:

$$f_\lambda(X,r) = X \cdot g^r$$

and

$$g_\lambda(X,r,y) = y - r, \quad h_\lambda(X,s,z) = z + s$$

Note that for every λ and every $X \in L_{\mathsf{DL},\lambda}$, and for $r \xleftarrow{\$} \mathbb{Z}_p^*$, it holds that $f_\lambda(X,r) = X \cdot g^r$ is distributed according to $\mathcal{U}_{\mathsf{DL},\lambda}$. Moreover, for every $Y = f_\lambda(X,r) = X \cdot g^r$ if $(Y,y) \in R_{L_{\mathsf{DL}}}$ then $(X, y-r) \in R_{L_{\mathsf{DL}}}$. Similarly, for every $(X,z) \in R_{L_{\mathsf{DL}}}$ and every $s \in \mathbb{Z}_p$, for $Y = f(X,s) = X \cdot g^s$ it holds that $(Y, z+s) \in R_{L_{\mathsf{DL}}}$, as desired. □

Remark 14. *L_{DL} has an instance-witness distribution $\mathcal{E}_{\mathsf{DL}} = \{\mathcal{E}_{\mathsf{DL},\lambda}\}_{\lambda \in \mathbb{N}}$, defined as follows: For each $\lambda \in \mathbb{N}$, the distribution $\mathcal{E}_{\mathsf{DL},\lambda}$ outputs (g^w, w) for $w \xleftarrow{\$} \mathbb{Z}_p^*$. Note that g^w is distributed according to $\mathcal{U}_{\mathsf{DL},\lambda}$.*

6.3 Proofs of Ignorance

In this section, we define proofs of ignorance and construct proof of ignorance protocols for random self-reducible languages.

6.3.1 Definition

We now define the notion of proof-of-ignorance (PoI) and trapdoor PoI for NP languages. Intuitively we want a proof of ignorance π for $x \in L$ to convince the verifier that the prover does not know a witness corresponding to x.

Definition 33 (Proof of Ignorance)**.** *Let $L \in \mathsf{NP}$, let R_L be the corresponding* NP*-relation, and let $\mathcal{D}$ be a distribution on the instances of L. A* **proof-of-ignorance** *(PoI) proof system for $(L, \mathcal{D})$ consists of a triplet of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify})$ *with the following syntax:*

Setup $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter and outputs a common reference string* CRS*.*

Instance Generation $(x, \pi) \leftarrow \mathsf{Gen}(\mathsf{CRS})$*: The generation algorithm takes as input the* CRS *and outputs an instance $x \in L$ and a proof-of-ignorance π.*

Verification $0/1 \leftarrow \mathsf{Verify}(\mathsf{CRS}, x, \pi)$*: The verification algorithm takes as input the* CRS*, instance x, and proof π, and outputs* 0 *or* 1*.*

We require the following properties to hold.

Completeness. *We require:*

- $\{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (x, \pi) \leftarrow \mathsf{Gen}(\mathsf{CRS})\ :\ x\}_{\lambda \in \mathbb{N}} \approx_s \{y \leftarrow \mathcal{D}_\lambda\ :\ y\}_{\lambda \in \mathbb{N}}$.
- *For all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (x, \pi) \leftarrow \mathsf{Gen}(\mathsf{CRS})\ :\ \mathsf{Verify}(\mathsf{CRS}, x, \pi) = 1] = 1.$$

Soundness. *There exists a* PPT *algorithm* Setup' *such that*

- $\{\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)\ :\ \mathsf{CRS}\}_{\lambda \in \mathbb{N}}, \approx_c \{\mathsf{CRS}' \leftarrow \mathsf{Setup}'(1^\lambda)\ :\ \mathsf{CRS}'\}_{\lambda \in \mathbb{N}}$
- *For any all-powerful $\mathcal{A}^*$, there exists a negligible function ν such that for all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{CRS}' \leftarrow \mathsf{Setup}'(1^\lambda)\ ;\ (x, \pi) \leftarrow \mathcal{A}^*(\mathsf{CRS}')\ :\ \mathsf{Verify}(\mathsf{CRS}', x, \pi) = 1\ \wedge\ x \in L] \leq \nu(\lambda)$$

Definition 34 (Trapdoor Proof of Ignorance)**.** *Let $L \in \mathsf{NP}$, let R_L be the corresponding* NP*-relation, and let $\mathcal{D}$ be a distribution over instances in L. A* **trapdoor proof-of-ignorance** *(*td*-PoI) proof system for $(L, \mathcal{D})$ consists of a tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify}, \mathsf{wit})$ *with the following syntax (we note that* Gen *and* Verify *are identical to those defined in Definition 33):*

Setup $(\mathsf{CRS}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter and outputs a common reference string* CRS *together with a corresponding trapdoor* td*.*

Instance Generation $(x, \pi) \leftarrow \mathsf{Gen}(\mathsf{CRS})$*: The generation algorithm takes as input the* CRS *and outputs an instance $x \in L$ and a proof of ignorance π.*

Verification $0/1 \leftarrow \mathsf{Verify}(\mathsf{CRS}, x, \pi)$*: The verification algorithm takes as input the* CRS*, instance x, and proof* π*, and outputs* 0 *or* 1*.*

Witness Generation $w \leftarrow \mathsf{wit}(\mathsf{CRS}, \mathsf{td}, x, \pi)$*: The witness generation algorithm takes as input the* CRS *together with a corresponding trapdoor, along with an instance x and a proof* π*, and outputs a string w.*

We require the following properties to hold.

Completeness *We require:*

- $\{(\mathsf{CRS}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (x, \pi) \leftarrow \mathsf{Gen}(\mathsf{CRS})\ :\ x\}_{\lambda \in \mathbb{N}} \approx_s \{y \leftarrow \mathcal{D}_\lambda\ :\ y\}_{\lambda \in \mathbb{N}}$.
- *For all* $\lambda \in \mathbb{N}$*,*

$$\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (x, \pi) \leftarrow \mathsf{Gen}(\mathsf{CRS})\ :\ \mathsf{Verify}(\mathsf{CRS}, x, \pi) = 1] = 1.$$

- *For every* $(\mathsf{CRS}, \mathsf{td})$ *in the image of* $\mathsf{Setup}(1^\lambda)$ *and for any* (x, π)*, if* $\mathsf{Verify}(\mathsf{CRS}, x, \pi) = 1$ *then*

$$\mathsf{wit}(\mathsf{CRS}, \mathsf{td}, x, \pi) \in R_L(x).$$

Soundness *For any poly-size* $\mathcal{A}^*$*, there exists a negligible function* ν *such that for all* $\lambda \in \mathbb{N}$*,*

$$\Pr[\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (x, w, \pi) \leftarrow \mathcal{A}^*(\mathsf{CRS})\ :\ \mathsf{Verify}(\mathsf{CRS}, x, \pi) = 1 \ \wedge\ (x, w) \in R_L] \leq \nu(\lambda)$$

Remark 15. *We note that the notions of PoI and trapdoor PoI are incomparable.*

Definition 35 (T-security)**.** *A trapdoor proof of ignorance (*td*-PoI) proof system for* $(L, \mathcal{D})$ *(as in Definition 34) is said to be* T*-secure if soundness holds for all adversaries of size* $\mathsf{poly}(T)$*.*

6.3.2 Constructions

We now construct a PoI protocol for random self-reducible languages and trapdoor PoI protocol for witness-preserving random self-reducible languages.

Proof of Ignorance for RSR Languages: Let L be hard on average and random self-reducible with respect to distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ (as per Definitions 28 and 30, respectively). Let

$$f_\lambda : \{0,1\}^\lambda \times \{0,1\}^{p(\lambda)} \to \{0,1\}^\lambda$$

be the corresponding reduction function. The PoI protocol for L is described as follows:

$\mathsf{Setup}(1^\lambda)$: Choose $z \leftarrow \mathcal{D}_\lambda$. Output $\mathsf{CRS} = (z, f_\lambda)$.

$\mathsf{Gen}(\mathsf{CRS})$: Choose $r \xleftarrow{\$} \{0,1\}^{p(\lambda)}$ and compute $x = f_\lambda(z, r)$. Output (x, π) where $\pi = r$.

$\mathsf{Verify}(\mathsf{CRS}, x, \pi)$: Output 1 if and only if $x = f_\lambda(z, \pi)$ where z is part of the CRS.

Theorem 6. *Let L be hard on average and random self-reducible with respect to distribution* $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ *(as per Definitions 28 and 30, respectively). The protocol described above is a Proof-of-Ignorance proof system for* $(L, \mathcal{D})$ *as per Definition 33.*

Proof. Completeness is straightforward: $\mathsf{Gen}(\mathsf{CRS})$ outputs (x,π) where $x = f_\lambda(z,r)$ and $\pi = r$, and $\mathsf{Verify}(\mathsf{CRS},x,\pi)$ checks that indeed $x = f_\lambda(z,\pi)$. Also from the definition of random self-reducibility of L, for every $x \in \{0,1\}^\lambda \cap L$ and for $r \xleftarrow{\$} \{0,1\}^{p(\lambda)}$, $f_\lambda(x,r) \approx_s y$ where $y \leftarrow \mathcal{D}_\lambda$, as desired.

Now we prove soundness. By definition, the fact that L is hard on average with respect to distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$, implies that there exists a distribution $\bar{\mathcal{D}} = \{\bar{\mathcal{D}}_\lambda\}_{\lambda\in\mathbb{N}}$, where $\bar{\mathcal{D}}_\lambda$ is over $\{0,1\}^\lambda \setminus L$, such that $\mathcal{D} \approx_c \bar{\mathcal{D}}$. Define Setup' as follows: On input the security parameter λ, output $\mathsf{CRS}' = (\bar{z}, f_\lambda)$ where $\bar{z} \leftarrow \bar{\mathcal{D}}_\lambda$ and f_λ is the reduction function of L as before. Suppose for contradiction, there exists $\mathcal{A}^*$ and polynomial s such that for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr[(x,\pi) = \mathcal{A}^*(\mathsf{CRS}') \ : \ \mathsf{Verify}(\mathsf{CRS}',x,\pi) = 1 \ \wedge \ x \in L] > \frac{1}{s(\lambda)}$$

where the probability is over $\mathsf{CRS}' \leftarrow \mathsf{Setup}'(1^\lambda)$. This implies that for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr[(x,\pi) = \mathcal{A}^*(\bar{z}, f_\lambda) \ : \ x = f_\lambda(\bar{z},r) \ \wedge \ x \in L] > \frac{1}{s(\lambda)}$$

where the probability is over $\bar{z} \leftarrow \bar{\mathcal{D}}_\lambda$, contradicting the fact that for every $\bar{z} \in \{0,1\}^\lambda \setminus L$ and every $r \in \{0,1\}^{p(\lambda)}$, $f_\lambda(\bar{z},r) \notin L$.

□

Trapdoor Proof of Ignorance for Witness-preserving RSR Languages: Let L be hard-to-extract and witness-preserving random self-reducible with respect to distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ (as per Definitions 29 and 31, respectively). Let $\mathcal{E} = \{\mathcal{E}_\lambda\}_{\lambda\in\mathbb{N}}$ be a corresponding instance-witness distribution on R_L as per Definition 32. Let

$$f_\lambda : \{0,1\}^\lambda \times \{0,1\}^{p(\lambda)} \to \{0,1\}^\lambda \text{ and } g_\lambda, h_\lambda : \{0,1\}^\lambda \times \{0,1\}^{p(\lambda)} \times \{0,1\}^{q(\lambda)} \to \{0,1\}^{q(\lambda)}$$

be the corresponding reduction functions. The trapdoor PoI protocol for L is described as follows:

$\mathsf{Setup}(1^\lambda)$: Choose $(z,w) \leftarrow \mathcal{E}_\lambda$. Output $\mathsf{CRS} = (z, f_\lambda, h_\lambda)$ and $\mathsf{td} = w$.

$\mathsf{Gen}(\mathsf{CRS})$: Choose $r \xleftarrow{\$} \{0,1\}^{p(\lambda)}$ and compute $x = f_\lambda(z,r)$. Output (x,π) where $\pi = r$.

$\mathsf{Verify}(\mathsf{CRS},x,\pi)$: Output 1 if and only if $x = f_\lambda(z,\pi)$ where z is part of the CRS.

$\mathsf{wit}(\mathsf{CRS},\mathsf{td},x,\pi)$: Output $h_\lambda(x,\pi,\mathsf{td})$.

Theorem 7. *Let L be hard-to-extract and witness-preserving random self-reducible with respect to distribution $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ (as per Definition 29 and 31, respectively). Assume that there exists an instance witness distribution $\mathcal{E} = \{\mathcal{E}_\lambda\}_{\lambda\in\mathbb{N}}$ corresponding to $\mathcal{D}$ that is efficiently sampleable. Then the protocol described above is a Trapdoor Proof of Ignorance proof system for $(L,\mathcal{D})$ as per Definition 34.*

Proof. Completeness is straightforward, we thus focus on proving soundness. To this end, suppose for contradiction that there exists a poly-size $\mathcal{A}^*$ and a polynomial s such that for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr[(x,w,\pi) = \mathcal{A}^*(\mathsf{CRS}) \ : \ \mathsf{Verify}(\mathsf{CRS},x,\pi) = 1 \ \wedge \ (x,w) \in R_L] > \frac{1}{s(\lambda)} \tag{6.2}$$

where the probability is over $\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$. The fact that L is *hard-to-extract* with respect to the distribution $\mathcal{D}$ implies that for every poly-size B^* there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$,

$$\Pr_{z \leftarrow D_\lambda}[w = B^*(z) \ : \ (z,w) \in R_L] \leq \mu(\lambda) \tag{6.3}$$

We use A^* from Equation (6.2) to construct a poly-size B^* that contradicts Equation (6.3). B^* on input z, sets $\mathsf{CRS} = (z, f_\lambda, h_\lambda)$, computes $(x, \pi, w) = A^*(\mathsf{CRS})$, and outputs $g_\lambda(x, \pi, w)$. By Equation (6.2), for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr[(x,w,\pi) = \mathcal{A}^*(\mathsf{CRS}) \ : \ x = f_\lambda(z,\pi) \ \wedge \ (x,w) \in R_L] > \frac{1}{s(\lambda)}$$

where $\mathsf{CRS} = (z, f_\lambda, h_\lambda)$ and the probability is over $z \leftarrow \mathcal{D}_\lambda$. By the definition of witness preserving random self-reducibility, if $(f_\lambda(z,\pi), w) \in R_L$ then $(z, g_\lambda(x,\pi,w)) \in R_L$. Hence, we conclude that for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr_{z \xleftarrow{\$} D_\lambda}[w = B^*(z) \ : \ (z,w) \in R_L] > \frac{1}{s(\lambda)},$$

contradicting Equation (6.3).

□

6.4 Witness Hiding Arguments from Proofs of Ignorance

In this section, we show how to use a PoI proof system to construct a 2-message witness hiding argument for NP with adaptive soundness.

6.4.1 Ingredients

We first describe the ingredients we use in our witness hiding protocol. We assume that there exists a super-polynomial function $T = T(\lambda)$ and a constant $\varepsilon > 0$, for which the following primitives exists:

- A T-secure Trapdoor Proof of Ignorance (td-PoI) system for any $(L', \mathcal{D}')$, as defined in Definition 34 (Section 6.3.1), denoted by

$$(\mathsf{PoI.Setup}, \mathsf{PoI.Gen}, \mathsf{PoI.Verify}, \mathsf{PoI.wit}).$$

- A T-secure non-interactive witness indistinguishable (NIWI) proof system with perfect soundness,** as defined in Definition 10, denoted by

$$(\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$$

- A T-secure bit-wise rerandomizable encryption scheme that is strong KDM secure with ε-bounded auxiliary input, as defined in Definitions 11, 12, and 13, denoted by

$$(\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec}).$$

The requirement of perfect soundness is not needed, and is only made for simplicity. We note that the NIWI proof system based on DLIN [?**] indeed has perfect soundness.

– A non-interactive statistically binding commitment scheme, as defined in Definition 15, denoted by Com. We assume that the hiding property of Com can be broken in time $T = T(\lambda)$. Namely, we assume that in time $\mathsf{poly}(T)$ one can brute-force break the commitment scheme; i.e., there exist a T-time adversary $\mathcal{A}$ such that for every $m \in \mathcal{M}$ and every $r \xleftarrow{\$} \{0,1\}^\lambda$,

$$\mathcal{A}(\mathsf{Com}(m,r)) = (m,r') \text{ s.t. } \mathsf{Com}(m,r') = \mathsf{Com}(m,r).$$

Theorem 8. *Assuming the ingredients above there exists a two-message WH argument for* NP *with adaptive soundness.*

We prove this theorem in Section 6.5.

Corollary 3. *Let* $T = n^{\omega(1)}$ *and let* $\varepsilon > 0$. *There exists a two-message WH protocol for* NP *with adaptive soundness, assuming the existence of a* T*-secure rerandomizable encryption that is strong KDM secure with* ε*-bounded auxiliary input, and assuming the* T*-security of DLIN.*

6.4.2 The Protocol Description

We next describe our 2-message witness hiding argument for any $L \in \mathsf{NP}$ and any distribution $\mathcal{D}$ over pairs in R_L. In what follows we assume without loss of generality, that $|\mathsf{PK}| = \lambda$.

The verifier's message: On input 1^λ such that $|x| \geq \lambda^{\frac{1}{\varepsilon}}$, the verifier V_1 does the following:

1. For every $b \in \{0,1\}$, do the following: Sample $(\mathsf{PK}_b, \mathsf{SK}_b) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, choose at random $r_b^1, r_b^2 \leftarrow \{0,1\}^\lambda$, and compute $(\mathsf{CRS}_b, \mathsf{td}_b) = \mathsf{PoI.Setup}(1^\lambda; r_b^1)$ and $\mathsf{ct}_b = \mathsf{PKE.Enc}_{\mathsf{PK}_b}(\mathsf{td}_b; r_b^2)$.
2. Consider the NP language

$$L^* \triangleq \big\{(\mathsf{CRS}, \mathsf{PK}, \mathsf{ct}) \ : \ \exists\, (\mathsf{td}, r^1, r^2) \text{ s.t. } (\mathsf{CRS}, \mathsf{td}) = \mathsf{PoI.Setup}(1^\lambda, r^1) \ \wedge \ \mathsf{ct} = \mathsf{PKE.Enc}_{\mathsf{PK}}(\mathsf{td}, r^2)\big\}$$

   and consider the NP language
3. For every $b \in \{0,1\}$, let $x_b^* = (\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)$. Choose $b^* \xleftarrow{\$} \{0,1\}$ and let $w^* = (\mathsf{td}_{b^*}, r_{b^*}^1, r_{b^*}^2)$. Generate a NIWI proof $\pi_{\mathsf{NIWI}} \leftarrow \mathsf{NIWI.Prove}((x_0^*, x_1^*), w^*)$.

Output $(\mathsf{pp}, \mathsf{st})$ where $\mathsf{pp} = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}}) = \big(\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, \pi_{\mathsf{NIWI}}\big)$ and $\mathsf{st} = (\mathsf{SK}_0, \mathsf{SK}_1)$.

The prover's message: On input $(1^\lambda, x, w)$, and public parameters

$$\mathsf{pp} = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}}) = \big(\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, \pi_{\mathsf{NIWI}}\big),$$

the prover does the following:

1. Check that $\mathsf{NIWI.Verify}((x_0^*, x_1^*), \pi_{\mathsf{NIWI}}) = 1$ and check that for every $b \in \{0,1\}$, $|\mathsf{PK}_b| \geq |x|^{\frac{1}{\varepsilon}}$. If either of this condition is not satisfied then abort.
2. For every $b \in \{0,1\}$, compute $(x'_b, \pi'_b) \leftarrow \mathsf{PoI.Gen}(\mathsf{CRS}_b)$, choose at random $r'_b, s'_b \leftarrow \{0,1\}^\lambda$ and compute $c'_b = \mathsf{Com}(\pi'_b; r'_b)$ and $\mathsf{ct}'_b = \mathsf{PKE.Enc}_{\mathsf{PK}_b}((\pi'_b, r'_b); s'_b)$.
3. Compute $c' \leftarrow \mathsf{Com}(0)$.
4. Consider the language

$$L_{\mathsf{PoI}} = \big\{(x, x'_0, x'_1, c') \mid \exists (w, b, w', r) \text{ s.t. } \big((x,w) \in R_L\big) \vee \big(\big(c' = \mathsf{Com}(w'; r)\big) \wedge \big((x'_b, w') \in R_{L'}\big)\big)\big\}.$$

   Generate a NIWI proof π'_{NIWI} for $(x, x'_0, x'_1, c') \in L_{\mathsf{PoI}}$, using a witness w for x.
5. Output $\big(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\big)$.

The verifier's verdict: V_2 on input $\big(1^\lambda, \mathsf{pp}, \mathsf{st}, (x, \mathsf{msg})\big)$ outputs 1, where

$$\mathsf{msg} = \big(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\big)$$

if and only if the all the following checks pass.

1. For every $b \in \{0,1\}$, compute $(\pi'_b, r'_b) = \mathsf{PKE.Dec}_{\mathsf{SK}_b}(\mathsf{ct}'_b)$, and check that $c'_b = \mathsf{Com}(\pi'_b; r'_b)$ and $\mathsf{PoI.Verify}(x'_b, \pi'_b, \mathsf{CRS}_b) = 1$.
2. Check that $\mathsf{NIWI.Verify}\big((x, x'_0, x'_1, c'), \pi'_{\mathsf{NIWI}}\big) = 1$.

6.5 Analysis of Witness Hiding Protocol

In this section we prove that the protocol defined in Section 6.4.2 satisfies Theorem 8. In particular, we prove that the protocol satisfies the completeness, soundness and witness hiding properties.

6.5.1 Completeness.

Completeness follows directly from the completeness of the underlying primitives.

6.5.2 Adaptive Soundness.

Recall that in the honest protocol, it holds that $|x| \leq \lambda^\varepsilon$. We prove a stronger soundness condition where the cheating prover can choose x adaptively without any length restriction.[††]

[††]The length restriction in the protocol is required only for witness hiding.

Assume for contradiction that there exists a non-uniform poly-size cheating prover P^*, a polynomial s, and an infinite set $\Lambda \subseteq \mathbb{N}$, such that for every $\lambda \in \Lambda$,

$$\Pr\Big[\big((P^*,V)(1^\lambda)=1\big) \ \wedge \ \big(x \notin L\big)\Big] > \frac{1}{s(\lambda)}. \tag{6.4}$$

where the probability is over $\mathsf{pp} \leftarrow V(1^\lambda)$ and where $(x,\mathsf{msg}) = P^*(\mathsf{pp})$. Parse

$$\mathsf{msg} = \Big(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\Big).$$

We use P^* to construct a $\mathsf{poly}(T)$-size adversary $\mathcal{A}$ that takes as input CRS generated according to $\mathsf{PoI.Setup}(1^\lambda)$, and outputs a tuple (x',w',π') such that $(x',w') \in L'$ and $\mathsf{PoI.Verify}(\mathsf{CRS},x',\pi') = 1$, contradicting the T-security of the td-PoI system. The algorithm $\mathcal{A}$ on input CRS, does the following:

1. Choose at random $b^* \leftarrow \{0,1\}$, and set $\mathsf{CRS}_{1-b^*} = \mathsf{CRS}$.
2. Choose at random $r^1_{b^*} \leftarrow \{0,1\}^\lambda$ and compute $(\mathsf{CRS}_{b^*}, \mathsf{td}_{b^*}) = \mathsf{PoI.Setup}(1^\lambda; r^1_{b^*})$.
3. Generate $(\mathsf{PK}_0,\mathsf{SK}_0),(\mathsf{PK}_1,\mathsf{SK}_1) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$.
4. Choose at random $r^2_{b^*} \leftarrow \{0,1\}^\lambda$ and compute $\mathsf{ct}'_{b^*} = \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(\mathsf{td}_{b^*}; r^2_{b^*})$.
5. Generate $\mathsf{ct}'_{1-b^*} \leftarrow \mathsf{PKE.Enc}_{\mathsf{PK}_{1-b^*}}(0)$.
6. Let $x^*_0 = (\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0)$ and $x^*_1 = (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1)$, and let $w^* = (\mathsf{td}_{b^*}, r^1_{b^*}, r^2_{b^*})$
7. Compute $\pi_{\mathsf{NIWI}} \leftarrow \mathsf{NIWI.Prove}(x^*_0, x^*_1, w^*)$.
8. Let $\mathsf{pp} = (x^*_0, x^*_1, \pi_{\mathsf{NIWI}})$.
9. Compute $\big(x, \mathsf{msg}\big) = P^*\left(\mathsf{pp}\right)$, and parse $\mathsf{msg} = \Big(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\Big)$.
10. Run in time $\mathsf{poly}(T)$ to find (w',r') such that $c' = \mathsf{Com}(w';r')$, and to find $(\pi'_{1-b^*}, r'_{1-b^*})$ such that $c'_{1-b^*} = \mathsf{Com}(\pi'_{1-b^*}; r'_{1-b^*})$.[‡‡]
11. Output $(x'_{1-b^*}, w', \pi'_{1-b^*})$.

We prove that there exists a polynomial q such that for every $\lambda \in \Lambda$,

$$\Pr\big[\mathcal{A}(\mathsf{CRS}) = (x',w',\pi') \text{ s.t. } \big((x',w') \in R_{L'}\big) \wedge \big(\mathsf{PoI.Verify}(\mathsf{CRS},x',\pi') = 1\big)\big] \geq \frac{1}{q(\lambda)}, \tag{6.5}$$

contradicting the T-security of the PoI scheme.

To this end, consider the following dishonest verifier $V_{\mathcal{A},b^*}$ that generates his first message with the same distribution as $\mathcal{A}$, while fixing his random bit choice to be b^*. Moreover, it outputs 1 if and only if the NIWI proof given by the prover is accepting, and for $(\pi'_{b^*}, r'_{b^*}) = \mathsf{PKE.Dec}_{\mathsf{SK}_{b^*}}(\mathsf{ct}'_{b^*})$ it holds that

$$c'_{b^*} = \mathsf{Com}(\pi'_{b^*}; r'_{b^*}) \ \text{ and } \ \mathsf{PoI.Verify}(x'_{b^*}, \pi'_{b^*}, \mathsf{CRS}_{b^*}) = 1.$$

Namely, $V_{\mathcal{A},b^*}$ does the same checks as the honest verifier, except that he does not check the conditions corresponding to $1-b^*$.

[‡‡] We note that the randomness computed in this step may not be the actual randomness used by P^*. This abuse of notation (or notational overload) is only to avoid cluttering on notation, and is of no significance.

Claim 10. *For every poly-size adversary $\mathcal{B}$, there exists a negligible function ν such that for every $\lambda \in \Lambda$,*

$$\left| \Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),b,\mathsf{SK}_b,w',r)=1\right] - \Pr\left[\mathcal{B}(\mathsf{pp}_{\mathcal{A},b},(x,\mathsf{msg}),b,\mathsf{SK}_b,w',r)=1\right] \right| \leq \nu(\lambda)$$

where the left probability is over $\mathsf{pp} = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}}) \leftarrow V(1^\lambda)$, where $(x,\mathsf{msg}) = P^*(\mathsf{pp})$, and $b \in \{0,1\}$ is such that π_{NIWI} is generated with a witness corresponding to x_b^*, and where SK_b is the secret key corresponding to PK_b where $x_b^* = (\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)$. The right probability is over $b \xleftarrow{\$} \{0,1\}$ and $\mathsf{pp}_{\mathcal{A},b} = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}}) \leftarrow V_{\mathcal{A},b}(1^\lambda)$, where $(x,\mathsf{msg}) = P^*(\mathsf{pp}_{\mathcal{A},b})$, and where SK_b is the secret key corresponding to PK_b where $x_b^* = (\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)$. In both probabilities (w',r) satisfies $c' = \mathsf{Com}(w';r)$, where c' is part of msg.[§§]

Proof. Suppose for contradiction there exists a poly-size adversary $\mathcal{B}$, polynomial p such that for infinitely many $\lambda \in \Lambda$

$$\Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),b,\mathsf{SK}_b,w',r)=1\right] - \Pr\left[\mathcal{B}(\mathsf{pp}_{\mathcal{A},b},(x,\mathsf{msg}),b,\mathsf{SK}_b,w',r)=1\right] > \frac{1}{p(\lambda)} \tag{6.6}$$

We use $\mathcal{B}$ to construct a $\mathsf{poly}(T)$-size adversary $\mathcal{M}$ that contradicts the T-security of the encryption scheme as per Lemma 1.

Algorithm $\mathcal{M}(1^\lambda)$ does the following:

1. For every $b \in \{0,1\}$, choose $r_b^1 \xleftarrow{\$} \{0,1\}^\lambda$ and compute $(\mathsf{CRS}_b, \mathsf{td}_b) = \mathsf{PoI.Setup}(1^\lambda; r_b^1)$.
2. Choose at random $d \xleftarrow{\$} \{0,1\}$, set $m_0 = 0$ and $m_1 = \mathsf{td}_{1-d}$ such that $|m_0| = |m_1|$, and send m_0 and m_1 as challenge messages.
3. Upon receiving from the challenger a pair $(\mathsf{PK}, \mathsf{ct})$, where $\mathsf{ct} = \mathsf{PKE.Enc}_{\mathsf{PK}}(m_{d^*})$ for a random $d^* \xleftarrow{\$} \{0,1\}$, do the following:
   (a) Generate $(\mathsf{PK}_d, \mathsf{SK}_d) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$, and let $\mathsf{PK}_{1-d} = \mathsf{PK}$.
   (b) Choose $r_d^2 \xleftarrow{\$} \{0,1\}^\lambda$, compute $\mathsf{ct}_d = \mathsf{PKE.Enc}_{\mathsf{PK}_d}(\mathsf{td}_d; r_d^2)$, and let $\mathsf{ct}_{1-d} = \mathsf{ct}$.
   (c) Let $w = (\mathsf{td}_d, r_d^1, r_d^2)$, and compute $\pi_{\mathsf{NIWI}} \leftarrow \mathsf{NIWI.Prove}\big(\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, w\big)$.
   (d) Let $\mathsf{pp} = (\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, \pi_{\mathsf{NIWI}})$, compute $(x,\mathsf{msg}) = P^*(\mathsf{pp})$, and parse
   $$\mathsf{msg} = \Big(\{(x_b', c_b', \mathsf{ct}_b')\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\Big).$$
   (e) Run in time $\mathsf{poly}(T)$ to find (w',r) such that $c' = \mathsf{Com}(w';r)$.
   (f) Output $\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r)$.

Note that if $d^* = 1$ then the input to $\mathcal{B}$ is distributed exactly as in the left side of Equation (6.6), whereas if $d^* = 0$ the input to $\mathcal{B}$ is distributed exactly as in the right side of Equation (6.6).

[§§] Recall that $\mathsf{msg} = \Big(\{(x_b', c_b', \mathsf{ct}_b')\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\Big)$.

$$\Pr\left[\mathcal{M}(\mathsf{PK},\mathsf{ct}) = d^*\right] =$$
$$\Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = d^*\right] =$$
$$\frac{1}{2}\cdot\Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = d^* \;\middle|\; d^*=1\right] + \frac{1}{2}\cdot\Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = d^* \;\middle|\; d^*=0\right] =$$
$$\frac{1}{2}\cdot\Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = 1\right] + \frac{1}{2}\cdot\Pr\left[\mathcal{B}(\mathsf{pp}_{\mathcal{A},d},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = 0\right] =$$
$$\frac{1}{2}\cdot\Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = 1\right] + \frac{1}{2}\cdot\left(1-\Pr\left[\mathcal{B}(\mathsf{pp}_{\mathcal{A},d},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = 1\right]\right) =$$
$$\frac{1}{2}+\frac{1}{2}\cdot\left(\Pr\left[\mathcal{B}(\mathsf{pp},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = 1\right] - \Pr\left[\mathcal{B}(\mathsf{pp}_{\mathcal{A},d},(x,\mathsf{msg}),d,\mathsf{SK}_d,w',r) = 1\right]\right) \geq$$
$$\frac{1}{2}+\frac{1}{2p(\lambda)}$$

where the last inequality follows from Equation (6.6) (for infinitely many $\lambda \in \Lambda$). This contradicts the T-semantic security of the underlying encryption, as desired. □

Let $\mathsf{pp}_{\mathcal{A},b^*} \leftarrow V_{\mathcal{A},b^*}(1^\lambda)$ and let $(x,\mathsf{msg}) \leftarrow P^*(\mathsf{pp}_{\mathcal{A},b^*})$. Parse $\mathsf{msg} = \left(\{(x'_b,c'_b,\mathsf{ct}'_b)\}_{b\in\{0,1\}},c',\pi'_{\mathsf{NIWI}}\right)$. For every $b \in \{0,1\}$ let E_b be the event that there exists (w',r) such that

$$c' = \mathsf{Com}(w';r) \;\text{ and }\; (x'_b,w') \in R_{L'}.$$

Claim 11. *There exists a polynomial p such that for every $\lambda \in \Lambda$,*

$$\Pr\left[\left(E_0 \vee E_1\right) \wedge \left((P^*,V_{\mathcal{A},b^*})(1^\lambda)=1\right)\right] \geq \frac{1}{p(\lambda)},$$

where the probability is over $b^* \xleftarrow{\$} \{0,1\}$ and over the randomness of $V_{\mathcal{A},b^*}$.

Proof. By our contradiction assumption (Equation (6.4)), and by the soundness of the NIWI proof system, there exists a negligible function μ such that for every $\lambda \in \Lambda$,

$$\Pr\left[\left(E_0 \vee E_1\right) \wedge \left((P^*,V)(1^\lambda)=1\right)\right] \geq \frac{1}{s(\lambda)} - \mu(\lambda),$$

where the probability is over the randomness of V. This, together with Claim 10, implies that there exists a negligible function ν such that for every $\lambda \in \Lambda$,

$$\Pr\left[\left(E_0 \vee E_1\right) \wedge \left((P^*,V_{\mathcal{A},b^*})(1^\lambda)=1\right)\right] \geq \frac{1}{s(\lambda)} - \nu(\lambda),$$

where the probability is over $b^* \xleftarrow{\$} \{0,1\}$ and over the randomness of $V_{\mathcal{A},b^*}$, as desired. □

Claim 12. *There exists a polynomial q such that for every $\lambda \in \Lambda$,*

$$\Pr\left[E_{1-b} \wedge \left((P^*,V_{\mathcal{A},b})(1^\lambda)=1\right)\right] \geq \frac{1}{q(\lambda)},$$

where the probability is over $b \xleftarrow{\$} \{0,1\}$ and the randomness of $V_{\mathcal{A},b}$.

Proof. Suppose for contradiction that there exists a negligible function μ and an infinite set $\Lambda_0 \subseteq \Lambda$ such that for every $\lambda \in \Lambda_0$,

$$\Pr\left[E_{1-b} \wedge \left((P^*, V_{\mathcal{A},b})(1^\lambda) = 1\right)\right] = \mu(\lambda). \tag{6.7}$$

This, together with Claim 11, implies that for every $\lambda \in \Lambda_0$,

$$\Pr\left[E_b \wedge \left((P^*, V_{\mathcal{A},b})(1^\lambda) = 1\right)\right] > \frac{1}{p(\lambda)} - \mu(\lambda). \tag{6.8}$$

Consider the verifier V_b that is identical to the honest verifier V, except that it uses the witness w_b to generate the NIWI, where w_b is the witness corresponding to x_b^*, and similarly to $V_{\mathcal{A},b}$, it does not do the check corresponding to $1-b$, rather only checks the NIWI of the prover and the check corresponding to b. In other words, V_b is identical to $V_{\mathcal{A},b}$ except that he generates x_{1-b}^* honestly (as opposed to $V_{\mathcal{A},b}$ who generates $x_{1-b}^* = (\mathsf{CRS}_{1-b}, \mathsf{PK}_{1-b}, \mathsf{ct}_{1-b})$ where ct_{1-b} is an encryption of 0).

By Claim 10, Equations (6.7) and (6.8) imply that there exists a negligible function ν such that for every $\lambda \in \Lambda_0$,

$$\Pr\left[E_{1-b} \wedge \left((P^*, V_b)(1^\lambda) = 1\right)\right] = \nu(\lambda) \tag{6.9}$$

and

$$\Pr\left[E_b \wedge \left((P^*, V_b)(1^\lambda) = 1\right)\right] > \frac{1}{p(\lambda)} - 2\nu(\lambda), \tag{6.10}$$

where the probabilities are over $b \xleftarrow{\$} \{0,1\}$ and over the randomness of V_b.

We next argue that these two equations contradict the T-security of the NIWI proof system. To this end, we construct a $\mathsf{poly}(T)$-size adversary $\mathcal{M}$ that wins the WI game as described in Definition 2 with non-negligible advantage, as follows.

1. For every $b \in \{0,1\}$ do the following:
   (a) Choose at random $r_b^1 \xleftarrow{\$} \{0,1\}^\lambda$ and compute $(\mathsf{CRS}_b, \mathsf{td}_b) = \mathsf{PoI.Setup}(1^\lambda, r_b^1)$.
   (b) Generate $(\mathsf{PK}_b, \mathsf{SK}_b) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$.
   (c) Choose at random $r_b^2 \xleftarrow{\$} \{0,1\}^\lambda$ and compute $\mathsf{ct}_b = \mathsf{PKE.Enc}_{\mathsf{PK}_b}(\mathsf{td}_b, r_b^2)$.
   (d) Let $x_b^* = (\mathsf{CRS}_b, \mathsf{PK}_0, \mathsf{ct}_b)$, and let $w_b = (\mathsf{td}_b, r_b^1, r_b^2)$.
2. Choose (x_0^*, x_1^*) to be the instance in the WI game (w.r.t. the NP language L_{OR}^*), and w_0 and w_1 to be the two witnesses.
3. Let π_{NIWI} be the challenge proof generated with respect to witness w_{b^*} for a randomly chosen $b^* \xleftarrow{\$} \{0,1\}$.
4. Compute $(x, \mathsf{msg}) = P^*(\mathsf{pp})$, where $\mathsf{pp} = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}})$.
5. Parse $\mathsf{msg} = \left(\{(x_b', c_b', \mathsf{ct}_b')\}_{b\in\{0,1\}}, c', \pi_{\mathsf{NIWI}}'\right)$.
6. Compute (w', r) such that $c' = \mathsf{Com}(w', r)$.
7. If there exists $b \in \{0,1\}$ such that the following three conditions are satisfied:

- π'_{NIWI} is accepting,
- $\pi'_b = \mathsf{PKE.Dec}_{\mathsf{SK}_b}(c'_b)$ satisfies that $\mathsf{PoI.Verify}(\mathsf{CRS}_b, x'_b, \pi'_b) = 1$,
- $(x'_b, w') \in R_{L'}$,

then output b. Otherwise, output a randomly chosen bit $b \xleftarrow{\$} \{0,1\}$.

We next argue that for every $\lambda \in \Lambda_0$,

$$\Pr[b = b^*] \geq \frac{1}{2} + \frac{1}{3p(\lambda)},$$

contradicting the T-security of the WI property.

To this end, denote by GOOD the event that the following three conditions hold:

- π'_{NIWI} is accepting.
- $\pi'_{b^*} = \mathsf{Dec}_{\mathsf{SK}_{b^*}}(c'_{b^*})$ satisfies $\mathsf{PoI.Verify}(\mathsf{CRS}_{b^*}, x'_{b^*}, \pi'_{b^*}) = 1$.
- There exists b such that $(x'_b, w') \in R_{L'}$.

Equations (6.9) and (6.10) imply that for every $\lambda \in \Lambda_0$,

$$\Pr[\mathsf{GOOD}] \geq \frac{1}{p(\lambda)} - \nu(\lambda)$$

and

$$\Pr[\mathsf{GOOD}] - \Pr\left[(b = b^*) \wedge \mathsf{GOOD}\right] \leq \nu(\lambda).$$

Therefore,

$$\begin{aligned}
&\Pr[b = b^*] = \\
&\Pr\left[(b = b^*) \wedge \mathsf{GOOD}\right] + \Pr\left[(b = b^*) \wedge \neg\mathsf{GOOD}\right] \geq \\
&\Pr[\mathsf{GOOD}] - \nu(\lambda) + \frac{1}{2} \cdot \left(1 - \Pr[\mathsf{GOOD}]\right) \geq \\
&\frac{1}{2} + \frac{1}{2} \cdot \Pr[\mathsf{GOOD}] - \nu(\lambda) \geq \\
&\frac{1}{2} + \frac{1}{2p(\lambda)} - 2\nu(\lambda) \geq \\
&\frac{1}{2} + \frac{1}{3p(\lambda)}
\end{aligned}$$

Contradicting the T-security of the NIWI proof system, as desired.

□

Denote by Z the event that both $(P^*, V_{\mathcal{A},b})(1^\lambda) = 1$ and E_{1-b}. By Claim 12, for every $\lambda \in \Lambda$,

$$\Pr[Z] \geq \frac{1}{q(\lambda)}$$

By the definition of $\mathcal{A}$, and the definition of the event E_{1-b}, it holds that

$$\Pr\left[\mathcal{A}(\mathsf{CRS}) = (x', w', \pi') \text{ s.t. } \left((x', w') \in R_{L'}\right) \wedge \left(\mathsf{PoI.Verify}(\mathsf{CRS}, x', \pi') = 1\right) \mid Z\right] = 1$$

Thus, we conclude that

$$\Pr\left[\mathcal{A}(\mathsf{CRS}) = (x', w', \pi') \text{ s.t. } \left((x', w') \in R_{L'}\right) \wedge \left(\mathsf{PoI.Verify}(\mathsf{CRS}, x', \pi') = 1\right)\right] \geq \Pr[Z] \geq \frac{1}{q(\lambda)},$$

contradicting the T-security of PoI.

6.5.3 Witness Hiding.

Suppose for the sake of contradiction that there exists a poly-size cheating verifier $V^* = (V_1^*, V_2^*)$, a function g such that $g(\kappa) \geq \kappa^{\frac{1}{\varepsilon}}$, a polynomial s, and an infinite set $\mathcal{K} \subseteq \mathbb{N}$, such that for every $\kappa \in \mathcal{K}$,

$$\Pr\left[V_2^*\left(x, V_1^*(1^\lambda, x), \mathsf{msg}_P\right) = w \text{ s.t. } (x, w) \in R_L\right] \geq \frac{1}{s(\kappa)} \tag{6.11}$$

where $\lambda = g(\kappa)$ and where the probability is over $(x, w) \leftarrow \mathcal{D}_\kappa$ and over $\mathsf{msg}_P \leftarrow P(1^\lambda, x, w, \mathsf{pp})$, where $(\mathsf{pp}, \mathsf{st}) = V^*(1^\lambda, x)$, and where pp is the message that V^* sends the prover and st is a secret state that is used by V_2^* to extract w.

Remark 16. *In what follows, V_1^*, V_2^*, P take as input 1^λ where $\lambda = g(\kappa)$ and (x, w) are sampled from $\mathcal{D}_\kappa$.*

Remark 17. *We assume without loss of generality that V_1^* always generates an accepting NIWI proof π_{NIWI} and samples $\mathsf{PK}_0, \mathsf{PK}_1$ such that for every $b \in \{0,1\}$, $|\mathsf{PK}_b| \geq |x|^{1/\varepsilon}$. Loosely speaking, this is without loss of generality since P aborts if the NIWI proof π_{NIWI} is rejected or if $\mathsf{PK}_0, \mathsf{PK}_1$ do not satisfy the length restriction, and hence the cheating verifier (V_1^*, V_2^*) does not gain anything by generating a rejecting π_{NIWI} or by setting $|\mathsf{PK}_b| < |x|^{1/\varepsilon}$ for any $b \in \{0,1\}$.*

Formally, this is argued as follows: Replace (V_1^*, V_2^*) with the following $(V_{h_1}^*, V_{h_2}^*)$:

- *$V_{h_1}^*$: On input $(1^\lambda, x)$, compute $(\mathsf{pp}, \mathsf{st}) = V_1^*(1^\lambda, x)$. Parse $\mathsf{pp} = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}})$. If*

$$\mathsf{NIWI.Verify}(x_0^*, x_1^*, \pi_{\mathsf{NIWI}}) = 1 \text{ and } |\mathsf{PK}_0| \geq |x|^{1/\varepsilon} \text{ and } |\mathsf{PK}_1| \geq |x|^{1/\varepsilon}$$

  output $\mathsf{pp}_h = \mathsf{pp}$. Else, compute pp_h as the honest verifier and output pp_h.

- *$V_{h_2}^*$: On input $(x, V_{h_1}^*(1^\lambda, x), \mathsf{msg}_P)$, compute $(\mathsf{pp}, \mathsf{st}) = V_1^*(1^\lambda, x)$, and parse $\mathsf{pp} = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}})$. If*

$$\mathsf{NIWI.Verify}(x_0^*, x_1^*, \pi_{\mathsf{NIWI}}) = 1 \text{ and } |\mathsf{PK}_0| \geq |x|^{1/\varepsilon} \text{ and } |\mathsf{PK}_1| \geq |x|^{1/\varepsilon}$$

  output $V_2^*(x, V_1^*(1^\lambda, x), \mathsf{msg}_P)$ where $\mathsf{msg}_P \leftarrow P(1^\lambda, x, w, \mathsf{pp})$. Else, output $V_2^*(x, V_1^*(1^\lambda, x), \perp)$.

The fact that $\mathsf{msg}_P = \perp$ when the NIWI proof of the verifier is rejected or when $|\mathsf{PK}_b| < |x|^{1/\varepsilon}$ for some $b \in \{0,1\}$, implies that the output of $V_{h_2}^*$ is identically distributed to the output of V_2^*. Hence, for every $\kappa \in \mathcal{K}$,

$$\Pr[V_{h_2}^*(x, V_{h_1}^*(1^\lambda, x), \mathsf{msg}_P) \in R_L(x)] = \Pr[V_2^*(x, V_1^*(1^\lambda, x), \mathsf{msg}_P) \in R_L(x)] \geq \frac{1}{s(\kappa)}.$$

where the last inequality holds by Equation (6.11), and where the probability is over $(x, w) \leftarrow \mathcal{D}_\kappa$ and $\mathsf{msg}_P \leftarrow P(1^\lambda, x, w, \mathsf{pp})$.

Remark 18. *In what follows, we often abuse notation, and denote by $x \leftarrow \mathcal{D}_\kappa$ to denote that x is sampled by sampling $(x,w) \leftarrow \mathcal{D}_\kappa$ and outputting x.*

Subset GOOD: We define the set $\mathsf{GOOD} \subseteq L$, where $x \in \mathsf{GOOD}$ if and only if

$$\Pr[V_2^*\left(x, V_1^*(1^\lambda, x), \mathsf{msg}_P\right) = w \text{ s.t. } (x,w) \in R_L] \geq \frac{1}{2s(\kappa)},$$

where the probability is over $\mathsf{msg}_P \leftarrow P(1^\lambda, x, w, \mathsf{pp})$.

Claim 13. *For every $\kappa \in \mathcal{K}$,*

$$\Pr[x \in \mathsf{GOOD}] \geq \frac{1}{2s(\kappa)}$$

where the probability is over $x \leftarrow \mathcal{D}_\kappa$.

Proof.

$$\begin{aligned}
\Pr[V_2^*(x, V_1^*(1^\lambda, x), \mathsf{msg}_P) \in R_L(x)] = &\Pr[V_2^*(x, V_1^*(1^\lambda, x), \mathsf{msg}_P) \in R_L(x) \mid x \in \mathsf{GOOD}] \cdot \Pr[x \in \mathsf{GOOD}] \\
&+ \Pr[V_2^*(x, V_1^*(1^\lambda, x), \mathsf{msg}_P) \in R_L(x) \mid x \notin \mathsf{GOOD}] \cdot \Pr[x \notin \mathsf{GOOD}] \\
&\leq \Pr[x \in \mathsf{GOOD}] + \frac{1}{2s(\kappa)} \cdot \Pr[x \notin \mathsf{GOOD}] \\
&\leq \Pr[x \in \mathsf{GOOD}] + \frac{1}{2s(\kappa)}
\end{aligned}$$

Hence, for every $\kappa \in \mathcal{K}$, the fact that $\Pr[V_2^*(x, V_1^*(1^\lambda, x), \mathsf{msg}_P) \in R_L(x)] > \frac{1}{s(\kappa)}$ implies that $\Pr[x \in \mathsf{GOOD}] \geq \frac{1}{2s(\kappa)}$, as desired.

□

Trapdoor Set of x: For every x we define the trapdoor set of x, denoted by $\mathsf{td}(x)$, as follows:

$$\mathsf{td}(x) = \{\mathsf{td} : \exists b \in \{0,1\}\ \exists (r_b^1, r_b^2) \text{ s.t. } \left((\mathsf{CRS}_b, \mathsf{td}) = \mathsf{Pol.Setup}(1^\lambda; r_b^1)\right) \wedge \left(\mathsf{ct}_b = \mathsf{Enc}_{\mathsf{PK}_b}(\mathsf{td}, r_b^2)\right)\}$$

where $V_1^*(1^\lambda, x) = (\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, \pi_{\mathsf{NIWI}})$. By the perfect soundness of the NIWI,

$$V_{\mathsf{NIWI}}\left(\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, \pi_{\mathsf{NIWI}}\right) = 1 \implies \mathsf{td}(x) \neq \emptyset.$$

We distinguish between the following two cases:

Case 1. There exists a poly-size computable function f such that for infinitely many $\kappa \in \mathcal{K}$,

$$\Pr[f(x) \in \mathsf{td}(x)] \geq \frac{1}{\mathsf{poly}(\kappa)}, \text{ where } x \leftarrow \mathcal{D}_\kappa | \mathsf{GOOD}. \tag{6.12}$$

In this case we construct a poly-size $\mathcal{A}$ that given $x \leftarrow \mathcal{D}_\kappa$ outputs a valid witness w with non-negligible probability (breaking the hardness of the language L). $\mathcal{A}$, on input x, does the following:

1. Compute $V_1^*(1^\lambda, x) = \left(\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b\in\{0,1\}}, \pi_{\mathsf{NIWI}}\right)$.

2. Compute $f(x) = \mathsf{td}$.

   If td is an invalid trapdoor with respect to both CRS_0 and CRS_1 then abort.

3. Otherwise, td is a valid trapdoor corresponding to CRS_{b^*} for some $b^* \in \{0,1\}$. Namely, there exists $(r^1_{b^*}, r^2_{b^*})$ for which

$$\big((\mathsf{CRS}_{b^*}, \mathsf{td}) = \mathsf{PoI.Setup}(1^\lambda; r^1_{b^*})\big) \ \wedge \ \big(\mathsf{ct}_{b^*} = \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(\mathsf{td}, r^2_{b^*})\big).$$

4. Compute $\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}$ as the honest prover does. Namely, do the following computations for every $b \in \{0,1\}$:

   - Sample $(x'_b, \pi'_b) \leftarrow \mathsf{PoI.Gen}(\mathsf{CRS}_b)$,
   - Sample $r'_b, s'_b \xleftarrow{\$} \{0,1\}^\lambda$ and compute $c'_b = \mathsf{Com}(\pi'_b; r'_b)$ and $\mathsf{ct}'_b = \mathsf{PKE.Enc}_{\mathsf{PK}_b}((\pi'_b, r'_b); s'_b)$.

5. Compute $w_{b^*} = \mathsf{PoI.wit}(\mathsf{CRS}_{b^*}, \mathsf{td}, x'_{b^*}, \pi'_{b^*})$.

6. Choose at random $u \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$, and compute $c' = \mathsf{Com}(w_{b^*}; u)$.

7. Generate a NIWI proof π'_{NIWI} for $(x, x'_0, x'_1, c') \in L_{\mathsf{PoI}}$, using the witness $(0, b^*, w_{b^*}, u)$.

8. Output $V^*_2\big(x, V^*_1(1^\lambda, x), (\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}})\big)$

$$\begin{aligned}
&\Pr_{x\leftarrow\mathcal{D}_\kappa}[\mathcal{A}(x) \in R_L(x)] \geq \\
&\Pr_{x\leftarrow\mathcal{D}_\kappa}[\mathcal{A}(x) \in R_L(x) \mid x \in \mathsf{GOOD}] \cdot \Pr_{x\leftarrow\mathcal{D}_\kappa}[x \in \mathsf{GOOD}] \geq \\
&\frac{1}{2s(\kappa)} \cdot \Pr_{x\leftarrow\mathcal{D}_\kappa}[\mathcal{A}(x) \in R_L(x) \mid x \in \mathsf{GOOD}] \geq \\
&\frac{1}{2s(\kappa)} \cdot \Pr_{x\leftarrow\mathcal{D}_\kappa}\big[\mathcal{A}(x) \in R_L(x) \mid \big(x \in \mathsf{GOOD}\big) \wedge \big(f(x) \in \mathsf{td}(x)\big)\big] \cdot \Pr_{x\leftarrow\mathcal{D}_\kappa}[f(x) \in \mathsf{td}(x) \mid x \in \mathsf{GOOD}] \geq \\
&\frac{1}{2s(\kappa)\cdot\mathsf{poly}(\kappa)} \cdot \Pr_{x\leftarrow\mathcal{D}_\kappa}\big[\mathcal{A}(x) \in R_L(x) \mid \big(x \in \mathsf{GOOD}\big) \wedge \big(f(x) \in \mathsf{td}(x)\big)\big] \geq \\
&\frac{1}{2s(\kappa)\cdot\mathsf{poly}(\kappa)}
\end{aligned}$$

contradicting the hardness of L, where the second inequality follows from Claim 13, the fourth inequality follows from Equation (6.12) (for infinitely many $\kappa \in \mathcal{K}$), and the last inequality follows from the witness indistinguishability property of the NIWI proof, since $\mathcal{A}(x)$ runs V^*_2 on input

$$\big(x, V^*_1(1^\lambda, x), (\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}})\big),$$

where the message $(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}})$ is distributed identically as a message generated by the honest prover, except that the NIWI proof π'_{NIWI} is generated using an alternative witness.

Case 2. For every poly-size computable function f there exists a negligible function μ and for every $\kappa \in \mathcal{K}$

$$\Pr[f(x) \in \mathsf{td}(x)] = \mu(\kappa) \tag{6.13}$$

where the probability is over $x \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$.

Claim 14. *For every $\kappa \in \mathcal{K}$ there exists $b_\kappa \in \{0,1\}$ such that the following holds: For every poly-size adversary $\mathcal{B}$ there exists a negligible function ν such that for every $\kappa \in \mathcal{K}$,*

$$\Pr\left[\mathcal{B}(x, V_1^*(1^\lambda, x), \mathsf{PKE.Enc}_{\mathsf{PK}_{b_\kappa}}(d)) = d\right] \leq \frac{1}{2} + \nu(\kappa),$$

where the probability is over $x \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$, $d \xleftarrow{\$} \{0,1\}$, and over the randomness of $\mathsf{PKE.Enc}_{\mathsf{PK}_{b_\kappa}}$, where PK_{b_κ} is computed by

$$\left((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}, \mathsf{st}^*\right) = V_1^*(1^\lambda, x).$$

Proof. Suppose for contradiction that there exists a poly-size algorithm $\mathcal{B}$, a polynomial q, and an infinite set $\mathcal{K}_0 \subseteq \mathcal{K}$, such that for every $\kappa \in \mathcal{K}_0$ and for every $b \in \{0,1\}$,

$$\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), \mathsf{PKE.Enc}_{\mathsf{PK}_b}(d)\big) = d\right] \geq \frac{1}{2} + \frac{1}{q(\kappa)} \tag{6.14}$$

where the probability is over $x \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$, $d \xleftarrow{\$} \{0,1\}$, and over the randomness of $\mathsf{PKE.Enc}_{\mathsf{PK}_b}$, where PK_b is computed by

$$\left((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\right) = V_1^*(1^\lambda, x).$$

For every $\kappa \in \mathcal{K}_0$ and every $b \in \{0,1\}$ we define a set $S_{\kappa,b}$ in the image of $\mathcal{D}_\kappa|\mathsf{GOOD}$, where $x \in S_{\kappa,b}$ if and only if

$$\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), \mathsf{PKE.Enc}_{\mathsf{PK}_b}(d)\big) = d\right] \geq \frac{1}{2} + \frac{1}{2q(\kappa)} \tag{6.15}$$

where the probability is over a randomly chosen $d \xleftarrow{\$} \{0,1\}$ and over the randomness of $\mathsf{PKE.Enc}_{\mathsf{PK}_b}$, and where PK_b is computed by

$$\left((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\right) = V_1^*(1^\lambda, x).$$

Note that for every $\kappa \in \mathcal{K}_0$ and every $b \in \{0,1\}$,

$$\begin{aligned}
&\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), \mathsf{PKE.Enc}_{\mathsf{PK}_b}(d)\big) = d\right] = \\
&\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), \mathsf{PKE.Enc}_{\mathsf{PK}_b}(d)\big) = d \;\middle|\; x \in S_{\kappa,b}\right] \cdot \Pr\left[x \in S_{\kappa,b}\right] + \\
&\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), \mathsf{PKE.Enc}_{\mathsf{PK}_b}(d)\big) = d \;\middle|\; x \notin S_{\kappa,b}\right] \cdot \Pr\left[x \notin S_{\kappa,b}\right] \leq \\
&\Pr\left[x \in S_{\kappa,b}\right] + \left(\frac{1}{2} + \frac{1}{2q(\kappa)}\right) \cdot \Pr\left[x \notin S_{\kappa,b}\right] \leq \\
&\Pr\left[x \in S_{\kappa,b}\right] + \left(\frac{1}{2} + \frac{1}{2q(\kappa)}\right).
\end{aligned}$$

This, together with Equation (6.14), implies that for every $\kappa \in \mathcal{K}_0$ and every $b \in \{0,1\}$,

$$\Pr_{x \leftarrow \mathcal{D}_\kappa | \mathsf{GOOD}}[x \in S_{\kappa,b}] \geq \frac{1}{2q(\kappa)}. \tag{6.16}$$

By Lemma 3 (in Section 3.3) and by Equation (6.15), there exists a non-uniform PPT algorithm $\mathcal{E}^*$, and a negligible function μ such that for for every $\kappa \in \mathcal{K}_0$, every $b \in \{0,1\}$, every $x \in S_{\kappa,b}$, and every $m = (m_1, \ldots, m_\kappa) \in \{0,1\}^\kappa$ and $r_1, \ldots, r_\kappa \in \{0,1\}^{\mathsf{poly}(\kappa)}$,

$$\Pr\left[\mathcal{E}^*(x, V_1^*(1^\lambda, x), \mathsf{ct}) = m\right] \geq 1 - \mu(\kappa) \tag{6.17}$$

where $\mathsf{ct} = \left(\mathsf{PKE.Enc}_{\mathsf{PK}_b}(m_1; r_1), \ldots, \mathsf{PKE.Enc}_{\mathsf{PK}_b}(m_\kappa; r_\kappa)\right)$.

We use $\mathcal{E}^*$ to construct a non-uniform PPT f such that $\Pr[f(x) \in \mathsf{td}(x)]$ is non-negligible, contradicting Equation (6.13). The function f, on input x in support of $\mathcal{D}_\kappa$, does the following:

- Compute $V_1^*(1^\lambda, x) = \left(\{(\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)\}_{b \in \{0,1\}}, \pi_{\mathsf{NIWI}}\right)$.
- Choose a random $b \xleftarrow{\$} \{0,1\}$ and output $\mathsf{td}'_b = \mathcal{E}^*(x, V_1^*(1^\lambda, x), \mathsf{ct}_b)$.

We next argue that for every $\kappa \in \mathcal{K}_0$,

$$\Pr\left[f(x) \in \mathsf{td}(x)\right] \geq \frac{1}{5q(\kappa)},$$

where the probability is over $x \leftarrow \mathcal{D}_\kappa | \mathsf{GOOD}$, contradicting Equation (6.13).

Let $E_{b,x}$ be the event that for $V_1^*(1^\lambda, x) = \left((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\right)$

$$\exists (r_b^1, r_b^2) \text{ s.t. } \left((\mathsf{CRS}_b, \mathsf{td}) = \mathsf{PoI.Setup}(1^\lambda; r_b^1)\right) \wedge \left(\mathsf{ct}_b = \mathsf{PKE.Enc}_{\mathsf{PK}_b}(\mathsf{td}, r_b^2)\right)$$

Since we assumed without loss of generality that π_{NIWI} is always accepting (see Remark 17), by the perfect soundness of the NIWI proof system it holds that

$$\Pr_{x \leftarrow \mathcal{D}_\kappa | \mathsf{GOOD}}\left[E_{0,x} \vee E_{1,x}\right] = 1. \tag{6.18}$$

Therefore,

$$\begin{aligned}
&\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[f(x)\in\mathsf{td}(x)\Big]=\\
&\frac{1}{2}\cdot\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[\mathcal{E}^*(x,V_1^*(1^\lambda,x),\mathsf{ct}_0)\in\mathsf{td}(x)\Big]+\frac{1}{2}\cdot\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[\mathcal{E}^*(x,V_1^*(1^\lambda,x),\mathsf{ct}_1)\in\mathsf{td}(x)\Big]\geq\\
&\frac{1}{2}\cdot\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[\mathcal{E}^*(x,V_1^*(1^\lambda,x),\mathsf{ct}_0)\in\mathsf{td}(x)\;\Big|\;x\in S_{\kappa,0}\Big]\cdot\Pr[x\in S_{\kappa,0}\Big]+\\
&\frac{1}{2}\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[\mathcal{E}^*(x,V_1^*(1^\lambda,x),\mathsf{ct}_1)\in\mathsf{td}(x)\;\Big|\;x\in S_{\kappa,1}\Big]\cdot\Pr\Big[x\in S_{\kappa,1}\Big]\geq\\
&\frac{1}{4q(\kappa)}\cdot\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[\mathcal{E}^*(x,V_1^*(1^\lambda,x),\mathsf{ct}_0)\in\mathsf{td}(x)\;\Big|\;x\in S_{\kappa,0}\Big]+\\
&\frac{1}{4q(\kappa)}\cdot\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[\mathcal{E}^*(x,V_1^*(1^\lambda,x),\mathsf{ct}_1)\in\mathsf{td}(x)\;\Big|\;x\in S_{\kappa,1}\Big]\geq\\
&\frac{1}{5q(\kappa)}\Big(\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[E_{0,x}\Big]+\Pr_{x\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}}\Big[E_{1,x}\Big]\Big)=\\
&\frac{1}{5q(\kappa)},
\end{aligned}$$

as desired, where the third equation follows from Equation (6.16), the fourth equation follows from Equation (6.17), and the last equation follows from Equation (6.18). □

In what follows we construct five non-uniform PPT provers, $P_1^*,P_2^*,P_3^*,P_4^*,P_5^*$, where P_5^* is the honest prover. We argue that for every $i\in[5]$, for every non-uniform PPT adversary $\mathcal{B}$, there exists a negligible function ν such that for every $\kappa\in\mathcal{K}$,

$$\Pr\Big[\mathcal{B}\big(x,V_1^*(1^\lambda,x),P_i^*(1^\lambda,x,w,\mathsf{pp}^*)\big)=w\Big]\leq\nu(\kappa)\tag{6.19}$$

where the probability is over $(x,w)\leftarrow\mathcal{D}_\kappa|\mathsf{GOOD}$ and over the random coin tosses of P_i^*, and where $(\mathsf{pp}^*,\mathsf{st}^*)=V_1^*(x)$. Note that this contradicts Equation (6.11), since Equation (6.11) implies that:

$$\begin{aligned}
\frac{1}{s(\kappa)}\leq&\Pr_{x\leftarrow\mathcal{D}_\kappa}[V_2^*(x,V_1^*(x),P^*(x,V_1^*(x)))\in R_L(x)]=\\
&\Pr_{x\leftarrow\mathcal{D}_\kappa}[V_2^*(x,V_1^*(x),P^*(x,V_1^*(x)))\in R_L(x)\mid\mathsf{GOOD}]\cdot\Pr[\mathsf{GOOD}]+\\
&\Pr_{x\leftarrow\mathcal{D}_\kappa}[V_2^*(x,V_1^*(x),P^*(x,V_1^*(x)))\in R_L(x)\mid\neg\mathsf{GOOD}]\cdot\Pr[\neg\mathsf{GOOD}]\leq\\
&\Pr_{x\leftarrow\mathcal{D}_\kappa}[V_2^*(x,V_1^*(x),P^*(x,V_1^*(x)))\in R_L(x)\mid\mathsf{GOOD}]+\\
&\Pr_{x\leftarrow\mathcal{D}_\kappa}[V_2^*(x,V_1^*(x),P^*(x,V_1^*(x)))\in R_L(x)\mid\neg\mathsf{GOOD}]\leq\\
&\Pr_{x\leftarrow\mathcal{D}_\kappa}[V_2^*(x,V_1^*(x),P^*(x,V_1^*(x)))\in R_L(x)\mid\mathsf{GOOD}]+\frac{1}{2s(\kappa)},
\end{aligned}$$

which in turn implies that

$$\Pr_{x\leftarrow\mathcal{D}_\kappa}[V_2^*(x,V_1^*(x),P^*(x,V_1^*(x)))\in R_L(x)\mid\mathsf{GOOD}]\geq\frac{1}{2s(\kappa)},$$

contradicting Equation (6.19) for $P_i^* = P_5^* = P$.

Prover P_1^*: We start by defining the non-uniform PPT prover P_1^* that on input $(1^\lambda, x, w, \mathsf{pp}^*)$, where $\mathsf{pp}^* = V_1^*(1^\lambda, x) = (x_0^*, x_1^*, \pi_{\mathsf{NIWI}})$ and where $x_b^* = (\mathsf{CRS}_b, \mathsf{PK}_b, \mathsf{ct}_b)$, ignores the witness w, and does the following: (Let $b^* = b_\kappa$, where b_κ is the bit from Claim 14.[¶¶])

1. Compute $(x'_{1-b^*}, \pi'_{1-b^*}) \leftarrow \mathsf{PoI.Gen}(\mathsf{CRS}_{1-b^*})$, choose at random $r'_{1-b^*}, s'_{1-b^*} \leftarrow \{0,1\}^\lambda$ and compute $c'_{1-b^*} = \mathsf{Com}(\pi'_{1-b^*}; r'_{1-b^*})$ and $\mathsf{ct}'_{1-b^*} = \mathsf{PKE.Enc}_{\mathsf{PK}_{1-b^*}}((\pi'_{1-b^*}, r'_{1-b^*}); s'_{1-b^*})$.
2. Generate a pair $(x'_{b^*}, w'_{b^*}) \in R_{L'}$ such that x'_{b^*} is distributed according to $\mathcal{D}'_\kappa$.
3. Generate $c'_{b^*} \leftarrow \mathsf{Com}(0)$ and $\mathsf{ct}'_{b^*} \leftarrow \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)$.
4. Choose at random $v' \leftarrow \{0,1\}^\lambda$ and compute $c' = \mathsf{Com}(w'_{b^*}; v')$.
5. Generate a NIWI proof for π'_{NIWI} for $(x, x'_0, x'_1, c') \in L_{\mathsf{PoI}}$, using witness (b^*, w'_{b^*}, v').
6. Output $\left(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\right)$.

Claim 15. *For every non-uniform* PPT *adversary $\mathcal{B}$, there exists a negligible function ν such that for every $\kappa \in \mathcal{K}$,*

$$\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), P_1^*(1^\lambda, x, w, \mathsf{pp}^*)\big) = w\right] \leq \nu(\kappa)$$

and

$$\Pr\left[\mathcal{B}\Big(x, V_1^*(1^\lambda, x), P_1^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(d)\Big) = d\right] \leq \frac{1}{2} + \nu(\kappa),$$

where the probabilities are over $x \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$, $d \xleftarrow{\$} \{0,1\}$, and over the randomness of P_1^*, where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(1^\lambda, x)$. In addition, the second probability is also over $\mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}$ *where*

$$\mathsf{pp}^* = \Big((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\Big).$$

Proof. The first equation follows from the fact that the messages of V_1^* and P_1^* are efficiently computable given only $(1^\lambda, x)$. The second equation follows from Claim 14, together with the fact that P_1^* is efficiently computable given only $(1^\lambda, x)$ and from the hardness of language L. □

Prover P_2^*: We next define a non-uniform PPT algorithm P_2^*, which is identical to P_1^* except that P_2^* generates $c'_{b^*} \leftarrow \mathsf{Com}(\pi'_{b^*})$ as opposed to $c'_{b^*} \leftarrow \mathsf{Com}(0)$.

In more detail, P_2^* on input $(1^\lambda, x, w, \mathsf{pp}^*)$, ignores the witness w, and does the following: (Let $b^* = b_\kappa$, where b_κ is the bit from Claim 14.[***])

[¶¶] P_1^* has the bit b_κ hard-wired into it.

[***] P_2^* has the bit b_κ hard-wired into it.

1. Compute $(x'_{1-b^*}, \pi'_{1-b^*}) \leftarrow \mathsf{PoI.Gen}(\mathsf{CRS}_{1-b^*})$, choose at random $r'_{1-b^*}, s'_{1-b^*} \leftarrow \{0,1\}^\lambda$ and compute $c'_{1-b^*} = \mathsf{Com}(\pi'_{1-b^*}; r'_{1-b^*})$ and $\mathsf{ct}'_{1-b^*} = \mathsf{PKE.Enc}_{\mathsf{PK}_{1-b^*}}((\pi'_{1-b^*}, r'_{1-b^*}); s'_{1-b^*})$.
2. Compute $(x'_{b^*}, \pi'_{b^*}) \leftarrow \mathsf{PoI.Gen}(\mathsf{CRS}_{b^*})$, choose at random $r'_{b^*} \leftarrow \{0,1\}^\lambda$ and compute $c'_{b^*} = \mathsf{Com}(\pi'_{b^*}; r'_{b^*})$.
3. Generate $\mathsf{ct}'_{b^*} \leftarrow \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)$.
4. Inefficiently, compute w'_{b^*} such that $(x'_{b^*}, w'_{b^*}) \in L'$.
5. Choose at random $v' \leftarrow \{0,1\}^\lambda$ and compute $c' = \mathsf{Com}(w'_{b^*}; v')$.
6. Generate a NIWI proof for π'_{NIWI} for $(x, x'_0, x'_1, c') \in L_{\mathsf{PoI}}$, using witness (b^*, w'_{b^*}, v').
7. Output $\left(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\right)$.

Claim 16. *For every non-uniform* PPT *adversary $\mathcal{B}$, there exists a negligible function ν such that for every $\kappa \in \mathcal{K}$*

$$\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*)\big) = w\right] \leq \nu(\kappa)$$

and

$$\Pr\left[\mathcal{B}\Big(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(d)\Big) = d\right] \leq \frac{1}{2} + \nu(\kappa),$$

where the probability is over $(x,w) \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$ and over the random coin tosses of P_2^*, and where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(1^\lambda, x)$. In addition, the second probability is also over $\mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}$ *where*

$$\mathsf{pp}^* = \Big((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\Big).$$

Claim 16 follows from Claim 15, together with the hiding property of the commitment scheme.

Prover P_3^*: We next define a non-uniform PPT algorithm P_3^*, which is identical to P_2^* except that P_3^* generates $\mathsf{ct}'_{b^*} \leftarrow \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(\pi'_{b^*}, r'_{b^*})$ where $c'_{b^*} = \mathsf{Com}(\pi'_{b^*}; r'_{b^*})$ as opposed to $\mathsf{ct}'_{b^*} \leftarrow \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)$.

In more detail, P_3^* on input $(1^\lambda, x, w, \mathsf{pp}^*)$, ignores the witness w, and does the following: (Let $b^* = b_\kappa$, where b_κ is the bit from Claim 14.[†††])

1. For every $b \in \{0,1\}$, compute $(x'_b, \pi'_b) \leftarrow \mathsf{PoI.Gen}(\mathsf{CRS}_b)$, choose at random $r'_b, s'_b \leftarrow \{0,1\}^\lambda$ and compute $c'_b = \mathsf{Com}(\pi'_b; r'_b)$ and $\mathsf{ct}'_b = \mathsf{PKE.Enc}_{\mathsf{PK}_b}((\pi'_b, r'_b); s'_b)$.
2. Inefficiently, compute w'_{b^*} such that $(x'_{b^*}, w'_{b^*}) \in L'$.
3. Choose at random $v' \leftarrow \{0,1\}^\lambda$ and compute $c' = \mathsf{Com}(w'_{b^*}; v')$.

[†††] P_3^* has the bit b_κ hard-wired into it.

4. Generate a NIWI proof for π'_{NIWI} for $(x, x'_0, x'_1, c') \in L_{\mathsf{Pol}}$, using witness (b^*, w'_{b^*}, v').
5. Output $\left(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\right)$.

Claim 17. *For every non-uniform* PPT *adversary $\mathcal{B}$, there exists a negligible function ν such that for every $\kappa \in \mathcal{K}$,*

$$\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), P_3^*(1^\lambda, x, w, \mathsf{pp}^*)\big) = w\right] \leq \nu(\kappa)$$

where the probability is over $(x,w) \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$ and over the random coin tosses of P_3^*, and where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(1^\lambda, x)$.

Proof. Claim 16 implies,

$$\left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)\right) \approx \left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(1)\right) \tag{6.20}$$

where the probability is over $(x,w) \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$ and over the random coin tosses of P_2^*, and where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(1^\lambda, x)$ and $\mathsf{pp}^* = \big((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\big)$.

Consider the following inefficient function f_R, parameterized with randomness R, that on input (PK^*, x) does the following:

1. Compute $V_1^*(1^\lambda, x) = (\mathsf{pp}^*, \mathsf{st}^*)$ where $\lambda = g(|x|)$. Parse $\mathsf{pp}^* = \Big((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\Big)$. Let b^* be such that $\mathsf{PK}_{b^*} = \mathsf{PK}^*$. If no such b^* exists, then output $\perp$.
2. Use the randomness R to compute $\mathsf{msg}_P = P_1^*(1^\lambda, x, \mathsf{pp}^*; R)$.
3. Parse $\mathsf{msg}_P = \left(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\right)$ and inefficiently compute (π'_{b^*}, r'_{b^*}) such that $c'_{b^*} = \mathsf{Com}(\pi'_{b^*}; r'_{b^*})$ and $\mathsf{Pol.Verify}(\mathsf{CRS}_{b^*}, x'_{b^*}, \pi'_{b^*}) = 1$. If no such (π'_{b^*}, r'_{b^*}) exists, output $\perp$.
4. Output (π'_{b^*}, r'_{b^*}).

Consider the distribution $\mathcal{D}^*$ that on input security parameter 1^λ does the following:

1. Compute $\kappa = g^{-1}(\lambda)$.[‡‡‡] Sample $x \leftarrow D_\kappa$.
2. Compute $V_1^*(1^\lambda, x) = (\mathsf{pp}^*, \mathsf{st}^*)$.
3. Parse $\mathsf{pp}^* = \Big((\mathsf{CRS}_0, \mathsf{PK}_0, \mathsf{ct}_0), (\mathsf{CRS}_1, \mathsf{PK}_1, \mathsf{ct}_1), \pi_{\mathsf{NIWI}}\Big)$ and output (x, PK_{b^*}).

By Equation (6.20), we have that for any randomness R,

$$\left(\mathsf{PK}_{b^*}, x, R, \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)\right) \approx \left(\mathsf{PK}_{b^*}, x, R, \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(1)\right)$$

[‡‡‡] Without loss of generality, we assume that this is an integer.

By KDM security with ε-bounded auxiliary input (as per Definition 13),

$$\left(\mathsf{PK}_{b^*}, x, R, \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)\right) \approx \left(\mathsf{PK}_{b^*}, x, R, \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(f_R(\mathsf{PK}_{b^*}, x))\right)$$

Since $V_1^*(1^\lambda, x), P_1^*(1^\lambda, x, w, \mathsf{pp}^*)$ can be efficiently computed given x,[§§§] it follows that

$$\left(x, V_1^*(1^\lambda, x), P_1^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)\right) \approx \left(x, V_1^*(1^\lambda, x), P_1^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(\pi'_{b^*}, r'_{b^*})\right)$$

By Claim 16,

$$\left(x, V_1^*(1^\lambda, x), P_1^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)\right) \approx \left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)\right)$$

By the hiding property of commitment, it also follows that

$$\left(x, V_1^*(1^\lambda, x), P_1^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(\pi'_{b^*}, r'_{b^*})\right) \approx \left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(\pi'_{b^*}, r'_{b^*})\right)$$

Hence we have,

$$\left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(0)\right) \approx \left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*), \mathsf{PKE.Enc}_{\mathsf{PK}_{b^*}}(\pi'_{b^*}, r'_{b^*})\right)$$

which is equivalent to

$$\left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*)\right) \approx \left(x, V_1^*(1^\lambda, x), P_3^*(1^\lambda, x, w, \mathsf{pp}^*)\right) \tag{6.21}$$

Therefore we conclude that if there exists a non-uniform PPT adversary $\mathcal{B}$ and a polynomial p such that for infinitely many $\kappa \in \mathcal{K}$,

$$\Pr\left[\mathcal{B}\left(x, V_1^*(1^\lambda, x), P_3^*(1^\lambda, x, w, \mathsf{pp}^*)\right) = w\right] \geq \frac{1}{p(\kappa)}$$

where the probability is over $(x, w) \leftarrow \mathcal{D}_\kappa | \mathsf{GOOD}$ and over the random coin tosses of P_3^*, and where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(x)$. Then by Equation (6.21) for infinitely many $\kappa \in \mathcal{K}$,

$$\Pr\left[\mathcal{B}\left(x, V_1^*(1^\lambda, x), P_2^*(1^\lambda, x, w, \mathsf{pp}^*)\right) = w\right] \geq \frac{1}{p(\kappa)} - \mathsf{negl}(\kappa)$$

where the probability is over $(x, w) \leftarrow \mathcal{D}_\kappa | \mathsf{GOOD}$ and over the random coin tosses of P_3^*, and where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(x)$. This directly contradicts Claim 16.

□

Prover P_4^*: We next define a non-uniform PPT algorithm P_4^*, which is identical to P_3^* except that uses the witness w corresponding to x in the NIWI proof. In more detail, P_4^* on input $(1^\lambda, x, w, \mathsf{pp}^*)$ does the following: (Let $b^* = b_\kappa$, where b_κ is the bit from Claim 14.)

[§§§] Recall that P_1^* does not use witness w.

1. For every $b \in \{0,1\}$, compute $(x'_b, \pi'_b) \leftarrow \mathsf{PoI.Gen}(\mathsf{CRS}_b)$, choose at random $r'_b, s'_b \leftarrow \{0,1\}^\lambda$ and compute $c'_b = \mathsf{Com}(\pi'_b; r'_b)$ and $\mathsf{ct}'_b = \mathsf{PKE.Enc}_{\mathsf{PK}_b}((\pi'_b, r'_b); s'_b)$.
2. Inefficiently, compute w'_{b^*} such that $(x'_{b^*}, w'_{b^*}) \in L'$.
3. Choose at random $v' \leftarrow \{0,1\}^\lambda$ and compute $c' = \mathsf{Com}(w'_{b^*}; v')$.
4. Generate a NIWI proof for π'_{NIWI} for $(x, x'_0, x'_1, c') \in L_{\mathsf{PoI}}$, using witness w.
5. Output $\left(\{(x'_b, c'_b, \mathsf{ct}'_b)\}_{b\in\{0,1\}}, c', \pi'_{\mathsf{NIWI}}\right)$.

Claim 18. *For every non-uniform* PPT *adversary $\mathcal{B}$, there exists a negligible function ν such that for every $\kappa \in \mathcal{K}$,*

$$\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), P_4^*(1^\lambda, x, w, \mathsf{pp}^*)\big) = w\right] \leq \nu(\kappa)$$

where the probability is over $(x,w) \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$ and over the random coin tosses of P_4^*, and where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(x)$.

Claim 18 follows from Claim 17, together with the witness indistinguishability of the underlying NIWI proof system.

Prover P_5^*: Finally, we define P_5^* to be the honest prover. Note that the only difference between P_5^* and P_4^* is in the way commitment c' is generated. P_4^* generates $c' \leftarrow \mathsf{Com}(w_{b^*})$ where $(x'_{b^*}, w_{b^*}) \in R_{L'}$ and the honest prover generates $c'_{b^*} \leftarrow \mathsf{Com}(0)$.

Claim 19. *For every non-uniform* PPT *adversary $\mathcal{B}$, there exists a negligible function ν such that for every $\kappa \in \mathcal{K}$,*

$$\Pr\left[\mathcal{B}\big(x, V_1^*(1^\lambda, x), P_5^*(1^\lambda, x, w, \mathsf{pp}^*)\big) = w\right] \leq \nu(\kappa)$$

where the probability is over $(x,w) \leftarrow \mathcal{D}_\kappa|\mathsf{GOOD}$ and over the random coin tosses of P_5^*, and where $(\mathsf{pp}^*, \mathsf{st}^*) = V_1^*(x)$.

Claim 19 follows from Claim 18, together with the hiding property of the commitment scheme.

□

Bibliography

[ACJ17] Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. Cryptography with updates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 445–472. Springer, 2017.

[ADKL19] Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic nizk and niwi proofs. 2019.

[AGM18] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In *IACR Cryptology ePrint Archive*, 2018.

[AGP14] Prabhanjan Ananth, Vipul Goyal, and Omkant Pandey. Interactive proofs under continual memory leakage. In *International Cryptology Conference*, pages 164–182. Springer, 2014.

[AN11] Tolga Acar and Lan Nguyen. Homomorphic proofs and applications. `https://www.microsoft.com/en-us/research/wp-content/uploads/2011/03/rac.pdf`, 2011.

[ANSF16] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. Blockstack: A global naming and storage system secured by blockchains. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 181–194, Denver, CO, 2016. USENIX Association.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[BBK+16] Nir Bitansky, Zvika Brakerski, Yael Kalai, Omer Paneth, and Vinod Vaikuntanathan. 3-message zero knowledge against human ignorance. In *Theory of Cryptography Conference*, pages 57–83. Springer, 2016.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Annual International Cryptology Conference*, pages 41–55. Springer, 2004.

[BCC+09a] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Advances in Cryptology-CRYPTO 2009*, pages 108–125. Springer, 2009.

[BCC+09b] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Advances in Cryptology-CRYPTO 2009*, pages 108–125. Springer, 2009.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the snark. *Journal of Cryptology*, 30(4):989–1066, 2017.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 111–120. ACM, 2013.

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: optimizations and applications. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2105–2122. ACM, 2017.

[BCKP17] Nir Bitansky, Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On virtual grey box obfuscation for general circuits. *Algorithmica*, 79(4):1014–1051, 2017.

[BCPR16] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. *SIAM Journal on Computing*, 45(5):1910–1952, 2016.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Guiseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 149–168. Springer, 2011.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. Cryptology ePrint Archive, Report 2001/069, 2001. `http://eprint.iacr.org/`.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[BKP18a] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: A paradigm for keyless hash functions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 671–684. ACM, 2018.

[BKP18b] Nir Bitansky, Dakshita Khurana, and Omer Paneth. Weak zero-knowledge beyond the black-box barrier. *IACR Cryptology ePrint Archive*, 2018:895, 2018.

[Bon16] Joseph Bonneau. Ethiks: Using ethereum to audit a coniks key transparency log. In *International Conference on Financial Cryptography and Data Security*, pages 95–105. Springer, 2016.

[BOV05] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. *IACR Cryptology ePrint Archive*, 2005:365, 2005.

[BP02] Mihir Bellare and Adriana Palacio. Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In *Annual International Cryptology Conference*, pages 162–177. Springer, 2002.

[BP04] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *Annual International Cryptology Conference*, pages 273–289. Springer, 2004.

[BP12] Nir Bitansky and Omer Paneth. Point obfuscation and 3-round zero-knowledge. In *Theory of Cryptography Conference*, pages 190–208. Springer, 2012.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-shamir and correlation intractability from strong kdm-secure encryption. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 91–122, 2018.

[CD09] Ran Canetti and Ronny Ramzi Dakdouk. Towards a theory of extractable functions. In *Theory of Cryptography Conference*, pages 595–613. Springer, 2009.

[CDGM19] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. Seemless: Secure end-to-end encrypted messaging with less trust. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1639–1656. ACM, 2019.

[CHKO08] Philippe Camacho, Alejandro Hevia, Marcos Kiwi, and Roberto Opazo. Strong accumulators from collision-resistant hashing. In *International Conference on Information Security*, pages 471–486. Springer, 2008.

[CHL+05] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 422–439. Springer, 2005.

[CKC+10] Ruben Cuevas, Michal Kryczka, Angel Cuevas, Sebastian Kaune, Carmen Guerrero, and Reza Rejaie. Is content publishing in bittorrent altruistic or profit-driven? In *Proceedings of the 6th International COnference*, page 11. ACM, 2010.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–300. Springer, 2012.

[CKLM13a] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. http://eprint.iacr.org/2013/179, 2013.

[CKLM13b] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable nizks and an application to compact shuffles. In *Theory of Cryptography*, pages 100–119. Springer, 2013.

[CL07] Melissa Chase and Anna Lysyanskaya. Simulatable vrfs with applications to multi-theorem nizk. In *Annual International Cryptology Conference*, pages 303–322. Springer, 2007.

[Dam92] Ivan Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with preprocessing. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 341–355. Springer, 1992.

[DGS03] Ivan Damgård, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In *Secure Electronic Voting*, pages 77–99. Springer, 2003.

[DK18] Apoorvaa Deshpande and Yael Kalai. Proofs of ignorance and applications to 2-message witness hiding. *IACR Cryptology ePrint Archive*, 2018:896, 2018.

[DN00] Cynthia Dwork and Moni Naor. Zaps and their applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 283–293. IEEE, 2000.

[DSYC17] Yi Deng, Xuyang Song, Jingyue Yu, and Yu Chen. On instance compression, schnorr/guillou-quisquater, and the security of classic protocols for unique witness relations. *IACR Cryptology ePrint Archive*, 2017:390, 2017.

[EK15] Mohammad Etemad and Alptekin Kupcu. Efficient key authentication service for secure end-to-end communications. Cryptology ePrint Archive, Report 2015/833, 2015. `https://eprint.iacr.org/2015/833`.

[EMBB17] Saba Eskandarian, Eran Messeri, Joe Bonneau, and Dan Boneh. Certificate transparency with privacy. *arXiv preprint arXiv:1703.02209*, 2017.

[FB14] T Fox-Brewster. Whatsapp adds end-to-end encryption using textsecure. *The Guardian, Nov*, 2014.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[FKP19] Cody Freitag, Ilan Komargodski, and Rafael Pass. On the impossibility of strong kdm security with auxiliary input. *Personal Communication*, 2019.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 416–426. ACM, 1990.

[Gen09] Craig Gentry. A fully homomorphic encryption scheme [ph. d. thesis]. *International Journal of Distributed Sensor Networks, Stanford University*, 2009.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np statements in zero-knowledge and a methodology of cryptographic protocol design. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 171–185. Springer, 1986.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7:1–32, 1994.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *CRYPTO*, volume 4117, pages 97–111. Springer, 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 339–358. Springer, 2006.

[GQ88] Louis C Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 123–128. Springer, 1988.

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *Annual International Cryptology Conference*, pages 192–208. Springer, 2009.

[Gro10] Jens Groth. Short non-interactive zero-knowledge proofs. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 341–358. Springer, 2010.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 469–477. ACM, 2015.

[HRS09] Iftach Haitner, Alon Rosen, and Ronen Shaltiel. On the (im) possibility of arthur-merlin witness hiding protocols. In *Theory of Cryptography Conference*, pages 220–237. Springer, 2009.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In *Annual International Cryptology Conference*, pages 408–423. Springer, 1998.

[JKKR17] Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Ron Rothblum. Distinguisher-dependent simulation in two rounds and its applications. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 158–189, 2017.

[Key14] Keybase.io. Keybase is now writing to the bitcoin blockchain. `https://keybase.io/docs/server_security/merkle_root_in_bitcoin_blockchain`, 2014. Accessed: 2018-10-05.

[Key19a] Keybase.io. Keybase is not softer than tofu. `https://keybase.io/blog/chat-apps-softer-than-tofu`, 2019. Accessed: 2019-05-05.

[Key19b] Keybase.io. Managing teams and keys with keybase. `https://keybase.io/docs-assets/blog/NCC_Group_Keybase_KB2018_Public_Report_2019-02-27_v1.3.pdf`, 2019. Accessed: 2019-05-05.

[Key19c] Keybase.io. Protocol security review. `https://rwc.iacr.org/2019/slides/keybase-rwc2019.pdf`, 2019. Accessed: 2019-05-05.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732. ACM, 1992.

[KP98] Joe Kilian and Erez Petrank. An efficient noninteractive zero-knowledge proof system for np with general assumptions. *Journal of Cryptology*, 11(1):1–27, 1998.

[KW18] Sam Kim and David J Wu. Multi-theorem preprocessing nizks from lattices. In *Annual International Cryptology Conference*, pages 733–765. Springer, 2018.

[Lis05] Moses Liskov. Updatable zero-knowledge databases. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 174–198. Springer, 2005.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. Technical report, 2013.

[MBB+15] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. Coniks: Bringing key transparency to end users. In *Usenix Security*, pages 383–398, 2015.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[MRK03] Silvio Micali, Michael Rabin, and Joe Kilian. Zero-knowledge sets. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium*, pages 80–91. IEEE, 2003.

[Nam14] Namecoin. `https://namecoin.org`, 2014. Accessed: 2018-10-05.

[NT16] Assa Naveh and Eran Tromer. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 255–271. IEEE, 2016.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. STOC 1990: 427-437, 1990.

[oSBS15] Conference of State Bank Supervisors. State regulatory requirements for virtual currency activities. *CSBS Model Regulatory Framework*, September, 2015.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 160–176. Springer, 2003.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[Pro12] LEAP Encryption Access Project. Nicknym. `https://leap.se/en/docs/design/nicknym`, 2012. Accessed: 2018-10-05.

[RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[Sch15] B Schneier. Apple's imessage encryption seems to be pretty good, 2015.

[sig16] signal.org. Identity binding. `https://www.signal.org/docs/specifications/x3dh`, 2016. Accessed: 2019-05-05.

[TD17] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via bitcoin. In *IEEE Symp. on Security and Privacy*, 2017.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography Conference*, pages 1–18. Springer, 2008.

[wha17] WhatsApp Security Vulnerability. `https://www.schneier.com/blog/archives/2017/01/whatsapp_securi.html`, 2017. Accessed: 2019-01-25.