University of California
Los Angeles

New Frontiers in Secure Computation

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Abhishek Jain

2012

Abstract of the Dissertation

New Frontiers in Secure Computation

by

Abhishek Jain

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2012

Professor Rafail Ostrovsky, Co-chair

Professor Amit Sahai, Co-chair

The notion of *secure computation* is central to cryptography. Introduced in the seminal works of Yao [FOCS'82, FOCS'86] and Goldreich, Micali and Wigderson [STOC'87], secure multi-party computation allows a group of (mutually) distrustful parties to jointly compute any functionality over their individual private inputs in such a manner that the honest parties obtain the correct outputs and no group of malicious parties learn anything beyond their inputs and the prescribed outputs. General feasibility results for secure computation were given by Yao [FOCS'86] and Goldreich et al. [STOC'87] more than two decades ago. Subsequent to these works, designing secure computation protocols that can tolerate more powerful adversaries and satisfy stronger notions of security has been a very active area of research. In this dissertation, we study two such new frontiers in the area of secure computation.

In the first part of this dissertation, we initiate a study of designing leakage-resilient interactive protocols. Specifically, we consider the scenario where an adversary, in addition to corrupting a subset of parties, can *leak* (potentially via physical attacks) arbitrary information from the secret state of any honest party.

This is in contrast to the standard notion of secure computation where it is assumed that the adversary only has "black-box" access to the honest parties. In particular, we formalize a meaningful definition of leakage-resilient zero knowledge proof systems and provide constructions that satisfy our notion. We also discuss various applications of our results.

The second part of this dissertation concerns with the general question of whether it is possible to securely run cryptographic protocols over an insecure network environment such as the Internet. It is well-known that the standard notion of secure computation is only relevant to the "stand-alone" setting where a single protocol is being executed in isolation; as such it does not guarantee security when multiple protocol sessions may be executed *concurrently* under the control of an adversary who is present across all sessions. We consider the open problem of constructing secure password-based authenticated key exchange protocols in such a setting in the "plain model" (i.e., without assuming any trusted infrastructure or random oracles). We give the first construction of such a protocol based on standard cryptographic assumptions. Our results are in fact much more general, and extend to other functionalities w.r.t. a (necessarily) weakened notion of concurrently secure computation.

The results presented in this dissertation stem from two papers which are joint works with Sanjam Garg and Amit Sahai, and with Vipul Goyal and Rafail Ostrovsky, respectively.

The dissertation of Abhishek Jain is approved.

Sheila Greibach

Richard Elman

Amit Sahai, Committee Co-chair

Rafail Ostrovsky, Committee Co-chair

University of California, Los Angeles

2012

Table of Contents

List of Figures

Acknowledgments

I have spent some of the most memorable years of my life attending graduate school at UCLA and I have numerous people to thank for it.

First, and foremost, I would like to thank my advisors, Rafail Ostrovsky and Amit Sahai, for showing faith in me and taking me as their student when I had only enthusiasm to show for on my resume. Over the years, Rafi and Amit have provided me continuous support and inspiration, and I feel very fortunate to have had them as my advisors. I look forward to having more opportunities to learn from and work with them in the future.

Rafi's breadth of knowledge in theoretical computer science is remarkable, and so is his ability to draw interesting connections between seemingly unrelated concepts. Indeed, the latter is, in my opinion, the magic secret behind his amazing ability to find one excellent research problem after another. Like all my crypto colleagues at UCLA, I have benefited greatly from this. Rafi is a constant source of energy and inspiration – nothing exemplifies this better than the fact that a typical day in Rafi's life consists of multiple(!) productive research meetings, and more. I was lucky to be Rafi's student and I sincerely thank him for his invaluable advice and support throughout my PhD.

Amit is one of the most brilliant people I have met. Among his many wonderful qualities, most striking is his magical ability to understand my chaotic thoughts (even when I don't understand them myself) and elevate them to clear, organized ideas. Amit has taught me how to think intuitively, and how to discard a "bad idea" quickly before it can lead one astray. I owe him greatly for my evolution as a researcher. Amit has helped me in making many important decisions, and I am thankful for his advice on career and life. Outside work, Amit and his

wife, Pragati, provided all of us a home away from home, by inviting us to join in celebration of Indian festivals like Diwali. It has been a privilege being Amit's student and I am going to miss the experience.

I cannot thank enough Vipul Goyal and Omkant Pandey for their friendship and continued support in my research career. Vipul introduced me to the world of research even before I started graduate school, and I thank him for mentoring me during my early years at UCLA. Over the years, Vipul has had a huge impact on my research career, and I thank him for always being there. (I only ask him that we eat at California Pizza Kitchen a little less often.) Omkant and Vipul taught me a lot about zero-knowledge and secure computation, and I thank them for answering my endless questions on these topics patiently. Omkant also taught me how to write better papers. I owe Omkant a spicy bowl of dal with garlic for all his help over the years.

I would like to thank Krzysztof Pietrzak and all the members of the cryptography group at CWI Amsterdam, including Ronald Cramer, Serge Fehr and Eike Kiltz, for hosting me during two visits. Krzysztof is freakishly smart and also one of the most fun people to have around. I thank him for helping me diversify my research interests and for several fruitful collaborations. Outside work, Krzysztof took me to the best bars and restaurants and made sure I had the best time in Amsterdam.

I thank Tal Rabin and all the members of the cryptography group at IBM T. J. Watson research center – David Cash, Rosario Gennaro, Craig Gentry, Shai Halevi, Hugo Krawczyk, Charanjit Jutla and Daniel Wichs – for hosting me during a summer internship. Tal is one of the most genuine person I have ever met and I am thankful for her kindness, continued support and interest in my career. I thank Daniel for being such a wonderful collaborator.

I would like to thank Yael Tauman Kalai for hosting me during my visits to Microsoft Research New England. Yael is a role model for me, and I wish I could emulate her work ethic. The amount of work that she manages to get done in the limited twenty-four hours of a day is beyond the scope of most human beings. My research discussions with Yael have been extremely productive, all thanks to her brilliance. I have had the most pleasant experience collaborating with her and I look forward to spending more time with her in the near future.

I thank all of my co-authors for doing all the hard work on our papers: Shweta Agrawal, Gilad Asharov, Elette Boyle, David Cash, Yevgeniy Dodis, Sanjam Garg, Shafi Goldwasser, Vipul Goyal, Yael Tauman Kalai, Eike Kiltz, Adriana Lopez-Alt, Tal Moran, Amit Sahai, Rafail Ostrovsky, Omkant Pandey, Krzysztof Pietrzak, Manoj Prabhakaran, Aris Tentes, Eran Tromer, Vinod Vaikuntanathan, Daniele Venturi, Daniel Wichs.

I thank Nishanth Chandran, Sanjam Garg, Hemanta Maji and Akshay Wadia for contributing to my understanding of cryptography, and for being such wonderful friends. I have immensely enjoyed numerous research (as well as non-research) discussions with each of them, and I thank them for that. I will fondly remember all the fun, late night technical discussions I had with Akshay and Sanjam when we were roommates. I also thank Ivan Visconti for teaching me many things about concurrent security and non-malleability. I thank the other members of the cryptography group for contributing to my fantastic experience at UCLA: Shweta Agrawal, Paul Bunn, Chongwon Cho, Clint Givens, Ran Gelles, Brett Hemenway, Bhavana Kanakurthi, Abishek Kumarasubramaniam, Steve Lu, Vanishree Hanumantha Rao, Alessandra Scafuro, Hakan Seyalioglu and Bhavani Shankar.

I would like to thank Adrian Perrig for showing faith in me and hosting me

for a year long internship at CMU. Adrian was very kind to me and I thank him for giving me the wonderful opportunity of working with his group.

Special thanks goes to Virendra Kumar. Viru has been a great friend since undergraduate and we embarked on our PhD journeys together. I cherish the fun times we have had together.

I thank Divyanshu Jain and Karthik Gururaj for being the best roommates one could ever ask for. DJ and KG are two of the nicest people I have ever met and I am thankful to them for tolerating all my idiosyncrasies without ever complaining. I thank DJ for all the fun times during our numerous travels, hikes, and photography expeditions. I thank KG for the wonderful food and allowing me to get by without cooking for almost an entire year. I will deeply miss having them around! I thank my other friends in Los Angeles – Ankit Kumar, Parth Patel, Srinivas Sista, Murali Vasudevan, Khushru Wadia – and all my photography friends, especially Norman Schwartz and Rick Smith, for enriching my life beyond work.

I thank Richard Elman and Sheila Greibach for agreeing to serve on my dissertation committee.

Finally, I would like to thank my father and my mother for their unconditional support and love, and for allowing me to choose my own path in life.

Vita

2002–2006	B.Tech. (Computer Science and Engineering), IIT BHU, India
2007–2009	M.S. (Computer Science), UCLA.
2007–2012	Ph.D. student, Department of Computer Science, UCLA.
2008–2009	Teaching Assistant, Department of Computer Science, UCLA.
2009–2012	Research Assistant, Department of Computer Science, UCLA.
Summer 2010	Research Intern at CWI Amsterdam.
Summer 2011	Research Intern at IBM T. J. Watson Research Center, NY.
Summer 2012	Research Intern at Microsoft Research New England.
2012	Symantec Outstanding Graduating Student Award

Publications

Shweta Agrawal, Vipul Goyal, Abhishek Jain, Manoj Prabhakaran and Amit Sahai, *"New Impossibility Results for Concurrent Composition and a Non-Interactive Completeness Theorem for Secure Computation"*. In **CRYPTO 2012** - Advances in Cryptology, Santa Barbara, USA, August 2012.

Elette Boyle, Shafi Goldwasser, Abhishek Jain and Yael Tauman Kalai, *"Multiparty Computation Secure Against Continual Memory Leakage"*. In **STOC 2012** – ACM Symposium on Theory of Computing, New York, USA, May 2012.

Gilad Asharov, Abhishek Jain, Adriana Lopez-Alt, Eran Tromer, Vinod Vaikuntanathan and Daniel Wichs, *"MPC with Low Communication, Computation, and Interaction via Threshold FHE"*. In **EUROCRYPT 2012** - Advances in Cryptology, Cambridge, UK, April 2012.

Sanjam Garg, Vipul Goyal, Abhishek Jain and Amit Sahai, *"Concurrently Secure Computation in Constant Rounds"*. In **EUROCRYPT 2012** - Advances in Cryptology, Cambridge, UK, April 2012.

Abhishek Jain, Krzysztof Pietrzak and Aris Tentes. *"Hardness Preserving Constructions of Pseudorandom Functions"*. In **TCC 2012** - Theory of Cryptrography Conference, Taormina, Italy, March 2012.

Yevgeniy Dodis, Abhishek Jain, Tal Moran and Daniel Wichs, *"Counterexamples to Hardness Amplification Beyond Negligible"*. In **TCC 2012** - Theory of Cryptrography Conference, Taormina, Italy, March 2012.

Sanjam Garg, Abhishek Jain and Amit Sahai, *"Leakage-Resilient Zero Knowledge"*. In **CRYPTO 2011** - Advances in Cryptology, Santa Barbara, USA, August 2011.

Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain and Daniele Venturi.

"Efficient Authentication from Hard Learning Problems". In **EUROCRYPT 2011** - Advances in Cryptology, Tallin, Estonia, May 2011.

Abhishek Jain and Krzysztof Pietrzak, *"Leakage Resilience Amplification via Parallel Repetition, Revisited"*. In **TCC 2011** - Theory of Cryptrography Conference, Providence, USA, March 2011.

Sanjam Garg, Vipul Goyal, Abhishek Jain and Amit Sahai, *"Bringing People of Different Beliefs Together to do UC"*. In **TCC 2011** - Theory of Cryptrography Conference, Providence, USA, March 2011.

Vipul Goyal, Abhishek Jain and Rafail Ostrovsky, *"Password-Authenticated Key Exchange on the Internet in the Plain Model"*. **CRYPTO 2010** - Advances in Cryptology, Santa Barbara, USA, August 2010.

Vipul Goyal and Abhishek Jain, *"On the Round Complexity of Covert Computation"*. In **STOC 2010** – ACM Symposium on Theory of Computing, Boston, USA, June 2010.

Vipul Goyal, Abhishek Jain, Omkant Pandey and Amit Sahai *"Bounded Ciphertext Policy Attribute-Based Encryption"*. In **ICALP 2008** – International Colloquium on Automata, Languages and Programming, Reykjavik, Iceland, July 2008.

CHAPTER 1

Introduction

The notion of secure computation, introduced in the seminal works of Yao [Yao82, Yao86] and Goldreich, Micali and Wigderson [GMW87], is one of the cornerstones in cryptography. In this setting, a group of (mutually) distrustful participants $P_1, \ldots, P_n$ who hold individual private inputs $x_1, \ldots, x_n$ respectively, wish to jointly compute a functionality f over their inputs by running a protocol Π. The security requirement from the protocol Π is that no subset of malicious parties can learn anything about the inputs of the honest parties other than what can be inferred from the function output. A motivating example for such a setting is the following problem (known as Yao's millionaires problem [Yao82]): two millionaires Alice and Bob wish to run a secure protocol to decide who is richer, without revealing how much money each of them has. General feasibility results for secure computation are known due to Yao [Yao86] and Goldreich et al. [GMW87] for any polynomial-time computable functionality based on standard cryptographic assumptions. In other words, it is possible to not only solve the Yao's millionaires problem, but in fact, any such problem where privacy of inputs is required during the computation of some functionality.

Subsequent to the works of Yao and Goldreich et al., secure computation has become a fundamental question in cryptography with much research activity over the last two decades. For example, a long line of research has focused on the design of efficient secure computation protocols. Another line of research (as in

this dissertation) concerns with the design of secure computation protocols that satisfy much stronger notions of security.

In this thesis, we consider two different frontiers in the area of secure computation. The first part of this thesis concerns with a very strong model of security where an adversary, who corrupts a subset of parties in the protocol, can also *leak* (via physical attacks) information from the secret state of an honest party, throughout the protocol execution. We initiate a study of designing leakage-resilient protocols, i.e., secure computation protocols that provide some meaningful security even against such class of adversaries. As our main results, we formalize a meaningful security definition for leakage-resilient zero knowledge proof systems and provide constructions for the same based on standard cryptographic assumptions. We also discuss several interesting applications of our results.

In the second part of the thesis, we consider the notion of concurrently secure computation that concerns with the setting where several secure computation protocol sessions may be executed concurrently under the control of an adversary that is present across all sessions. A typical example of such a setting is protocols executed over modern networked environments such as the Internet. In particular, we consider the open problem of constructing concurrently-secure password-based key exchange protocols and provide the first positive result for the same based on standard cryptographic assumptions without assuming any trusted infrastructure or random oracles. Our results are in fact much broader, and in fact, extend to any functionality w.r.t. a (necessarily) weakened notion of concurrent security in the standard model.

1.1 Leakage-Resilient Protocols

Zero knowledge proof systems, introduced in the seminal work of Goldwasser, Micali and Rackoff [GMR85], have proven fundamental to cryptography. Very briefly, a zero knowledge proof system is an interactive proof between two parties – a prover, and a verifier – with the remarkable property that the verifier does not learn anything beyond the validity of the statement being proved. Subsequent to their introduction, zero knowledge proofs have been studied in various adversarial settings such as concurrency attacks [DNS98], malleability attacks [DDN00], reset attacks [CGGM00], to list a few, with very successful results. Over the years, zero knowledge proofs (and its various strengthened notions) have turned to be extremely useful, finding numerous applications in the design of various cryptographic protocols.

We note that the standard definition of zero knowledge proofs, like most classical security notions, assumes that an adversary is given only black-box access to the honest party algorithms. Unfortunately, over the last two decades, it has become increasingly evident that such an assumption may be unrealistic when arguing security in the real world where the physical implementation (e.g. on a smart card or a hardware token) of an algorithm is under attack. Motivated by such a scenario, in this thesis, we initiate a study of zero knowledge proof systems in the presence of *side-channel attacks* [Koc96, AK96, QS01, GMO01, OST06, HSH+08]. Specifically, we study zero knowledge proofs in the intriguing setting where a cheating verifier, in addition to receiving a proof of some statement, is able to obtain arbitrary bounded leakage on the *entire state* (*including the witness and the random coins*) of the prover *during the entire protocol execution*. We note that while there has been an extensive amount of research work on leakage-resilient cryptography in the past few years, to the best of our knowledge, almost all prior

work mainly deals with leakage resilient *primitives* such as encryption and signature schemes [DP08, AGV09, Pie09, DKL09, ADW09a, NS09, KV09, DGK+10, FKPR10, ADN+10, KP10, BKKV10, DHLW10a, DHLW10b, LRW11, MTVY11, BSW11, LLW11], or involve general compilers that handle limited class of attacks [ISW03, IPSW06, GR10, JV10, FRR+10, Ajt11], while very limited effort has been dedicated towards constructing leakage-resilient *interactive protocols*. To the best of our knowledge, the recent works on correlation extractors [IKOS09], and leakage-resilient identification and authenticated key agreement protocols [ADW09a, DHLW10b, DHLW10a] come closest to being considered in the latter category. However, we stress that in all these works, either leakage attacks are allowed only *prior* to the protocol execution, or very limited leakage is allowed *during* the protocol execution; in contrast, we consider the setting where the adversary can obtain leakage on the *entire state* of the honest party during the protocol execution.

We find it imperative to stress that handling leakage attacks on interactive protocols can be particularly challenging. On the one hand, for the leakage attacks to be meaningful, we would want to allow leakage on the secret state of the protocol participants. However, the state of a party typically includes a secret value (witness and random coins of the prover in the case of zero knowledge proofs) and any leakage on that secret value might immediately violate a security property (e.g., the zero knowledge property) of the protocol. Then, coming back to setting of zero knowledge proofs, it is not immediately clear how to even define "leakage-resilient zero knowledge."

How to define Leakage-Resilient Zero Knowledge? One possibility is to pursue an assumption such as *only computation leaks information* [MR04] (i.e., assuming that there is no leakage in the absence of computation). While this

is a valuable and interesting approach in general, we note that this assumption is often problematic (e.g. cold-boot attacks [HSH+08]). Furthermore, it is not difficult to see that such an approach (on its own) cannot yield the standard security of zero-knowledge.[1] In our work here, therefore, we do *not* make any such assumption. We seek a general definition maximizing the potential applicability of that definition to different application scenarios.

Another possibility is to allow a "leakage-free pre-processing phase" prior to the actual protocol execution, in an attempt to render the leakage attacks during the protocol useless. We note, however, that allowing pre-processing would limit the applicability of our notion. In particular, such a definition would be problematic for scenarios where the statement to be proven is generated "online" (thereby eliminating the possibility of pre-processing the witness "safely"). Furthermore, we give strong evidence that such an approach is unlikely to yield better guarantees than what we are able to achieve (see Section 3.5.2 for further discussion on this issue).

Indeed, our goal is to obtain a meaningful and appropriate definition of zero knowledge in the model where an adversarial verifier can obtain leakage on any content (state) of the prover machine at any time. We do not consider any "leakage-free" time-period; in particular, any pre-processing phase is subject to leakage as well. However, in such a setting, it is important to note that since the adversary could simply choose to leak on the *witness* (and no other prover state), the zero knowledge *simulator* must be able to obtain similar amount of leakage in order to perform correct simulation. We shall see that even with this limitation, our notion turns out to be both quite nontrivial to obtain and very useful in application scenarios.

[1] To see this, consider the case of witness-indistinguishability where the adversary simply leaks one of the bit where the witnesses differ.

Our Definition – Informally. To this end, we consider a definition of leakage-resilient zero knowledge that provides the intuitive guarantee that *the protocol does not yield anything beyond the validity of the statement and the leakage obtained by the adversary.* In other words, whatever an adversary "learns" from the protocol (with leakage) should be no more than what she can learn from only the leakage without running the protocol. To formalize the above intuition, as a first step, we consider a *leakage oracle* that gets as private input the witness of the honest prover; the zero knowledge simulator is then given access to such a leakage oracle. More concretely, we consider a parameter λ, and say that an interactive proof system is λ-*leakage-resilient zero knowledge* (LR-ZK) if for every cheating verifier, there exists a simulator with access to a leakage oracle (that gets the honest prover's witness as private input) that outputs a view of the verifier (indistinguishable from the real execution), with the following requirement. Let ℓ bits be an upper bound on the total amount of leakage obtained by the adversarial verifier. Then the simulator is allowed to obtain at most $\lambda \cdot \ell$ bits of leakage. (In Section 3.5.1, we show that constructing an LR-ZK proof system with $\lambda < 1$ is in fact impossible.)

Applications of Our Definition. Now that we have a definition for LR-ZK proof system, one may question how meaningful it is. As we now discuss, the above definition indeed turns out to be very useful. Intuitively, our definition is appropriate for a scenario where a leakage-resilient primitive A is being used in conjunction with a zero knowledge proof system (where the proof system is used to prove some statement about A), in the design of another cryptographic protocol B. The reason for this is that our definition of LR-ZK allows us to directly reduce the leakage-resilience property of B on the leakage-resilience property of A.

As an application of our LR-ZK interactive proof system, we first construct a universally composable (UC) multiparty computation protocol in the *leaky token model* (which is a relaxation of the model of Katz [Kat07] in that a malicious token user is now allowed to leak arbitrary bounded information on the *entire state* of the token). Very briefly, we use *leakage-resilient hard relations* [DHLW10b] and hardware tokens that implement the prover algorithm of our LR-ZK proof system where we prove the validity of an instance of the hard relation; then the leakage on the state of the token can be easily "reduced" to leakage on (the witness corresponding to) an instance of the hard relation.

Next, we are able to extend the notion of LR-ZK to the non-interactive setting in a natural way. Then, as an application of LR-NIZKs, we give generic constructions of *fully* leakage-resilient (FLR) signature schemes (where leakage is allowed on the *entire state* as opposed to only the secret key). Very briefly, we use leakage-resilient hard relations in conjunction with "simulation-extractable" LR-NIZKs (see below); we are then able to reduce the leakage-resilience property of the signature scheme to that of the hard relation. We now summarize our results.

1.1.1 Our Results

We first study the possibility of constructing leakage-resilient zero knowledge protocols and obtain the following results:

- We construct a $(1 + \epsilon)$-leakage-resilient zero knowledge interactive proof system (where ϵ is any positive constant) based on standard general assumptions (specifically, the existence of a statistically hiding commitment scheme that is public-coin w.r.t. the receiver). To the best of our knowledge, this is the first instance of a cryptographic *interactive protocol* where

an adversary is allowed to obtain arbitrary bounded leakage on the *entire state* of the honest parties *during* the protocol execution.

- Next, we consider the non-interactive setting and show that any NIZK proof system with *honest prover state reconstruction* property [GOS06] is an LR-NIZK proof system for $\lambda = 1$. As a corollary, we obtain an LR-NIZK proof system from [GOS06] based on the decisional linear assumption.

We supplement our above positive results by proving the impossibility of constructing an LR-ZK proof (or argument) system for $\lambda < 1$. Then, as applications of leakage-resilient zero knowledge, we obtain the following results:

- We initiate a new line of research to relax the assumption on the "tamper-proofness" of hardware tokens used in the design of various cryptographic protocols. In particular, assuming semi-honest oblivious transfer, we give a construction of a universally composable (UC) multiparty computation protocol in the *leaky token model*, where the token user is allowed to obtain arbitrary bounded leakage on the *entire state* of the token. We stress that all prior works on designing cryptographic protocols using hardware tokens, including the work on UC secure computation [Kat07, CGS08, MS08, DNW09], made the implicit assumption that the tokens are completely leakage-resilient.

- Next, we extend the notion of leakage-resilient NIZKs to incorporate the property of *simulation-extractability* [Sah99, DDO$^+$01] (also see [PR05] in the context of interactive proofs), in particular, the "true" variant [DHLW10b]. We are then able to adapt the approach of Katz and Vaikuntanathan [KV09], and in particular, Dodis et al [DHLW10b, DHLW10a] (who use

a leakage-resilient hard relation in conjunction with a true simulation-extractable NIZK argument system to construct leakage-resilient signatures) to the setting of *full leakage*. As a result, we obtain simple, generic constructions of *fully* leakage-resilient signature schemes in the bounded leakage model as well as the continual leakage model. Similar to [DHLW10b, DHLW10a], our signature scheme inherits the leakage-resilience properties (and the leakage bounds) of the hard relation used in its construction.[2] In contrast to the recent constructions of FLR signature schemes by [MTVY11, BSW11, LLW11] in the standard model[3], our scheme is also secure in the *noisy leakage model* [NS09]. We supplement our result by showing that a true simulation-extractable leakage-resilient NIZK argument system is implied by the UC-NIZK of Groth et al. [GOS06], which can be based on the decisional linear assumption.

We study two more questions which are very closely related to the setting of leakage-resilient zero knowledge. First, we consider the scenario in which a malicious prover can obtain arbitrary leakage on the random coins of the verifier during the protocol execution. The question that we investigate is whether it is possible to construct interactive proofs that remain sound even in such a scenario. We refer to such proofs as *leakage-sound proofs*. Secondly, we consider the question of constructing an interactive proof system that *simultaneously* satisfies the two notions of leakage-soundness (c.f. Definition 12) and leakage-resilient zero knowledge (c.f. Definition 8). We call such an interactive proof system *simulta-*

[2]Specifically, our signature scheme is fully leakage-resilient (FLR) in the bounded (resp., continual) leakage model if the hard relation is leakage-resilient in the bounded (resp., continual) leakage model. As such, if we use the key pairs from the encryption scheme of Lewko et al [LLW11] as a hard relation, then our signature scheme can tolerate leakage during the update process as well.

[3]Earlier, FLR signature schemes were constructed either only in the random oracle model [ADW09a, DHLW10b, BKKV10], or were only "one-time" [KV09]

neous leakage-resilient zero knowledge. We obtain positive results for both these settings. We refer the reader to Section 3.4 for details.

1.1.2 Our Techniques

We now briefly discuss the main techniques used to obtain our positive results on leakage-resilient zero knowledge proof systems. Recall that our goal is to realize a definition where a cheating verifier does not learn anything from the protocol beyond the validity of the statement and the leakage information obtained from the prover. Further, recall that in our definition, simulator is given access to a leakage oracle that gets the honest prover's witness as private input and accepts leakage queries on the witness string. (In contrast, the verifier is allowed to make leakage queries on the entire state, including the witness and the random coins used by the prover thus far in the protocol execution.) Then, during the simulation, on receiving a leakage query from the verifier, our simulator attempts to convert it into a "valid" query to the leakage oracle. Now, note that the simulator may be cheating in the protocol execution (which is typically the case since it does not possess a valid witness); then, since the verifier can arbitrarily leak on both the witness and the random coins (which completely determine the actions of the prover thus far), at every point in the protocol execution, the simulator must find a way to "explain its actions so far". Note that this is reminiscent of *adaptive security* [Bea96, CFGN96, CLOS02, LZ09] in the context of secure computation protocols. We stress, however, that adaptive security does not suffice to achieve the property of leakage-resilient zero knowledge in the interactive proofs setting, as we explain below.

Recall that the notion of adaptive security corresponds to the setting where an adversary is allowed to corrupt parties *during* the protocol execution (as opposed

to static corruption, where the parties can only be corrupted *before* the protocol begins). Once a party is corrupted, the adversary learns the entire state (including the input and random coins) of that party. The adversary may choose to corrupt several parties (in the case of multi-party protocols) throughout the course of the protocol. The notion of adaptive security guarantees security for the remaining uncorrupted parties.

While adaptive corruption itself is not our focus, note that in our model, a cheating verifier may obtain leakage on the prover's state at *several points* during the protocol execution. Furthermore, the honest prover may not even be aware as to what was leaked. Our goal is to guarantee that the adversary does not learn anything beyond the leaked information. Then, in order to provide such a guarantee, note that our simulator must *continue to simulate the prover even after leakage happens*, in a way that is consistent with the leaked information even though it does not know the prover's witness or what information was leaked. In contrast, the simulator for adaptively secure protocols does *not* need to simulate a party once it is corrupted.[4] In summary, we wish to guarantee some security for the honest party even *after* leakage happens, while adaptive security does not provide any such guarantees. We stress that this difference is crucial, and explains why known techniques for achieving adaptive security do not suffice for our purposes. Nevertheless, as we explain below, adaptive security serves as a good starting point for our purpose.

Recall that the main challenge in the setting of adaptive security is that whenever an adversary chooses to corrupt a party, the simulator must be able to explain its random coins, in a way that is consistent with the party's input and the messages it generated so far in the protocol. The main technique for over-

[4]Indeed, for this reason, known adaptively secure ZK protocols are not leakage-resilient.

coming this challenge is to allow the simulator to *equivocate*. For our purposes, we will also make use of equivocation so that the leakage queries can be answered correctly by the simulator. However, since our simulator would need to simulate the prover even after leakage happens (without the knowledge of the prover's witness or the information that was leaked), we do not want this equivocation to interfere with the simulation of prover's messages. In other words we want to be able to simulate the prover's messages independent of what information is being leaked but still remain consistent with it. Our solution is to have two separate and independent ways of cheating at the simulator's disposal. It will use one way to cheat in the protocol messages and the second way is reserved for answering the leakage queries correctly. Furthermore, we would need to make sure that the simulator does not "step on its own toes" when using the two ways to cheat *simultaneously*.

We now briefly discuss the actual construction of our protocol in order to illustrate the above ideas. We recall two well-known ways of constructing constant-round zero knowledge protocols – the Feige-Shamir [FS89] approach of using equivocal commitments (also used in adaptive security), and the Goldreich-Kahan [GK96] approach of requiring the verifier to commit to its challenges in advance. Now, armed with the intuition that our simulator will need two separate ways of cheating, we use both the above techniques *together*. Our protocol roughly consists of two phases: in the first phase, the verifier commits to a challenge string using a standard challenge-response based extractable commitment scheme (in a manner similar to [Ros04]); in the second phase, we execute the Blum-Hamiltonicity protocol instantiated with an equivocal commitment scheme. While leakage during the first phase can be handled easily by our simulator, handling leakage during the second phase makes use of the ideas discussed above.

Unfortunately, although the above construction seems to satisfy most of our requirements, it fails on the following account. Recall that our goal is to obtain a leakage-resilient zero knowledge protocol with nearly optimal precision (i.e., $\lambda = 1 + \epsilon$) with respect to the leakage queries of the simulator. Now note that in the above construction, the simulator would need to extract the verifier's challenge in the first phase by means of rewinding before proceeding to phase two of the protocol. Then, depending upon the verifier's behavior, the simulator may need to perform several rewinds in order to succeed in extraction. Now, note that a cheating verifier may be able to make a *different* leakage query during each rewind, thus forcing our simulator to make a new query as well to its leakage oracle. As a result, depending upon the number of such rewinds, the total leakage obtained by the simulator may potentially become a polynomial factor of the leakage obtained by the adversary in a real execution.

In order to obtain a precision in the leakage queries of the simulator, we borrow techniques from the work on *precise zero knowledge* pioneered by Micali and Pass [MP06]. We remark that in the context of precise ZK, (for fundamental reasons of modeling) it is typically not possible to obtain a precision of almost 1. In our case, however, we are able to achieve a precision of $\lambda = 1 + \epsilon$ (where ϵ is any positive constant) with respect to the leakage queries of the simulator.

Finally, we note that in the case of non-interactive zero knowledge, since the simulator does not need to simulate any "future messages" after the leakage, we are indeed able to show that an adaptively secure NIZK is also a leakage-resilient NIZK. Specifically, we show that any NIZK with *honest prover state reconstruction* property, as defined by Groth et al. [GOS06] (in the context of adaptive security), is also a leakage-resilient NIZK with $\lambda = 1$.

1.2 Concurrently-Secure Protocols

The problem of password authenticated key exchange (PAKE) involves a pair of parties who wish to establish a high entropy session key in an authenticated manner when their *a priori* shared secret information only consists of a (possibly low entropy) password. More formally, the problem of PAKE can be modeled as a two-party functionality $\mathcal{F}$ involving a pair of parties P_1 and P_2; if the inputs (passwords) of the parties match, then $\mathcal{F}$ outputs a uniformly distributed session key, else it outputs $\perp$. Hence the goal of PAKE is to design a protocol that securely realizes the functionality $\mathcal{F}$. Unfortunately, positive results for secure multi-party computation (MPC) [Yao86, GMW87] do not immediately translate to this setting; the reason being that known solutions for secure MPC require the existence of authenticated channels[5] – which is in fact the end goal of PAKE. Therefore, very informally speaking, secure multi-party computation and PAKE can be viewed as complementary problems.

The problem of password authenticated key exchange was first studied by Bellovin and Meritt [BM92]. This was followed by several additional works proposing protocols with only heuristic security arguments (see [Boy00] for a survey). Subsequently, starting with the work of Bellare et al [BPR00], PAKE was formally studied in various models, including the random oracle/ideal cipher model, common reference string (CRS) model, and the plain model (which is the focus of this work). We briefly survey the state of the art on this problem.

The works of Bellare et al [BPR00] and Boyko et al [BMP00] deal with defining and constructing PAKE protocols in the ideal cipher model and random oracle

[5]A recent work of Barak et al [BCL+05] is an exception to this, in that it does not require authenticated channels for secure computation. More specifically, Barak et al [BCL+05] show that 2-bounded concurrent two party computation can be translated to a stand-alone secure password authenticated key exchange. More details are given later in the section.

model respectively. In the CRS model, Katz, Ostrovsky and Yung [KOY01] gave the first efficient construction for PAKE without random oracles based on the DDH assumption. Their result were subsequently improved by Gennaro and Lindell [GL03] and Genarro [Gen08]. Again in the CRS model, Canetti et al [CHK+05] proposed new definitions and constructions for a PAKE protocol in the framework of Universal Composability [Can01]. They further proved the *impossibility* of such a construction in the plain model.

Goldreich and Lindell [GL01] formulated a new simulation-based definition for PAKE and gave the first construction for a PAKE protocol in the *plain model*. Their construction was further simplified (albeit at the cost of a weaker security guarantee) by Nguyen and Vadhan [NV04]. Recently, Barak et al [BCL+05] gave a very general construction for a PAKE protocol that is secure in the bounded-concurrent setting (see below) in the plain model.

To date, [GL01, NV04] and [BCL+05] remain the only known solutions for PAKE in the plain model. However, an important limitation of [GL01] (as well as [NV04]) is that their solution is only relevant to the stand-alone setting where security holds only if a single protocol session is executed on the network. A more natural and demanding setting is where several protocol sessions may be executed concurrently (a typical example being protocols executed over the Internet). In such a setting, an adversary who controls parties across different sessions may be able to mount a coordinated attack; as such, stand-alone security does not immediately translate to concurrent security [FS90]. In the context of PAKE, this problem was partially addressed by Barak et al [BCL+05] who gave a construction that maintains security in the setting of *bounded-concurrency*. In this setting, an *a priori* bound is known over the number of sessions that may be executed concurrently at any time; this bound is crucially used in the design of the

protocol. It is natural to consider the more general setting of full concurrent self-composition, where any polynomially many protocol sessions (with no a priori bound) with the same password may be executed in an arbitrary interleaved manner by an adversary who may corrupt any number of parties. Although the works of KOY01,GL03,CSKLM05 solve this problem (where [KOY01, GL03] are secure under self-composition, while [CHK$^+$05] is secure under general-composition), they require a trusted setup in the form of a common reference string. Indeed, to date, no constructions are known for a PAKE protocol that is secure in the plain model in the setting of concurrent self-composition.

1.2.1 Our Contribution

In this thesis, we resolve this open problem. We give the first construction of a PAKE protocol in the plain model that allows for concurrent executions of the protocol between parties with the same password. Our techniques rely on several previous works, most notably the works of Barak, Prabhakaran and Sahai [BPS06] and Pandey et al. [PPS$^+$08] (which in turn relies on the work of Micali and Pass [MP06]).

Our construction is proven secure as per the definition of Goldreich and Lindell [GL01] in the concurrent setting. We stress that Lindell's impossibility result [Lin04] for concurrent self-composition is *not* applicable here since (a) Goldreich and Lindell used a *specific* definition that is different from the standard paradigm for defining secure computation[6], and (b) further, they only consider the scenario where the honest parties hold fixed inputs (while Lindell's impossi-

[6]Note that in the standard simulation paradigm, the output distributions of the "real" and "ideal" worlds must be computationally indistinguishable; in contrast, the definition of Goldreich and Lindell [GL01] allows these distributions to be $\mathcal{O}(1/|D|)$ apart (where D is the password dictionary).

bility result crucially requires adaptive inputs).

In fact, our security definition is stronger and arguably more cleaner than the one by Goldreich and Lindell [GL01]. The definition in [GL01], for example, does not consider the case where the adversary may have some a priori information on the password of the honest parties in a protocol execution. We consider an improved simulation-based security model similar to that proposed by [BMP00]. More specifically, in our model, the simulator in the ideal world is empowered to make a *constant* number of queries per (real world) session to the ideal functionality (as opposed to just one). Our security definition then requires computational indistinguishability of the output distributions of real and ideal world executions in keeping with the standard paradigm for secure computation. As noted in [GL06], this improved definition implies the original definition of Goldreich and Lindell (see appendix 4.3 for a proof sketch).

In our above main construction, we only consider the setting when a number of concurrent executions are run where the honest parties hold the same password (or independently chosen passwords)[7]. However, a natural question is to consider the setting where the passwords of honest parties in different sessions might be *correlated in any arbitrary way*. Towards this end, we note that our construction can be easily extended to this setting. However, we require the ideal simulator to be able to query the ideal functionality an *expected constant* number of times per session.[8] This in turn means that for the setting of correlated passwords, our constructions will satisfy a security definition which is slightly weaker than the original definition of Goldreich and Lindell [GL01]. Obtaining a construction

[7] An example is when a server expects a specific password for authentication and several parties are trying to authenticate simultaneously.

[8] Jumping ahead, in case the honest parties were using the same password, the simulator is able to "trade" ideal functionality calls in one session for another. Hence, the simulator is able to even out the number of calls to a fixed constant in each session.

as per the definition in [GL01] in this setting is left as an open problem in the current work.

Implications for Concurrently Secure Computation in the Plain Model. We in fact note that our techniques and constructions are quite general. Our construction can be instantiated with a basic semi-honest secure computation protocol for any PPT computable functionality. This would lead to a concurrently secure protocol for that functionality as per the security definition where we allow the simulator to make an expected constant number of calls to the ideal function per (real world) session. (In the client-server setting where the server is honest and same input in each of its session, our positive result holds if the simulator can make only require a constant number of ideal functionality calls per session). The meaningfulness of such a definition is shown in the case of password based key exchange which is the focus of this work (more precisely, by comparing it with the definition of [GL06]). However we anticipate that the above general construction with such security guarantees might be acceptable in many other settings as well.

A related model is that of *resettably secure computation* proposed by Goyal and Sahai [GS09]. In resettably secure computation, the ideal simulator is given the power to reset and query the trusted party any (polynomial) number of times. However there are important differences. Goyal and Sahai [GS09] consider only the "fixed role" setting and only one of the parties can be thought of as accepting concurrent sessions. This means that the key technical problems we face in the current work (arising out of the possibility of mauling attacks in the concurrent setting) do not arise in [GS09]. Secondly, [GS09] do not try to optimize (or even bound) the number of queries the ideal simulator makes to the trusted party per session.

1.2.2 Overview of Main Ideas

Note that in the setting of concurrent self-composition, an adversary may corrupt different parties across the various sessions. Consider for instance two different sessions where one of the parties is corrupted in each session. We can view one of these sessions as a "left" session and the other as a "right session", while the corrupted parties can be jointly viewed as an adversarial man-in-the-middle. An immediate side-effect of this setting is that it allows an adversary to possibly "maul" a "left" session in order to successfully establish a session key with an honest party (say) P in a "right" session *without* the knowledge of P's secret password. Clearly, in order to provide any security guarantee in such a setting, it is imperative to achieve independence between various protocol sessions executing on the network. Note that this is akin to guaranteeing non-malleability across various sessions in the concurrent setting. Then, as a first step towards solving this problem, we borrow techniques from the construction of concurrent non-malleable zero knowledge argument due to Barak, Prabhakaran and Sahai [BPS06] (BPS-CNMZK). In fact, at a first glance, it might seem that compiling a semi-honest two-party computation protocol (that emulates the PAKE functionality in the stand-alone setting) with the BPS-CNMZK argument or some similar approach might fully resolve this problem. However, such an approach fails on account of several reasons. We highlight some important problems in such an approach.

We first note that the simulation of BPS-CNMZK is based on a rewinding strategy. In a concurrent setting, the adversary is allowed to control the scheduling of the messages of different sessions. Then for a given adversarial scheduling, it is possible that the simulator of BPS-CNMZK may rewind past the beginning of a session (say) s when "simulating" another session. Now, every time session

s is re-executed, an adversary may be able to change his input (i.e., make a new password guess possibly based on the auxiliary information it has). In such a case, the simulator would have to query the ideal functionality for that session more than once; therefore, we need to allow the simulator to make extra (i.e., more than one) queries per session to ideal functionality. In order to satisfy our definition, we would need to limit the number of queries to a *constant* per session. However, the simulator for BPS-CNMZK, if used naively, may require large polynomially many queries per session to the ideal functionality, and therefore, fail to satisfy our definition.

In order to overcome this problem, we build on the techniques of precise simulation, introduced by Micali and Pass [MP06] in the context of (stand-alone) zero knowledge and later extended to the setting of concurrent zero knowledge by Pandey et al [PPS+08]. Specifically, Pandey et al [PPS+08] use a time-oblivious rewinding schedule that (with a careful choice of system parameters) ensures that the the time spent by the simulator in the "look-ahead" threads[9] is only within a *constant* factor of the time spent by the simulator in the "main" thread. We remark that we do not require this precision in simulation time; instead we require that the number of queries made by the simulator in the look-ahead threads is only within a constant factor of the number of queries made in the main thread. For this purpose, we consider an imaginary experiment in which our adversary takes a disproportionately large amount of time in generating the message after which the simulator has to query the trusted party. Our rewinding strategy is determined by running the PPSTV [PPS+08] simulator using the next message generation timings of such an (imaginary) adversary (even though our simulator

[9]Very roughly speaking, a "thread of execution" between the simulator and the adversary is a simulation of a prefix of an actual execution. The simulator may run multiple threads of execution, and finally output a single thread, called the *main thread*. Any other thread is referred to as a *look-ahead thread*. See appendix 4.4.4 for more details.

is fully black-box and does not even measure the timings for the real adversary). (Please see the simulator description for more details).

We further note that in the security proof of the above approach, the simulator must be able to extract the inputs of the adversary in *all* the sessions in order to simulate its view. However, the extractor of [BPS06] is unsuitable for this task since it can extract adversary's inputs (in the setting of BPS-CNMZK) only on a *session-by-session* basis. To further elaborate, let us first recall the setting of BPS-CNMZK, where an adversary is interacting with some honest provers as well as some honest verifiers. Now, in order to extract the input of an adversarial prover in a particular session s, the extractor in [BPS06] honestly runs all the uncorrupted verifiers except the verifier in session s. We stress that the extractor is able to run the honest verifiers by itself since they do not possess any secret inputs; clearly, such an extraction technique would fail in our setting since the simulator does not know the inputs of the honest parties.

To address this problem, we require each party in our protocol to commit to its input and randomness inside a separate preamble [PPS+08, PRS02] that allows extraction of the committed values in a concurrent setting. However, we note that such a preamble requires a complicated rewinding strategy for extraction of committed value, and so is the case for simulating the BPS-CNMZK argument. Indeed, it seems that we might need to *compose* the (possibly conflicting) individual rewinding strategies of BPS-CNMZK and the additional preamble into a new uniform rewinding strategy. Fortunately, by ensuring that we use the same kind of preamble (for committing to the input of a party) as the one used inside BPS-CNMZK, we are able to avoid such a scenario, and crucially, we are able to use the BPS-CNMZK strategy as a single coherent rewinding strategy. The above idea also gives us a *new construction of a concurrent non-malleable zero-*

knowledge protocol where the extraction can be *automatically done in-line* along with the simulation. We believe this implication to be of independent interest.

Finally, the construction in [BPS06] is only analyzed for the setting where the theorems to be proven by the honest parties are fixed in advance before any session starts (in keeping with the impossibility results of Lindell [Lin04]). Towards that end, our protocol only makes use of BPS-CNMZK in the very beginning of the protocol to prove a statement which could be generated by the honest parties before the start of any session.

CHAPTER 2

Preliminaries for Leakage-Resilient Protocols

2.1 Basic Definitions: Interactive Case

We first recall the standard definitions of interactive proofs and zero knowledge [GMR85]. For convenience, we will follow the notation and presentation of [PR05]. Let P (called the *prover*) and V (called the *verifier*) denote a pair of interactive Turing machines that are running a protocol with each other on common input x. Throughout our text, we will always assume V to be a polynomial-time machine. Let $\langle P, V\rangle(x)$ be the random variable representing the output of V at the end of the protocol. If the machine P is polynomial-time, it is assumed that it has a private input w.

Definition 1 (Interactive proof system) *A pair of interactive Turing machines $\langle P, V\rangle$ is called an* interactive proof system *for a language $\mathcal{L}$ if the following two conditions hold:*

- Completeness: *For every $x \in \mathcal{L}$,*

$$\Pr[\langle P, V\rangle(x) = 1] \geq 1 - \mathrm{negl}(|x|)$$

- Soundness: *For every $x \notin \mathcal{L}$, and every interactive Turing machine P^*,*

$$\Pr[\langle P^*, V\rangle(x) = 1] \leq \mathrm{negl}(|x|)$$

If the soundness condition in the above definition is valid only against PPT Turing machines, then we say that $\langle P, V \rangle$ is an *argument* system.

Zero Knowledge. An interactive proof $\langle P, V \rangle$ is said to be *zero-knowledge* if, informally speaking, the verifier V learns nothing beyond the validity of the statement being proved. This intuition is formalized by requiring that the view of every probabilistic polynomial-time (PPT) cheating verifier V^*, represented by $\text{view}_{V^*}(x, z)$, generated as a result of its interaction with P can be "simulated" by a PPT machine $\mathcal{S}$ (referred to as the *simulator*). Here, the verifier's view consists of the common input x, its random tape, and the sequence of prover messages that it receives during the protocol execution. The auxiliary input of V^* and $\mathcal{S}$ is denoted by $z \in \{0,1\}^*$.

Definition 2 (Zero knowledge) *An interactive proof system $\langle P, V \rangle$ for a language $\mathcal{L}$ is said to be* zero knowledge *if for every* PPT *verifier V^*, there exists a* PPT *algorithm $\mathcal{S}$ such that for every $x \in \mathcal{L}$, every $z \in \{0,1\}^*$, $\text{view}_{V^*}(x, z)$ and $\mathcal{S}(x, z)$ are computationally indistinguishable.*

One can consider stronger variants of zero knowledge where the output of $\mathcal{S}$ is statistically close (or identical) to the verifier's view. In this thesis, unless otherwise specified, we will focus on the computational variant only.

2.2 Basic Definitions: Non-Interactive Case

Here we recall the standard definition of non-interactive zero knowledge (NIZK) proof systems. For convenience, we will follow the notation and presentation of [GOS06].

Let $\mathcal{R}$ be an efficiently computable relation that consists of pairs (x, w), where x is called the statement and w is the witness. Let $\mathcal{L}$ denote the language consisting of statements in $\mathcal{R}$. A non-interactive proof system for a language $\mathcal{L}$ consists of a setup algorithm K, a prover P and a verifier V. The setup algorithm K generates a common reference string σ. The prover P takes as input (σ, x, w) and checks whether $(x, w) \in \mathcal{R}$; if so, it produces a proof string π, else it outputs `fail`. The verifier V takes as input (σ, x, π) and outputs 1 if the proof is valid, and 0 otherwise.

Definition 3 (Non-interactive proof system) *A tuple of algorithms (K, P, V) is called a* non-interactive proof system *for a language $\mathcal{L}$ with a* PPT *relation $\mathcal{R}$ if the following two conditions hold:*

- *Completeness: For all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k); (x, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow P(\sigma, x, w) : V(\sigma, x, \pi) = 1 \text{ if } (x, w) \in \mathcal{R}] \geq 1 - \text{negl}(k)$$

- *Soundness: For all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k); (x, \pi) \leftarrow \mathcal{A}(\sigma) : V(\sigma, x, \pi) = 1 \text{ if } x \notin \mathcal{L}] \leq \text{negl}(k)$$

If the soundness condition holds only against PPT adversaries, then we say that (K, P, V) is a non-interactive *argument* system.

Definition 4 (Zero Knowledge) *A non-interactive proof system (K, P, V) for a relation $\mathcal{R}$ is said to be* zero knowledge *if there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k) : \mathcal{A}^{P(\sigma,\cdot,\cdot)}(\sigma) = 1] \stackrel{c}{\equiv} \Pr[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^k) : \mathcal{A}^{\mathcal{S}'(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1],$$

where $\mathcal{S}'(\sigma, \tau, x, w) = \mathcal{S}_2(\sigma, \tau, x)$ if $(x, w) \in \mathcal{R}$ and outputs `fail` *otherwise.*

We now state an extension of the zero knowledge property, called *honest prover state reconstruction*, that is central to our positive result on leakage-resilient NIZK. We recall the notion as defined by Groth, Ostrovsky and Sahai [GOS06].

Definition 5 (Honest Prover state reconstruction.) *A non-interactive proof system (K, P, V) for a relation $\mathcal{R}$ is said to support* honest prover state reconstruction *if there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that for all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k) : \mathcal{A}^{PR(\sigma,\cdot,\cdot)}(\sigma) = 1] \stackrel{c}{\equiv} \Pr[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^k) : \mathcal{A}^{SR(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1],$$

where $PR(\sigma, x, w)$ computes $r \leftarrow \{0,1\}^{\ell_P(k)}$; $\pi \leftarrow P(\sigma, x, w; r)$ and returns $(\pi, w, r,)$ and $SR(\sigma, \tau, x, w)$ computes $\rho \leftarrow \{0,1\}^{\ell_S(k)}$; $\pi \leftarrow \mathcal{S}_2(\sigma, \tau, x; \rho)$; $r \leftarrow \mathcal{S}_3(\sigma, \tau, x, w, \rho)$ and returns (π, w, r); both of the oracles outputting `fail` *if $(x, w) \notin \mathcal{R}$.*

2.3 Building Blocks

In this section, we briefly recall some basic cryptographic primitives that we will use in our various constructions in Chapter 3.

2.3.1 Naor's Statistically Binding Commitment Scheme [Nao89]

We briefly recall Naor's statistically binding bit commitment scheme based on one way functions. The commitment phase consists of two rounds: first, the verifier sends a $3k$ bit random string r, where k is the security parameter. The committer chooses a seed s for a pseudo-random generator $g : \{0,1\}^k \to \{0,1\}^{3k}$; if it wishes to commit to 0, then it sends $g(s)$, else it sends $g(s) \oplus r$. The decommitment phase simply involves the committer sending s to the receiver.

2.3.2 Public-coin Statistically Hiding String Commitment

We will also use a statistically hiding commitment scheme that is public-coin with respect to the receiver. Such schemes can be constructed in constant rounds using collision-resistant hash functions [NY89, HM96, DPP97].

2.3.3 Leakage-Resilient Hard Relations

Here we recall the notion of leakage-resilient hard relations as defined by Dodis, Haralambiev, Lopez-Alt, Wichs [DHLW10b].

To model leakage attacks, the adversary is given access to a *leakage oracle*, which she can adaptively access to learn leakage on the secret value. A leakage oracle $L_x^{k,\ell}(\cdot)$ is parametrized by a secret value x, a leakage parameter ℓ, and a security parameter k. A query to the leakage oracle consists of a function $f_i : \{0,1\}^* \to \{0,1\}^{\ell_i}$, to which the oracle answers with $f_i(x)$. We only require that the functions f_i be efficiently computable, and the total number of bits leaked is $\sum_i \ell_i \leq \ell$.

Definition 6 (Leakage-resilient hard relation.) *A relation $\mathcal{R}$ with a* PPT *sampling algorithm* $\text{KGEN}(\cdot)$ *is an ℓ-leakage resilient hard relation if:*

- *For any $(x,y) \leftarrow \text{KGEN}(1^k)$, we have $(x,y) \in \mathcal{R}$.*
- *There is a poly-time algorithm that decides if $(x,y) \in \mathcal{R}$.*
- *For all* PPT *adversaries $\mathcal{A}^{L_x^{k,\ell}(\cdot)}$ with access to the leakage oracle $L_x^{k,\ell}(\cdot)$, we have that*

$$\Pr\left[\mathcal{R}(x^*, y) = 1 \mid (x,y) \leftarrow \text{KGEN}(1^k); x^* \leftarrow \mathcal{A}^{L_x^{k,\ell}(\cdot)}(y)\right] \leq \text{negl}(k)$$

Notice that without loss of generality, we can assume that $\mathcal{A}$ queries $L_x^{k,\ell}(\cdot)$ only once with a function f whose output is ℓ bits.

We also recall the notion of second-preimage resistant (SPR) relation, as defined in [DHLW10b].

Definition 7 (Second-preimage resistant relation.) *A relation $\mathcal{R}$ with a randomized* PPT *sampling algorithm* $\text{KGEN}(\cdot)$ *is second-preimage resistant if:*

- *For any* $(x, y) \leftarrow \text{KGEN}(1^k)$*, we have that* $(x, y) \in \mathcal{R}$*.*
- *There is a poly-time algorithm that decides if* $(x, y) \in \mathcal{R}$
- *For any* PPT *algorithm* $\mathcal{A}$*, we have that*

$$\Pr\left[\mathcal{R}(x^*, y) = 1 \wedge x^* \neq x \mid (x, y) \leftarrow \text{KGEN}(1^k); x^* \leftarrow \mathcal{A}(y)\right] \leq \text{negl}(k)$$

The average-case pre-image entropy of the SPR relation is defined as $\mathbf{H}_{avg}(\mathcal{R}) = \widetilde{\mathbf{H}}_\infty(X \mid Y)$ *, where the random variables* (X, Y) *are distributed according to* $\text{GEN}(1^k)$*, and* $\widetilde{\mathbf{H}}_\infty(X \mid Y)$ *is the average-conditional min-entropy of* X *conditioned on* Y*.*

Leakage-resilient hard relations from SPR relations. Dodis et al show that any SPR relation $\mathcal{R}$ is an ℓ-leakage-resilient hard relation with $\ell = \mathbf{H}_{avg}(\mathcal{R}) - \omega(\log k)$. Finally, we note that SPR relations are implied by the existence of one-way functions. We refer the reader to [ADW09b, DHLW10b] for more details.

2.3.4 Encryption with pseudorandom ciphertexts

A public-key cryptosystem $(K_{\text{pseudo}}, E, D)$ has pseudorandom ciphertexts of length $\ell_E(k)$ if for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} & \Pr\left[(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{E_{pk}(\cdot)}(pk) = 1\right] \\ \approx \; & \Pr\left[(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{R_{pk}(\cdot)}(pk) = 1\right], \end{aligned} \tag{2.1}$$

where $R_{pk}(m)$ runs $c \leftarrow \{0,1\}^{\ell_E(k)}$ and every time returns a fresh c. We require that the cryptosystem has errorless decryption.

Trapdoor permutations over domain $\{0,1\}^{\ell_E(k)-1}$ imply pseudorandom cryptosystems as we can use the Goldreich-Levin hard-core bit [GL89] of a trapdoor permutation to make a one-time pad. Trapdoor permutations over $\{0,1\}^{\ell_E(k)-1}$ can for instance be constructed from the RSA assumption asusming $\ell_E(k)$ is large enough [CFGN96]. These can also be constructed from other special number theoretic assumptions as described in [GOS06].

2.3.5 Tag-based simulation-sound trapdoor commitment

A tag-based simulation-sound trapdoor commitment scheme is a tuple of four algorithms denoted as $(K_{\text{tag}-\text{com}}, \text{commit}, \text{Tcom}, \text{Topen})$. The key generation algorithm $K_{\text{tag}-\text{com}}$ produces a commitment key ck as well as a trapdoor key TK. There is a commitment algorithm that takes as input the commitment key ck, a message m and any tag $\texttt{tag}$ and outputs a commitment $c = \text{commit}_{ck}(m, \texttt{tag}; r)$. To open a commitment c with tag $\texttt{tag}$ we reveal m and the randomness r. Anybody can now verify $c = \text{commit}_{ck}(m, \texttt{tag}; r)$. As usual, the commitment scheme must be both hiding and binding.

In addition to these two algorithms, there are also a couple of trapdoor algorithms $\text{Tcom}, \text{Topen}$ that allow us to create an equivocal commitment and later open this commitment to any value we prefer. We create an equivocal commitment and an equivocation key as $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(\texttt{tag})$. Later we can open it to any message m as $r \leftarrow \text{Topen}_{\text{EK}}(c, m, \texttt{tag})$, such that $c = \text{commit}_{ck}(m, \texttt{tag}; r)$. We require that equivocal commitments and openings are indistinguishable from

real openings. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} & \Pr\left[(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k) : \mathcal{A}^{\mathcal{R}(\cdot,\cdot)}(ck) = 1\right] \\ \approx \quad & \Pr\left[(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k) : \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(ck) = 1\right], \end{aligned} \tag{2.2}$$

where $\mathcal{R}(m, \texttt{tag})$ returns a randomly selected randomizer and $\mathcal{O}(m, \texttt{tag})$ computes $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(m, \texttt{tag})$; $r \leftarrow \text{Topen}_{\text{EK}}(c, m, \texttt{tag})$ and returns r. Both oracles ignore tags that have already been submitted once.

The tag-based simulation-soundness property means that a commitment using `tag` remains binding even if we have made equivocations for commitments using different tags. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} \Pr\Big[(ck, \text{TK}) \leftarrow K(1^k); (c, \texttt{tag}, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(ck) : \texttt{tag} \notin Q \wedge \\ c = \text{commit}_{ck}(m_0, \texttt{tag}; r_0) = \text{commit}_{ck}(m_1, \texttt{tag}; r_1) \wedge m_0 \neq m_1\Big] \approx 0, \end{aligned} \tag{2.3}$$

where $\mathcal{O}(\text{commit}, \texttt{tag})$ computes $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(\texttt{tag})$, returns c and stores $(c, \texttt{tag}, \text{EK})$, and $\mathcal{O}(\text{open}, c, m, \texttt{tag})$ returns $r \leftarrow \text{Topen}_{ck}(\text{EK}, c, m, \texttt{tag})$ if the tuple $(c, \texttt{tag}, \text{EK})$ has been stored, and where Q is the list of tags for which equivocal commitments have been made by $\mathcal{O}$.

The term tag-based simulation-sound commitment comes from Garay, MacKenzie and Yang [GMY06], while the definition presented here is from MacKenzie and Yang [MY04]. The latter work offers a construction based on one-way functions.

CHAPTER 3

Leakage-Resilient Zero Knowledge

In this chapter, we present our results on leakage-resilient zero knowledge proof systems. First, in Section 3.1 we introduce the notion of leakage-resilient zero knowledge protocols and give a concrete construction of a leakage-resilient zero knowledge proof system in the interactive setting. Next we extend our results to the non-interactive setting in Section 3.2. We discuss two concrete applications for our results in Section3.3. We further discuss the notions of leakage soundness and simulataneous leakage-resilient zero knowledge in Section 3.4. We conclude with some impossibility results on leakage-resilient zero knowledge in Section 3.5

3.1 Leakage-Resilient Zero Knowledge: Interactive Case

In this section, we discuss our results on leakage-resilient zero knowledge in the interactive setting. We start by describing our model and our definition of leakage-resilient zero knowledge.

3.1.1 Our Definition

We consider the scenario where a malicious verifier can obtain arbitrary bounded leakage on the entire state (including the witness and the random coins) of the prover during the protocol execution. We wish to give a meaningful definition

of zero knowledge interactive proofs in such a setting. To this end, we first modify the standard model for zero knowledge interactive proof system in order to incorporate leakage attacks and then proceed to give our definition. We refer the reader to Section 2.1 for the standard definitions of interactive proofs and zero knowledge.

We model the prover P and the verifier V as interactive turing machines that have the ability to flip coins during the protocol execution (such that the random coins used by a party in any round are determined only at the beginning of that round). In order to incorporate leakage attacks, we allow a malicious verifier V^* to make *adaptive* leakage queries on the state of the prover during the protocol execution. A leakage query to the prover consists of an efficiently computable function f_i (described as a circuit), to which the prover responds with $f_i(\texttt{state})$, where `state` is a variable that denotes the "current state" of the prover at any point during the protocol execution. The variable `state` is initialized to the witness of the prover. At the completion of each step of the protocol execution (that corresponds to the prover sending a protocol message to the verifier), the random coins used by the prover in that step are appended to `state`. That is, $\texttt{state} := \texttt{state} \| r_i$, where r_i denote the random coins used by the prover in that step. The verifier may make any arbitrary polynomial number of such leakage queries during the protocol execution. Unlike prior works, we do not require an a-priori bound on the total leakage obtained by the verifier in order to satisfy our definition (described below). Nevertheless, in order for our definition to be meaningful, we note that the total leakage obtained by the verifier must be smaller than the witness size.

We model the zero knowledge simulator $\mathcal{S}$ as a PPT machine that has access to a leakage oracle $L_w^{k,\lambda}(\cdot)$ that is parameterized by the honest prover's witness

w, a leakage parameter λ (see below), and the security parameter k. A query to the oracle consists of an efficiently computable function $f(\cdot)$, to which the oracle answers with $f(w)$. In order to bound the total leakage available to the simulator, we consider a parameter λ and require that if the verifier obtains ℓ bits of total leakage in the real execution, then the total leakage obtained by the simulator (from the leakage oracle) must be bounded by $\lambda \cdot \ell$ bits. Finally, we require that the view output by the simulator be computationally indistinguishable from the verifier's view in the real execution. We formalize this in the definition below.

Definition 8 (Leakage-Resilient Zero Knowledge) *An interactive proof system $\langle P, V \rangle$ for a language $\mathcal{L}$ with a witness relation $\mathcal{R}$ is said to be* λ-leakage-resilient zero knowledge *if for every* PPT *machine V^* that makes any arbitrary polynomial number of leakage queries on P's state (in the manner as described above) with ℓ bits of total leakage, there exists a* PPT *algorithm $\mathcal{S}$ that obtains at most $\lambda \cdot \ell$ bits of total leakage from a leakage oracle $L_w^{k,\lambda}(\cdot)$ (as defined above) such that for every $(x, w) \in \mathcal{R}$, every $z \in \{0,1\}^*$,* $\text{view}_{V^*}(x, z)$ *and* $\mathcal{S}^{L_w^{k,\lambda}(\cdot)}(x, z)$ *are computationally indistinguishable.*

Some observations on the above definition are in order.

Leakage parameter λ. Note that when $\lambda = 0$, no leakage is available to the simulator (as is the case for the standard zero knowledge simulator). In this case, our definition guarantees the standard zero knowledge property. It is not difficult to see that it is impossible to realize such a definition. In fact, as we show in Section 3.5.1, it is impossible to realize the above definition for any $\lambda < 1$.

On the other hand, in Section 3.1.2, we give a positive result for $\lambda = 1 + \epsilon$, where ϵ is any positive constant. The meaningfulness of our positive result stems from the observation that when λ is close to 1, very roughly, our definition

guarantees that *a malicious verifier does not learn anything from the protocol beyond the validity of the statement being proved and the leakage obtained from the prover.*

Leakage-oblivious simulation. Note that in our definition of leakage resilient zero-knowledge, (apart from the total output length) there is no restriction on the nature of leakage queries that the simulator may make to the leakage oracle. Then, since the simulator has indirect access to the honest prover's witness (via the leakage oracle), it may simply choose to leak on the witness (regardless of the leakage queries of the verifier) in order to help with the simulation of protocol messages instead of using the leakage oracle to *only* answer the leakage queries of the verifier. We stress that this issue should not affect any potential application of leakage resilient zero-knowledge that one may think of. Nonetheless, we think that this is an important issue since it relates to the meaningfulness of the definition. To this end, we note that this issue can easily handled by putting a restriction on how the simulator accesses the leakage oracle. Specifically, we can model the interaction between the simulation and the oracle such that the simulator is not allowed to look at the oracle's responses to its queries. The simulator is simply allowed to provide a translation function that is applied to the leakage queries of the verifier in order to create queries for the oracle. The oracle's responses are sent directly to the verifier and the simulator does not get to seem them. We call such simulators *leakage-oblivious*. We note that the simulator that we construct for our protocol $\langle P, V \rangle$ (described in the next subsection) is leakage-oblivious.[1]

[1]Indeed, since we cannot rule out of obfuscation of arbitrary functionalities, we do not know how to obtain a formal proof without making the simulator leakage-oblivious.

3.1.2 Our Protocol

We now proceed to give our construction of a leakage-resilient zero knowledge interactive proof system as per Definition 8. Very roughly speaking, our protocol can be seen as a combination of Feige-Shamir [FS89] and Goldreich-Kahan [GK96], in that we make use of equivocal commitments from the prover's side, as well as require the verifier to commit to all its challenges in advance. Note that while either of the above techniques would suffice for standard simulation, interestingly, we need to use them *together* to help the simulator handle leakage queries from a cheating verifier. We now describe our protocol in more detail.

Let P and V denote the prover and verifier respectively. Our protocol $\langle P, V \rangle$ proceeds in three stages, described as follows. In *Stage 1*, V commits to its challenge and a large random string r' using a challenge-response based PRS [PRS02] style preamble instantiated with a public-coin statistically hiding commitment scheme (see Section 2.3). In *Stage 2*, P and V engage in coin-flipping (that was initiated in Stage 1 when V committed to r') to jointly compute a random string r. Finally, in *Stage 3*, P and V run k (where k denotes the security parameter) parallel repetitions of the 3-round Blum Hamiltonicity protocol, where P uses Naor's commitment scheme (see Section 2.3) to commit to the permuted graphs in the first round. Here, for each bit commitment i, P uses a different substring r_i (of appropriate length) of r as the first message of Naor's commitment scheme. Protocol $\langle P, V \rangle$ is described in Figure 3.1.

Intuitively, the purpose of multiple challenge response slots in Stage 1 is to allow the simulator to extract the values committed by V^* with minimal use of the leakage oracle. With the knowledge of the extracted values, the simulator can force the output of the coin-flipping to a specific distribution of its choice. This, in turn, allows the simulator to convert Naor's commitment scheme into an

equivocal commitment scheme during simulation.

Theorem 1 *If public-coin statistically hiding commitment schemes exist, then the protocol $\langle P, V \rangle$, parameterized by ϵ, is a $(1+\epsilon)$-leakage-resilient zero knowledge proof system.*

We note that statistically hiding commitment schemes imply one-way functions, which in turn suffice for Naor's statistically binding commitment scheme used in our construction.

3.1.3 Proof of Security

We start by arguing that protocol $\langle P, V \rangle$ is complete and sound. Then we argue that the protocol is $(1 + \epsilon)$-leakage-resilient zero knowledge.

Completeness. The completeness of our protocol follows directly from the completeness of Blum's Hamiltonicity protocol.

Soundness. Before we jump into the proof, we recall and build some notation related to Naor's commitment scheme (cf. Section 2.3) that we will need in our proof. This commitment scheme is statistically binding as long as the first message sent by the receiver does not come from a special set $\mathcal{B} \subset \{0,1\}^{3k}$, where $\mathcal{B}$ is the set of all strings $r = g(s_0) \oplus g(s_1)$ such that $s_0, s_1 \in \{0,1\}^k$ and $g : \{0,1\}^k \to \{0,1\}^{3k}$ is a pseudorandom generator. It follows from inspection that $\frac{|\mathcal{B}|}{2^{3k}}$ is negligible in k. However, observe that if the first message of receiver is in fact chosen from the set $\mathcal{B}$, then Naor's commitment is no longer statistically binding and allows for equivocation.

The proof of soundness of $\langle P, V \rangle$ follows in two steps. First we argue that no

Common Input: A k-vertex graph G.

Private Input to P: A Hamiltonian Cycle H in graph G.

Parameters: $n = \omega(\log k)$, $t = 3k^4$, positive constant ϵ s.t. $\frac{1}{\epsilon}$ is an integer.

Stage 1 (Commitment phase)

$V \rightleftarrows P$**:** Commit to a t-bit random string r' and $(\frac{n^2}{\epsilon})$-pairs of random shares $\left\{r'^0_{i,j}, r'^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$ (such that $r'^0_{i,j} \oplus r'^1_{i,j} = r'$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$) using a public-coin statistically hiding commitment scheme. Similarly commit to a k-bit random string ch and $(\frac{n^2}{\epsilon})$-pairs of random shares $\left\{ch^0_{i,j}, ch^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$ (such that $ch^0_{i,j} \oplus ch^1_{i,j} = ch$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$) using a public-coin statistically hiding commitment scheme.

Challenge-response slots: For every $i \in [\frac{n}{\epsilon}]$,

$P \rightarrow V$**:** Choose n-bit random strings $\alpha_i = \alpha_{i,1}, \ldots, \alpha_{i,n}$ and $\beta_i = \beta_{i,1}, \ldots, \beta_{i,n}$. Send α_i, β_i to V.

$V \rightarrow P$**:** For every $j \in [n]$, V^* decommits to $r'^{\alpha_{i,j}}_{i,j}$ and $ch^{\beta_{i,j}}_{i,j}$.

Stage 2 (Coin-flipping completion phase)

$P \rightarrow V$**:** Choose a t-bit random string r'' and send it to V.

$V \rightarrow P$**:** Decommit to r' and $r'^0_{i,j}$, $r'^1_{i,j}$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$. Let $r = r' \oplus r''$.

Stage 3 (Blum Hamiltonicity protocol)

$P \rightarrow V$**:** Let $r = r_1, \ldots, r_{k^3}$, where $|r_i| = 3k$ for every $i \in [k^3]$. For every $i \in [k]$,

- Choose a random permutation π_i and prepare an isomorphic copy of G, denoted $G_i = \pi_i(G)$.
- For every $j \in [k^2]$, commit to bit b_j in the adjacency matrix of G_i using Naor's commitment scheme with $r_{i \times j}$ as the first message.

$V \rightarrow P$**:** Decommit to ch and $ch^0_{i,j}$, $ch^1_{i,j}$ for every $i \in [\frac{n}{\epsilon}], j \in [n]$.

$P \rightarrow V$**:** Let $ch = ch_1, \ldots, ch_k$. For each $i \in [k]$, if $ch_i = 0$, decommit to every edge in G_i and reveal the permutation π_i. Else, decommit to the edges in the Hamiltonian Cycle in G_i.

Figure 3.1: Protocol $\langle P, V \rangle$

cheating prover P^* can force the string r computed via coin flipping to lie in the set $\mathcal{B}$. Then, given that $r \notin \mathcal{B}$, it follows that the prover's commitments in Stage 3 are statistically binding. From this soundness follows by a standard argument in the same manner as [PRS02, Ros04]. Next, we give more details.

Stage 1 all together can be thought of as a statistically hiding commitment to r' and challenge string ch. We note that coin flipping phase (Stage 2) generates an output r which is $t = 3k^4$ bits long and is used for k^3 Naor's bit commitments. For simplicity of exposition, we restrict ourselves to the first $3k$ bits of r. These correspond to the bits that will be used as the first message of Naor's commitment scheme for the first bit commitment by the prover in Stage 3. We argue that a cheating prover can not force these $3k$ bits to lie in set $\mathcal{B}$. We can argue about the remaining bits of r in an analogous manner. Consider $r_0, r_1 \in \{0,1\}^{3k}$ with the property that there does not exist any $r^* \in \{0,1\}^{3k}$ such that $r_0 \oplus r^* \in \mathcal{B}$ and $r_1 \oplus r^* \in \mathcal{B}$. We argue that if a cheating prover P^* can force the first $3k$ bits of r to lie in $\mathcal{B}$ with non-negligible probability ε, then we can construct an adversary $\mathcal{A}$ that can distinguish between a statistically hiding commitment to r_0 and a commitment to r_1 with probability $\frac{1}{2} + \frac{\varepsilon}{2}$, thus obtaining a contradiction. Consider the adversary $\mathcal{A}$ that takes an input a statistically hiding commitment to r_b (where r_b is either r_0 or r_1) from an external challenger, and uses P^* to determine b. $\mathcal{A}$ forwards the commitment to r_b from the external challenger to P^* as its commitment to the first $3k$ bits of r'. It generates the commitments to the remaining bits of r' and the challenge string ch on its own. Let r'' be the string sent by P^*. Let r^* be the first $3k$ bits of r''. $\mathcal{A}$ outputs $b = 0$ if $r_0 \oplus r^* \in \mathcal{B}$, and $b = 1$ if $r_1 \oplus r^* \in \mathcal{B}$, and randomly guesses $b \in \{0,1\}$ otherwise.

Finally, note that the commitment to challenge ch from the verifier is statistically hiding and Naor's commitment from the prover is statistically binding.

From this it follows by standard argument that if a cheating prover can convince verifier of a false theorem then we can use this prover to break the statistical hiding property of the statistically hiding commitment scheme.

Leakage-Resilient Zero Knowledge. Now we argue that the protocol $\langle P, V \rangle$ (cf. Figure 3.1) is $(1 + \epsilon)$-leakage-resilient zero knowledge. For this we need to construct a simulator that simulates the view of every cheating verifier V^*. Our simulator has access to a leakage oracle $L_w^{k,\lambda}(\cdot)$ to help it with the leakage queries, such that if V^* queries for a total of ℓ bits of leakage then the simulator is allowed to leak $(1+\epsilon)\cdot\ell$ bits. Without loss of generality, we assume that immediately after every protocol message sent by the prover, the cheating verifier makes exactly one leakage query. However, we do not restrict the output length or the nature of these queries. In particular, these queries could change adaptively depending on the messages of the prover and the leakage itself. We stress that the above assumption has been made only to simplify exposition, and indeed, our simulator can handle arbitrary number of leakage queries as well.

We start by describing our simulator in the next section. We then discuss bounds on the total leakage required by the simulator, and finally give a proof that the view of a cheating verifier interacting with a real prover is computationally indistinguishable from the view of the verifier interacting with our simulator.

3.1.3.1 Description of $\mathcal{S}$

We start by giving an informal description of the simulator, and then proceed to a more formal treatment.

Informal description of $\mathcal{S}$. The purpose of Stage 1 in our protocol is to help the simulator in the extraction of r' and ch. Once a successful extraction of these values is completed, the simulator can simulate in a "straight line" manner. Further, note that typically extraction in a "stand-alone setting" (in the absence of leakage) can be performed in expected polynomial time by rewinding multiple times in only one slot. Therefore, at first it might seem unnatural that we use $n = \omega(\log(k))$ slots. However we stress that rewinding in our case is complicated by the fact that the simulator has to respond to the leakage queries of the verifier. Whenever the simulator rewinds V^*, it might ask for a *new* leakage query (different from the one it asked on the "main" thread of execution); as a result, the total leakage required by the simulator might grow with the number of rewindings.

We deal with this issue by using the following rewinding strategy. Consider the i^{th} challenge response slot in Stage 1. We will refer to the main thread of execution with the verifier (that is output by the simulator) as the *main thread* and the execution thread created as a result of rewinding as the *look-ahead* thread. Now, consider the leakage query made by V^* immediately after the after the simulator sends a random challenge in the the i^{th} slot on the main thread. Suppose that the output length of this query is ℓ^m bits. The simulator will respond to this query using the leakage oracle $L_w^{k,\lambda}(\cdot)$ (in the manner as described later). Now, the simulator rewinds V^* *once* (in that slot) and creates a look-ahead thread, where it sends a new random challenge. The verifier may now ask for a new leakage query. Suppose that the output length of this query is ℓ^a bits. If $\ell^a \leq \ell^m$, then the simulator responds to this query using the leakage oracle $L_w^{k,\lambda}(\cdot)$ and aborts the look-ahead thread otherwise. The simulator will follow the same strategy for each slot.

Now, based on a standard "swapping argument", we can say that in each slot, in which V^* does not abort in the main thread, the simulator is able to extract r' and ch with a probability at least $1/2$. If V^* does not cause an abort in the main thread, then the simulator has n rewinding opportunities, and it will be able to extract r' and ch with overwhelming probability. This is still not good enough as the simulator might need leakage that is twice in size than what the verifier queries (whereas we want a precision of $(1+\epsilon)$). We fix this issue by having the simulator rewind in only those slots in which the leakage queries have "short output length."

We now proceed to give a formal description of the simulator. We split the simulator into three key parts, that correspond to the three stages of the protocol $\langle P, V\rangle$ (cf. Figure 3.1). We go over these step by step.

Description of $\mathcal{S}$ in Stage 1. Recall that in Stage 1 of the protocol, the verifier commits to a string r' and its shares shares $\left\{r'^0_{i,j}, r'^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$, as well as a challenge string ch and its shares $\left\{ch^0_{i,j}, ch^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$. Following these commitments, there are $\frac{n}{\epsilon}$ challenge-response slots between $\mathcal{S}$ and the verifier. For each $p \in \{0,\ldots,n-1\}$ consider the set of slots $\frac{p}{\epsilon}+1$ to $\frac{p+1}{\epsilon}$. Let ℓ^{avg} denote the average output length of the verifier's queries among these slots. The simulator chooses one of the $1/\epsilon$ slots at random. Let i be the chosen slot. The simulator rewinds to the point in the i^{th} slot where the challenge was sent and sends a new random challenge. At this point V^* might make a new leakage query. Let the output length of this query be ℓ_i^{la} bits. If $\ell_i^{\mathsf{la}} \leq \ell_{\mathsf{avg}}$, then $\mathcal{S}$ uses the leakage oracle to answer the leakage query (in the manner as discussed below). Now, if V^* decommits correctly (as per the random challenge), then the simulator uses the decommitted values (obtained on the main thread and the look-ahead thread) to extract both r' and ch. It then aborts the look ahead. Further note that we con-

sider one such slot i for each set of slots $\frac{p}{\epsilon}+1$ to $\frac{p+1}{\epsilon}$ (where $p \in \{0,\ldots,n-1\}$). If the simulator fails to extract r' and ch before the completion of Stage 1, then it aborts. This rewinding strategy is demonstrated more formally in Figure 3.2. Note that leakage queries have been explicitly marked by the $\twoheadleftarrow$ arrow.

Leakage Queries in Stage 1. Let $R(\cdot)$, which takes the prover's witness w as input, be a function that outputs the value of random coins of an honest prover which when used along with the prover's witness will result in the messages generated by the simulator. More specifically, the honest prover strategy with the prover's witness w and the random coins $R(w)$ will generate the exact same messages as the simulator. The function $R(\cdot)$ is initialized with the null string. Now note that in this stage, all messages played by an honest prover are public coin. Then, $R(\cdot)$ at any point in Stage 1 is just the concatenation of all the protocol messages sent by the simulator so far. Now consider a leakage query f of the adversarial verifier that takes as input the prover's witness and the random coins used by the prover so far. On receiving such a query f, the simulator creates a new query f' (that takes as input only the prover's witness w) such that $f'(w) = f(w, R(w))$. It then queries the leakage oracle with f' to obtain $f'(w)$ and returns it to the cheating verifier.

Description of $\mathcal{S}$ in Stage 2. Let r' be the random string (not including the challenge ch) extracted by the simulator in Stage 1. For every $v \in \{0,\ldots,k^3-1\}$, the simulator chooses $r''_v = r'_v \oplus g(s^0_v) \oplus g(s^1_v)$ (where $s^0_v, s^1_v \in \{0,1\}^k$ are randomly chosen) and sends it to V^*. Here, r''_v and r'_v denote $3k$ bit long substrings of r'' and r' respectively, between positions $3vk+1$ and $3(v+1)k$. Now, if V^* decommits to a value different from the extracted string r', then $\mathcal{S}$ aborts.

Leakage Queries in Stage 2. All messages played by an honest prover in Stage 2

Common Input: A k-vertex graph G.

Private Input to $L(\cdot)$: A Hamiltonian Cycle H in graph G (same as real prover).

Parameters: Security parameter 1^k, $n = \omega(\log(k))$, $t = 3k^4$, ϵ is a positive constant. Without loss of generality, we assume that $\frac{1}{\epsilon}$ is an integer.

$V^* \rightleftarrows \mathcal{S}$**:** $\mathcal{S}$ acts just like a real prover and obtains the commitments to r' $\left\{r'^{0}_{i,j}, r'^{1}_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$, ch and $\left\{ch^0_{i,j}, ch^1_{i,j}\right\}_{i=1,j=1}^{i=\frac{n}{\epsilon},j=n}$ from V^*.

$\mathcal{S} \nleftarrow V^*$**:** V^* could make multiple leakage queries in the above step. $\mathcal{S}$ uses $L_w^{k,\lambda}(\cdot)$ to answer all these leakage queries (in the manner as described in the main text). V^* could abort as well, in which case $\mathcal{S}$ aborts.

Challenge Response: For every $p \in 0, \ldots, (n-1)$,

1. For every $q \in 1, \ldots, 1/\epsilon$, do the following. Let $i = p/\epsilon + q$.
   (a) $\mathcal{S} \rightarrow V^*$: Choose n-bit random strings $\alpha_i = \alpha_{i,1}, \ldots, \alpha_{i,n}$ and $\beta_i = \beta_{i,1}, \ldots, \beta_{i,n}$. Send α_i, β_i to V.

$\mathcal{S} \nleftarrow V^*$: $\mathcal{S}$ uses $L_w^{k,\lambda}(\cdot)$ to answer the leakage queries (in the manner as described in the main text). Let ℓ_{avg} denote the average output length of all the queries in the $1/\epsilon$ slots.

   (b) $V^* \rightarrow \mathcal{S}$: For every $j \in [n]$, V^* decommits to $r'^{\alpha_{i,j}}_{i,j}$ and $ch^{\beta_{i,j}}_{i,j}$.

2. $\mathcal{S} \rightarrow V^*$: $\mathcal{S}$ rewinds V^* to Step 1a of slot i, where $i \in \{\frac{p}{\epsilon}+1, \ldots, \frac{p+1}{\epsilon}\}$ is chosen uniformly at random. It chooses fresh n-bit random strings $\alpha'_i = \alpha'_{i,1}, \ldots, \alpha'_{i,n}$ and $\beta'_i = \beta'_{i,1}, \ldots, \beta'_{i,n}$ and sends α'_i, β'_i to V.

$\mathcal{S} \nleftarrow V^*$: Let the output length of the leakage query be ℓ_i^{la} bits. If $\ell_i^{\mathsf{la}} \leq \ell_{\mathsf{avg}}$, then $\mathcal{S}$ uses $L_w^{k,\lambda}(\cdot)$ to answer the leakage queries. Otherwise it aborts.

3. $V^* \rightarrow \mathcal{S}$: For $j \in [n]$, V^* opens $r'^{\alpha_{i,j}}_{i,j}$ and $ch^{\beta_{i,j}}_{i,j}$ or it aborts. In either case $\mathcal{S}$ aborts the look ahead thread.

Note on leakage queries. All messages played by the simulator in Stage 1 are public coin; therefore, any leakage query from V^* can be reduced to a leakage query on only the witness (as described in the main text).

Figure 3.2: Rewindings in Stage 1.

are also public coin and just like in Stage 1, $R(\cdot)$ at any point in Stage 2 is just the concatenation of all the protocol messages sent by the simulator so far. The leakage queries of the cheating verifier are handled using $R(\cdot)$ in the same way as described earlier in Stage 1.

Description of $\mathcal{S}$ in Stage 3. Let ch denote the challenge string extracted by the simulator at the end of Stage 1. Let $ch = ch_1, \ldots, ch_k$. For each $i \in [k]$, if $ch_i = 0$, then the simulator chooses a random permutation π_i and commits to $G_i = \pi_i(G)$; otherwise, it commits to a random k-cycle graph G_i. Depending upon the verifier's challenge, it reveals the permutation π_i and decommits to the graph G_i or it decommits to the edges corresponding to the cycle in G_i. If the challenge string sent by V^* is different from ch, then $\mathcal{S}$ aborts.

Leakage Queries in Stage 3. The leakage queries in this Stage need to be handled carefully. Observe that during Stage 3, for every $i \in [k]$, an *honest prover* chooses a random permutation π_i and commits to $G_i = \pi_i(G)$. Note that during this process, an honest prover would have flipped coins to generate a random permutation π_i and commitments to G_i. To emulate honest prover behavior, $\mathcal{S}$ must be able to reconstruct the permutation π_i and the randomness used in generating commitments to G_i as a function of the witness only. Furthermore, this randomness must be consistent with what $\mathcal{S}$ later reveals (while decommitting) in Stage 3. There are two cases.

1. If $ch_i = 0$, then, as mentioned earlier, $\mathcal{S}$ chooses a random permutation π_i and commits to $G_i = \pi_i(G)$. Let R' denote the random coins used by $\mathcal{S}$ to generate the commitments. In this case $\mathcal{S}$ updates $R(\cdot)$ as $R(\cdot)\|\pi_i\|R'$.

2. The case when $ch_i = 1$ is slightly more involved. In this case, as mentioned earlier, $\mathcal{S}$ commits to a random k-cycle graph G_i. For the edges that are not

part of the cycle, it commits in a way so that it can equivocate. Observe that later in the simulation $\mathcal{S}$ will actually reveal the cycle and decommit the edges on the cycle. Hence the cycle and the openings to the commitments that correspond to the edges of the cycle are fixed. Now, intuitively, by "using the witness" $\mathcal{S}$ can map the cycle in G with the cycle in G_i and obtain a permutation π_i. $\mathcal{S}$ then computes $G^* = \pi_i(G)$ and uses equivocation to explain commitments that it sent earlier as if they were for G^*. However, it must do all this in a setting where it has access to the witness only via the leakage oracle.

More formally, consider a string ρ that consists of the following values. First, for each edge belonging to the cycle in G_i, ρ consists of the random coins that $\mathcal{S}$ used when it committed to bit 1 for that edge. Further, for each other edge (not in the cycle) or a non-edge, ρ consists of both the random coins that result in a commitment to bit 1 and the random coins that result in a commitment to bit 0. Note that the simulator can compute the random coins for both cases since it can equivocate. Now consider the function $R'(G, \rho, w)$ that works as follows. It first superimposes the cycle graph G_i onto G such that the cycle in G (determined by w) maps to the cycle in G_i. Note that this can be done in multiple ways. The function R' picks one such mapping randomly. It then obtains the permutation π_i that would lead to this mapping. Now, let G^* denote the graph such that $\pi_i(G) = G^*$. Note that G^* consists of the same k-cycle as in G_i, while the remaining structure of the graph may be different. Now, the function R' determines the random coins (from ρ) that when used to commit to (the adjacency matrix for) G^* would result in the same commitment string as the one that $\mathcal{S}$ sent earlier (when it committed to G_i). For the edges corresponding to the cycle in G^*, R' selects from ρ the (unique) random coins corresponding

to the edges belonging to the cycle in G_i. Further, for each other edge (not in the cycle) or non-edge in G^*, (depending upon whether it is an edge or a non-edge) R' selects the *appropriate* corresponding random coins from ρ (where the correspondence is determined by the mapping obtained above). Let R'' denote the concatenation of all the random coins selected from ρ in the above manner. Finally, R' outputs $\pi_i \| R''$. Now the simulator updates the function $R(\cdot)$ as $R(\cdot) \| R'(G, \rho, \cdot)$.

The leakage queries of the cheating verifier are handled using $R(\cdot)$ in the same way as described earlier in Stage 1.

3.1.3.2 Total leakage queries by $\mathcal{S}$

Lemma 1 *If in a protocol execution V^* makes queries with a total leakage of ℓ bits then the simulator $\mathcal{S}$ only requires $(1+\epsilon) \cdot \ell$ bits of leakage.*

Proof. This follows directly from the construction of our simulator. Consider the first $\frac{1}{\epsilon}$ slots in Stage 1 of the protocol. Let ℓ_1 be the total leakage obtained by the verifier during these slots on the main thread. Now, consider the slot i (out of these $\frac{1}{\epsilon}$ slots) where the simulator performs a single rewind. Let ℓ_i^{la} be the output length of the leakage query made by the verifier on the look-ahead thread created. Recall that ℓ_i^{la} is at most the average output length of the leakage queries made during the first $\frac{1}{\epsilon}$ slots on the main thread. In other words, $\ell_i^{\mathsf{la}} \leq \epsilon \cdot \ell_1$. Thus, the total leakage obtained by the verifier during the first $\frac{1}{\epsilon}$ slots is $(1+\epsilon) \cdot \ell_1$.

The same reasoning applies to each set of $\frac{1}{\epsilon}$ slots, and therefore, the total leakage is upper bounded by $(1+\epsilon) \cdot \ell$, where $\ell = \sum_{j=1}^{n} \ell_j$. □

3.1.3.3 Indistinguishability of the views

We now need to prove that view of V^* generated in interaction with the real prover is indistinguishable from the view generated when interacting with the simulator $\mathcal{S}$. We start by describing our hybrids.

$\mathcal{H}_0$**:** This hybrid corresponds to the view of the verifier V^* in interaction with $\mathcal{S}$ when it has the witness and follows honest prover strategy. This corresponds to the real interaction. Leakage queries are answered directly based on the witness and the public coins used by the simulator.

$\mathcal{H}_1$**:** This hybrid is just like in $\mathcal{H}_0$, except that the simulator $\mathcal{S}$ rewinds V^* in n challenge response slots of Stage 1 as explained in Figure 3.2. $\mathcal{S}$ aborts if the main thread reaches end of Stage 1 but r' and ch have not been extracted. The simulator has the witness and the leakage queries are answered in the same way as in $\mathcal{H}_0$.

$\mathcal{H}_2$**:** This hybrid is just like in $\mathcal{H}_1$, except that $\mathcal{S}$ aborts if V^* opens r' and ch differently from the extracted values. Leakage queries are answered in the same way as in $\mathcal{H}_1$.

$\mathcal{H}_3$**:** This hybrid is same as $\mathcal{H}_2$, except that instead of sending a random string r'' to V^*, $\mathcal{S}$ does the following. For every $v \in \{0, \ldots, k^3 - 1\}$, $\mathcal{S}$ chooses $r''_v = r'_v \oplus g(s^0_v) \oplus g(s^1_v)$ (where $s^0_v, s^1_v \in \{0,1\}^k$ are randomly chosen) and sends it to V^*. (Here, r''_v and r'_v denote $3k$ bit long substrings of r'' and r' respectively, between positions $3vk + 1$ and $3(v + 1)k$.) Further, the v^{th} commitment in Stage 3 is made by sending $g(s^0_v)$. It can be opened to 0 by sending s^0_v and to 1 by sending s^1_v. The simulator has the witness and the leakage queries are answered in the same way as in $\mathcal{H}_2$.

$\mathcal{H}_4$**:** $\mathcal{H}_4$ is different from $\mathcal{H}_3$ only in the commitments that prover makes in the Stage 3. Let $ch = ch_1, \ldots, ch_k$ be the challenge string that $\mathcal{S}$ extracted in Stage 1. For each $i \in [k]$, if $ch_i = 0$, then $\mathcal{S}$ chooses a random permutation π_i and commits to $G_i = \pi_i(G)$. Else, $\mathcal{S}$ commits to a random k-cycle graph G_i. Depending upon the verifier's challenge, it reveals the permutation π_i and decommits to the graph G_i or it decommits to the edges corresponding to the cycle in G_i. Leakage queries are handled as described in the description of the simulator. Note that simulator only needs access to a leakage oracle to answer the leakage queries. Note that $\mathcal{H}_4$ corresponds to the simulator described earlier.

Indistinguishability of $\mathcal{H}_0$ and $\mathcal{H}_1$. The only difference between hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ is that the simulator may abort in $\mathcal{H}_1$ at the end of Stage 1. Now, consider the event E that the prover in $\mathcal{H}_1$ reaches the end of the Stage 1 (Commitment Phase) but fails to extract r and ch (and thus aborts). From Lemma 2 (given below) it follows that the probability of event E is negligible and therefore the hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ are statistically close.

Indistinguishability of $\mathcal{H}_1$ and $\mathcal{H}_2$. We note that V^* can not open commitments to r' and ch differently from the extracted values, because the commitment being used is computationally binding. If V^* opens any of the commitments in two different ways with a non-negligible probability then we can use V^* to construct an adversary that breaks the computational binding property of the used commitment scheme. Then, it follows that $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable.

Indistinguishability of $\mathcal{H}_2$ and $\mathcal{H}_3$. If an adversary can distinguish between hybrids $\mathcal{H}_2$ and $\mathcal{H}_3$ then we can use this adversary to break the security of the pseudo-random generator used in our instantiation of Naor's commitment scheme. For this consider a sequence of hybrids $\mathcal{H}_{2,0} \ldots \mathcal{H}_{2,k^3}$. In $\mathcal{H}_{2,i}$ the bits in r that correspond to the first i commitments are created in a way as specified in the hybrid $\mathcal{H}_3$, the rest are created as in $\mathcal{H}_2$. Observe that hybrid $\mathcal{H}_{2,0}$ is same as hybrid $\mathcal{H}_2$ and hybrid $\mathcal{H}_{2,k^3}$ is same as hybrid $\mathcal{H}_3$. Now we argue that if an adversary $\mathcal{D}$ can distinguish between hybrids $\mathcal{H}_{2,i}$ and $\mathcal{H}_{2,i+1}$ then we can use this adversary to construct an adversary $\mathcal{A}$ that can distinguish a random string from a pseudorandom string. The argument depends on the value being committed. $\mathcal{A}$ obtains a string a and it is supposed to guess if it is random or pseudorandom. It picks a random string s^1 and evaluates $g(s^1)$. It forces the bits of r that corresponding to the i^{th} commitment to $g(s^1) \oplus a$. And sends its commitment as $g(s^1)$ if it needs to commit to 0 and as a if it needs to commit to 1. The distinguishing advantage of $\mathcal{D}$ directly translates to the distinguishing advantage of $\mathcal{A}$.

Indistinguishability of $\mathcal{H}_3$ and $\mathcal{H}_4$. Finally, we note that hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$ are identical. The only change in $\mathcal{H}_4$ form $\mathcal{H}_3$ is that the simulator does not know the witness and hence does not know the openings of the commitments that it does not open in the protocol. But even though $\mathcal{S}$ does not know the openings and can not compute them efficiently, the openings itself come from the same distribution as $\mathcal{H}_3$. And the simulator having access to the leakage oracle can evaluate these openings and answer leakage queries correctly. It follows from the description of the simulator that it responds to the leakage queries in exactly the same manner as hybrid $\mathcal{H}_3$, therefore the view of the cheating verifier with respect to the leakage queries is identical in hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$.

Lemma 2 *Consider the event E that the simulator reaches the end of the Stage 1 but fails to extract r' and ch. Then,*

$$\Pr[E] \leq \text{negl}(k)$$

Proof. For every $j \in [n]$,consider the event E_j such that:

1. V^* responds to the challenge in slot $i \in \{\frac{j-1}{\epsilon}+1, \ldots, \frac{j}{\epsilon}\}$ where i is the slot that the simulator chooses at random in order to perform a single rewind. Let ℓ_i^{m} denote the output length of the leakage query made by the verifier during slot i.

2. When V^* is rewound in the slot i, then V^* makes a query with output length ℓ_i^{la} in the look ahead thread such that $\ell_i^{\mathsf{la}} > \ell^{\mathsf{avg}}$ or it aborts after a leakage query of length $\ell^{\mathsf{la}} \leq \ell^{\mathsf{avg}}$. Here ℓ^{avg} is the average output length of the leakage queries made by the verifier during slots $\{\frac{j-1}{\epsilon}+1, \ldots, \frac{j}{\epsilon}\}$ on the main thread.

Lets say that the challenge sent in the main thread is c and the challenge sent in the look ahead is c' such that the event E_i happens. We ignore the case in which $c = c'$ as this happens with negligible probability. It can been seen that since both c and c' are chosen randomly, it is equally likely that challenge c' was chosen in the main thread and c was chosen in the look ahead thread. Now note that with probability ϵ, we have that $\ell_i^{\mathsf{m}} \leq \ell^{\mathsf{avg}}$. Conditioned on this event, note that V^* would not abort in the look ahead thread and if it does not abort in the main thread then the output length of the leakage query in the look ahead thread would be smaller than the output length of the leakage query in the main thread. In a nutshell, we have argued that for every choice of challenges which leads to event E_j there exists another choice which does not lead to event E_j.

Hence, $\Pr[E_j] \leq (1 - \epsilon/2)$. This holds for every j by the same argument. Note that each E_j is an independent event and since the simulator gets to rewinds in $n = \omega(\log k)$ different slots the probability that it fails to extract in all of them is $\text{negl}(k)$.

$$\begin{aligned}
\Pr[E] &= \Pr[\bigwedge_{j=1}^{n} E_j] \\
&= \bigwedge_{j=1}^{n} \Pr[E_j] \\
&\leq \left(1 - \frac{\epsilon}{2}\right)^n \\
&= \text{negl}(k)
\end{aligned}$$

□

3.1.4 Leakage-Resilient Zero Knowledge Proofs of Knowledge

Very informally, an interactive proof system is a *proof of knowledge* if not only does the prover convince the verifier of the validity of the statement, but it also possesses a witness for the statement. This intuition is formalized by showing the existence of an *extractor* machine, that is able to extract a witness from a prover that succeeds in convincing an honest verifier. The proof of knowledge property can be useful in several applications. In particular, in our construction of a UC secure computation protocol (see Section 3.3.1) in the "leaky token model", we will need a λ-leakage-resilient zero knowledge proof of knowledge (LR-ZKPOK) system.

We note that the protocol $\langle P, V \rangle$ described earlier is not a proof of knowledge. Very roughly, note that since the verifier challenge (to be used in Stage 3) is committed to in advance in Stage 1, the standard extractor algorithm for Blum's Hamiltonicity protocol cannot be used here. To this end, we now briefly discuss

how to modify protocol $\langle P, V \rangle$ to incorporate the proof of knowledge property.

In the modified protocol, in Stage 3, the verifier simply reveals the value *ch* (without decommitting) and additionally engages in an execution of a *public-coin* zero knowledge proof of knowledge $\langle P', V' \rangle$ to prove that the revealed value *ch* is correct. Now, during the rewindings, the extractor algorithm can simply send a random challenge string and use the simulator for $\langle P', V' \rangle$ to convince the prover that the revealed value is correct.

We note that while the above modification seems to work fine for extraction purposes, we need to verify that it does not adversely affect the leakage-resilient zero knowledge property of the original protocol. Specifically, note that since a cheating verifier is allowed to make arbitrary leakage queries in our model, we would require that protocol $\langle P', V' \rangle$ remains *sound* even when P' (played by the verifier of $\langle P, V \rangle$) can obtain arbitrary leakage information from V' (played by the prover of $\langle P, V \rangle$). To this end, we note that since $\langle P', V' \rangle$ is *public-coin*, leakage queries do not reveal any useful information to P' as long as it cannot leak on *future random coins*, which is indeed the case in our model. We note that proof of Theorem 1 given in previous subsection can be easily extended to account for these changes.

3.2 Leakage-Resilient NIZK

In this section, we discuss our results on leakage-resilient NIZKs. To begin with, we describe our (leakage) model and give our definition of leakage-resilient NIZKs. We refer the reader to Section 2.2 for the standard definition of non-interactive zero knowledge proof systems. Below, we will follow the notation introduced in Section 2.2.

3.2.1 Our Definition

We consider the scenario where a malicious verifier can obtain arbitrary leakage on the witness and the random coins used by an honest prover to generate the proof string. To model leakage attacks, we allow the cheating verifier to make adaptive leakage queries on the honest prover's witness and the random coins used to generate the proof string. A leakage query to the prover consists of an efficiently computable function f, to which the prover replies with $f(w\|r)$, where w and r denote the prover's witness and random coins respectively. It is easy to see that in the non-interactive proofs setting, a cheating verifier who is allowed multiple leakage queries enjoys no additional power than one who is allowed only one leakage query. Therefore, for simplicity of exposition, from now on, we only consider cheating verifiers who make only one leakage query. We note that our definition given below can be easily adapted to incorporate multiple leakage queries. [2]

We model the zero knowledge simulator $\mathcal{S}$ as a PPT machine that has access to a leakage oracle $L_w^k(\cdot)$ that is parameterized by the honest prover's witness w and the security parameter k. (Unlike the interactive proofs setting, here we do not consider the leakage parameter λ for simplicity of exposition.) The leakage oracle accepts queries of the form f (where $f(\cdot)$ is an efficiently computable function) and outputs $f(w)$. In order to bound the total leakage available to the simulator, we require that if the verifier obtains ℓ bits of total leakage from the honest prover, then the total leakage obtained by the simulator (from the leakage oracle) must be bounded by ℓ bits.

[2] As in the case of leakage-resilient zero knowledge interactive proofs, we do not require an a-priori bound on the total leakage obtained by the verifier in order to satisfy our definition (described below). Nevertheless, in order for our definition to be meaningful, we note that the total leakage obtained by the verifier must be smaller than the witness size.

Definition 9 (LR-NIZK) *A non-interactive proof system (K, P, V) for a* PPT *relation $\mathcal{R}$ is said to be a* leakage-resilient NIZK *if there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that for all adversaries $\mathcal{A}$,*

$$\Pr[\sigma \leftarrow K(1^k) : \mathcal{A}^{PR(\sigma,\cdot,\cdot,\cdot)}(\sigma) = 1] \stackrel{c}{\equiv} \Pr[(\sigma,\tau) \leftarrow \mathcal{S}_1(1^k) : \mathcal{A}^{SR^{L^k_w(\cdot)}(\sigma,\tau,\cdot,\cdot,\cdot)}(\sigma) = 1],$$

where $PR(\sigma, x, w, f)$ computes $r \leftarrow \{0,1\}^{\ell_P(k)}$; $\pi \leftarrow P(\sigma, x, w; r)$; $y = f(w\|r)$ and returns (π, y), while $SR^{L^k_\lambda(\cdot)w}(\sigma, \tau, x, w, f)$ computes $r \leftarrow \{0,1\}^{\ell_S(k)}$; $\pi \leftarrow \mathcal{S}_2(\sigma, \tau, x; r)$; $f' \leftarrow \mathcal{S}_3(\sigma, \tau, x, r, f)$; $y \leftarrow L^k_w(f')$ and returns (π, y). Here, the leakage query f' made to $L^k_w(\cdot)$ is such that its output length is no more than the output length of f. Both the oracles PR and SR output `fail` *if $(x, w) \notin \mathcal{R}$.*

3.2.2 Our Result

We now show that every NIZK proof system with the honest prover state reconstruction property (see Section 2.2 for a formal definition) is in fact a leakage-resilient NIZK. An immediate corollary is that the Groth et al. [GOS06] NIZK proof system is a leakage-resilient NIZK proof system.

Theorem 2 *A NIZK proof system (K, P, V) for a relation $\mathcal{R}$ with honest prover state reconstruction is a leakage resilient NIZK for $\mathcal{R}$.*

Proof. Given that (K, P, V) is a NIZK proof system with honest prover state reconstruction, let $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2, \mathcal{S}'_3)$ denote a simulator for (K, P, V) as per Definition 5. Then, given such a simulator $\mathcal{S}'$, we show how to construct a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ that satisfies Definition 9.

The machine $\mathcal{S}_1$ is identical to $\mathcal{S}'_1$ in that on input 1^k, it samples a CRS string σ along with a trapdoor τ. Similarly, the machine $\mathcal{S}_2$ is identical to $\mathcal{S}'_2$ in that on input a CRS string σ, trapdoor τ, statement x and randomness ρ, it outputs a proof string π. The machine $\mathcal{S}_3$ works as follows. It takes as input a

CRS string σ, trapdoor τ, statement x, randomness ρ, and a leakage query f, and outputs the description of a function f' (that only takes the witness w as input), described as follows. The function f' on input the witness w first runs the machine $\mathcal{S}'_3(\sigma, \tau, x, w, \rho)$ to obtain a random string r and then computes and outputs $f(w\|r)$. Note that f' has the CRS σ, trapdoor τ, statement x, and randomness ρ hardwired in it. Furthermore, it follows from the description of f' that the output lengths of f' and f are equal.

We now argue that the simulated view of the adversary is indistinguishable from its real view. To this end, first note that the adversary's real (resp., simulated) view only consists of the proof string π^* (resp., π) and the leakage y^* (resp, y) obtained from the honest prover (resp., simulator $\mathcal{S}$). Further, note that y^* is a function of the witness w and the honest prover's randomness (say) r^* (used to compute π^*), while y is a function of w and the honest prover's state r reconstructed by $\mathcal{S}'_3$. Then, observe that to argue the indistinguishability of adversary's views, it suffices to argue that the joint distribution of (π, w, r) is indistinguishable from the joint distribution of (π', w, r'). However, we note that this already follows from the honest prover state reconstruction property of (K, P, V). This completes the proof. □

3.3 Applications of Leakage-Resilient Zero Knowledge

3.3.1 Universally Composable Security with Leaky Tokens

Starting with the work of Goldreich and Ostrovsky on software protection [GO96], tamper-proof hardware tokens have been used for a variety of cryptographic tasks such as achieving universal composability [Kat07, CGS08, MS08, DNW09], one-time-programs [GKR08], unconditionally secure protocols [GIS+10, GIMS10],

compilers for leakage-resilient computation [JV10, GR10], etc. To the best of our knowledge, all prior works using tamper-proof hardware tokens make the assumption that the tokens are completely leakage-resilient (i.e., a token does not leak any information to an adversary in possession of the token). Here, we start a new line of research to investigate whether it is possible to relax this assumption for various cryptographic tasks. In particular, in this section, we study the feasibility of doing universally composable secure computation using "leaky" tokens. More specifically, we start with the tamper-proof hardware token model of Katz [Kat07] and modify it appropriately to incorporate "bounded" leakage. Then, by making use of leakage-resilient hard relations [DHLW10b] and our leakage-resilient zero knowledge proof of knowledge, we give a construction for a universally composable multi-party computation protocol in the leaky token model.

The rest of this section is organized as follows. We first recall the hardware token model of Katz and describe our modification to incorporate leakage attacks in Section 3.3.1.1. Next, in Section 3.3.1.2, we recall the notion of "UC-puzzles" [LPV09] that is central to our positive result. Finally, we describe our positive result in Section 3.3.1.3.

3.3.1.1 Tamper-proof Hardware Setup

In the tamper-proof hardware model [Kat07], it is assumed that all parties in the system can exchange tamper-proof hardware tokens with each other. Specifically, in this model, a party can take some software code and "seal" it inside a tamper-proof hardware token; the party can then give this token to another party, who can then access the embedded software in a black-box manner. Here, the first party is referred to as the token *creator*, while the other party is referred to as

the token's *user*. This setup is modeled by a "wrapper" functionality $\mathcal{G}_{wrap}$ that accepts two types of messages: the first type is used by a party P to create a hardware token (encapsulating an interactive protocol M) and to "send" this token to another party P'. $\mathcal{G}_{wrap}$ enforces that P can send at most one token to P' which is used for all their protocol interactions throughout their lifetimes (and not just for the interaction labeled by the *sid* used when the token is created). Once the token is "created" and "sent" to P', this party can interact with the token in an arbitrary black-box manner. This is formalized by allowing P' to send messages of its choice to M via the wrapper functionality $\mathcal{G}_{wrap}$. Note that each time M is invoked, fresh random coins are chosen for M. Finally, note that $\mathcal{G}_{wrap}$ prevents the token creator P from sending any messages to the token once it is "sent" to P'. The functionality $\mathcal{G}_{wrap}$ (as defined in [Kat07])) is described in Figure 3.3.

The Leaky Token Model. We wish to weaken the assumption about the "tamper-proofness" of the hardware tokens by allowing "bounded" leakage of the secret state of a token to its user. To this end, we consider a modified wrapper functionality $\mathcal{G}_{wrap}^{\ell}$ parametrized by a leakage-parameter ℓ that defines the "total" leakage available to a token user over all the executions of the token. More concretely, the new wrapper functionality $\mathcal{G}_{wrap}^{\ell}$ is defined in the same manner as $\mathcal{G}_{wrap}$, except that $\mathcal{G}_{wrap}^{\ell}$ accepts special `leak` queries (from the token user) that consist of a length-decreasing function $f_i : \{0,1\}^* \to \{0,1\}^{\ell_i}$ (described as a circuit), to which the functionality answers with $f(M, state)$, where M denotes the code of the interactive Turing machine encapsulated in the token and *state* denotes the current state of M consisting of all the protocol messages received from the user and the random coins used so far by M in the current protocol execution. The token user can make any arbitrary polynomial number of such

Functionality $\mathcal{G}_{\mathbf{wrap}}$

$\mathcal{G}_{wrap}$ is parameterized by a polynomial p and an implicit security parameter k.

Creation. Upon receiving $(\texttt{create}, sid, P, P', M, n)$ from P, where P' is another user in the system and M is an interactive Turing machine, do:

- Send $(\texttt{create}, sid, P, P')$ to P'.
- If there is no tuple of the form $(P, P', \star, \star, \star, \star)$ stored, then store $(P, P', M, n, 0, \emptyset)$.

Execution. Upon receiving $(\texttt{run}, sid, P, \mathsf{msg})$ from P', find the unique stored tuple $(P, P', M, n, i, state)$ (if no such tuple exists, then do nothing). Then, choose random $r \leftarrow \{0,1\}^{p(k)}$. Run $M(\mathsf{msg}; r; state)$ for at most p(k) steps, and let out be the response (set $\mathsf{out} = \perp$ if M does not respond in the allotted time). Send (sid, P, out) to P', and:

Case 1 ($\mathbf{i < n - 1}$): Store $(P, P', M, n, i + 1, (\mathsf{msg} \| r \| state))$ and erase $(P, P', M, n, i, state)$.

Case 2 ($\mathbf{i = n - 1}$): Store $(P, P', M, n, 0; \emptyset)$ and erase $(P, P', M, n, i, state)$.

Figure 3.3: The wrapper functionality [Kat07].

leakage queries over multiple protocol executions with M; we only require that the functions f_i be efficiently computable, and the total number of bits leaked (over all executions) is $\sum_i \ell_i = \ell$. The functionality $\mathcal{G}^{\ell}_{wrap}$ is described in Figure 3.4.

We stress that by allowing leakage on M, we essentially allow the token user to obtain leakage on any secret values hardwired into M. We also stress that it is important for our purposes that the wrapper functionality $\mathcal{G}^{\ell}_{wrap}$ flip fresh random coins (to be used by M) during each round of a protocol execution between M

and the token user. (In contrast, in the original model of Katz [Kat07], the wrapper functionality may choose and a fix a random tape for M before the start of a protocol execution.[3])

3.3.1.2 UC-Security via UC-Puzzles

In order to obtain our positive result, we build on the recent work of Lin, Pass and Venkitasubramaniam [LPV09] which puts forward a unified framework for designing UC secure protocols from known setup assumptions like CRS [CF01, CLOS02], tamper-proof hardware tokens [Kat07], key registration [BCNP04], etc. As observed by Lin et al., it is implicit from prior works (see e.g. [CLOS02]) that the task of constructing UC-secure protocols for any well-formed functionality [CLOS02] reduces to the task of constructing a "concurrent simulation-sound" zero knowledge protocol (ssZK) with "UC simulation" property[4].[5] Very informally, these properties can be described as follows (the text is taken almost verbatim from [LPV09]):

UC simulation: For every PPT adversary $\mathcal{A}$ receiving "honest" proofs of statements x using witness w, where (x, w) are chosen by the environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ (that only gets statements x as input) such that no $\mathcal{Z}$ can distinguish (except with negligible probability) whether it is interacting with $\mathcal{A}$ or $\mathcal{S}$.

Concurrent simulation-soundness: An adversary who receives an unbounded

[3]Note that in this case, if a token user were allowed leakage queries, then it would be able to leak on the entire random tape of M at the start of the protocol execution. We do not consider such a model in this thesis.

[4]Formally, this can be modeled as implementing a specific "zero knowledge proof of membership" functionality.

[5]Intuitively, this is because given a functionality f, we can start with a semi-honest secure computation protocol Π for f, and then "compile" Π with an ssZK protocol to obtain a UC-secure protocol against active adversaries.

Functionality $\mathcal{G}^{\ell}_{\mathbf{wrap}}$

$\mathcal{G}^{\ell}_{wrap}$ is parameterized by a polynomial p, a leakage parameter ℓ and an implicit security parameter k.

Creation. Upon receiving $(\texttt{create}, sid, P, P', M, n)$ from P, where P' is another user in the system and M is an interactive Turing machine, do:

- Send $(\texttt{create}, sid, P, P')$ to P'.
- If there is no tuple of the form $(P, P', \star, \star, \star, \star, \star)$ stored, then store $(P, P', M, n, 0, \emptyset, \ell)$.

Execution. Upon receiving $(\texttt{run}, sid, P, \mathsf{msg})$ from P', find the unique stored tuple $(P, P', M, n, i, state, \delta)$ (if no such tuple exists, then do nothing). Then, choose random $r \leftarrow \{0,1\}^{p(k)}$. Run $M(\mathsf{msg}; r; state)$ for at most p(k) steps, and let out be the response (set $\mathsf{out} = \perp$ if M does not respond in the allotted time). Send (sid, P, out) to P', and:

Case 1 ($\mathbf{i < n - 1}$): Store $(P, P', M, n, i + 1, (\mathsf{msg} \| r \| state), \delta)$ and erase $(P, P', M, n, i, state, \delta)$.

Case 2 ($\mathbf{i = n - 1}$): Store $(P, P', M, n, 0; \emptyset, \delta)$ and erase $(P, P', M, n, i, state, \delta)$.

Leakage. Upon receiving $(\texttt{leak}, sid, P, f)$, find the unique stored tuple $(P, P', M, n, i, state, \delta)$ (if no such tuple exists, then do nothing). If $|f| > \delta$, then do nothing. Otherwise, do:

- Compute $z = f(M \| state)$ and send (sid, P, z) to P'.
- Store $(P, P', M, n, i, state, \delta - |f|)$ and erase $(P, P', M, n, i, state, \delta)$.

Figure 3.4: The new wrapper functionality $\mathcal{G}^{\ell}_{wrap}$ that allows ℓ bits of leakage.

number of concurrent *simulated* proofs, of statements chosen by $\mathcal{Z}$, cannot prove any false statements (except with negligible probability).

Lin et al. consider a modular approach towards constructing an ssZK protocol. They observe that a general technique for realizing the "UC simulation" property is to have the simulator obtain a "trapdoor" which is hard to compute for the adversary. This is formalized in the form of (two party) "UC-puzzle" protocols that enable the simulator to obtain such a trapdoor string (but prevent the adversary from doing so), as described below.

UC-puzzle. Let $\mathcal{G}$ denote a setup functionality. A UC-puzzle is a pair $(\langle S, R\rangle, \mathcal{R})$, where $\langle S, R\rangle$ is a protocol between two parties—a sender S, and a receiver R—in the $\mathcal{G}$-hybrid model and $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ is an associated PPT computable relation. A UC-puzzle must satisfy the following two properties.

Soundness No PPT adversarial receiver R^* after an execution with an honest sender S can find (except with negligible probability) a trapdoor $\sigma \in \mathcal{R}(\texttt{trans})$, where `trans` is the transcript of the puzzle execution.

Statistical Simulatability Let $\mathcal{A}$ be a real world adversary (in an environment $\mathcal{Z}$) that participates as a sender in multiple concurrent executions of a UC-puzzle. Then, for every such $\mathcal{A}$, there exists a simulator $\mathcal{S}$ interacting only with $\mathcal{Z}$ such that no (possibly unbounded) $\mathcal{Z}$ can distinguish between an execution with $\mathcal{A}$ from an execution with $\mathcal{S}$, except with negligible probability. Further, for every completed puzzle execution, except with negligible probability, $\mathcal{S}$ outputs a trapdoor $\sigma \in \mathcal{R}(\texttt{trans})$, where `trans` is the transcript of that puzzle execution.

Now that we have a means for "UC simulation", in order to achieve "simulation-soundness", Lin et al define and construct a strongly non-malleable witness indistinguishable (SNMWI) argument of knowledge from one way functions. Lin

et al. then give a construction for an ssZK protocol from a UC-puzzle and an SNMWI protocol.

We note that following the work of [LPV09], the task of constructing UC secure protocols from any setup assumption reduces to the task of constructing a UC-puzzle (in the hybrid model of the corresponding setup). We obtain our positive result by following the same route, i.e., constructing a UC-puzzle in the leaky token model. We in fact construct a "family of UC-puzzles" in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model. More details follow in the next subsection.

3.3.1.3 Our Protocol

Recall that in the hardware token model, each pair of parties in the system exchange hardware tokens with each other. Now consider a system with m parties $P_1, \ldots, P_m$. For each pair of parties (P_i, P_j), we will construct two different UC-puzzles, (a) one where P_i (resp., P_j) acts as the puzzle sender (resp., receiver) and (b) the other where the roles of P_i and P_j are reversed. This gives us a family of m^2 UC-puzzles.

Now, given such a family of UC-puzzles, we can construct a family of ssZK protocols where the protocols in the family are concurrent simulation-sound with respect to each other. Specifically, for each pair of parties (P_i, P_j), we can construct two different ssZK protocols, (a) one where P_i (resp., P_j) acts as the prover (resp., verifier), and (b) the other, where the roles of P_i and P_j are reversed. Finally, in order to construct a UC-secure protocol for any well-formed functionality f, we can start with a semi-honest protocol Π for f, and then "compile" Π with the above family of ssZK protocols in the following manner. Whenever a party P_i sends a protocol message to P_j, it proves that it has "behaved honestly so far in the protocol" by running an execution of the "appropriate" ssZK protocol (i.e.,

where P_i and P_j play the roles of the prover and verifier respectively) from the above family.

We now give the construction of a family of UC-puzzles in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model. Specifically, we construct a family of protocol and relation pairs denoted as $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$, where $i, j \in [m]$. Here the choice of notation is to highlight that party P_i (resp., P_j) plays the role of the sender (resp., receiver) in protocol $\langle S_{ij}, R_{ij}\rangle$. We will then prove that each pair $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$ is a UC-puzzle in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model.

Our construction of a UC-puzzle in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model is very similar to that of Lin et al [LPV09] (in the $\mathcal{G}_{wrap}$-hybrid model). Specifically, instead of using a standard witness-hiding proof of knowledge protocol, we use a λ-leakage-resilient zero knowledge proof of knowledge (LR-ZKPOK) protocol (see Section 3.1.2). Further, instead of using an ordinary one-way function, we use an ℓ'-leakage-resilient hard relation, as defined by Dodis, Haralambiev, Lopez-Alt, Wichs [DHLW10b], for $\ell' = \lambda \cdot \ell$. We refer the reader to Section 2.3.3 for a discussion on leakage-resilient hard relations. We now proceed to describe our construction.

Description of $\langle \mathbf{S_{ij}}, \mathbf{R_{ij}}\rangle$. The interactive Turing machine S_{ij}, when invoked with the inputs the identity of the sender P_i, the identity of the receiver P_j and the session id sid, proceeds as follows. It first checks whether this is the first time interacting with party P_j. If so, it first samples a pair (x, y) from an ℓ'-leakage resilient hard relation $\mathcal{R}_{\ell'}$ and then "creates" and "gives" P_j a token, which encapsulates the interactive Turing machine M that gives a λ-LR-ZKPOK of the statement that there exists an x such that $(x, y) \in \mathcal{R}_{\ell'}$. In order to "give" the token to P_j, S_{ij} sends the message $(\texttt{create}, sid, P_i, P_j, M, n)$ to $\mathcal{G}^{\ell}_{wrap}$,

where n denotes the round-complexity of our λ-LR-ZKPOK protocol. To actually challenge P_j, S_{ij} simply sends y as the puzzle to the receiver.

The interactive Turing machine R_{ij}, on receiving y from S_{ij}, engages in an execution of our λ-LR-ZKPOK protocol with M (via $\mathcal{G}^{\ell}_{wrap}$) where M proves that there exists an x such that $(x, y) \in \mathcal{R}_{\ell'}$. More specifically, in order to send a protocol message msg to M, R_{ij} sends $(\mathtt{run}, sid, P_i, \mathsf{msg})$ to $\mathcal{G}^{\ell}_{wrap}$. An adversarial receiver R_{ij} may additionally send leakage queries $(\mathtt{leak}, sid, P, f)$ to $\mathcal{G}^{\ell}_{wrap}$, who responds with $f(M\|r)$ (where r denotes the random coins used by M "so far") as long as the total leakage (over all queries) is bounded by ℓ.

Description of $\mathcal{R}_{\mathbf{ij}}$. The puzzle relation $\mathcal{R}_{ij}$ is simply $\{(x, y)|(x, y) \in \mathcal{R}_{\ell'}\}$.

This completes the description of $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$. We now prove that the pair $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$ is a UC-puzzle in the $\mathcal{G}^{\ell}_{wrap}$-hybrid model. To this end, we first argue that it satisfies the $\mathsf{Soundness}$ property.

$(\langle \mathbf{S_{ij}}, \mathbf{R_{ij}}\rangle, \mathcal{R}_{\mathbf{ij}})$ satisfies Soundness. The $\mathsf{Soundness}$ property follows from the following hybrid argument:

$\mathcal{H}_{\mathbf{0}}$: This hybrid corresponds to the real execution between S_{ij} and R^*_{ij} as described above. $\mathcal{G}^{\ell}_{wrap}$ answers any leakage query from R^*_{ij} as long as the total leakage is bounded by ℓ. Let p_0 denote the probability that R^*_{ij} outputs a trapdoor $x \in \mathcal{R}(y)$ in this experiment.

$\mathcal{H}_{\mathbf{1}}$: This hybrid is the same as $\mathcal{H}_0$, except that we replace the honest execution of the λ-LR-ZKPOK between the token and R^*_{ij} (via $\mathcal{G}^{\ell}_{wrap}$) with a simulated execution. Specifically, we run the simulator for our LR-ZKPOK protocol that provides a simulated proof[6] to R^*_{ij}. The leakage queries made by R^*_{ij}

[6]Note that simulation of our LR-ZKPOK involves rewinding of the adversary which is not

are answered by the simulator in the following manner. On receiving a leakage query f from R^*_{ij}, the simulator prepares a query f' to the leakage oracle in the same manner as described in Section 3.1.2), except for the following change. The function f' now has the code of the honest prover algorithm for our λ-LR-ZKPOK hardwired in it; f' internally computes the machine code M (using the above information) in order to compute leakage on M. Here, the leakage oracle is implemented by the puzzle sender. Note that by definition (of λ-leakage resilient zero knowledge), the simulator (and therefore in turn, R^*_{ij}) obtains at most $\lambda \cdot \ell$ bits of leakage. Let p_1 denote the probability that R^*_{ij} outputs a trapdoor $x \in \mathcal{R}(y)$ (where y is the puzzle) in $\mathcal{H}_1$.

Now, note that it follows from the λ-leakage resilient zero knowledge property of our LR-ZKPOK that the views of R^*_{ij} in $\mathcal{H}_0$ and $\mathcal{H}_1$ are computationally indistinguishable. Therefore, we have that $|p_1 - p_0| \leq \text{negl}(k)$ (where k is the security parameter).

$\mathcal{H}_2$: This hybrid is the same as $\mathcal{H}_1$, except that the puzzle y is taken from an external party who samples $(x, y) \in \mathcal{R}_{\ell'}$. The leakage queries from the simulator are forwarded to the external party and the responses are sent back to the simulator. Let p_2 denote the probability that R^* outputs a trapdoor $x \in \mathcal{R}(y)$ in $\mathcal{H}_2$.

Note that the views of R^*_{ij} in $\mathcal{H}_1$ and $\mathcal{H}_2$ are identical. Therefore, we have that $p_1 = p_2$. Now, observe that in $\mathcal{H}_2$, R^*_{ij} obtains no information on x

allowed in the UC framework. However, we stress that the rewinding is performed here only for a "soundness" argument, which can be done outside the UC framework. Elaborating further, we note that the ZK proof being given is independent of everything else in the system (except, of course, the instance of the hard relation). Therefore, we can think of the proof in isolation of the rest of the system. Now, in this setting the adversary (or, more generally the whole environment) can be used to break the zero knowledge property of our protocol in the stand-alone setting. We note that this idea has been used in several previous works, see [BS05, CGS08].

apart from $\lambda \cdot \ell$ bits of leakage. Then, since $\mathcal{R}_{\ell'}$ is an ℓ'-leakage resilient hard relation (where $\ell' = \lambda \cdot \ell$), it follows that p_2 must be negligible (in the security parameter). Finally, since $|p_2 - p_0| \leq \text{negl}(k)$, we have that $p_0 \leq \text{negl}(k)$.

We now argue that $(\langle S_{ij}, R_{ij} \rangle, \mathcal{R}_{ij})$ satisfies the Statistical Simulatability property.

$(\langle \mathbf{S_{ij}}, \mathbf{R_{ij}} \rangle, \mathcal{R}_{\mathbf{ij}})$ satisfies Statistical Simulation. The proof for Statistical Simulatability property follows exactly as in [LPV09]. We recall the argument here for completeness. The text below is taken almost verbatim from [LPV09].

To simulate a concurrent puzzle execution with an adversarial sender $\mathcal{A}$ and the environment $\mathcal{Z}$, $\mathcal{S}$ internally emulates each execution with $\mathcal{A}$ and acts as the wrapper functionality $\mathcal{G}_{wrap}^{\ell}$ for $\mathcal{A}$. Whenever $\mathcal{A}$ sends a message of the form $(\texttt{create}, sid, P_i, P_j, M^*)$ to $\mathcal{G}_{wrap}^{\ell}$, $\mathcal{S}$ obtains the message. Later, to extract the trapdoor of a puzzle y challenged by $\mathcal{A}$ (controlling P_i) to P_j, $\mathcal{S}$ simply rewinds M^* in the LR-ZKPOK protocol to extract the witness. Note that since M^* cannot receive messages from other parties except P_j, it would never expect any new messages from parties other than P_j during rewindings. Therefore, the extraction can be finished in isolation without intervening the adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Hence we achieve perfect simulation.

Leakage Parameter ℓ. As discussed in Section 2.3.3, assuming one-way functions, it is possible to construct ℓ-leakage-resilient hard relations in the bounded leakage model for optimal value of ℓ, namely, $\ell = (1-o(1))\xi$, where ξ is the length of the secret (i.e., the witness for an instance of the hard relation). (See Section 2.3.3 for more details.) Combining this with our result on $(1+\epsilon)$-LR-ZKPOK (where ϵ is a positive constant) in section 3.1.2, we have that $\ell = \frac{(1-o(1))\xi}{1+\epsilon}$.

Family of ssZK protocols. We note that given the family of UC-puzzles $(\langle S_{ij}, R_{ij}\rangle, \mathcal{R}_{ij})$, the construction of a family of ssZK protocols easily follows from the techniques as described in [LPV09]. We refer the reader to [LPV09] for more details.

3.3.2 Fully Leakage-Resilient Signatures

In this section, we give a generic constructions of fully leakage-resilient (FLR) signature schemes by building on our notion of leakage-resilient NIZKs.

In order to discuss our approach, we first briefly recall the leakage-resilient signature scheme in [DHLW10b] (which in turn is based on the construction of [KV09]). Dodis et al. gave a generic construction of a leakage-resilient signature scheme in the bounded-leakage model from a leakage-resilient hard relation, and a tag-based *true simulation-extractable* (tSE) NIZK argument system. Very roughly, a tSE-NIZK system guarantees the existence of an extractor algorithm that can extract the witness for a NIZK proof output by an adversary that has oracle access to simulated proofs of true statements under tags of his choice (the tag used in the proof output by the adversary must be different from the tags used in the simulated proofs). We note that the approach of Dodis et al. is quite general, in that if we use a hard relation that is secure in the continual-leakage (CTL) model (as opposed to only the bounded-leakage model), the resultant signature scheme is also secure in the CTL model. Indeed, this is the approach followed in [DHLW10a].

In order to construct FLR signatures (that allow leakage on the entire state as opposed to only the secret key), we extend our notion of leakage-resilient NIZK to incorporate true simulation-extractability. Then, given a *true simulation-extractable leakage-resilient* (tSE-LR) NIZK argument system, we note that the

construction of [DHLW10b] (resp., [DHLW10a]) can be easily modified to obtain FLR signatures in the bounded-leakage model (resp., CTL model). Finally, we note that a tSE-LR-NIZK argument system is implicit from the UC-secure NIZK of [GOS06].

The rest of this section is organized as follows. We first define the notion of true simulation-extractable leakage-resilient NIZK and give a construction for the same in Section 3.3.2.1. Next, we present our construction of an FLR signature scheme in the bounded-leakage model in Section 3.3.2.2. Finally, in Section 3.3.2.3, we briefly discuss fully leakage-resilient signatures in the CTL model.

3.3.2.1 True Simulation-Extractable Leakage-Resilient NIZK

In this section we define tag-based tSE-LR-NIZK system and give a construction for the same. Our definition can be seen as an extension of the notion of tSE-NIZKs, as defined in [DHLW10b]. Very roughly, tSE-LR-NIZK extends the notion of tSE-NIZK by allowing the adversary to obtain (in addition to simulated proofs) leakage on the witness and randomness used to generate the simulated proofs. We note that our original definition of LR-NIZK (c.f. Definition 9) does not include tags, but we stress that it can be easily extended to do so.

Definition 10 (True simulation-extractability) *Let (K, P, V) be a leakage-resilient NIZK system for a relation $\mathcal{R}$ with a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ and a leakage oracle $L_w^k(\cdot)$. We say that (K, P, V) is* true simulation-extractable *with tags if there exists a* PPT *extractor algorithm $\mathcal{E}$ such that for all adversaries $\mathcal{A}$, we have* $\Pr[\mathcal{A} \text{ wins}] \leq \text{negl}(k)$ *in the following experiment:*

1. $(\sigma, \tau) \leftarrow \mathcal{S}_1(1^k)$.
2. $(x^*, \mathtt{tag}^*, \pi^*) \leftarrow \mathcal{A}^{SR^{L_w^k(\cdot)}(\sigma,\tau,\cdot,\cdot,\cdot,\cdot)}$, *where* $SR^{L(\cdot)}(\sigma, \tau, x, w, \mathtt{tag}, f)$ *computes*

$r \leftarrow \{0,1\}^{\ell_S(k)}$; $\pi \leftarrow \mathcal{S}_2(\sigma, \tau, x, \mathtt{tag}; r)$; $f' \leftarrow \mathcal{S}_3(\sigma, \tau, x, r, f)$; $y \leftarrow L_w^k(f')$ *and returns* (π, y) *(or* `fail` *if* $x \notin \mathcal{L}$*). Note that* $\mathcal{A}$ *can query* $SR^{L(\cdot)}$ *multiple times in an adaptive manner.*

3. $w^* \leftarrow \mathcal{E}(\sigma, \tau, x^*, \mathtt{tag}^*, \pi^*)$.

4. $\mathcal{A}$ *wins if:*

   - *the pair* $(x^*, \mathtt{tag}^*)$ *was not part of a simulator query, and*
   - $V(\sigma, x^*, \mathtt{tag}^*, \pi^*) = 1$*, and*
   - $\mathcal{R}(x^*, w^*) = 0$.

Our Construction. A tag based tSE-LR-NIZK argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ follows directly from the UC-secure NIZK constructed by Groth, Ostrovsky and Sahai [GOS06]. In fact it is relatively easier to construct tSE-LR-NIZK (as opposed to obtaining UC-security). In our construction, we can use tags directly while this was not feasible in the construction of UC-NIZK in [GOS06]. Also, unlike their construction, here we do not consider perfect security. For the sake of completeness, we give the complete construction and proof here. A large part of the construction and the proof has been taken verbatim from [GOS06].

We will use a non-interactive zero knowledge argument system (K, P, V) with honest prover state reconstruction (cf. Definition 4 and Definition 5). Let (S_1, S_2, S_3) denote the simulator for (K, P, V). In addition, we will use a public-key cryptosystem $(K_{\text{pseudo}}, E, D)$ with pseudorandom ciphertexts, and a tag-based simulation-sound trapdoor commitment scheme denoted as the tuple $(K_{\text{tag}-\text{com}}, \text{commit}, \text{Tcom}, \text{Topen})$. We refer the reader to Section 2.3 for their formal definitions.

The issues that come up in the UC NIZK construction of Groth, Ostro-

vsky and Sahai [GOS06] also come up in our construction of true simulation-extractable leakage resilient NIZKs. The two key hurdles that come up in the construction are:

1. First, the simulator $\mathcal{S}$ has to simulate the NIZK arguments (let Π be one of them) without knowing the witness. Furthermore, given the witness $\mathcal{S}$ must be able to simulate the randomness that would explain Π. $\mathcal{S}$ needs to do this in order to answer the leakage queries.

2. The second problem is that if an adversary generates an acceptable NIZK argument Π for a statement C then $\mathcal{S}$ must use Π and output a witness w such that $C(w) = 1$.

The main idea to overcome these hurdles is to commit to the witness w and make a NIZK argument with honest prove state reconstruction such that the commitment contains a witness w such that $C(w) = 1$. The honest prover state reconstruction property of the NIZK argument helps us to simulating the leakage queries. But, this leaves us with the commitment scheme. On one hand, when $\mathcal{S}$ simulates NIZK arguments we want to make equivocal commitments that can be opened arbitrarily since $\mathcal{S}$ does not know the witness and may need to answer leakage queries. On the other hand, when $\mathcal{S}$ sees an adversarially generated NIZK proof then we want to be able to extract the witness.

We construct such a commitment scheme, just like in [GOS06], from the tools specified in the previous section in a manner related to the construction of a UC commitment by Canetti et al. [CLOS02]. We use a tag-based simulation-sound trapdoor commitment scheme to commit to each bit of w. If w has length ℓ this gives us commitments $c_1, \ldots, c_\ell$. $\mathcal{S}$ can use the trapdoor key TK to create equivocal commitments that can be opened to arbitrary bits. This enables $\mathcal{S}$ to

simulate the leakage queries made by the verifier.

We still have an extraction problem since $\mathcal{S}$ may not be able to extract a witness from tag-based commitments created by the adversary. To solve this problem we encrypt the openings of the commitments. Now $\mathcal{S}$ can extract witnesses, but we have reintroduced the problem of equivocation. In a simulated commitment there may be two different openings of a commitment c_i to respectively 0 and 1, however, if the opening is encrypted then we are stuck with one possible opening. This is where the pseudorandomness property of the cryptosystem comes in handy. $\mathcal{S}$ can simply make two ciphertexts, one containing an opening to 0 and one containing an opening to 1. Since the ciphertexts are pseudorandom, $\mathcal{S}$ can later open the ciphertext containing the desired opening and plausibly claim that the other ciphertext was chosen as a random string. To recap, the idea so far to commit to a bit b is to make a commitment c_i to this bit, and create a ciphertext $c_{i,b}$ containing an opening of c_i to b, while choosing $c_{i,1-b}$ as a random string.

The commitment scheme is once again equivocable, however, once again we must be careful that $\mathcal{S}$ can extract a message from an adversarial commitment during the simulation. We stress that this is not a problem as the adversary can not produce equivocable commitments using a tag different from the tags on which it gets commitments from $\mathcal{S}$.

The resulting protocol can be seen in Figure 3.5. We use the notation from above.

Theorem 3 *The protocol $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ described in Figure 3.5 is a true simulation-extractable leakage-resilient non-interactive zero knowledge argument system.*

Proof. Soundness and completeness of $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ follow directly from the soundness and completeness of the underlying NIZK. We are left to argue two things.

Common reference string generation:

1. $(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k)$
2. $(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k)$
3. $(\sigma, \tau) \leftarrow S_1(1^k)$
4. Return $\Sigma = (ck, pk, \sigma)$

Proof: On input (Σ, C, w) such that $C(w) = 1$ do

1. For $i = 1$ to ℓ select r_i at random and let $c_i := \text{commit}_{ck}(w_i, \texttt{tag}; r_i)$
2. For $i = 1$ to ℓ select R_{w_i} at random and set $c_{i,w_i} := E_{pk}(r_i; R_{w_i})$ and choose $c_{i,1-w_i}$ as a random string.
3. Let $c := (c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1})$
4. Create an NIZK proof π for the statement that there exists w and randomness such that c has been produced as described in steps 1,2 and 3 and $C(w) = 1$.
5. Return $\Pi = (\texttt{tag}, c, \pi)$

Verification: On input (Σ, C, Π)

1. Parse $\Pi = (\texttt{tag}, c, \pi)$
2. Verify the NIZK proof π
3. Return 1 if the check works out, else return 0.

Figure 3.5: Simulation Extractable Leakage Resilient NIZK argument $(\mathcal{K}, \mathcal{P}, \mathcal{V})$.

First we need to argue that the protocol $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is leakage resilient non-interactive zero-knowledge. Secondly, we need to argue that we can extract a witness from a valid proof generated by an adversary.

SIMULATING Σ. $\mathcal{S}$ chooses the common reference string in the following way: It selects, $(ck, \text{TK}) \leftarrow K_{\text{tag}-\text{com}}(1^k)$; $(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k)$ and $(\sigma, \tau) \leftarrow S_1(1^k)$. It

sets the CRS as $\Sigma := (ck, pk, \sigma)$. This means $\mathcal{S}$ is able to create and equivocate simulation-sound trapdoor commitments, decrypt pseudorandom ciphertexts and simulate NIZK proofs and later upon learning a witness simulate convincing randomness used for generating the proof.

Simulating Proofs. $\mathcal{S}$ needs to simulate a proof that there exists w such that $C(w) = 1$, however, it may not use w. $\mathcal{S}$ uses $\mathtt{tag}$ specified by $\mathcal{A}$ and forms ℓ equivocal commitments $(c_i, \mathrm{EK}_i) \leftarrow \mathrm{Tcom}_{\mathrm{TK}}(\mathtt{tag})$. $\mathcal{S}$ then simulates openings of the c_i's to both 0 and 1. For all $i = 1$ to ℓ and $b = 0$ to 1 it computes $\rho_{i,b} \leftarrow \mathrm{Topen}_{\mathrm{EK}_i}(c_i, b, \mathtt{tag})$. It selects $r_{i,b}$ at random and sets $c_{i,b} := E_{pk}(\rho_{i,b}; r_{i,b})$. $\mathcal{S}$ sets $c := (c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1})$. Let x be the statement that there exists a witness w and randomness such that c has been correctly generated using w and $C(w) = 1$. $\mathcal{S}$ chooses randomness ρ and simulates the NIZK proof for x being true as $\pi \leftarrow S_2(\sigma, \tau, x; \rho)$. Let $\Pi = (\mathtt{tag}, c, \pi)$ and return it as the simulated proof.

Simulating Leakage. For any simulated proof Π generated by $\mathcal{S}$ it might need to answer a leakage query on the witness and randomness used to generate the proof Π. For this the simulator has access to a leakage oracle $L(\cdot)$.

We now describe how given the witness $\mathcal{S}$ can simulate the randomness that would lead P_i to produce such an proof Π. Since $\mathcal{S}$ created $c_i, c_{i,0}, c_{i,1}$ such that $c_{i,0}$ contains a 0-opening of c_i and $c_{i,1}$ contains a 1-opening of c_i it can produce good looking randomness to claim that the party committed to w_i. This gives us convincing randomness for constructing all these commitments and for producing the ciphertext c. $\mathcal{S}$ can now run the simulator algorithm S_3 to simulate randomness that would lead the prover to have produced the proof π. Hence any leakage query made on the witness and the randomness can be reduced to a leakage query made just on the witness and the simulator can use the leakage

oracle to answer that query.

EXTRACTION. For an adversarially generated valid proof Π, $\mathcal{S}$ must extract a witness w. $\mathcal{S}$ parses c as $c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1}$. Since $\mathcal{S}$ knows the decryption key DK, it can then decrypt all $c_{i,b}$. This gives $\mathcal{S}$ plaintexts $\rho_{i,b}$. It checks for each i whether $c_i = \text{commit}_{ck}(b, \texttt{tag}; \rho_{i,b})$ and in that case b is a possible candidate for the i-th bit of w.

If successful in all of this, $\mathcal{S}$ lets w be these bits. However, if any of the bits are ambiguous, *i.e.*, w_i could be both 0 and 1, or if any of them are inextractable, then $\mathcal{S}$ outputs `fail`.

We will later argue that the probability of the NIZK argument Π being valid, yet $\mathcal{S}$ not being able to extract a witness is negligible.

HYBRIDS. We wish to argue that no PPT adversarial verifier $\mathcal{A}$ can distinguish between its interaction with a real prover and its interaction with the simulator $\mathcal{S}$. In order to do so we define several hybrid experiments and show that $\mathcal{A}$ cannot distinguish between any of them. Then we argue that our simulator [7] $\mathcal{S}$ can in fact also extract the witness from a valid proof generated by $\mathcal{A}$. We will now give the full description of the hybrid experiments and the security proof.

H1: This is real interaction between adversary $\mathcal{A}$ and $\mathcal{S}$. $\mathcal{S}$ obtains the witness for every theorem it proves. It use the witness in an honest way and answers leakage queries honestly as well.

H2: We modify H1 in the way $\mathcal{S}$ creates tag-based simulation-sound trapdoor commitments $c_1, \ldots, c_\ell$ to the bits of the witness. Let `tag` be the tag specified by the adversary. Instead of creating c_i by selecting r_i at random and setting $c_i = \text{commit}_{ck}(w_i, \texttt{tag}; r_i)$, we create an equivocal com-

[7] Note that our $\mathcal{S}$ is playing the role of the extractor $\mathcal{E}$ as well.

mitment $(c_i, \mathrm{EK}_i) \leftarrow \mathrm{Tcom}_{\mathrm{TK}}(\mathtt{tag})$ and subsequently produce randomness $\rho_{i,w_i} \leftarrow \mathrm{Topen}_{\mathrm{EK}_i}(c_i, w_i, \mathtt{tag})$. We continue the proof using ρ_{i,w_i} instead of r_i.

H1 and H2 are indistinguishable because it is hard to distinguish tag-based commitments and their openings from tag-based equivocal commitments and their equivocations to the same messages (Equation (2.2)).

H3: In H3, we make another modification to the procedure followed by $\mathcal{S}$. We are already creating c_i as an equivocal commitment and equivocating it with randomness ρ_{i,w_i} that would open it to contain w_i. We run the equivocation procedure once more to also create convincing randomness that would explain c_i as a commitment to $1 - w_i$. This means, we compute $\rho_{i,1-w_i} \leftarrow \mathrm{Topen}_{\mathrm{EK}_i}(c_i, 1 - w_i, \mathtt{tag})$. Instead of selecting $c_{i,1-w_i}$ as a random string, we choose to encrypt $\rho_{i,1-w_i}$ as $c_{i,1-w_i} = E_{pk}(\rho_{i,1-w_i}; r_{i,1-w_i})$ for a randomly chosen $r_{i,1-w_i}$. We still pretend that $c_{i,1-w_i}$ is a randomly chosen string when we carry out the NIZK proof π or when the leakage queries need to be answered.

H2 and H3 are indistinguishable because of the pseudorandomness property of the cryptosystem, see Equation (2.1). Suppose we could distinguish H2 and H3, then we could distinguish between an encryption oracle and an oracle that supplies randomly chosen strings.

H4: Instead of making NIZK arguments using honest prover strategy we use the zero-knowledge with honest prover state reconstruction simulators. We use $\pi \leftarrow S_2(\sigma, \tau, \cdot; \rho)$ with ρ random to simulate the honest provers' NIZK arguments that c has been correctly generated. Finally, on input the witness we can use $r \leftarrow S_3(\sigma, \tau, x, \pi, \cdot, \rho)$ to create convincing randomness that would make the prover output π on the witness for c being correctly generated.

So any leakage query on the randomness and the witness can be reduced to a leakage query of the witness alone.

The zero-knowledge with honest prover state reconstruction property of the NIZK proof implies that H3 and H4 are indistinguishable.

Simulation. Note that the simulator $\mathcal{S}$ in hybrid H4 already simulates the view of the adversary $\mathcal{A}$ in a way that is indistinguishable from its view while interacting with honest prover. This concludes the proof that the simulator $\mathcal{S}$ correctly simulates the view of the adversary. Now we need to argue that if the adversary $\mathcal{A}$ can in fact output a valid proof Π with a tag $\mathtt{tag}$ such that $\mathcal{S}$ never gave a proof using the tag $\mathtt{tag}$ then we can use the proof to extract a witness. Consider the following subsequent hybrids.

H5: Again, we look at the adversarially generated NIZK argument $\Pi = (\mathtt{tag}, c, \pi)$ for some C. Parse c as $c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1}$. Then we use the decryption key DK to attempt to decrypt the $c_{i,b}$'s to get $\rho_{i,b}$ such that $c_{i,b} = \mathrm{commit}_{ck}(b, \mathtt{tag}; \rho_{i,b})$. We output $\mathtt{failure}$ if we encounter a

$$c_i = \mathrm{commit}_{ck}(0, \mathtt{tag}, \rho_{i,0}) = \mathrm{commit}_{ck}(1, \mathtt{tag}, \rho_{i,1}).$$

Tag-based simulation-soundness, see Equation (2.3), of the commitment scheme implies that H4 and H5 are indistinguishable. To see this consider the tag $\mathtt{tag}$. Outputting $\mathtt{failure}$ corresponds to breaking the binding property of the commitment scheme, unless we have previously created an equivocal commitment with tag $\mathtt{tag}$. But we already ruled out that possibility.

H6: As in H5, we try to extract $\rho_{i,0}, \rho_{i,1}$'s. We output $\mathtt{failure}$ if there is an i such that we cannot decrypt either $c_{i,0}$ or $c_{i,1}$ to give us $\rho_{i,b}$ so $c_i =$

$\text{commit}_{ck}(b, \texttt{tag}; \rho_{i,b})$. We ruled out the possibility of both $\rho_{i,0}$ and $\rho_{i,1}$ being an opening of c_i in H5, so if everything is OK so far we have a uniquely defined w such that for all i we have $c_i = \text{commit}_{ck}(w_i, \texttt{tag}; \rho_{i,w_i})$. We output `failure` if $C(w) \neq 1$.

Call c well-formed if $c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1}$ are such that for all $i = 1$ to ℓ at least one of the $c_{i,0}, c_{i,1}$ will have a proper $\rho_{i,b}$ so $c_i = \text{commit}_{ck}(b, \texttt{tag}; \rho_{i,b})$, and if all of these openings are unique then the bits constitute a witness w for $C(w) = 1$. Observe, from the soundness of NIZK [8] it follows that with overwhelming probability c is well-formed and we have negligible chance of outputting `failure`. This means H5 and H6 are indistinguishable.

Extraction. Observe that $\mathcal{S}$ in H6 has already obtained a witness w corresponding to the the valid proof Π generated by the adversary $\mathcal{A}$. Our simulator can output this as its output and this concludes the proof that the NIZK argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is indeed simulation extractable. □

Remark on common random string. We note that in our scheme the CRS consists of three components. It consists of a public key of a pseudorandom encryption scheme, a public key of a tag-based simulation sound trapdoor commitment scheme and a CRS for the underlying NIZK proof (as explained earlier). We stress that actually all these components can be chosen randomly, i.e., the sampled without actually learning the associated secret parameters. As explained in [GOS06], we can construct public-key encryption with pseudorandom ciphertexts under the decisional linear assumption. A public key of such a scheme consists of three random generators of a prime order group which can be sampled

[8] Groth et. al. [GOS06] argue that it is problematic if the language about which the theorem is being proved is chosen depending on the CRS. We ignore this as this does not affect our application. However it can be noted that the same argument holds for our NIZK as well.

without the knowledge of the corresponding secret values. As noted in [MY04] tag-based simulation-sound commitment scheme can be constructed using a signature scheme and we know a number of signature schemes in which the public key can be sampled without the knowledge of the secret key. Brent's signature scheme serves as one such example in the setting of bilinear groups. Finally, as noted in [GOS06], the CRS for the underlying NIZK proof system can be chosen to be a common random string at the cost of having a proof system that is only statistically sound, which suffices in our setting. In summary, we have argued that the CRS in our scheme can be common random string.

3.3.2.2 Fully Leakage-Resilient Signatures in the Bounded Leakage Model

We first recall the definition of FLR signatures in the bounded-leakage model from [BSW11]. Some of the text below is taken verbatim from [BSW11]. Very roughly, we say that a signature scheme is fully leakage-resilient in the bounded-leakage model if it is existentially unforgeable against any PPT adversary that can obtain polynomially many signatures over messages of her choice, as well as bounded leakage information on the secret key and the randomness used by the signing algorithm throughout the lifetime of the system.[9] We define a variable `state` that is initialized to the secret key. On each signature query from the adversary, the random coins used by the signing algorithm are appended to `state`. The adversary can leak any PPT information on `state` as long as the total amount is bounded by the leakage parameter ℓ.

[9]We note that in the original definition of [BSW11], the adversary can obtain leakage even on the randomness used in the key-generation algorithm. In our main discussion, for the sake of simplicity, we do not consider this case. We stress, however, that our construction satisfies the original definition of [BSW11], as discussed later in the section.

Definition 11 (FLR security – bounded leakage model) *A signature scheme (*KeyGen, Sign, Verify*) is ℓ-fully-leakage-resilient in the bounded leakage model if for all* PPT *adversaries $\mathcal{A}$, we have that* $\Pr[\mathcal{A} \text{ wins}] \leq \text{negl}(k)$ *in the following experiment:*

1. *Compute $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k, \ell)$, and set* $\texttt{state} = sk$. *Give pk to the adversary.*

2. *Run the adversary $\mathcal{A}$ on input tuple $(1^k, pk, \ell)$. The adversary may make adaptive queries to the signing oracle and the leakage oracle, defined as follows:*

   Signing queries: *On receiving a query m_i, the signing oracle samples $r_i \leftarrow \{0,1\}^*$, and computes $\Phi_i \leftarrow \mathsf{Sign}_{sk}(m_i; r_i)$. It updates* $\texttt{state} := \texttt{state} \| r_i$ *and outputs Φ_i.*

   Leakage queries: *On receiving as input the description of a polynomial-time computable function $f_j \colon \{0,1\}^* \to \{0,1\}^{\ell_j}$, the leakage oracle outputs* $f(\texttt{state})$.

3. *At some point, $\mathcal{A}$ stops and outputs (m^*, Φ^*).*

4. *$\mathcal{A}$ wins in the experiment iff:*

   - $\mathsf{Verify}_{pk}(m^*, \Phi^*) = 1$, *and*
   - *m^* was not queried to the signing oracle, and*
   - $\sum_j \ell_j \leq \ell$.

Our Construction. We now give a generic construction of fully leakage-resilient signatures based on leakage-resilient hard relations and tSE-LR-NIZK arguments. Let $\mathcal{R}_\ell$ be an ℓ-leakage-resilient hard relation with a PPT sampling algorithm

$\text{KGEN}(\cdot)$. Let (K, P, V) be a tag-based tSE-LR-NIZK argument system for a relation $\mathcal{R}$. The signature scheme (KeyGen, Sign, Verify) is described as follows.

- $\mathsf{KeyGen}(1^k, \ell)$: Sample $(x, y) \leftarrow \text{KGEN}(1^k)$, $\sigma \leftarrow K(1^k)$. Output $sk = x$ and $pk = (\sigma, y)$.
- $\mathsf{Sign}_{sk}(m)$: Output $\Phi = \pi$, where $\pi \leftarrow P(\sigma, y, m, x)$. (Here m is the tag in the argument.)
- $\mathsf{Verify}_{pk}(m, \Phi)$: Output $V(\sigma, y, m, \Phi)$.

Theorem 4 *If $\mathcal{R}_\ell$ is an ℓ-leakage-resilient hard relation and (K, P, V) is a tag-based true simulation-extractable leakage-resilient NIZK argument system, then (KeyGen, Sign, Verify) is an ℓ-fully-leakage-resilient signature scheme in the bounded leakage model.*

Proof. Consider the following series of experiments:

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the fully-leakage-resilience experiment as described in Definition 11. Let p_0 denote the probability that $\mathcal{A}$ outputs a successful forgery in this experiment.

Hybrid $\mathcal{H}_1$. This hybrid is the same as $\mathcal{H}_0$, except for the following changes. First, during the key generation process, instead of sampling a CRS honestly, we now run the simulator of the NIZK system to generate the CRS. Further, on receiving a query m_i from $\mathcal{A}$, instead of giving an honestly generated NIZK argument to $\mathcal{A}$, the signing oracle works as follows. It runs the simulator for our NIZK system with a leakage query f_i and obtains a simulated argument π_i and the description of a function f_i'. Here, the function f_i is such that it takes as input the witness and random coins of the NIZK prover algorithm (simulator in this case) and simply outputs all the random coins. Further, f_i' is the function

output by the simulator that takes as input only the witness and produces the same output as f_i (c.f. Definition 9). The signing oracle outputs $\Phi_i = \pi_i$ and gives f'_i (as private input) to the leakage oracle.

The leakage oracle on receiving a leakage query f_j from $\mathcal{A}$ works as follows. Let $f'_1, \ldots, f'_i$ denote the list of functions that the leakage oracle has received from the signing oracle so far. Then, the leakage oracle first prepares a function f^*_j that takes as input only the secret key sk (which is the witness for each proof generated by the simulator above), described as follows. The function f^*_j on input the secret key sk first computes $r_1 \leftarrow f'_1(sk)$, $\ldots$, $r_i \leftarrow f'_i(sk)$ to generate $\mathtt{state} = sk \| r_1 \| \ldots \| r_i$ and then outputs $f_j(\mathtt{state})$.

Let p_1 denote the probability that $\mathcal{A}$ outputs a successful forgery in this experiment. Now, it follows from the leakage-resilient zero knowledge property of (K, P, V) that the views of $\mathcal{A}$ in $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable. Then, we have that $|p_1 - p_0| \leq \text{negl}(k)$.

Hybrid $\mathcal{H}_2$. This hybrid is the same as $\mathcal{H}_1$, except that the public key component y is now taken from an external party P who samples a pair $(x, y) \leftarrow \text{KGEN}(1^k)$ such that $(x, y) \in \mathcal{R}_\ell$. Further, instead of computing the response a leakage query f_j on its own, the leakage oracle now prepares a query f^*_j (as in the previous hybrid) and forwards it to the external party P. The response from P is sent back to $\mathcal{A}$.

Let p_2 denote the probability that $\mathcal{A}$ outputs a successful forgery in this experiment. Now, note that the views of $\mathcal{A}$ in $\mathcal{H}_1$ and $\mathcal{H}_2$ are identical. Then, we have that $p_2 = p_1$.

Now, let (m^*, Φ^*) denote the forgery output by $\mathcal{A}$. We now run the extractor for the simulation-extractable leakage-resilient on input the CRS, the CRS trapdoor, $\mathtt{tag}^* = m^*$, and $\pi^* = \Phi^*$ to obtain a witness $x^* = w^*$. It follows

from the simulation-extractability of our NIZK argument system that x^* is such that $(x^*, y) \in \mathcal{R}_\ell$, except with negligible probability. That is, we have obtained a pre-image of y with probability $p = p_2 - \text{negl}(k)$. Then, it follows from the ℓ-leakage resilience of $\mathcal{R}_\ell$ that $p \leq \text{negl}(k)$. Combining this with above, we have that $p_0 \leq \text{negl}(k)$. This concludes the proof. □

Leakage parameter ℓ. As discussed in Section 2.3.3, assuming one-way functions, it is possible to construct ℓ-leakage-resilient hard relations in the bounded leakage model for optimal value of ℓ, namely, $\ell = (1-o(1))\xi$, where ξ is the length of the secret (i.e., the witness for an instance of the hard relation). (We refer the reader to Section 2.3.3 for more details.) Then, instantiating our signature scheme with such a hard relation, we have that $\ell = (1 - o(1))\xi$, where ξ is the length of the secret key.

Leakage during Key Generation. We note that the signature scheme described above can in fact tolerate leakage during the key generation algorithm (thus satisfying the original definition of Boyle et al [BSW11]) if it is possible to sample CRS for the tSE-LR-NIZK argument system in an *oblivious* manner (i.e., without first computing a trapdoor string). Note that this is possible if the CRS is a common *random* string. As we discussed in the previous subsection, our construction of tSE-LR-NIZK argument system indeed satisfies this property.

3.3.2.3 Fully Leakage-Resilient Signatures in the Continual Leakage Model

In Section 3.3.2.2, we considered FLR signature schemes in the bounded-leakage model, where the adversary is allowed to obtain only some bounded leakage on

the secret state during the entire lifetime of the system. A more realistic model is the *continual-leakage model* (CTL), first studied by Dodis et al. [DHLW10a] and Brakerski et al [BKKV10]. We briefly recall the CTL model in the context of FLR signature schemes [BSW11, MTVY11]. Very roughly, in this model, the adversary is allowed to leak continuously from the secret state, with no bound on the total leakage obtained during the lifetime of the system. However, there are two restrictions: First, it is assumed that the a user can "refresh" (or update) the secret key regularly, and that the total leakage between two successive updates is bounded. Second, there is no leakage during the update process.[10] As in the bounded-leakage model, a variable `state` is considered that is initialized to the secret key, and is constantly updated with the randomness used by the signing algorithm. However, at the end of an update, `state` is set to the updated secret key (such that no leakage is possible on the old secret state). We refer the reader to [BSW11] and [MTVY11] for a detailed definition of a fully leakage-resilient signature scheme in the continual leakage model.

We now briefly discuss how to extend our construction of FLR signature scheme from Section 3.3.2.2 to the CTL model. We note that if we substitute the leakage-resilient hard relation in our previous construction with a *continual leakage-resilient hard relation* [DHLW10a], we immediately obtain a FLR signature scheme in the CTL model. An alternative way of looking at this is as follows. If we substitute the tSE-NIZK used in the construction of a (standard) leakage-resilient signature scheme in the CTL model in [DHLW10a] with our true simulation-extractable leakage-resilient NIZK, we immediately obtain

[10] As observed in [BKKV10], and by Waters (noted in [DHLW10a]), there is general technique that can be used to tolerate up to logarithmic bits of leakage during the update process. More recently, Lewko et al [LLW11] give a construction for FLR signature scheme and an encryption scheme that tolerates constant fraction of leakage during the update process. We note that if we use the key pairs of the encryption scheme of [LLW11] as a hard relation, then our construction of FLR signatures will inherit the leakage bounds of their encryption scheme.

an FLR signature scheme in the CTL model. The construction and proof details easily follow from Section 3.3.2.2 and [DHLW10a] and are therefore omitted.

3.3.2.4 Security in the Noisy Leakage Model

We note that FLR signature schemes in the bounded leakage model (as well as the CTL model) were given only very recently (in the standard model) by Malkin et al. [MTVY11] and Boyle et al.[BSW11]. However, these schemes are not secure in the *noisy leakage model*, formalized by Naor and Segev [NS09]. Noisy leakage is a realistic generalization of bounded leakage, in which the leakage is not necessarily of bounded length, and it is only guaranteed that the secret key still has some min-entropy even given the leakage. We note that our signature scheme, when instantiated with a hard relation secure in the noisy leakage model, is also secure in this model.

At a high level, constructions of reductions, from adversaries breaking unforgeability of known FLR signature schemes [MTVY11, BSW11] to underlying hard problems, rely on partitioning the message space into two parts - the *first* on which the reduction can generate signatures and the *second* on which it can not. These reductions break the underlying hard problem when all the adversary's signature queries come from the first partition while the forgery comes from the second partition. Further the signatures generated by the reduction on messages of first partition do not information theoretically fix the secret key. Therefore leakage of a signature from this partition would allow an adversary to break unforgeability without severely reducing the entropy of the secret key. Because of these reasons, the above scheme are not secure in the noisy leakage model.

On the other hand, in our scheme every signature information theoretically fixes the secret key. However, in the proof, a reduction can not answer the

adversary's signature queries with these signatures that information theoretically fix the secret key. Our reduction solves this problem by providing "simulated signatures" instead which do not fix the secret key information theoretically, yet are computationally indistinguishable from the "real signatures." This allows us to achieve security in the noisy leakage model.

3.4 Leakage-Soundness and Simultaneous Leakage-Resilient Zero Knowledge

3.4.1 Leakage-Sound Interactive Proofs

We now consider the opposite scenario where a malicious prover can obtain arbitrary leakage on the random coins of the verifier during the protocol execution. The question that we wish to investigate is whether it is possible to construct interactive proofs that remain sound even in such a scenario. Towards that goal, (as done previously) we model P and V as interactive turing machines that have the ability to flip coins during the protocol execution. At any point during the protocol execution, a malicious prover P^* may send a leakage query f (where $f(\cdot)$ is an arbitrary PPT length-decreasing function, described as a circuit) to the verifier. An honest verifier V, on receiving such a leakage query, computes f on her random coins used thus far in the protocol (i.e., the prover cannot leak on the future random coins of the verifier) and returns the output to the prover. In order to bound the leakage obtained by a cheating prover, we consider a leakage parameter ℓ and require that $|f(\cdot)| \leq \ell$ for every leakage query $f(\cdot)$.The prover may make any arbitrary polynomial number of leakage queries during the protocol execution, as long as the *total* leakage size is bounded by ℓ.

Informally speaking, we say that an interactive proof system is *leakage-sound*

if it satisfies the soundness property even with respect to a cheating prover that can obtain leakage on the random coins of the verifier.

Definition 12 (Leakage-sound Interactive Proofs) *An interactive proof system $\langle P, V\rangle$ for a language $\mathcal{L}$ is said to be ℓ-*leakage-sound interactive proof system *if for every $x \notin \mathcal{L}$, and every interactive Turing machine P^* that makes any arbitrary polynomial number of leakage queries on the verifier's random coins (thus far in the protocol execution; in the manner as described above) such that the total leakage size is bounded by ℓ, the following holds:*

$$\Pr[\langle P^*, V\rangle(x) = 1] \leq \mathrm{negl}(|x|)$$

If the soundness condition in the above definition is valid only against PPT Turing machines, then we say that $\langle P, V\rangle$ is a leakage-sound interactive *argument* system. We note that any public coin interactive proof system is already leakage-sound for any arbitrary amount of leakage from the verifier.

3.4.2 Simultaneous Leakage-Resilient Zero Knowledge

We finally consider the scenario where a cheating prover can obtain leakage on the random coins of an honest verifier while at the same time, a cheating verifier can obtain leakage on the honest prover's witness and random coins. We wish to investigate whether it is possible to construct an interactive proof system that *simultaneously* satisfies the two notions of leakage-soundness (c.f. Definition 12) and leakage-resilient zero knowledge (c.f. Definition 8). We call such an interactive proof system *simultaneous leakage-resilient zero knowledge*, as stated below formally.

Definition 13 (Simultaneous Leakage-resilient Zero Knowledge Proofs) *An interactive proof system $\langle P, V\rangle$ for a language $\mathcal{L}$ is said to be ℓ-*simultaneous

leakage-resilient zero knowledge proof system *if it is ℓ-leakage-sound as per Definition 12 and leakage-resilient zero knowledge as per Definition 8.*

We note that our protocol presented in Figure 3.1 is already *simultaneous leakage-resilient zero knowledge*. In order to argue leakage soundness we start by observing that the commitment from the verifier to the prover is statistically hiding. At a high level this means that the commitment provided by the verifier can be opened to any value, and therefore leakage on the committed value and the randomness used in generating the commitment can be reduced to a leakage query (running possibly in unbounded time) on the message alone. In our protocol, the verifier provides a commitment to its challenge string. Therefore given the leakage, as long as at least $\omega(\log k)$ bit of entropy remains in the challenge string, soundness will be preserved. Finally, in order to achieve ℓ-leakage-soundness we will need to consider $\ell+\omega(\log k)$ repetitions of the Blum's protocol in our protocol presented in Figure 3.1.

Finally we note that, our construction of leakage resilient NIZKs (In Section 3.2) is *simultaneous leakage-resilient zero knowledge* for any arbitrary amount of leakage from the verifier. This follows trivially from the fact that in our construction of NIZKs the verifier is deterministic, and is not involved in any interaction.

3.5 Impossibility Results

3.5.1 Impossibility of LR-ZK for $\lambda < 1$

Theorem 5 *There exists a language $\mathcal{L}$ such that there exists no interactive proof system $\langle P, V\rangle$ that is λ-*leakage-resilient zero knowledge *where $\lambda < 1$.*

Proof Sketch. Consider a very simple language $\mathcal{L}$ that consists of every string $x \in \{0,1\}^*$. The witness relation $\mathcal{R}$ associated with $\mathcal{L}$ consists of pairs (x, w) such that for a given instance x, every string $w \in \{0,1\}^{|x|}$ is a witness. In this setting we will construct an adversarial verifier V^* and a distinguisher D such that D, that gets the prover's witness as auxiliary input, can distinguish between $\text{view}_{V^*}(x, z)$ and $\mathcal{S}^{L_w^{k,\lambda}(\cdot)}(x, z)$ with non-negligible probability.

Consider the scenario where for a given instance x, the prover's witness w is sampled uniformly at random among all possible witnesses. Consider a V^* that works as follows. It makes a leakage query that leaks the whole witness.[11] Finally V^* outputs the leaked witness as part of its view. The construction of D is straight forward. D gets the prover's witness w as auxiliary input. It outputs 1 if the view of V^* contains w and 0 otherwise.

It is easy to see that there's no way that the simulator can output the correct witness every time when at most $\lambda \cdot |w|$ bits of leakage are available to it. An easy way to argue this is as follows: fix the random coins of the simulator, but keep the randomization over the witnesses. Then note that the simulator will get the witness wrong at least $1/2$ the time, for any fixed random tape. Therefore, averaging over his random tapes, the simulator must still get it wrong at least $1/2$ the time. Hence, the distinguisher D will be able to distinguish between $\text{view}_{V^*}(x, z)$ and $\mathcal{S}^{L_w^{k,\lambda}(\cdot)}(x, z)$ with non-negligible probability. □

Remark 1 *In the proof sketch we gave the proof for a trivial language where every string was in the language and every string was a witness. We stress that this was only done for simplicity. It is easy to construct an NP-complete*

[11] Note that leakage of the entire witness is not necessary for the proof to work. In particular, it is easy to see that the proof works even with partial leakage of the witness.

language and argue in a similar way. Finally, the impossibility holds even in case of bounded leakage as well by an analogous argument.

3.5.2 LR-ZK with Pre-Processing

In this subsection we argue that it would be difficult to construct a leakage resilient zero knowledge proof system in a setting where there is a "leakage-free" pre-processing phase prior to the actual protocol execution, but the simulator does not have any access to a leakage oracle (unlike our model). In order to establish our argument, we will assume that it is not possible for a simulator to *reverse-engineer* the leakage queries of an adversarial verifier (or in other words, it is possible for an adversarial verifier to *obfuscate* its leakage queries). Consider a language $\mathcal{L}$ and a prover $P = (P_1, P_2)$ in the protocol $\langle P, V \rangle$ that wants to prove that $x \in \mathcal{L}$. Let w denote the witness that is given to P as private input. Before the start of the actual protocol, P_1 runs a private "leakage-free" pre-processing phase on w to generate a valid witness w' for an instance $x' \in \mathcal{L}'$. The new witness w' is given as input to P_2. P_2 now interacts with the verifier and attempts to prove that $x' \in \mathcal{L}'$.

Note that in order to argue the correctness and soundness of $\langle P, V \rangle$, we will need that $x' \in \mathcal{L}'$ if and only if $x \in \mathcal{L}$. Now, since the simulator will not have access to a valid witness w, it will not have access to w' as well. However, a cheating verifier may simply make a leakage query that checks if w' is indeed a valid witness for x' and encrypts the output (under a secret key known only to the verifier). Now assuming that the simulator can not reverse engineer the leakage query, simulator will not be able to respond to the query correctly (since otherwise, we can contradict the soundness of the zero knowledge proof system).

The argument presented above makes strong unproven assumptions and might

not seem satisfactory for that reason. Nonetheless, we stress that our goal here is not to obtain a strong impossibility result in this direction but rather to highlight the fact that this direction is not promising and hence not worth pursuing in the interactive setting when strong guarantees against leakage attacks are desired.

CHAPTER 4

Concurrent Security Preliminaries

In this chapter, we discuss the preliminaries for our results on concurrently-secure password-based authenticated key exchange.

4.1 Background

Goldreich and Lindell [GL01] proposed a definition for password authenticated key exchange based on the "simulation paradigm" [Can00]. In particular, they model the problem of PAKE as a three-party functionality $\mathcal{F}$ involving honest parties P_1 and P_2 and an adversary $\mathcal{A}$. They define appropriate "ideal" and "real" models of computation, and require that any adversary in the real model can be *emulated* (in the specific sense described below) by an adversary in the ideal model. We give more details below.

Ideal Model. In the ideal model, the parties send their input passwords to a trusted party that evaluates $\mathcal{F}$; if the passwords match, then the trusted party sends a uniformly distributed session key to the parties, else it sends $\perp$. On the other hand, the adversary $\mathcal{A}$ receives no output, and in particular, no information on the password or the session key. However, $\mathcal{A}$ is allowed to control whether or not both the honest parties receive the output (since $\mathcal{A}$ possesses the ability to abort the real execution, see below). The ideal distribution is defined as the output of the honest parties along with the output of the adversary $\mathcal{A}$ resulting

from the ideal process.

Real Model. In the real model, parties engage in an execution of a real password-authenticated key exchange protocol. In this model, the adversary $\mathcal{A}$ controls the communication link between the honest parties; as such it is allowed to modify the protocol messages of the honest parties. The real distribution is defined as the output of the honest parties along with the output of the adversary $\mathcal{A}$.

Note that in the real model, $\mathcal{A}$ can attempt to impersonate an honest party by guessing its secret password p and participating in the protocol. Assuming that the passwords are chosen uniformly from a dictionary D, each guess of $\mathcal{A}$ will be correct with probability $1/|D|$. Each guess allows $\mathcal{A}$ to learn some information (whether or not a guess is correct) and since $|D|$ may be small, it is not possible to obtain a protocol that emulates an ideal world execution of F up to computational indistinguishability.

Goldreich and Lindell [GL01] formalize the above limitation in the following manner. They propose a definition where, very informally, the ideal and the real distributions must be distinguishable for any PPT machine at most with probability $\mathcal{O}(1/|D|)$. We refer the reader to [GL01] for more details.

We note that the above definition does not consider the case where the adversary may have some a priori information on the password of the honest parties participating in a session. To this end, we instead consider an improved simulation-based definition that implies the above definition, yet seems more natural and closer to the standard paradigm for defining secure computation. Looking ahead, we note that our security model is similar to the one used by Boyko et al. [BMP00]. Further, as noted in [GL06], the improved definition implies the original definition of [GL01] (we give a proof sketch later). More details

are given in the next subsection.

4.2 Our Model

We first summarize the main differences in our model with respect to [GL01]. We first note that even in the stand-alone setting, if an adversary $\mathcal{A}$ controls the communication link between two honest parties, then $\mathcal{A}$ can execute separate "left" and "right" executions with the honest parties. Therefore, these executions can be viewed as two concurrent executions where $\mathcal{A}$ is the common party. In keeping with this observation, in our model, the adversary $\mathcal{A}$ is cast as a party participating in the protocol instead of being a separate entity who controls the communication link (as in [GL01]). We stress that this modeling allows us to assume that the communication between protocol participants takes place over authenticated channels. More details follow.

Description of $\mathcal{F}$. We model the problem of password-authenticated key exchange as a two-party functionality $\mathcal{F}$ involving parties P_1 and P_2 (where either party may be adversarial). If the inputs (password from a dictionary D) of P_1 and P_2 match, then $\mathcal{F}$ sends them a uniformly distributed session key (whose length is determined by the security parameter), else it sends $\perp$.

Further, in contrast to the stand-alone setting of [GL01] (where security holds only if a single protocol session is executed on the network), we consider the more general setting of *concurrent self-composition*, where polynomially many (in the security parameter) protocols with the same password may be executed on the network in an arbitrarily interleaved manner. In this setting, an adversary $\mathcal{A}$ may corrupt several parties across all the different sessions.

To formalize the above requirements and define security, we extend the stan-

dard paradigm for defining secure computation. We define an ideal model of computation and a real model of computation, and require that any adversary in the real model can be *emulated* (in the specific sense described below) by an adversary in the ideal model. In particular, we allow the adversary in the ideal world to make a constant number of (output) queries to the trusted party for each protocol session. In the definition below, we focus only on the case where the honest parties hold the same password p. However it can be extended to the case of arbitrarily correlated passwords (or, in fact, general secure computation) in a natural way where the simulator in the ideal world might make an expected constant number of calls to the ideal functionality for every session in the real world.

We consider a static adversary that chooses whom to corrupt before execution of the protocol. Finally, we consider *computational* security only and therefore restrict our attention to adversaries running in probabilistic polynomial time. We denote computational indistinguishability by $\stackrel{c}{\equiv}$, and the security parameter by k. Let D be the dictionary of passwords.

IDEAL MODEL. In the ideal model, there is a trusted party that computes the password functionality $\mathcal{F}$ (described above) based on the inputs handed to it by the players. Let there be n parties $P_1, \ldots, P_n$ where different pairs of parties are involved in one or more sessions, such that the total number of sessions is polynomial in the security parameter k. Let $M \subset [n]$ denote the subset of corrupted parties controlled by an adversary. An execution in the ideal model with an adversary who controls the parties M proceeds as follows:

I. Inputs: The honest parties hold a fixed input which is a password p chosen from a dictionary D. The input of a corrupted party is not fixed in advance.

II. Session initiation: If a party P_i wishes to initiate a session with another party P_j, it sends a $(\mathsf{start\text{-}session}, i, j)$ message to the trusted party. On receiving a message of the form $(\mathsf{start\text{-}session}, i, j)$, the trusted party sends $(\mathsf{new\text{-}session}, i, j, \ell)$ to both P_i and P_j, where ℓ is the index of the new session.

III. Honest parties send inputs to trusted party: Upon receiving the message $(\mathsf{new\text{-}session}, i, j, k)$ from the trusted party, an honest party P_i sends its real input along with the session identifier. More specifically, P_i sets its session ℓ input $x_{i,\ell}$ to be the password p and sends $(\ell, x_{i,\ell})$ to the trusted party.

IV. Corrupted parties send inputs to trusted party: A corrupted party P_i sends a message $(\ell, x_{i,\ell})$ to the trusted party, for any $x_{i,\ell} \in D$ of its choice.

V. Trusted party sends results to adversary: For a session ℓ involving parties P_i and P_j, when the trusted party has received messages $(\ell, x_{i,\ell})$ and $(\ell, x_{j,\ell})$, it computes the output $\mathcal{F}(x_{i,\ell}, x_{j,\ell})$. If at least one of the parties is corrupted, then the trusted party sends $(\ell, \mathcal{F}(x_{i,\ell}, x_{j,\ell}))$ to the adversary[1]. On the other hand, if both P_i and P_j are honest, then the trusted party sends the output message $(\ell, \mathcal{F}(x_{i,\ell}, x_{j,\ell}))$ to them.

VI. Adversary instructs the trusted party to answer honest players: For a session ℓ involving parties P_i and P_j where exactly one party is honest, the adversary, depending on its view up to this point, may send the (output, k) message in which case the trusted party sends the most recently computed session n output $(\ell, \mathcal{F}(x_{i,\ell}, x_{j,\ell}))$ to the honest party. (Intuitively, for each

[1]Note that here, the ideal functionality does not restrict the adversary to a *fixed* constant number of queries per session. However, in our security definition, we will require that the ideal adversary only makes a constant number of queries per session.

session ℓ where exactly one party is honest, we allow the adversary to choose which one of the λ output values would be received by the honest party.)

VII. Adversary makes more queries for a session: The corrupted party P_i, depending upon its view up to this point, can send the message $(\mathsf{new\text{-}query}, \ell)$ to the trusted party. In this case, execution of session ℓ in the ideal world comes back to stage IV. P_i can then choose its next input *adaptively* (i.e., based on previous outputs).

VIII. Outputs: An honest party always outputs the value that it received from the trusted party. The adversary outputs an arbitrary (PPT computable) function of its entire view (including the view of all corrupted parties) throughout the execution of the protocol.

Let $\mathcal{S}$ be a probabilistic polynomial-time ideal-model adversary that controls the subset of corrupted parties $M \subset [n]$. Then the ideal execution of $\mathcal{F}$ (or the ideal distribution) with security parameter k, password $p \in D$ and auxiliary input z to $\mathcal{S}$ is defined as the output of the honest parties along with the output of the adversary $\mathcal{S}$ resulting from the ideal process described above. It is denoted by $\text{IDEAL}^{\mathcal{F}}_{M,\mathcal{S}}(k, p, z)$.

REAL MODEL. We now consider the real model in which a real two-party password-based key exchange protocol is executed.

Let $\mathcal{F}, P_1, \ldots, P_n, M$ be as above. Let Σ be the password-based key exchange protocol in question. Let $\mathcal{A}$ be probabilistic polynomial-time (PPT) machine such that for every $i \in M$, the adversary $\mathcal{A}$ controls the party P_i.

In the real model, a polynomial number (in the security parameter k) of sessions of Σ may be executed concurrently, where the scheduling of all messages throughout the executions is controlled by the adversary. We *do not* assume that

all the sessions have a unique session index. We assume that the communication between the parties takes place over authenticated channels[2]. An honest party follows all instructions of the prescribed protocol, while an adversarial party may behave arbitrarily. At the conclusion of the protocol, an honest party computes its output as prescribed by the protocol. Without loss of generality, we assume the adversary outputs exactly its entire view of the execution of the protocol.

The real concurrent execution of Σ (or the real distribution) with security parameter k, password $p \in D$ and auxiliary input z to $\mathcal{A}$ is defined as the output of all the honest parties along with the output of the adversary resulting from the above process. It is denoted as $\text{REAL}^{\Sigma}_{M,\mathcal{A}}(k, p, z)$.

Having defined these models, we now define what is meant by a concurrently-secure password-authenticated key exchange protocol.

Definition 1 *Let $\mathcal{F}$ and Σ be as above. Let D be the dictionary of passwords. Then protocol Σ for computing $\mathcal{F}$ is a concurrently secure password authenticated key exchange protocol if for every probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a probabilistic expected polynomial-time adversary $\mathcal{S}$ such that $\mathcal{S}$ makes a constant number of queries to the ideal functionality per session, and, for every $z \in \{0,1\}^*$, $p \in D$, $M \subset [n]$,*

$$\left\{\text{IDEAL}^{\mathcal{F}}_{M,\mathcal{S}}(k, p, z)\right\}_{k \in N} \stackrel{c}{\equiv} \left\{\text{REAL}^{\Sigma}_{M,\mathcal{A}}(k, p, z)\right\}_{k \in N}$$

Remark 1. We remark that even if the total number of sessions is such that there are sufficient number of corrupted participants to do brute-force attack and

[2]As mentioned earlier, this is a reasonable assumption since in our model, the adversary is a protocol participant instead of being a separate entity that controls the communication links (as in [GL01]).

guess the password, the two aforementioned distributions should remain indistinguishable to satisfy our definition above.

Remark 2. Note that in the setting of concurrent self composition, an adversary may be able to maul the conversation (with an honest party) of a particular session in order to successfully establish a session key with an honest party in another session *without* the knowledge of the secret password. Clearly, in order to provide any security guarantee in such a setting, it is imperative to achieve independence between various protocol sessions executing on the network. We note that this property is implicit in our security definition.

We now state our main result.

Theorem 1 (Main Result) *Assume the existence of one-way permutations and 1-out-of-2 oblivious transfer. Let $\mathcal{F}$ be the two-party PAKE functionality as described above. Then, there exists a protocol Σ that securely realizes $\mathcal{F}$ as per Definition 1.*

We prove the above theorem by constructing such a protocol Σ in section 5.1.

Finally, we note that our security definition implies the original definition of Goldreich and Lindell [GL01] (adapted to the concurrent setting). This is formally proven in the next subsection.

4.3 Implication to Goldreich-Lindell's Definition

Goldreich and Lindell define the stand-alone security of a PAKE protocol by requiring the ideal and real distributions to be at most $\mathcal{O}(1/|D|) + \mu(k)$ apart, where D is the dictionary of passwords and μ is a negligible function in the security parameter k. Further, it was noted in [GL01] that in the case of m

sequential sessions (referring to the same password), the former definition can be suitably modified to allow a distinguishing gap of $\mathcal{O}(m/|D|) + \mu(k)$ rather than $\mathcal{O}(1/|D|) + \mu(k)$. We note that this new definition works even for the case of m *concurrent* sessions (referring to the same password). We restate this definition below (assuming suitable definitions of the ideal and real models for the case when m sessions are being executed concurrently).

Definition 2 (adapted from [GL01]) *Let $\mathcal{F}$ be as above. Let D be the dictionary of passwords. A protocol Σ for password-authenticated key exchange is concurrently secure if for every probabilistic polynomial-time real model adversary A, there exists a probabilistic polynomial time ideal model adversary S such that for every password $p \in D$, and every auxiliary input z,*

$$\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}(p, z) \overset{\mathcal{O}(\frac{m}{|D|})}{\equiv} \text{REAL}_{\mathcal{A}}^{\Sigma}(p, z),$$

where $\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}(p, z)$ and $\text{REAL}_{\mathcal{A}}^{\Sigma}(p, z)$ are the output distributions in the ideal and real worlds respectively.

We stress that definition 2 is meaningful only if the adversary has no a priori information on the password. That is, the auxiliary input z in the above definition must not contain any information on the password. We now claim that definition 1 (given in section 4.2) implies definition 2, as stated below.

Lemma 3 *If a PAKE protocol is concurrently secure as per definition 1, then it is also secure as per definition 2.*

Before we give a proof of lemma 3, we first make the following observations. The definition of Goldreich and Lindell cannot be satisfied if an adversary has a priori information on the password; in particular, the real and ideal distributions

may be distinguishable with probability 1 in this case. In contrast, our definition (see definition 1) can still be realized for such an adversary (and provides meaningful guarantees even for such a case), as evident in theorem 2. Therefore, in order to prove lemma 3, we will consider weaker adversaries for our definition; in particular, we will only consider adversaries that have no a-priori information on the password[3]. We now give a proof sketch.

Proof of Lemma 3. Let Σ be a PAKE protocol that is concurrently secure as per definition 1. Let $m = \text{poly}(k)$ be the total number of sessions. Then, given a real world adversary $\mathcal{A}$ for Σ, there exists an ideal world adversary $\mathcal{S}$ such that $\mathcal{S}$ makes a constant number of queries per session, and produces an ideal distribution that is computationally indistinguishable from the real distribution. We will use $\mathcal{S}$ to construct another ideal world adversary $\mathcal{S}'$ that makes no queries and produces an ideal distribution that is $\mathcal{O}(m/|D|) + \mu(k)$ apart from the real distribution. We note that this is sufficient to prove lemma 3. We stress that here we are only considering adversaries that have no a-priori information on the password.

Description of $\mathcal{S}'$. The ideal world adversary $\mathcal{S}'$ works by running $\mathcal{S}$. Whenever $\mathcal{S}$ makes any query in the ideal world, $\mathcal{S}'$ returns $\perp$ (i.e., $\mathcal{S}'$ replies that the password is incorrect). Finally, when $\mathcal{S}$ stops and outputs a value (its view), $\mathcal{S}'$ outputs the same value.

Let λ be a constant such that $\mathcal{S}$ makes a total of $m \cdot \lambda$ queries. Let E_i denote the event that the answer to the ith query of $\mathcal{S}$ is wrong. In other words, E_i is the event that the password guessed by $\mathcal{S}$ in the ith query is correct. Then, since $\mathcal{S}$ has no prior information on the password, $\Pr[E_i] = \frac{1}{|D|}$ (where probability is

[3]Note that these are the only valid adversaries as per definition of [GL01].

over the random coins of $\mathcal{S}$). We can use the union bound to compute an upper bound on the probability that at least one of the total $m \cdot \lambda$ answers is wrong. Specifically, we have,

$$\Pr[E_1 + \ldots, +E_{m\cdot\lambda}] \leq \frac{m \cdot \lambda}{|D|}.$$

Therefore, all the answers of $\mathcal{S}'$ must be correct with probability at least $1 - \frac{m\cdot\lambda}{|D|}$.

Now, from definition 1, the ideal distribution produced by $\mathcal{S}$ must be computationally indistinguishable from the real distribution conditioned on the event that all the answers of $\mathcal{S}'$ are correct. Then, it follows that the distinguishing gap between these distributions is at most $\mathcal{O}(m/|D|) + \mu(k)$, where μ is a negligible function in the security parameter k. □

4.4 Building Blocks

We now briefly mention some of the main cryptographic primitives that we use in our construction.

4.4.1 Statistically Binding String Commitments

In our protocol, we will use a (2-round) statistically binding string commitment scheme, e.g., a parallel version of Naor's bit commitment scheme [Nao91] based on one-way functions. For simplicity of exposition, in the presentation of our results, we will actually use a non-interactive perfectly binding string commitment.[4] Such a scheme can be easily constructed based on a 1-to-1 one way function. Let

[4]It is easy to see that the construction given in Section 5.1 does not necessarily require the commitment scheme to be non-interactive, and that a standard 2-round scheme works as well. As noted above, we choose to work with non-interactive schemes only for simplicity of exposition.

$\text{COM}(\cdot)$ denote the commitment function of the string commitment scheme. For simplicity of exposition, in the sequel, we will assume that random coins are an implicit input to the commitment function.

4.4.2 Statistically Witness Indistinguishable Arguments

In our construction, we shall use a statistically witness indistinguishable (SWI) argument $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ for proving membership in any **NP** language with perfect completeness and negligible soundness error. Such a scheme can be constructed by using $\omega(\log k)$ copies of Blum's Hamiltonicity protocol [Blu87] in parallel, with the modification that the prover's commitments in the Hamiltonicity protocol are made using a statistically hiding commitment scheme [NOVY98, HHK$^+$05].

4.4.3 Semi-Honest Two Party Computation

We will also use a semi-honest two party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ that emulates the PAKE functionality $\mathcal{F}$ (as described in section 4.2) in the stand-alone setting. The existence of such a protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ follows from [Yao86, GMW87, Kil88].

4.4.4 Extractable Commitment Scheme

We will also use a challenge-response based extractable statistically-binding string commitment scheme that has been used in several prior works, most notably [PRS02, Ros04, PPS$^+$08]. In particular, we will use the commitment scheme of [PPS$^+$08] due to its specific properties (useful in our context), as discussed below.

Protocol $\langle C, R\rangle$. Let $\text{COM}(\cdot)$ denote the commitment function of a non-interactive perfectly binding string commitment scheme (as described in Section 4.4). Let k denote the security parameter. The commitment scheme $\langle C, R\rangle$ is described as follows.

COMMIT PHASE:

- To commit to a string `str`, C chooses k^2 independent random pairs of strings $\{\alpha^0_{i,j}, \alpha^1_{i,j}\}^k_{i,j=1}$ such that $\alpha^0_{i,j} \oplus \alpha^1_{i,j} = \beta$ for all $i, j \in [k]$. C commits to all these strings using COM, with fresh randomness each time. Let $B \leftarrow \text{COM}(\texttt{str})$, and $A^0_{i,j} \leftarrow \text{COM}(\alpha^0_{i,j})$, $A^1_{i,j} \leftarrow \text{COM}(\alpha^1_{i,j})$ for every $i, j \in [k]$.

For every $j \in [k]$, do the following:

- R sends a random n-bit challenge string $v_j = v_{1,j}, \ldots, v_{k,j}$.
- For every $i \in [k]$, if $v_{i,j} = 0$, C opens $A^0_{i,j}$, otherwise it opens $A^1_{i,j}$ by sending the decommitment information.

OPEN PHASE: C opens all the commitments by sending the decommitment information for each one of them. R verifies the consistency of the revealed values.

This completes the description of $\langle C, R\rangle$.

Modified Commitment Scheme $\langle C', R'\rangle$. Due to technical reasons, we will also use a minor variant, denoted $\langle C', R'\rangle$, of the above commitment scheme. Protocol $\langle C', R'\rangle$ is the same as $\langle C, R\rangle$, except that for a given receiver challenge string, the committer does not "open" the commitments, but instead simply reveals the appropriate committed values (without revealing the randomness used

to create the corresponding commitments). More specifically, in protocol $\langle C', R' \rangle$, on receiving a challenge string $v_j = v_{1,j}, \ldots, v_{k,j}$ from the receiver, the committer uses the following strategy: for every $i \in [k]$, if $v_{i,j} = 0$, C' sends $\alpha^0_{i,j}$, otherwise it sends $\alpha^1_{i,j}$ to R'. Note that C' does not reveal the decommitment values associated with the revealed shares.

When we use $\langle C', R' \rangle$ in our main construction, we will require the committer C' to prove the "correctness" of the values (i.e., the secret shares) it reveals in the last step of the commitment protocol. In fact, due to technical reasons, we will also require the the committer to prove that the commitments that it sent in the first step are "well-formed". Looking ahead, these proofs will be done via an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$. Below we formalize both these properties in the form of a *validity* condition for the commit phase.

Proving Validity of the Commit Phase. We say that commit phase between C' and R' is *valid* with respect to a value $\hat{\mathtt{str}}$ if there exist values $\{\hat{\alpha}^0_{i,j}, \hat{\alpha}^1_{i,j}\}^k_{i,j=1}$ such that:

1. For all $i, j \in [k]$, $\hat{\alpha}^0_{i,j} \oplus \hat{\alpha}^1_{i,j} = \hat{\mathtt{str}}$, and
2. Commitments B, $\{A^0_{i,j}, A^1_{i,j}\}^k_{i,j=1}$ can be decommitted to $\hat{\mathtt{str}}$, $\{\hat{\alpha}^0_{i,j}, \hat{\alpha}^1_{i,j}\}^k_{i,j=1}$ respectively.
3. For any challenge $v_j = v_{1,j}, \ldots, v_{k,j}$, let $\bar{\alpha}^{v_{1,j}}_{1,j}, \ldots, \bar{\alpha}^{v_{k,j}}_{k,j}$ denote the secret shares revealed by C in the commit phase. Then, for all $i \in [k]$, $\bar{\alpha}^{v_{i,j}}_{i,j} = \hat{\alpha}^{v_{i,j}}_{i,j}$.

We can define *validity* condition for the commitment protocol $\langle C, R \rangle$ in a similar manner.

4.4.4.1 Precise Concurrent Extraction

Now consider the scenario where multiple sessions of the commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$ are being executed concurrently between honest receivers and a cheating committer. A simulator S_{cec} for the concurrently extractable commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$ is a machine that uses a rewinding schedule to "simulate" the concurrent sessions (i.e., produce a transcript indistinguishable from the real execution) and simultaneously extract the committed value `str` (sometimes referred to as the **preamble secret**) in each session, except with negligible probability.

Simulator S_{cec}. It was shown in [PPS+08] that there exists a simulator S_{cec} that uses a "time-oblivious" rewinding strategy such that when $\langle C, R\rangle$ (and similarly $\langle C', R'\rangle$) contains k rounds, there exists a simulator S_{cec} that is able to simulate the concurrent sessions and extract the committed in each session in time that is only a *constant* multiple of the running time of the concurrent committer.

In this work, we do not focus on precision in running time of the simulator. However, we shall crucially use the precision in running time of the simulator S_{cec} in order to argue that the total number of output queries made by the simulator (that internally uses S_{cec}) of our protocol are only a *constant* per session. Below we introduce some terminology and summarize two main properties of S_{cec} for our context.

Consider polynomially many concurrent sessions of $\langle C, R\rangle$ and $\langle C', R'\rangle$ that we wish to simulate. The simulator S_{cec} produces an ordered list of "threads of execution", where a thread of execution (consisting of the views of all the parties) is a perfect simulation of a prefix of an actual execution. In particular, the *main thread*, is a perfect simulation of a complete execution, and this is the

execution thread that is output by the simulator. Any other thread is referred to as a *look-ahead thread*. Here, each thread shares a possibly empty prefix with the previous thread.

The goal of S_{cec} is, for each commitment that it comes across in any session in any thread, to extract the committed value before that commitment is concluded in that thread. We recall the following two properties of S_{cec} that are useful to our context.

Lemma 4 *(Informal statement [PPS+08]) For any concurrent adversarial committer, there exists a simulator algorithm S_{cec} such that the running time of S_{cec} is within a constant factor of T, where T is the running time of the adversarial committer.*

S_{cec} is said to "get stuck" if it fails in extracting the committed value in a session on a thread such that the commit phase of that session in that thread is concluded. The probability of S_{cec} getting "stuck" is negligible, as stated below.

Lemma 5 *(implicit in [PPS+08]) Consider a concurrent adversarial committer and a receiver running polynomially many (in the security parameter) sessions of a protocol consisting of the commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$. Then except with negligible probability, in every thread of execution output by S_{cec}; if the receiver accepts a commit phase of $\langle C, R\rangle$ or $\langle C', R'\rangle$ as valid, then at the point when that commit phase is concluded, S_{cec} would have already recorded the secret committed value of that commitment.*

We note that in our main construction, commitments sent in the commit phase of an execution of $\langle C', R'\rangle$ are never later opened via the opening phase. However, the above lemma is still applicable to $\langle C', R'\rangle$ as well as long as the proofs of

validity given along with $\langle C', R' \rangle$ are sound. In our construction, the proofs of validity will be given via executions of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$. We note that for each of these executions, the statement for $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ will have a "trapdoor condition" that will allow our simulator to cheat; however, in our security proof, we will ensure that that the trapdoor condition is *false* for each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ where the adversary plays the role of the prover. Therefore, by relying on the soundness of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$, we will still be able to use lemma 5.

4.4.5 Concurrent Non-Malleable Zero Knowledge Argument

Concurrent non-malleable zero knowledge (CNMZK) considers the setting where a man-in-the-middle adversary is interacting with several honest provers and honest verifiers in a concurrent fashion: in the "left" interactions, the adversary acts as verifier while interacting with honest provers; in the "right" interactions, the adversary tries to prove some statements to honest verifiers. The goal is to ensure that such an adversary cannot take "help" from the left interactions in order to succeed in the right interactions. This intuition can be formalized by requring the existence of a machine called the simulator-extractor that generates the view of the man-in-the-middle adversary and additionally also outputs a witness from the adversary for each "valid" proof given to the verifiers in the right sessions.

Recently, Barak, Prabhakaran and Sahai [BPS06] gave the first construction of a concurrent non-malleable zero knowledge (CNMZK) argument for every language in **NP** with perfect completeness and negligible soundness error.

In our main construction, we will use a specific CNMZK protocol, denoted $\langle P, V \rangle$, based on the CNMZK protocol of Barak et al. [BPS06] to guarantee non-malleability. Specifically, we will make the following two changes to Barak

et al's protocol: (a) Instead of using an $\omega(\log k)$-round extractable commitment scheme [PRS02], we will use the k-round extractable commitment scheme $\langle C, R\rangle$ (described above). (b) Further, we require that the non-malleable commitment scheme being used in the protocol be public-coin w.r.t. receiver[5]. We now describe the protocol $\langle P, V\rangle$.

Protocol $\langle P, V\rangle$. Let P and V denote the prover and the verifier respectively. Let L be an NP language with a witness relation R. The common input to P and V is a statement $\pi \in L$. P additionally has a private input w (witness for π). Protocol $\langle P, V\rangle$ consists of two main phases: (a) the *preamble phase*, where the verifier commits to a random secret (say) σ via an execution of $\langle C, R\rangle$ with the prover, and (b) the *post-preamble phase*, where the prover proves an NP statement. In more detail, protocol $\langle P, V\rangle$ proceeds as follows.

Preamble Phase.

1. P and V engage in the execution of $\langle C, R\rangle$ where V commits to a random string σ.

Post-preamble Phase.

2. P commits to 0 using a statistically-hiding commitment scheme. Let c be the commitment string. Additionally, P proves the knowledge of a valid decommitment to c using a statistical zero-knowledge argument of knowledge (SZKAOK).

[5]The original BPS-CNMZK construction only required a public-coin extraction phase inside the non-malleable commitment scheme. We, however, require that the entire commitment protocol be public-coin. We note that the non-malleable commitment protocol of [DDN00] only consists of standard perfectly binding commitments and zero knowledge proof of knowledge. Therefore, we can easily instantiate the DDN construction with public-coin versions of these primitives such that the resultant protocol is public-coin.

3. V now reveals σ and sends the decommitment information relevant to $\langle C, R\rangle$ that was executed in step 1.

4. P commits to the witness w using a public-coin non-malleable commitment scheme.

5. P now proves the following statement to V using SZKAOK:

   (a) *either* the value committed to in step 4 is a valid witness to π (i.e., $R(\pi, w) = 1$, where w is the committed value), *or*

   (b) the value committed to in step 2 is the trapdoor secret σ.

   P uses the witness corresponding to the first part of the statement.

Straight-line Simulation of $\langle P, V\rangle$. A nice property of protocol $\langle P, V\rangle$ is that it allows *straight-line* simulation of the prover if the trapdoor secret σ is available to the simulator S. (Note that S can run the simulator S_{cec} during the execution of $\langle C, R\rangle$ in order to extract σ from V.) Below we describe the straight-line simulation strategy for the post-preamble phase (assuming that the simulator S already knows the trapdoor secret σ).

1. S creates a statistically hiding commitment to σ (instead of a string of all zeros) and follows it with an honest execution of SZKAOK to prove knowledge of the decommitment value.

2. On receiving the decommitment information corresponding to the preamble phase, S first verifies its correctness (in the same manner as an honest prover). If the verification fails, S stops the simulation.

3. S commits to an all zeros string (instead of a valid witness to π) using the non-malleable commitment scheme.

4. S engages in the execution of SZKAOK with the adversarial verifier, where it uses the (trapdoor) witness corresponding to the *second* part of the statement. (Note that the trapdoor witness is available to S since it committed to σ in step 2 of the protocol.)

CHAPTER 5

Concurrent Password-Authenticated Key Exchange

5.1 Our Construction

In this section, we describe our concurrently secure password-authenticated key exchange protocol Π. Since our result is applicable to general functionalities as well (see Section 1.2.1), we describe our construction for a general functionality $\mathcal{F}$, without necessarily referring to the password setting.

In order to describe our construction, we first recall the notation associated with the primitives that we use in our protocol. Let $\text{COM}(\cdot)$ denote the commitment function of a non-interactive perfectly binding commitment scheme. Let $\langle C, R \rangle$ denote the k-round extractable commitment scheme and $\langle C', R' \rangle$ be its modified version as described in Section 4.4.4. Let $\langle P, V \rangle$ denote the modified version of the CNMZK argument of Barak et al. [BPS06] as described in Section 4.4.5. Further, let $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ denote a SWI argument and let $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ denote a *semi-honest* two party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ that securely computes $\mathcal{F}$ in the stand-alone setting (as per the standard definition of secure computation).

Let P_1 and P_2 be two parties with inputs x_1 and x_2. Let k be the security parameter. Protocol $\Pi = \langle P_1, P_2 \rangle$ proceeds as follows.

I. Trapdoor Creation Phase.

1. $P_1 \Rightarrow P_2$: P_1 creates a commitment $com_1 = \text{COM}(0)$ to bit 0 and sends com_1 to P_2. P_1 and P_2 now engage in the execution of $\langle P, V \rangle$ where P_1 proves that com_1 is a commitment to 0.

2. $P_2 \Rightarrow P_1$: P_2 now acts symmetrically. That is, it creates a commitment $com_2 = \text{COM}(0)$ to bit 0 and sends com_2 to P_1. P_2 and P_1 now engage in the execution of $\langle P, V \rangle$ where P_2 proves that com_2 is a commitment to 0.

Informally speaking, the purpose of this phase is to aid the simulator in obtaining a "trapdoor" to be used during the simulation of the protocol.

II. Input Commitment Phase. In this phase, the parties commit to their inputs and random coins (to be used in the next phase) via the commitment protocol $\langle C', R' \rangle$.

1. $P_1 \Rightarrow P_2$: P_1 first samples a random string r_1 (of appropriate length, to be used as P_1's randomness in the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ in Phase III) and engages in an execution of $\langle C', R' \rangle$ (denoted as $\langle C', R' \rangle_{1 \to 2}$) with P_2, where P_1 commits to $x_1 \| r_1$. Next, P_1 and P_2 engage in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ where P_1 proves the following statement to P_2: (a) *either* there exist values $\hat{x}_1$, $\hat{r}_1$ such that the commitment protocol $\langle C', R' \rangle_{1 \to 2}$ is *valid* with respect to the value $\hat{x}_1 \| \hat{r}_1$, *or* (b) com_1 is a commitment to bit 1.

2. $P_2 \Rightarrow P_1$: P_2 now acts symmetrically. Let r_2 (analogous to r_1 chosen by P_1) be the random string chosen by P_2 (to be used in the next phase).

Informally speaking, the purpose of this phase is aid the simulator in extracting the adversary's input and randomness.

III. Secure Computation Phase. In this phase, P_1 and P_2 engage in an execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ where P_1 plays the role of P_1^{sh}, while P_2 plays the role of P_2^{sh}. Since $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ is secure only against semi-honest adversaries, we first enforce that the coins of each party are truly random, and then execute $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, where with every protocol message, a party gives a proof using $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ of its honest behavior "so far" in the protocol. We now describe the steps in this phase.

1. $P_1 \leftrightarrow P_2 : P_1$ samples a random string r_2' (of appropriate length) and sends it to P_2. Similarly, P_2 samples a random string r_1' and sends it to P_1. Let $r_1'' = r_1 \oplus r_1'$ and $r_2'' = r_2 \oplus r_2'$. Now, r_1'' and r_2'' are the random coins that P_1 and P_2 will use during the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$.

2. Let t be the number of rounds in $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, where one round consists of a message from P_1^{sh} followed by a reply from P_2^{sh}. Let transcript $T_1 j$ (resp., $T_2 j$) be defined to contain all the messages exchanged between P_1^{sh} and P_2^{sh} before the point P_1^{sh} (resp., P_2^{sh}) is supposed to send a message in round j. For $j = 1, \ldots, t$:

   (a) $P_1 \Rightarrow P_2$: Compute $\Delta_{1,j} = P_1^{\mathsf{sh}}(T_1 j, x_1, r_1'')$ and send it to P_2. P_1 and P_2 now engage in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$, where P_1 proves the following statement:

      i. *either* there exist values $\hat{x}_1$, $\hat{r}_1$ such that (a) the commitment protocol $\langle C', R' \rangle_{1 \to 2}$ is *valid* with respect to the value $\hat{x}_1 \| \hat{r}_1$, and (b) $\Delta_{1,j} = P_1^{\mathsf{sh}}(T_1 j, \hat{x}_1, \hat{r}_1 \oplus r_1')$

      ii. *or*, com_1 is a commitment to bit 1.

   (b) $P_2 \Rightarrow P_1$: P_2 now acts symmetrically.

This completes the description of the protocol $\Pi = \langle P_1, P_2 \rangle$. Note that Π consists of several instances of SWI, such that the proof statement for each SWI instance consists of two parts. Specifically, the second part of the statement states that prover committed to bit 1 in the trapdoor creation phase. In the sequel, we will refer to the second part of the proof statement as the *trapdoor* condition. Further, we will call the witness corresponding to the first part of the statement as *real* witness and that corresponding to the second part of the statement as the *trapdoor* witness.

We now claim the following:

Theorem 2 *The proposed protocol* Π *is a concurrently secure PAKE protocol as per Definition 1.*

We will prove Theorem 2 in the following manner. First, in Section 5.2, we will construct a simulator $\mathcal{S}$ for protocol Π that will simulate the view of $\mathcal{A}$ in the ideal world. We will show that $\mathcal{S}$ makes only a constant number of queries per session while simulating the view of $\mathcal{A}$. Finally, we Section 5.3, we will argue that the output distributions of the real and ideal world executions are computationally indistinguishable.

5.2 Description of Simulator

Let there be n parties in the system where different pairs of parties are involved in one or more sessions of Π, such that the total number of sessions m is polynomial in the security parameter k. Let $\mathcal{A}$ be an adversary who controls an arbitrary number of parties.. For simplicity of exposition, we will assume that exactly one party is corrupted in each session. We note that if the real and ideal distributions are indistinguishable for this case, then by using standard techniques we can easily

remove this assumption.

We describe the construction of our simulator in section 5.2.1. In section 5.2.2, we will argue that the simulator makes only a constant number of queries per session while simulating the view of $\mathcal{A}$. We first fix some notation.

NOTATION. In the sequel, for any session $\ell \in [m]$, we will use the notation H to denote the honest party and $\mathcal{A}$ to denote the corrupted party. Let $\langle P, V \rangle_{H \to \mathcal{A}}$ denote an instance of $\langle P, V \rangle$ where H plays the role of the prover and $\mathcal{A}$ plays the verifier. Similarly, let $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{H \to \mathcal{A}}$ denote each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ where H and $\mathcal{A}$ plays the roles of prover P and verifier V respectively. Now, recall that H plays the role of committer C in one instance of $\langle C, R \rangle$ inside execution of $\langle P, V \rangle$, where it commits to a preamble secret (denoted σ_H), and in one instance of $\langle C', R' \rangle$, where it commits to its input x_H and randomness r_H (to be used in the secure computation phase). We will reserve the notation $\langle C, R \rangle_{H \to \mathcal{A}}$ for the former case, and we will refer to the latter case by $\langle C', R' \rangle_{H \to \mathcal{A}}$. Further, we define $\langle P, V \rangle_{\mathcal{A} \to H}$, $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}$, $\langle C, R \rangle_{\mathcal{A} \to H}$, $\langle C', R' \rangle_{\mathcal{A} \to H}$ in the same manner as above, except that the roles of H and $\mathcal{A}$ are interchanged. Also, let $x_{\mathcal{A}}$ and $r_{\mathcal{A}}$ denote the input and random coins, respectively, of $\mathcal{A}$ (to be used in the secure computation phase).

Consider any session $\ell \in [m]$ between H and $\mathcal{A}$. Consider the last message from $\mathcal{A}$ before H sends a message to $\mathcal{A}$ during the secure computation phase in session ℓ. Note that this message could either be the first message of the secure computation phase or the last message of the input commitment phase, depending upon whether $\mathcal{A}$ or H sends the first message in the secure computation phase. In the sequel, we will refer to this message from $\mathcal{A}$ as the $\mathsf{special}$ message. Intuitively, this message is important because our simulator will need to query the ideal functionality every time it receives such a message from $\mathcal{A}$. Looking

ahead, in order to bound the number of queries made by our simulator, we will be counting the number of $\mathsf{special}$ messages sent by $\mathcal{A}$ during the simulation.

5.2.1 Simulator $\mathcal{S}$

The simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{ext}}, \mathcal{S}_{\mathsf{main}})$ consists of two parts, namely, $\mathcal{S}_{\mathsf{ext}}$ and $\mathcal{S}_{\mathsf{main}}$. Informally speaking, the goal of $\mathcal{S}_{\mathsf{ext}}$ is to extract the committed value in each execution of the extractable commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$ where $\mathcal{A}$ acts as the committer. These extracted values are passed on to $\mathcal{S}_{\mathsf{main}}$, who uses them crucially to simulate the view of $\mathcal{A}$. We now give more details.

Description of $\mathcal{S}_{\mathsf{ext}}$. We first describe the strategy of $\mathcal{S}_{\mathsf{ext}}$. Roughly speaking, $\mathcal{S}_{\mathsf{ext}}$ essentially handles all communication with $\mathcal{A}$; however, in each session $\ell \in [m]$, $\mathcal{S}_{\mathsf{ext}}$ by itself only answers $\mathcal{A}$'s messages during the execution of the commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$ where $\mathcal{A}$ plays the role of the committer; $\mathcal{S}_{\mathsf{ext}}$ in turn communicates with the main simulator $\mathcal{S}_{\mathsf{main}}$ (described below) to answer all other messages from $\mathcal{A}$. We now give more details.

Let S_{cec} denote the simulator for the commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$ as described in Section 4.4.4. The machine $\mathcal{S}_{\mathsf{ext}}$ is essentially the same as the simulator S_{cec} that interacts with $\mathcal{A}$ in order to extract the committed value in each instance of $\langle C, R\rangle_{H\to\mathcal{A}}$ and $\langle C', R'\rangle_{H\to\mathcal{A}}$. Specifically, in order to perform these extractions, $\mathcal{S}_{\mathsf{ext}}$ employs the time-oblivious rewinding strategy of S_{cec} for an "imaginary" adversary, described below. During the simulation, whenever $\mathcal{S}_{\mathsf{ext}}$ receives a message from $\mathcal{A}$ in an execution of $\langle C, R\rangle_{H\to\mathcal{A}}$ or $\langle C', R'\rangle_{H\to\mathcal{A}}$, then it answers it on its own in the same manner as S_{cec} does (i.e., by sending a random challenge string). However, on receiving any other message, it simply passes it to the main simulator $\mathcal{S}_{\mathsf{main}}$ (described below), and transfers its response to $\mathcal{A}$.

Whenever $\mathcal{S}_{\mathsf{ext}}$ extracts a committed value from an execution of $\langle C, R\rangle_{H\to\mathcal{A}}$ or $\langle C', R'\rangle_{H\to\mathcal{A}}$ at any point during the simulation, it immediately passes it to $\mathcal{S}_{\mathsf{main}}$. Whenever $\mathcal{S}_{\mathsf{ext}}$ fails to extract any of the committed values from $\langle C, R\rangle_{H\to\mathcal{A}}$ or $\langle C', R'\rangle_{H\to\mathcal{A}}$, then it aborts with the special symbol $\perp$.

Message generation timings of $\mathcal{A}$. We note that in order to employ the time-oblivious rewinding strategy of S_{cec}, $\mathcal{S}_{\mathsf{ext}}$ needs to know the amount of time that $\mathcal{A}$ takes to send each message in the protocol (see [PPS+08]). We remark that we do not seek precision in simulation time (guaranteed by the rewinding strategy of S_{cec}); instead we only require that the number of queries made by the simulator in the look-ahead threads is only within a constant factor of the number of the number of sessions. To this end, we consider an imaginary experiment in which $\mathcal{A}$ takes a disproportionately large amount of time in generating the message after which our simulator has to query the trusted party. Then the rewinding strategy of $\mathcal{S}_{\mathsf{ext}}$ is determined by running S_{cec} using the next message generation timings of such an (imaginary) adversary, explained as follows.

Consider all the messages sent by $\mathcal{A}$ during a protocol execution. Assign q time units to the $\mathsf{special}$ message, where q is the round complexity (linear in the security parameter) of our protocol; any other message from $\mathcal{A}$ is simply assigned one time unit. Intuitively, by assigning more weight to the $\mathsf{special}$ message, we ensure that if the running time of our simulator is only within a constant factor of the running time of $\mathcal{A}$ in the real execution, then the number of $\mathsf{special}$ messages sent by $\mathcal{A}$ during the simulation must be a constant as well. Looking ahead, this in turn will allow us to prove that the number of queries made by the simulator are only a constant.

Description of $\mathcal{S}_{\mathsf{main}}$. We now describe the strategy of $\mathcal{S}_{\mathsf{main}}$ in each phase

of the protocol, for each session $\ell \in [m]$. We stress that $\mathcal{S}_{\mathsf{main}}$ uses the same strategy in the main-thread as well as all look-ahead threads (unless mentioned otherwise). For the sake of simplicity, below we describe the case in which the honest party sends the first message in the protocol. The other case, in which the adversary sends the first message, can be handled in an analogous manner and is omitted.

Trapdoor Creation Phase. In the first step, instead of committing to bit 0, $\mathcal{S}_{\mathsf{main}}$ sends com_1 as a commitment to bit 1. Now, recall that $\mathcal{S}_{\mathsf{ext}}$ interacts with $\mathcal{A}$ during the preamble phase in $\langle P, V\rangle_{H\to\mathcal{A}}$ and extracts the preamble secret $\sigma_{\mathcal{A}}$ from $\mathcal{A}$ at the conclusion of the preamble. Then, on receiving $\sigma_{\mathcal{A}}$ from $\mathcal{S}_{\mathsf{ext}}$, $\mathcal{S}_{\mathsf{main}}$ simulates the *post-preamble* phase of $\langle P, V\rangle_{H\to\mathcal{A}}$ in a straight-line manner (by using $\sigma_{\mathcal{A}}$); in the same manner as explained in Section 4.4.5. On the other hand, if $\mathcal{S}_{\mathsf{ext}}$ returns $\perp$, then $\mathcal{S}_{\mathsf{main}}$ executes the post-preamble phase of $\langle P, V\rangle_{H\to\mathcal{A}}$ by following the party strategy.

In the second step of this phase, $\mathcal{S}_{\mathsf{main}}$ simply uses the honest party strategy to interact with $\mathcal{A}$.

As we show later, except with negligible probability, $\mathcal{S}_{\mathsf{ext}}$ always succeeds in extracting the preamble secret $\sigma_{\mathcal{A}}$ in as long as the commitment protocol $\langle C, R\rangle_{\mathcal{A}\to H}$ is *valid* since $\langle C, R\rangle$ is a perfectly binding commitment scheme. In other words, $\mathcal{S}_{\mathsf{ext}}$ only outputs $\perp$ if the commitment protocol $\langle C, R\rangle_{\mathcal{A}\to H}$ is not *valid*. Note that in this case, when $\mathcal{S}_{\mathsf{main}}$ executes Step 3 in an honest fashion, $\mathcal{A}$ would fail with probability 1 in successfully decommitting to the preamble secret during the post-preamble phase of $\langle P, V\rangle_{H\to\mathcal{A}}$. As a consequence, $\mathcal{S}_{\mathsf{main}}$ (who is following the honest party strategy) will abort the session.

Input Commitment Phase. In this phase, $\mathcal{S}_{\mathsf{main}}$ first commits to a (sufficiently large) random string (unlike the honest party that commits to its input x_H

and randomness r_H) in the execution of the commitment protocol $\langle C', R' \rangle_{H \to \mathcal{A}}$. $\mathcal{S}_{\mathsf{main}}$ then engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{H \to \mathcal{A}}$ with $\mathcal{A}$, where (unlike the honest party that uses the real witness) $\mathcal{S}_{\mathsf{main}}$ uses the trapdoor witness. Note that the trapdoor witness is available to $\mathcal{S}_{\mathsf{main}}$ since it committed to bit 1 in the trapdoor commitment phase.

$\mathcal{S}_{\mathsf{main}}$ does not do anything in the second step of this phase. Instead, as mentioned above, $\$\mathcal{S}_{\mathsf{ext}}$ interacts with $\mathcal{A}$ and extracts the input and randomness pair $(x_{\mathcal{A}}, r_{\mathcal{A}})$ of $\mathcal{A}$ from $\langle C', R' \rangle_{\mathcal{A} \to H}$. The pair $(x_{\mathcal{A}}, r_{\mathcal{A}})$ is given to $\mathcal{S}_{\mathsf{main}}$.

Secure Computation Phase. Let S_{sh} denote the simulator for the semi-honest two-party protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ used in our construction. $\mathcal{S}_{\mathsf{main}}$ internally runs the simulator S_{sh} on adversary's input $x_{\mathcal{A}}$. S_{sh} starts executing, and, at some point, it makes a call to the trusted party in the ideal world with some input (say) $x_{\mathcal{A}}$. $\mathcal{S}_{\mathsf{main}}$ uses the following strategy to manage queries to the trusted party.

$\mathcal{S}_{\mathsf{main}}$ maintains a counter c to count the total number of queries (including all sessions) made to the trusted party on the look-ahead threads so far in the simulation (note that there will be exactly m queries on the main thread). Now, when S_{sh} makes a call to the trusted party, $\mathcal{S}_{\mathsf{main}}$ computes a session index s in the following manner. If the query corresponds to the main thread, then $\mathcal{S}_{\mathsf{main}}$ sets $s = \ell$, else it computes $s = c \mod m$. Now, if $\mathcal{S}_{\mathsf{main}}$ has already queried the trusted party at least once for session s, then it first sends the $(\mathsf{new\text{-}query}, s)$ message to the trusted party. Otherwise, it simply sends the message (s, x) to the trusted party.[1][2] The response from the trusted party is passed on to S_{sh}. If

[1] We stress that the simulator is able to "trade" the ideal functionality calls in one session for another since the inputs of the honest parties are the same across all the sessions.

[2] Note that by choosing the session index for the output query in the above fashion, $\mathcal{S}_{\mathsf{main}}$ is able to equally distribute the queries across all the sessions. Looking ahead, in the next subsection, we will argue that the total number of queries across all the sessions are only within

the query corresponds to the main thread, $\mathcal{S}_{\mathsf{main}}$ sends the message (output, s) to the trusted party, indicating it to send the output to the honest party in session s.[3]

Having received the trusted party's response from $\mathcal{S}_{\mathsf{main}}$, S_{sh} runs further, and finally halts and outputs a transcript $\Delta_{H,1}, \Delta_{\mathcal{A},1}, \ldots, \Delta_{H,t}, \Delta_{\mathcal{A},t}$ of the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, and an associated random string $\hat{r}_{\mathcal{A}}$. $\mathcal{S}_{\mathsf{ext}}$ now performs the following steps.

1. $\mathcal{S}_{\mathsf{main}}$ first computes a random string $\tilde{r}_{\mathcal{A}}$ such that $\tilde{r}_{\mathcal{A}} = r_{\mathcal{A}} \oplus \hat{r}_{\mathcal{A}}$ and sends it to $\mathcal{A}$.

2. Now, in each round $j \in [t]$, $\mathcal{S}_{\mathsf{main}}$ sends $\Delta_{H,j}$. It then engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{H \to \mathcal{A}}$ with $\mathcal{A}$ where it uses the trapdoor witness (deviating from honest party strategy that used the real witness). Next, on receiving $\mathcal{A}$'s next message $\Delta_{\mathcal{A},j}$ in the protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, $\mathcal{S}_{\mathsf{main}}$ engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}$ with $\mathcal{A}$ where it uses the honest verifier strategy. Finally at any stage, if the j^{th} message of the adversary is not $\Delta_{\mathcal{A},j}$ and the proof $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}$ given immediately after this messages is accepted, then the simulator aborts all communication and outputs $\perp$. (Later, we establish in the proof of Lemma 7 that $\mathcal{S}_{\mathsf{ext}}$ outputs $\perp$ with only negligible probability.)

This completes the description of our simulator $\mathcal{S} = \{\mathcal{S}_{\mathsf{ext}}, \mathcal{S}_{\mathsf{main}}\}$. In the next subsection, we bound the total number of queries made by $\mathcal{S}$.

a constant factor of the number of sessions. Then, this strategy of distributing the queries will ensure that the queries per session are also a constant.

[3]Note that $s = \ell$ in this case. We stress that by setting $s = \ell$ for a query on the main thread, $\mathcal{S}_{\mathsf{main}}$ ensures that the honest party in session ℓ receives the correct output. (Note that an honest party does not receive any output for an output query on a look-ahead thread.)

5.2.2 Total Queries by $\mathcal{S}$

Lemma 6 *Let m be the total number of sessions of $\Pi = (P_1, P_2)$ being executed concurrently. Then, the total number of queries made by $\mathcal{S}$ to the trusted party is within a constant factor of m.*

Proof. Let T be the total running time of the adversary in the real execution, as per the time assignment strategy described in section 5.2.1. Now, since $\mathcal{S}$ employs the time-oblivious rewinding strategy of S_{cec}, it follows from lemma 4 (see section 4.4.4) that the total running time of $\mathcal{S}$ is within a constant factor of T. Let us now assume that our claim is false, i.e., the total number of queries made by $\mathcal{S}$ is a super-constant multiple of m. We will show that in this case, the running time of $\mathcal{S}$ must be super-constant multiple of T, which is a contradiction. We now give more details.

Let q be the round complexity of Σ. Then, as per the time assignment strategy given in section 5.2.1, $T = (q-1+q)\cdot m$ (recall that the $\mathsf{special}$ message is assigned a weight of q time units, while each of the remaining $q-1$ messages is assigned one time unit). Now, let λ be a value that is super-constant in the security parameter such that $\mathcal{S}$ makes $\lambda \cdot m$ total queries during the simulation. Note that each output query corresponds to a unique $\mathsf{special}$ message. Let T' be the total running time of $\mathcal{S}$. We calculate T' as follows:

$$\begin{aligned} T' &\geq q \cdot (\lambda \cdot m) + (q-1) \cdot m \\ &> q \cdot (\lambda \cdot m) \\ &> \frac{\lambda \cdot q}{(q-1+q)} \cdot (q-1+q) \cdot m \\ &> \frac{\lambda \cdot q}{(q-1+q)} \cdot T \end{aligned}$$

Since $\frac{\lambda \cdot q}{(q-1+q)}$ is a super-constant in the security parameter, we have that T' is a

super-constant multiple of T, which is a contradiction. Hence the claim follows. □

The corollary below immediately follows from lemma 6 and the description of $\mathcal{S}$ in section 5.2.1.

Corollary 1 *$\mathcal{S}$ makes a constant number of queries per session to the trusted party.*

5.3 Indistinguishability of the Views

We consider two experiments $\mathcal{H}_0$ and $\mathcal{H}_1$, where $\mathcal{H}_0$ corresponds to the real execution of Σ while $\mathcal{H}_1$ corresponds to the ideal computation of $\mathcal{F}$, as described below.

Experiment $\mathcal{H}_0$: The simulator $\mathcal{S}$ is given the inputs of all the honest parties. By running honest programs for the honest parties, it generates their outputs along with $\mathcal{A}$'s view. This corresponds to the real execution of the protocol. The output of the hybrid corresponds to the outputs of the honest parties and the view of the adversary $\mathcal{A}$.

Experiment $\mathcal{H}_1$: $\mathcal{S}$ simulates all the sessions without the inputs of the honest parties (in the same manner as explained in the description of $\mathcal{S}$) and outputs the view of $\mathcal{A}$. Each honest party outputs the response it receives from the trusted party. Again the output of the hybrid corresponds to the outputs of the honest parties and the view of the adversary $\mathcal{A}$.

Let v^i be a random variable that represents the output of $\mathcal{H}_i$. We now claim that the output distributions of $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable, as stated below:

Lemma 7 $v^0 \stackrel{c}{\equiv} v^1$

We will prove this lemma using a carefully designed series of intermediate hybrid experiments. More details are given below.

5.3.1 Getting Started

We will prove Lemma 7 by contradiction. Suppose that the hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ are distinguishable in polynomial time, i.e., there exists a PPT distinguisher D that can distinguish between the two hybrids with a non-negligible probability. We will now consider a series of hybrid experiments $\mathcal{H}_{i:j}$, where $i \in [1, 2m]$, and $j \in [1, 6]$. We define two additionally hybrids – first, a dummy hybrid $\mathcal{H}_{0:6}$ that represents the real world execution (i.e., $\mathcal{H}_0$, as defined above), and second, an additional hybrid $\mathcal{H}_{2m+1:1}$ that corresponds to the simulated execution in the ideal world (i.e., $\mathcal{H}_1$, as defined above). For each intermediate hybrid $\mathcal{H}_{i:j}$, we define a random variable $v^{i:j}$ that represents the output (including the view of the adversary and the outputs of the honest parties) of $\mathcal{H}_{i:j}$.

Below, we will establish (via the intermediate hybrid arguments) that no polynomial time distinguisher can distinguish between $v^{0:6}$ and $v^{2m+1:1}$ with a non-negligible probability, which is a contradiction. Before we jump into description of our hybrids, we first establish some notation and terminology.

In the sequel, we will make use of the notation described in Section 5.2. In particular, whenever necessary, we will augment our notation with a super-script that denotes the session number. We now describe some additional notation that will be used in the proof.

First Message Notation. For any session $\ell \in [m]$, consider the *first* message that H sends to $\mathcal{A}$ during the *post-preamble phase* inside $\langle P, V \rangle_{H \to \mathcal{A}}$. We will refer to this message as an FM of **type I**. Further, in that session, consider the *first* message that H sends to $\mathcal{A}$ during the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ in the secure computation phase. We will refer to this message as an FM of **type II**.

Consider an ordered numbering of all the occurrences of FM (irrespective of its type) across the m sessions. Note that there may be up to $2m$ FM's in total on any execution thread. In particular, there will be exactly $2m$ FM's on the *main* thread. For any execution thread, let FM_i denote the ith FM. Let $s(i)$ be the index of the protocol session that contains FM_i. In the sequel, our discussion will mainly involve the FM's on the main thread. Therefore, we omit the reference to the main thread and unless otherwise stated, it will be implicit that the FM's in our discussion correspond to the main thread.

Soundness Condition. Looking ahead, while proving the indistinguishability of the outputs of our hybrid experiments, we will need to argue that in each session $\ell \in [m]$, the soundness property holds for $\langle P, V \rangle_{\mathcal{A} \to H}$ and that the trapdoor condition is false for each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}$. In the sequel, we will refer to this as the *soundness condition*.

Consider the CNMZK instance $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ in session ℓ. Let $\pi^{\ell}_{\mathcal{A}}$ denote the proof statement for $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$, where, informally speaking, $\pi^{\ell}_{\mathcal{A}}$ states that $\mathcal{A}$ committed to bit 0 (earlier in the trapdoor creation phase). Note that the soundness condition "holds" if we prove that in each session $\ell \in [m]$, $\mathcal{A}$ commits to a valid witness to the statement π^{ℓ} in the non-malleable commitment (NMCOM) inside $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$. To this end, we define m random variables, $\{\rho^{\ell}_{i:j}\}^{m}_{\ell=1}$, where $\rho^{\ell}_{i:j}$ is the value committed in the NMCOM inside $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ as per $v^{i:j}$.

Now, before we proceed to the description of our hybrids, we first claim that the soundness condition holds in the real execution. We will later argue that the soundness condition still holds as we move from one hybrid to another.

Lemma 8 *Let $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ and $\pi^{\ell}_{\mathcal{A}}$ be as described above corresponding to the real execution. Then, for each session $\ell \in [m]$, if the honest party does not abort the session (before the first message of the Secure Computation Phase is sent) in the view $v^{0:6}$, then $\rho^{\ell}_{0:6}$ is a valid witness to the statement $\pi^{\ell}_{\mathcal{A}}$, except with negligible probability.*

Intuitively, the above lemma immediately follows due the knowledge soundness of the statistical zero knowledge argument of knowledge used in $\langle P, V\rangle$. We refer the reader to [Claim 2.5, [BPS06]] for a detailed proof.

Public-coin property of NMCOM. We now describe a strategy that we will repeatedly use in our proofs in order to argue that for every session $\ell \in [m]$, the value contained in NMCOM inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ remains indistinguishable as we change our simulation strategy from one hybrid experiment to another. Intuitively, we will reduce our indistinguishability argument to a specific cryptographic property (that will be clear from context) that holds in a stand-alone setting. Specifically, we will consider a stand-alone machine M^* that runs $\mathcal{S}$ and $\mathcal{A}$ internally. Here we explain how for any session ℓ, M^* can "expose" the NMCOM inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ to an external party R (i.e., M^* will send the commitment messages from $\mathcal{A}$ to R and vice-versa, instead of handling them internally). Note that $\mathcal{S}$ may be rewinding $\mathcal{A}$ during the simulation. However, since R is a stand-alone receiver; M^* can use its responses only on a single thread of execution.

In order to deal with this problem, we will use the following strategy. When $\mathcal{A}$ creates the NMCOM inside $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$, any message in this NMCOM from $\mathcal{A}$ on the main-thread is forwarded externally to R; the responses from R are forwarded internally to $\mathcal{A}$ on the main-thread. On the other hand, any message in this NMCOM from $\mathcal{A}$ on a look-ahead thread is handled internally; M^* creates a response on its own and sends it internally to $\mathcal{A}$ on that look-ahead thread. We stress that this possible because NMCOM is a public-coin protocol.

In the sequel, whenever we use the above strategy, we will omit the details of the interaction between M^* and R.

5.3.2 Description of the Hybrids

For $i \in [1, 2m]$, the hybrid experiments are described as follows.

Experiment $\mathcal{H}_{i:1}$: Same as $\mathcal{H}_{i-1:6}$, except that $\mathcal{S}$ performs rewindings *upto* FM_i (as described in Section 5.2). Specifically, the rewindings are performed with the following restrictions:

- No new-look ahead threads are created beyond FM_i on the main thread (i.e., the execution is straight-line beyond FM_i).
- Consider any look-ahead thread that is created before the execution reaches FM_i on the main-thread. Then, any such look-ahead thread is terminated as soon as the execution reaches the i^{th} FM *on that thread*[4].

Additionally, $\mathcal{S}$ extracts and records the committed value in each execution of $\langle C, R \rangle_{\mathcal{A} \to H}$ and $\langle C', R' \rangle_{\mathcal{A} \to H}$ that concludes before FM_i. $\mathcal{S}$ outputs the abort

[4]Note that the FM_i's on different executions threads may not be identical, and in particular, may correspond to different sessions

symbol $\perp$ if it "gets stuck". Otherwise, it outputs the view of the adversary in the main thread of this simulation as $v^{i:1}$.

We now claim that,

$$v^{i-1:6} \stackrel{c}{\equiv} v^{i:1} \tag{5.1}$$

$$\forall \ell \quad \rho^{\ell}_{i-1:6} \stackrel{c}{\equiv} \rho^{\ell}_{i:1} \tag{5.2}$$

Hybrid $\mathcal{H}_{i-1:6:1}$. In order to prove our claim, we will first consider an intermediate hybrid experiment $\mathcal{H}_{i-1:6:1}$ where $\mathcal{S}$ employs the same strategy as described above, except that whenever it fails to extract the committed values from $\langle C, R \rangle_{\mathcal{A} \to H}$ and $\langle C', R' \rangle_{\mathcal{A} \to H}$, it does not abort, but instead continues the simulation and outputs the main thread. Now, since the main thread in this experiment remains unchanged from $\mathcal{H}_{i-1:6}$, it follows that:

$$v^{i-1:6} \stackrel{s}{\equiv} v^{i-1:6:1} \tag{5.3}$$

where $\stackrel{s}{\equiv}$ denotes statistical indistinguishability. We further claim that:

$$\forall \ell \quad \rho^{\ell}_{i-1:6} \stackrel{c}{\equiv} \rho^{\ell}_{i-1:6:1} \tag{5.4}$$

Let us assume that equation 5.4 is false. That is, $\exists \ell \in [m]$ such that $\rho^{\ell}_{i-1:6}$ and $\rho^{\ell}_{i-1:6:1}$ are distinguishable by a probabilistic polynomial time (PPT) distinguisher. In this case, we can create an unbounded adversary that extracts the value contained in the non-malleable commitment inside $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ and is then able to distinguish between the main threads in $\mathcal{H}_{i-1:6}$ and $\mathcal{H}_{i-1:6:1}$, which is a contradiction.

We now argue that in hybrid $\mathcal{H}_{i-1:6:1}$, $\mathcal{S}$ is able to extract (except with negligible probability) the committed value in each execution of $\langle C, R \rangle_{\mathcal{A} \to H}$ and $\langle C', R' \rangle_{\mathcal{A} \to H}$ that concludes before FM_i. Towards this, we first note that by construction, simulator's strategy in this experiment is identical for each thread,

irrespective of whether it is the main-thread or a look-ahead thread. Now consider an imaginary adversary who aborts once the execution reaches FM_i on any thread. Note that lemma 5 holds for such an adversary (i.e. the probability that the simulator fails to extract the committed value of a "concluded" commitment $\langle C, R\rangle$ or $\langle C', R'\rangle$ is negligible). Then, if the adversary does not abort (as is the case with $\mathcal{A}$), the probability that the simulation successfully extracts the committed values must be only higher. Hence our claim follows for case 1.

For case 2, we note that lemma 5 is applicable if we can argue that the *soundness condition* holds (specifically, we require that the trapdoor condition is false for each instance of SWI in $\langle C, R\rangle^{\ell}_{\mathcal{A}\to H}$ if $\langle C, R\rangle^{\ell}_{\mathcal{A}\to H}$ concludes before FM_i). Note that this is already implied by equation 5.4. Hence, our claim follows for case 2 as well.

Proving Equations 5.1 *and* 5.2. Note that the only difference between $\mathcal{H}_{i-1:6:1}$ and $\mathcal{H}_{i:1}$ is that $\mathcal{S}$ outputs the abort symbol $\perp$ if S_{cec} "gets stuck". We have shown that this event happens only with negligible probability. Hence our claim follows.

Experiment $\mathcal{H}_{i:2}$: Same as $\mathcal{H}_{i:1}$, except that if FM_i is of type I, then $\mathcal{S}$ simulates the post-preamble phase of $\langle P, V\rangle^{s(i)}_{H\to\mathcal{A}}$ in a *straight-line* manner, as explained in Section 4.4.5. For completeness, we recall it below. Recall that no look-ahead threads are started once the execution reaches FM_i on the main thread. Thus, all the changes in the main thread, as explained below, are performed *after* FM_i.

1. In the post-preamble phase of $\langle P, V\rangle^{s(i)}_{H\to\mathcal{A}}$, $\mathcal{S}$ first commits to $\sigma^{s(i)}_{\mathcal{A}}$ (instead of a string of all zeros) using the statistically hiding commitment scheme SCOM and follows it up with an honest execution of SZKAOK to prove

knowledge of the decommitment.

2. Next, after receiving the decommitment to the preamble phase of $\langle P, V \rangle^{s(i)}_{H \to \mathcal{A}}$, $\mathcal{S}$ commits to an all zeros string (instead of a valid witness to $\pi^{s(i)}_H$) using the the non-malleable commitment scheme NMCOM.

3. Finally, $\mathcal{S}$ proves the following statement using SZKAOK: (a) either the value committed to in SCOM earlier is a valid witness to $\pi^{s(i)}_H$, or (b) the value committed to in SCOM earlier is $\sigma^{s(i)}_{\mathcal{A}}$. Here it uses the witness corresponding to the *second* part of the statement. Note that this witness is available to $\mathcal{S}$ since it already performed step 1 above. Below, we will refer to this witness as the *trapdoor* witness, while the witness corresponding to the first part of the statement will be referred to as the *real* witness.

Now we prove that,

$$v^{i:1} \stackrel{c}{\equiv} v^{i:2} \tag{5.5}$$

$$\forall \ell \quad \rho^{\ell}_{i:1} \stackrel{c}{\equiv} \rho^{\ell}_{i:2} \tag{5.6}$$

In order to prove the above equations, we will create three intermediate hybrids $\mathcal{H}_{i:1:1}$, $\mathcal{H}_{i:1:2}$, and $\mathcal{H}_{i:1:3}$. Hybrid $\mathcal{H}_{i:1:1}$ is identical to $\mathcal{H}_{i:1}$, except that it changes its strategy to perform step 1 (as described above). Hybrid $\mathcal{H}_{i:1:2}$ is identical to $\mathcal{H}_{i:1:1}$, except that it changes its strategy to perform step 3. Finally, hybrid $\mathcal{H}_{i:1:3}$ is identical to $\mathcal{H}_{i:1:2}$, except that it changes its strategy to perform step 2. Note that $\mathcal{H}_{i:1:3}$ is identical to $\mathcal{H}_{i:2}$.

We now claim the following:

$$v^{i:1} \stackrel{c}{\equiv} v^{i:1:1} \tag{5.7}$$

$$\forall \ell \quad \rho^{\ell}_{i:1} \stackrel{c}{\equiv} \rho^{\ell}_{i:1:1} \tag{5.8}$$

$$v^{i:1:1} \stackrel{c}{\equiv} v^{i:1:2} \tag{5.9}$$

$$\forall \ell \quad \rho^{\ell}_{i:1:1} \stackrel{c}{\equiv} \rho^{\ell}_{i:1:2} \tag{5.10}$$

$$v^{i:1:2} \stackrel{c}{\equiv} v^{i:1:3} \tag{5.11}$$

$$\forall \ell \quad \rho^{\ell}_{i:1:2} \stackrel{c}{\equiv} \rho^{\ell}_{i:1:3} \tag{5.12}$$

Note that equation 5.5 follows by combining the results of equations 5.7, 5.9, and 5.11. Similarly, equation 5.6 follows by combining the results of equations 5.8, 5.10, and 5.12. We now prove the above set of equations.

Let $\pi_H^{s(i)}$ denote the proof statement in $\langle P, V \rangle^{s(i)}_{H \to \mathcal{A}}$. Let $\sigma_{\mathcal{A}}^{s(i)}$ denote the preamble secret committed by the $\mathcal{A}$ in the preamble phase of $\langle P, V \rangle^{s(i)}_{H \to \mathcal{A}}$ that $\mathcal{S}$ has already extracted.

Proving Equations 5.7 and 5.8. We first note that SCOM and SZKAOK can together be viewed as a statistically hiding commitment scheme. Let $\overline{\text{SCOM}}$ denote this new commitment scheme. Then, equation 5.7 simply follows from the hiding property of $\overline{\text{SCOM}}$.

In order to prove equation 5.8, let us first assume that the claim is false, i.e., $\exists \ell \in [m]$ such that $\rho^{\ell}_{i:1}$ and $\rho^{\ell}_{i:1:1}$ are distinguishable by a PPT distinguisher D. We will create a standalone machine M^* that is identical to $\mathcal{H}_{i:1}$, except that instead of simply committing to a string of all zeros using $\overline{\text{SCOM}}$ in $\langle P, V \rangle^{s(i)}_{H \to \mathcal{A}}$, M^* takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$. Additionally, M^* "exposes" the NMCOM in $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Let us describe the interaction between M^* and C in more detail. M^* first

sends the preamble secret $\sigma_{\mathcal{A}}^{m\text{BPS}}s(i)$ to C. Now, when C starts the execution of $\overline{\text{SCOM}}$ in $\langle P, V\rangle_{H\to\mathcal{A}}^{s(i)}$, M^* forwards the messages from C to $\mathcal{A}$; the responses from $\mathcal{A}$ are forwarded externally to C. Note that if C commits to a string of all zeros in the $\overline{\text{SCOM}}$ execution, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:1:1}$. On the other hand, if C commits to the preamble secret $\sigma_{\mathcal{A}}^{m\text{BPS}}s(i)$, then the (C, M^*, R) system is equivalent to $\mathcal{H}_{i:1:2}$. We will now construct a computationally unbounded distinguisher D' that distinguishes between these two executions, thus contradicting the statistically hiding property of $\overline{\text{SCOM}}$. D' simply extracts the value inside the NMCOM received by R and runs D on this input. D' outputs whatever D outputs. By our assumption, D's output must be different in these two experiments; this implies that D' output is different as well, which is a contradiction.

Proving Equations 5.9 and 5.10. Equation 5.9 simply follows due to the witness indistinguishability property of SZKAOK. Equation 5.10 follows from the fact that SZKAOK is *statistically* witness indistinguishable. The proof details are almost identical to the proof of equation 5.8 and therefore omitted.

Proving Equations 5.11 and 5.12. Equation 5.11 simply follows from the hiding property of NMCOM. To see this, we can construct a standalone machine M that internally runs $\mathcal{S}$ and $\mathcal{A}$ and outputs the view generated by $\mathcal{S}$. M is identical to $\mathcal{H}_{i:1:2}$ except that in phase IV of $\langle P, V\rangle_{H\to\mathcal{A}}^{s(i)}$, instead of simply committing (using NMCOM) to a valid witness (to the proof statement $y^{s(i)}$), it takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$.

In order to prove equation 5.12, we will use the non-malleability property of NMCOM. Let us assume that equation 5.12 is false, i.e., $\exists \ell \in [m]$ such that $\rho_{i:1:2}^{\ell}$ and $\rho_{i:1:3}^{\ell}$ are distinguishable by a PPT machine. We will construct a standalone machine M^* that is identical to the machine M described above, except that

it will "expose" the non-malleable commitment inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Now, if E commits to the witness to y^{ℓ}, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:1:2}$, whereas if E commits to a random string, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:1:3}$. From the non-malleability property of NMCOM, we establish that the value committed by M^* to R must be computationally indistinguishable in both cases.

Experiment $\mathcal{H}_{i:3}$: Same as $\mathcal{H}_{i:2}$, except that if FM_i is of type I, then the simulator commits to bit 1 instead of 0 in phase I of session $s(i)$. Let $\Pi^{s(i)}_{\mathrm{COM},H\to\mathcal{A}}$ denote this commitment.

We now claim that,

$$v^{i:2} \stackrel{c}{\equiv} v^{i:3} \tag{5.13}$$

$$\forall \ell \quad \rho^{\ell}_{i:2} \stackrel{c}{\equiv} \rho^{\ell}_{i:3} \tag{5.14}$$

Proving Equations 5.13 and 5.14. Equation 5.13 simply follows from the (computationally) hiding property of the commitment scheme COM.

In order to prove equation 5.14, we will leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in $\langle P, V\rangle$. Let us first assume that equation 5.14 is false, i.e., $\exists \ell \in [m]$ such that $\rho^{\ell}_{i:2}$ and $\rho^{\ell}_{i:3}$ are distinguishable by a PPT distinguisher. Note that it cannot be the case that the NMCOM inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ concludes *before* $\mathcal{S}$ sends the non-interactive commitment $\Pi^{s(i)}_{\mathrm{COM},H\to\mathcal{A}}$ in session $s(i)$, since in this case, the execution of NMCOM is independent of $\Pi^{s(i)}_{\mathrm{COM},H\to\mathcal{A}}$. Now consider the case when the NMCOM inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ concludes *after* $\mathcal{S}$ sends $\Pi^{s(i)}_{\mathrm{COM},H\to\mathcal{A}}$.

We will create a standalone machine M^* that is identical to $\mathcal{H}_{i:2}$, except that

instead of committing to bit 0 in $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$, it takes this commitment from an external sender C and forwards it internally to $\mathcal{A}$. Additionally, it "exposes" the NMCOM inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Note that if C commits to bit 0 then the (C, M^*, R) system is identical to $\mathcal{H}_{i:2}$, otherwise it is identical to $\mathcal{H}_{i:3}$. Now, recall that NMCOM is an extractable commitment scheme. Therefore, we now run the extractor (say) E of NMCOM on (C, M^*) system. Note that E will rewind M^*, which in turn may rewind the interaction between C and M^*. However, since COM is a non-interactive commitment scheme, M^* simply re-sends the commitment string received from C to $\mathcal{A}$ internally. Now, if the extracted values are different when C commits to bit 0 as compared to when it commits to bit 1, then we can break the (computationally) hiding property of COM, which is a contradiction.

Experiment $\mathcal{H}_{i:4}$: Same as $\mathcal{H}_{i:3}$, except that if FM_i is of type I, then $\mathcal{S}$ uses the following modified strategy. In session $s(i)$, $\mathcal{S}$ uses the trapdoor witness (instead of the real witness) in each instance of SWI where the honest party plays the role of the prover. Note that the trapdoor witness for each of these SWI must be available to the simulator at this point since it earlier committed to bit 1 in phase I of session $s(i)$.

We now claim that,

$$v^{i:3} \stackrel{c}{\equiv} v^{i:4} \tag{5.15}$$

$$\forall \ell \quad \rho^{\ell}_{i:3} \stackrel{c}{\equiv} \rho^{\ell}_{i:4} \tag{5.16}$$

Proving Equations 5.15 and 5.16. Equation 5.15 simply follows from the witness indistinguishability of SWI by a standard hybrid argument.

In order to prove equation 5.16, let us first consider the simpler case where $\mathcal{S}$ uses the trapdoor witness only in the *first* instance (in the order of execution) of SWI in session $s(i)$ where the honest party plays the role of the prover. In this case, we can leverage the "statistical" nature of the witness indistinguishability property of SWI in a similar manner as in the proof of equation 5.10. Then, by a standard hybrid argument, we can extend this proof for multiple SWI.

Experiment $\mathcal{H}_{i:5}$: Same as $\mathcal{H}_{i:4}$, except that if FM_i is of type I, then $\mathcal{S}$ uses the following strategy in the execution of $\langle C', R' \rangle^{s(i)}_{H \to \mathcal{A}}$ in session $s(i)$. Recall that $\langle C', R' \rangle^{x}_{H \to \mathcal{A}}$ denotes the instance of $\langle C', R' \rangle$ in session $s(i)$ where the honest party commits to its input x_H and randomness r_H (to be used in the secure computation phase).

1. Instead of honest commitments to $x_H \| r_H$ and its secret shares, $\mathcal{S}$ sends commitments to random strings as the first message.

2. On receiving any challenge string from $\mathcal{A}$, instead of honestly revealing the values committed to in the commit phase (as per the challenge string), $\mathcal{S}$ sends random strings to $\mathcal{A}$.

We now claim that,

$$v^{i:4} \stackrel{c}{\equiv} v^{i:5} \tag{5.17}$$

$$\forall \ell \quad \rho^{\ell}_{i:4} \stackrel{c}{\equiv} \rho^{\ell}_{i:5} \tag{5.18}$$

In order to prove these equations, we will define two intermediate hybrids $\mathcal{H}_{i:4:1}$ and $\mathcal{H}_{i:4:2}$. Experiment $\mathcal{H}_{i:4:1}$ is the same as $\mathcal{H}_{i:4}$, except that $\mathcal{S}$ also performs steps 1 as described above. Experiment $\mathcal{H}_{i:4:2}$ is the same as $\mathcal{H}_{i:4:1}$, except that $\mathcal{S}$ also performs step 2 as described above. Therefore, by definition, $\mathcal{H}_{i:4:2}$ is identical to $\mathcal{H}_{i:5}$.

We now claim the following:

$$v^{i:4} \stackrel{c}{\equiv} v^{i:4:1} \tag{5.19}$$

$$\forall \ell \quad \rho^{\ell}_{i:4} \stackrel{c}{\equiv} \rho^{\ell}_{i:4:1} \tag{5.20}$$

$$v^{i:4:1} \stackrel{c}{\equiv} v^{i:4:2} \tag{5.21}$$

$$\forall \ell \quad \rho^{\ell}_{i:4:1} \stackrel{c}{\equiv} \rho^{\ell}_{i:4:2} \tag{5.22}$$

Note that equation 5.17 follows by combining the results of equations 5.19 and 5.21. Similarly, equation eq:b45 follows by combining the results of equations 5.20 and 5.22. We now prove the above set of equations.

Proving Equations 5.19 and 5.20. Equation 5.19 simply follows from the (computational) hiding property of the commitment scheme COM.

In order to prove equation 5.20, let us first consider the simpler case where $\mathcal{S}$ only modifies the first commitment in the commit phase in $\langle C, R \rangle^{s(i)}_{H \to \mathcal{A}}$. In this case, we can leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in $\langle P, V \rangle$ in a similar manner as in the proof of equation 5.14. Then, by a standard hybrid argument, we can extend this proof to the case where $\mathcal{S}$ modifies all the commitments in the commit phase in $\langle C, R \rangle^{s(i)}_{H \to \mathcal{A}}$.

Proving Equations 5.21 and 5.22. Note that the main-thread is identical in hybrids $\mathcal{H}_{i:4:1}$ and $\mathcal{H}_{i:4:2}$ since we are only changing some random strings to other random strings; furthermore, the strings being changed are not used elsewhere in the protocol. Equations 5.21 and 5.22 follow as a consequence.

Experiment $\mathcal{H}_{i:6}$: Same as $\mathcal{H}_{i:5}$, except that if FM_i is of type II, $\mathcal{S}$ "simulates" the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ in session $s(i)$, in the following manner. Let S_{sh} be the simulator for the semi-honest two party protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ used in our

construction. $\mathcal{S}$ internally runs the simulator S_{sh} for the semi-honest two party protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ on $\mathcal{A}$'s input in session $s(i)$ that was extracted earlier. When S_{sh} makes a query to the trusted party with some input, $\mathcal{S}$ selects a session index s' and forwards the query to the trusted party in the same manner as explained earlier in Section 5.2.1. The response from the trusted party is passed on to S_{sh}. Further, $\mathcal{S}$ decides whether the output must be sent to the honest party in the same manner as explained earlier. S_{sh} finally halts and outputs a transcript of the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, and an associated random string for the adversary.

Now, $\mathcal{S}$ forces this transcript and randomness on $\mathcal{A}$ in the same manner as described in section 5.2.1. We claim that during the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, each reply of $\mathcal{A}$ must be consistent with this transcript, except with negligible probability. Note that we have already established from the previous hybrids that the *soundness condition* holds (except with negligible probability) at this point. This means that the trapdoor condition is false for each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle^{s(i)}_{\mathcal{A} \to H}$. Then our claim follows from the soundness property of SWI used in our construction.

We now claim that:

$$v^{i:5} \stackrel{c}{\equiv} v^{i:6} \tag{5.23}$$

$$\forall \ell \quad \rho^{\ell}_{i:5} \stackrel{c}{\equiv} \rho^{\ell}_{i:6} \tag{5.24}$$

Proving Equation 5.23. Informally speaking, equation 5.23 follows from the semi-honest security of the two-party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ used in our construction. We now give more details.

We will construct a standalone machine M that is identical to $\mathcal{H}_{i:5}$, except that instead of engaging in an honest execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ with $\mathcal{A}$ in session $s(i)$, it obtains a protocol transcript from an external sender C and forces it on

$\mathcal{A}$ in the following manner. M first queries the ideal world trusted party on the extracted input of $\mathcal{A}$ for session $s(i)$ in the same manner as explained above for $\mathcal{S}$. Let $x_{\mathcal{A}}^{s(i)}$ denote the extracted input of $\mathcal{A}$. Let $x_H^{s(i)}$ denote the input of the honest party in session $s(i)$. Let O be the output that M receives from the trusted party. Now M sends $x_H^{s(i)}$ along with $x_{\mathcal{A}}^{s(i)}$ and O to C and receives from C a transcript for $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ and an associated random string. M forces this transcript and randomness on $\mathcal{A}$ in the same manner as $\mathcal{S}$ does. Now, the following two cases are possible:

1. C computed the transcript and randomness by using *both* the inputs - $x_H^{s(i)}$ and $x_{\mathcal{A}}^{s(i)}$ - along with the output O. In this case, the transcript output by C is a real transcript of an honest execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$.

2. C computed the transcript and randomness by using only adversary's input $x_{\mathcal{A}}^{s(i)}$, and the output O. In this case C simply ran the simulator S_{sh} on input $x_{\mathcal{A}}^{s(i)}$ and answered its query with O. The transcript output by C in this case is a simulated transcript for $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$.

In the first case, the (C, M) system is identical to $\mathcal{H}_{i:5}$, while in the second case, the (C, M) system is identical to $\mathcal{H}_{i:6}$. By the (semi-honest) security of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, we establish that the output of M must be indistinguishable in both the cases, except with negligible probability. This proves equation 5.23.

Proving Equation 5.24. We will leverage the semi-honest security of the two-party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ and the extractability property of the non-malleable commitment scheme in $\langle P, V \rangle$ to prove equation 5.24.

Specifically, we will construct a standalone machine M^* that is identical to M as described above, except that it "exposes" the NMCOM in $\langle P, V \rangle_{\mathcal{A} \to H}^{\ell}$ to an external receiver R by relying on the public-coin property of NMCOM, as

described earlier. Note that if C produces a transcript $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ according to case 1 (as described above), then the (C, M^*, R) system is identical to $\mathcal{H}_{i:5}$. On the other hand, if C produces a transcript for $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ according to case 2, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:6}$. We can now run the extractor E of NMCOM on (C, M^*) system. Note that E will rewind M^*, which in turn may rewind the interaction between C and M^*. However, since this interaction consists of a single message from C, M^* simply re-uses (if necessary) the transcript received from C in order to interact with $\mathcal{A}$ internally. Now, if the extracted values are different in case 1 and case 2, then we can break the semi-honest security of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, which is a contradiction.

References

[ADN+10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In *EUROCRYPT*, pages 113–134, 2010.

[ADW09a] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[ADW09b] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In *ICITS*, pages 1–18, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[Ajt11] Miklos Ajtai. Secure computation with information leaking to an adversary. In *STOC*, 2011.

[AK96] Ross Anderson and Markus Kuhn. Tamper resistance: a cautionary note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–11, 1996.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *CRYPTO*, pages 361–377, 2005.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[Bea96] Donald Beaver. Adaptive zero knowledge and computational equivocation (extended abstract). In *STOC*, pages 629–638, 1996.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[Blu87] Manual Blum. How to prove a theorem so no one else can claim it. In *International Congress of Mathematicians*, pages 1444–1451, 1987.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Security and Privacy*, 1992.

[BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *EUROCRYPT*, pages 156–171, 2000.

[Boy00] Victor Boyko. Ph.d. thesis. on all-or-nothing transforms and password-authenticated key exchange. MIT, EECS Department, 2000.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, pages 139–155, 2000.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS*, pages 345–354, 2006.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition using super-polynomial simulation. In *Proc. 46th FOCS*, 2005.

[BSW11] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In *EUROCRYPT*, 2011.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge. In *Proc. 32th STOC*, pages 235–244, 2000.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In *EUROCRYPT*, pages 545–562, 2008.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT*, pages 404–421, 2005.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO ' 2001*, pages 566–598, 2001.

[DGK+10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[DHLW10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DHLW10b] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proc. 30th STOC*, pages 409–418, 1998.

[DNW09] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties. In *TCC*, 2009.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[DPP97] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. *J. Cryptology*, 10(3):163–194, 1997.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *CRYPTO*, pages 526–545, 1989.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.

[Gen08] Rosario Genarro. Faster and shorter password-authenticated key exchange. In *ACM Conference on Computer and Communications Security*, 2008.

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge PCPs, and unconditional cryptography. In *CRYPTO*, pages 173–190, 2010.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–189, Summer 1996.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[GL01] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *CRYPTO*, pages 408–432, 2001.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT*, pages 524–543, 2003.

[GL06] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. *J. Cryptology*, 19(3):241–340, 2006.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th STOC*, pages 291–304, 1985.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the 19th annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, 2006.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *EUROCRYPT*, pages 339–358, 2006.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In *EUROCRYPT*, pages 54–71, 2009.

[HHK+05] Iftach Haitner, Omer Horvitz, Jonathan Katz, Chiu-Yuen Koo, Ruggero Morselli, and Ronen Shaltiel. Reducing complexity assumptions for statistically-hiding commitment. In *EUROCRYPT*, pages 58–77, 2005.

[HM96] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In *CRYPTO*, pages 201–215, 1996.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot

attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Extracting correlations. In *FOCS*, pages 261–270, 2009.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[Kat07] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT*, pages 475–494, 2001.

[KP10] Eike Kiltz and Krzysztof Pietrzak. Leakage resilient elgamal encryption. In *ASIACRYPT*, pages 595–612, 2010.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In *Theory of Cryptography Conference (TCC)*, volume 1, pages 203–222, 2004.

[LLW11] Allison Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In *STOC*, 2011.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 179–188. ACM, 2009.

[LRW11] Allison Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, 2011.

[LZ09] Yehuda Lindell and Hila Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. In *TCC*, pages 183–201, 2009.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC*, pages 306–315, 2006.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[MS08] Tal Moran and Gil Segev. David and goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, pages 527–544, 2008.

[MTVY11] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In *EUROCRYPT*, 2011.

[MY04] Philip D. MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In *EUROCRYPT*, pages 382–400, 2004.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness (extended abstract). In *CRYPTO*, pages 128–136, 1989.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[NOVY98] Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect zero-knowledge arguments for *NP* using any one-way permutation. *J. Cryptology*, 11(2):87–108, 1998.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[NV04] Minh-Huyen Nguyen and Salil P. Vadhan. Simpler session-key generation from short random passwords. In *TCC*, pages 428–445, 2004.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proc.* 21*st STOC*, pages 33–43, 1989.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *CT-RSA*, pages 1–20, 2006.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[PPS+08] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Precise concurrent zero knowledge. In *EUROCRYPT*, pages 397–414, 2008.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *STOC*, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, 2002.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[Ros04] Alon Rosen. A note on constant-round zero-knowledge proofs for NP. In *TCC*, pages 191–202, 2004.

[Sah99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proc.* 40*th FOCS*, pages 543–553, 1999.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proc.* 23*rd FOCS*, pages 80–91, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proc.* 27*th FOCS*, pages 162–167, 1986.