Pursuing the *Limits* of Cryptography

by

Arka Rai Choudhuri

Abstract

While cryptography has fairly ancient roots, *modern cryptography* has gone beyond the traditional notion of encryption, allowing for new applications such as digital signatures, digital watermarking, software obfuscation among others. While cryptography might seem like a magical tool for one's privacy needs, there are mathematical limitations to what cryptography can help us achieve. In this thesis we focus on understanding what lies on the boundary of what cryptography enables. In particular, we focus on three specific aspects that we elaborate on below. Common to these aspects is that their focus lies in the design of *protocols*, wherein two or more participants interact towards a pre-specified goal.

Necessity of Randomness in Zero-Knowledge Protocols. A Zero-Knowledge protocol consists of an interaction between two parties, designated prover and verifier, where the prover is trying to convince the verifier of the validity of a statement without revealing anything beyond the validity. We study the necessity of randomness in such protocols, a natural question since *true randomness* is a scarce resource. Prior works have shown that for most settings, the prover necessarily *requires* randomness to run any such protocol. We show, somewhat surprisingly, that for many reasonable settings, one can design protocols where a prover requires *no* randomness.

Minimizing Interaction in Secure Computation Protocols. The next part of the thesis focuses on one of the most general notions in cryptography, that of *secure computation*. It allows mutually distrusting parties to jointly compute a function over a network without revealing anything but the output of the computation. Considering that these protocols are going to be run on high-latency networks such as the internet, it is imperative that we design protocols suitable for this environment. To this end, we want to minimize the interaction between participants of the protocol. Prior works have established lower bounds on the number of times participants need to interact, and in our work we show that these lower bounds are tight by constructing new protocols that are optimal in their assumptions.

Circumventing Impossibilities with Blockchains. In some cases, there are desired usages of secure computations protocols that are provably impossible on the (regular) Internet, i.e. existing protocols can no longer be proven secure when multiple concurrent instances of the protocol are executed. But we show that by assuming the existence of a secure blockchain, a minimal additional trust assumption, we can push past the boundaries of what is cryptographically possible by constructing *new* protocols that are provably secure on the Internet.

Thesis Readers

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

Dr. Matthew Green
Associate Professor
Department of Computer Science
Johns Hopkins University

Dr. Susan Hohenberger
Research Professor
Department of Computer Science
Johns Hopkins University

Acknowledgments

My list of people to thank is long, so please bear with me. First, and foremost, I must thank my advisor Abhishek Jain. Despite my own self-doubts entering the program, making the transition to doing theoretical cryptography became significantly easier with Abhishek as my advisor. His excitement and drive for solving a wide array of cryptographic problems is something I would like to emulate going forward, and it still amazes me at the number of problems Abhishek is working on at any given time. My many long discussions with Abhishek, both technical and otherwise, have played a pivotal role in shaping my own views on cryptography and research more broadly.

I would next like to thank Matthew Green. Coming into the PhD program, I was incredibly naive with my perception of “theory” research, and was under the impression that Matt’s interests would be too applied for my liking. Luckily for me I was able to work with Matt on a few projects, and wish I had even a fraction of Matt’s theoretical grasp of ideas. Matt’s ability to come up with theoretical grounded problems motivated by practice is incredible.

I would like to thank Susan Hohenberger for being a constant source of encouragement, and agreeing to serve on my thesis committee.

I was fortunate to have been hosted over two summers by Krzysztof Pietrzak at IST Austria, and Nir Bitansky at Tel Aviv. Krzysztof, and the entire lab at IST - Hamza, Karen, Michael, Chethan, Samarth, Sasha - were amazing to be around, and I was treated to the wonderful city of Vienna, while expanding my research horizons exploring new facets of cryptography. At Tel Aviv, working with Nir, Idan and Alon, I was able to gain an improved understanding of some of the foundational questions in cryptography. I’m incredibly grateful to both Krzysztof and Nir for giving me the opportunity to work with them.

Additionally, Nils Fleischhacker, Vipul Goyal, Alon Rosen have played pivotal roles as research mentors. I’m grateful for the their patience as I stumbled along, asking stupid questions while I learned new things and I apologize for any tardiness on my part in the process. I hope to be able to leave a similar impression when given the opportunity to mentor.

I would also like to thank Nils for his `tikzpeople` LaTeX package that I continue to use in all my talks, and for him to continue to humor my constant badgering with regards to the package.

I would like to thank my various lab mates over my time at Hopkins - Christina, Ian, Gabe, Aarushi, Alishah, Dave, Zhengzhong, Gabby, Max, Tushar, and more recent additions Aditya, Pratyush, Harry, Logan and Atheer. I reserve a special mention for Alishah and Aarushi who started the program the same time I did. We have spent countless evenings and nights in the lab talking about both research and (more often than not) anything but research, and I shall miss those conversations. I have seen our lab grow in the time I’ve been here at Hopkins, and I’m incredibly proud of where it is, and I will be watching excitedly from afar as it continues to grow.

I would like to thank all my collaborators, Prabhanjan Ananth, Nir Bitansky, Michele Ciampi, Aarushi Goel, Vipul Goyal, Matthew Green, Pavel Hubáček, Abhishek Jain, Zhengzhong Jin, Chethan Kamath, Gabriel Kaptchuk, Ian Miers, Rafail Ostrovsky, Krzysztof Pietrzak, Alon Rosen and Guy Rothblum.

In the department, I've been lucky to have made many friends with whom many hours have been spent procrastinating by the coffee machine - Fabian, Rohit, Eli, Yasamin, Jaron, Aditya, Ama, Ravi, Enayat, Teodor, Razieh. I apologize if I have missed anybody. Outside of the work, I was able to rely on soccer to provide for an outlet of stress - my ever evolving weekly pick-up soccer group, my various intramural soccer teams and my weekly leagues downtown. I have grown quite attached to Baltimore over the years, and will miss the city dearly.

I would like to thank my parents, brother and my larger family for their unconditional support throughout. I would like to thank Dhivya, who has been a constant support through my highs and lows - I cannot imagine completing the PhD without you by my side.

Finally, over the last 20 odd months, and counting, the world has been ravaged by COVID-19. All things considered, I was in a position of incredible privilege that I was only mildly affected by the pandemic, which would not have been possible without the incredible support system that I had around me (and the technology that enabled it).

Funding. This work was funded in part by DARPA/ARL Safeware Grant W911NF-15-C-0213, NSF Grants CNS-1908181, CNS-1414023, CNS-1814919, NSF CAREER 1942789, Samsung Global Research Outreach award, Johns Hopkins University Catalyst award and the Office of Naval Research Grant N00014-19-1-2294

Contents

List of Figures

Chapter 1

Introduction

Modern Cryptography combines various facets of complexity and information theory, among others areas, in order to cover a wide range of applications primarily dealing with controlled access to information. While cryptography allows for many wonderful things, it is also important to characterize what is possible through cryptography. In this thesis, we focus on three specific aspects that we elaborate on below. Common to these aspects is that their focus lies in the design of *protocols*, wherein two or more participants interact towards a pre-specified goal.

Necessity of Randomness in Zero-Knowledge Protocols. A Zero-Knowledge protocol consists of an interaction between two parties, designated prover and verifier, where the prover is trying to convince the verifier of the validity of a statement. Such protocols satisfy the natural notion of *completeness* and *soundness*, wherein the validity of a true statement can be easily conveyed, while a false statement cannot easily be proven to be valid. In addition, it is also required that an adversarial verifier does not learn anything beyond the validity of the statement. A typical application of zero-knowledge protocols is when the prover is in possession of both a problem p and its solution s, and wants to use s in order to convince the verifier that the problem p has a solution, without revealing anything about the solution s.[1]

Intuitively, one would think that this would require a prover to be randomized in order to *hide* the solution (given randomness is essential in cryptography for the purposes of hiding, as in encryption for instance). In fact, there are partial results attesting to exactly this intuition [GO94]. In this work we ask if those partial results can be extended to rule out the existence of *deterministic provers*, or whether these partial results are tight. Somewhat surprisingly, our results prove the latter, by constructing zero-knowledge protocols with a deterministic prover (in settings not ruled impossible by [GO94]).

Minimizing Interaction in Secure Computation Protocols. The next part focuses on one of the most general notions in cryptography, that of *secure computation*. It allows mutually distrusting parties to jointly compute a function over a network without revealing anything but the output of the computation. Considering that these protocols are going to be run on high-latency networks such as the internet, it is imperative that we design protocols suitable for this environment. To this end, we want to minimize the interaction between participants

[1]For those familiar, the solution corresponds to the witness of an NP problem (statement).

of the protocol. The metric used to measure interactions is *rounds*, where a single round constitutes every participant sending a message.

Garg, Mukherjee, Pandey and Polychroniadou [GMPP16] established a three round lower bound for *any* secure computation protocol. We ask the question of whether their lower bound is tight. Our results show that not only is their lower bound tight with respect to the number of rounds, but also with respect to the necessary cryptographic assumptions.

Circumventing Impossibilities of Protocols on the Internet. The last part of this thesis focuses on desired properties from secure computation protocols that have been proven impossible to achieve. Specifically, when secure computation protocols get deployed on the internet, it will not be the case that only a single instance of the protocol is executed, but several such instances will be executed concurrently. In fact, these different executions might have overlapping participants. Do our protocols stay secure in this setting? Unfortunately, we know that even in the simplest setting where we have multiple concurrent instances of the *same* protocol running on the internet, it is impossible to design protocols secure against a actively deviating participants.

In this work we ask what is the minimal trust we can consider to circumvent the impossibility results. Our work shows that the existence of the blockchain (or more broadly global ledgers) allows us to circumvent these impossibilities. In fact our work more broadly studies the interaction of various protocols with the blockchain, and the challenges that arise from their interaction.

We now delve into these problems in more detail.

1.1 Deterministic Prover Zero-Knowledge

Goldwasser, Micali, and Rackoff [GMR89a] founded the concept of *zero-knowledge proofs* on two main elements: *interaction and randomness*. While both interaction and verifier randomness are known to be essential for zero knowledge, the answer as to whether the prover must also be randomized is not as definite. Goldreich and Oren [GO94] showed that prover randomness is essential in order to achieve *auxiliary-input zero-knowledge* for non-trivial languages. According to this notion, motivated by composition [GK96b], anything that a verifier can learn from the proof, on top of the auxiliary information z it already possesses, can be efficiently simulated given the same auxiliary information z.

So when is deterministic-prover zero knowledge possible? So far, deterministic prover zero knowledge have only been shown to exist in the *honest-verifier setting*. Here Faonio, Nielsen, and Venturi [FNV17] proved that any NP language $\mathcal{L}$ that has a *witness encryption scheme* [GGSW13], also has a deterministic-prover honest-verifier (perfect) zero-knowledge argument, or proof, if the language $\mathcal{L}$ has a *hash proof system* [CS02]. A similar result was recently shown by Dahari and Lindell [DL20]. In the same work, Dahari and Lindell also show a statistically sound honest-verifier zero knowledge protocol with an unbounded honest prover for all of NP assuming doubly-enhanced injective one-way functions. In the malicious verifier setting, they give a protocol satisfying a non-standard distributional notion of zero knowledge. In their definition, the prover has access to a pair of witnesses sampled from a distribution, which satisfy a certain entropy guarantee.

Whether zero knowledge with a truly deterministic prover is possible considering any meaningful form of malicious verifiers remains unknown.

1.1.1 This Work

We prove in that deterministic-prover zero knowledge for non-trivial languages is feasible for the class of malicious verifiers with bounded auxiliary input.

Theorem 1 (Informal). *Assuming non-interactive witness-indistinguishable proofs and subexponentially-secure indistinguishability obfuscation and one-way functions, there exist two-message deterministic-prover arguments for* $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ *that are zero-knowledge against bounded-auxiliary-input verifiers.*[2]

Theorem 2 (Informal). *Assuming also keyless hash functions that are collision-resistant against bounded-auxiliary-input quasipolynomial-time attackers, there exist similar arguments for all of* NP.

By zero knowledge against bounded-auxiliary-input verifiers we formally mean that for any polynomial bound b, there exists a corresponding deterministic-prover argument that is zero knowledge against (malicious) verifiers with non-uniform auxiliary input of size at most b. This, in particular, includes the class of *uniform verifiers*, considered in the original zero-knowledge definition of [GMR89a]. We stress that the running time of the verifier may be an arbitrary polynomial, potentially larger than b. Also, indistinguishability of simulated and real proofs holds against non-uniform distinguishers of arbitrary polynomial size. Same goes for soundness, which holds against non-uniform provers of arbitrary polynomial size.

Together with the impossibility result of Goldreich and Oren for unbounded auxiliary input, the above results give a complete picture of when exactly deterministic-prover zero knowledge is feasible. We note that two-message zero knowledge against unbounded auxiliary input is by itself known to be impossible. Our result indeed circumvents this impossibility (for bounded auxiliary input), but this was already known (with a randomized prover) [BCPR14].

On the Necessity of Strong Assumptions and Predictable Arguments. To demonstrate the feasibility of deterministic-prover zero knowledge, we rely on hardness assumptions that are arguably strong. We show that this is inherent. Specifically, we show that deterministic prover zero-knowledge arguments for NP imply witness encryption for NP, which at this point is only known based on strong assumptions, such as indistinguishability obfuscation.

The implication to witness encryption, in fact, follows from a more general implication to *predictable arguments*. Predictable arguments, introduced by Faonio, Nielsen, and Venturi [FNV17], are arguments where the honest verifier's (private) random coins efficiently determine a unique accepting transcript — in order to convince the verifier, the prover must be consistent with this transcript throughout the entire protocol. We prove that any deterministic-prover zero-knowledge argument against bounded-auxiliary-input verifiers can be turned into a predictable argument. The transformation, in fact, preserves the honest prover algorithm, and in particular also zero knowledge.

Theorem 3 (Informal). *Any deterministic-prover zero-knowledge argument against bounded-auxiliary-input verifiers can be made predictable.*

We also give a transformation that only requires honest-verifier zero knowledge and works provided that the argument is expressive enough (e.g., for all NP or even just $\mathsf{NP} \cap \mathsf{co\text{-}NP}$). The fact

[2]Indistinguishability obfuscation implies non-interactive witness indistinguishable proofs, but with a randomized verifier [BP15], which is insufficient for our purpose. The verifier can be derandomized under a worst-case Nisan-Wigderson [NW94] type derandomization assumption [BV17]. Non-interactive witness indistinguishable proofs with a deterministic verifier are also known from standard assumptions on bilinear maps [GOS06].

that deterministic-prover zero knowledge arguments imply witness encryption, then follows from [FNV17] where predictable arguments are shown to imply witness encryption.

Corollary 1 (of Predictability). *Any deterministic-prover zero-knowledge argument against bounded-auxiliary-input verifiers for a language $\mathcal{L}$ implies a witness encryption scheme for $\mathcal{L}$.*

We use additional known results regarding predictable arguments [FNV17] to deduce similar results for deterministic-prover zero knowledge:

Corollary 2 (of Predictability). *Any deterministic-prover zero-knowledge argument against bounded-auxiliary-input verifiers can be reduced to two messages and made laconic.*

Here by *laconic* [GVW01, FNV17] we mean that the prover sends a *single* bit and the soundness error is negligibly close to $1/2$; or more generally, the prover sends ℓ bit in order to obtain a soundness error negligibly close to $2^{-\ell}$.

Non-Black-Box Zero-Knowledge Simulation. The zero-knowledge simulator in our constructed arguments makes non-black-box use of the verifier's code. This is known to be inherent — black-box simulation is impossible in the setting of two (or even three) message zero knowledge against bounded-auxiliary-input verifiers [GK96b, BCPR14].

1.2 Round Optimal Secure MPC from Oblivious Transfer

The ability to securely compute on private datasets of individuals has wide applications of tremendous benefits to society. Secure multiparty computation (MPC) [Yao86, GMW87a] provides a solution to the problem of computing on private data by allowing a group of parties to jointly evaluate any function over their private inputs in such a manner that no one learns anything beyond the output of the function.

Since its introduction nearly three decades ago, MPC has been extensively studied along two fundamental lines: necessary *assumptions* [GMW87a, Kil88, IPS08], and *round complexity* [GMW87a, BMR90, KOS03, KO04, Pas04, PW10, Wee10, Goy11, GMPP16, ACJ17, BHP17, COSV17a, COSV17b].[3]

Even for the case of malicious adversaries who may corrupt any number of parties, both of these topics, individually, are by now pretty well understood:

– It is well known that oblivious transfer (OT) is both necessary and sufficient [Kil88, IPS08] for MPC.

– A recent sequence of works have established that *four rounds* are both necessary [GMPP16] and sufficient [ACJ17, BHP17, BGJ$^+$18, HHPV18] for MPC (with respect to black-box simulation). However, the assumptions required by these works are far from optimal, ranging from sub-exponential hardness assumptions [ACJ17, BHP17] to polynomial hardness of specific forms of encryption schemes [HHPV18] or specific number-theoretic assumptions [BGJ$^+$18].

In this work, we consider the well studied goal of building round-efficient MPC while minimizing the underlying cryptographic assumptions. Namely:

[3]A detailed discussion on related works can be found in Section 1.2.2.

Can we construct round optimal MPC from minimal assumptions?

Precisely, we ask whether it is possible to construct four round MPC from four round OT. This was explicitly left as an open problem in the elegant work of Benhamouda and Lin [BL18] who constructed k-round MPC from k-round OT for $k \geq 5$.

1.2.1 Our Results

In this work, we resolve the above question in the affirmative. Namely, we construct four round malicious-secure MPC based only on four round (malicious-secure) OT. Our protocol admits black-box simulation and achieves security against malicious adversaries in the dishonest majority setting.

Theorem 4 (Informal). *Assuming the existence of four round OT, there exists a four round MPC protocol for any efficiently computable functionality in the plain model.*

This settles the long line of research on constructing round efficient MPC from minimal cryptographic assumptions.

Our Approach. To obtain our result, we take a conceptually different approach from the works of [ACJ17, BHP17, BGJ$^+$18, HHPV18] for enforcing honest behavior on (possibly malicious) protocol participants. Unlike these works, we do not require the parties to give an *explicit* proof of honest behavior within the first three rounds of the protocol. Instead, we devise a *multiparty conditional disclosure of secrets* mechanism that ensures that the final round messages of the honest parties become "opaque" if even a single participant behaved maliciously. A key property of this mechanism is that it allows for each party to obtain a *public* witness that attests to honest behavior of all the parties, without compromising the security of any party. We refer the reader to Section 4.1 for details.

On the Minimal Assumptions. We study MPC in the standard broadcast communication model, where in each round, every party broadcasts a message to the other parties. In this model, k-round MPC implies k-round *bidirectional* OT, where each round consists of messages from both the OT sender and the receiver. However, it is not immediately clear whether it also implies k-round OT in the standard, *alternating-message* model for two-party protocols where each round consists of a message from only one of the two parties. As such, the minimal assumption for k-round MPC is, in fact, k-round bidirectional OT (as opposed to alternating-message OT).

Towards establishing the optimality of Theorem 4, we observe that k-round bidirectional OT implies k-round alternating-message OT.

Theorem 5. *k-round bidirectional OT implies k-round alternating-message OT.*

Our transformation is unconditional and generalizes a message rescheduling strategy previously considered by Garg et al. [GMPP16] for the specific case of three round coin-tossing protocols. In fact, this transformation is even more general and applies to any two-party functionality, with the restriction that only one party learns the output in the alternating-message protocol.

An important corollary of Theorem 5 is that it establishes the missing piece from the result of Benhamouda and Lin [BL18] who constructed k-round MPC from any k-round alternating-message OT for $k \geq 5$. Their result, put together with our main result in Theorem 4 provides a *full resolution* of the fundamental question of basing round efficient MPC on minimal assumptions.

In the sequel, for simplicity of exposition, we refer to alternating-message OT as simply OT.

1.2.2 Related Work

Round-Complexity of MPC. The round complexity of MPC has been extensively studied over the years in a variety of models. Here, we provide a short survey of malicious-secure MPC protocols in the plain model. We refer the reader to [BGJ$^+$18] for a more comprehensive survey.

Beaver et al. [BMR90] initiated the study of constant round MPC in the honest majority setting. Several follow-up works subsequently constructed constant round MPC against dishonest majority (which is the focus of the present work) [KOS03, Pas04, PW10, Wee10, Goy11]. Garg et al. [GMPP16] established a lower bound of four rounds for MPC. They constructed five and six round MPC protocols using indistinguishability obfuscation and LWE, respectively, together with three-round robust non-malleable commitments.

The first four round MPC protocols were constructed by Ananth et al. [ACJ17] and Brakerski et al. [BHP17] based on different sub-exponential-time hardness assumptions. [ACJ17] also constructed a five round MPC protocol based on polynomial-time hardness assumptions. Ciampi et al. constructed four-round protocols for multiparty coin-tossing [COSV17a] and two-party computation [COSV17b] from polynomial-time assumptions. Benhamouda and Lin [BL18] gave a general transformation from any k-round OT with alternating messages to k-round MPC, for $k \geq 5$. More recently, independent works of Badrinarayanan et al. [BGJ$^+$18] and Halevi et al. [HHPV18] constructed four round MPC protcols for general functionalities based on different polynomial-time assumptions. Specifically, [BGJ$^+$18] rely on DDH (or QR or N-th Residuosity), and [HHPV18] rely on Zaps, affine-homomorphic encryption schemes and injective one-way functions (which can all be instantiated from QR).

Conditional Disclosure of Secrets. The notion of CDS has also been extensively studied over the years in a variety of models. The works most relevant to ours are [AIR01, BP12, AJ17, BKP19] that consider the computational setting with *two* parties, a sender and a receiver. The sender holds an instance x (of an NP language) and a message m, while the receiver holds x and the corresponding witness w. If the witness is valid for x, then the receiver obtains m, whereas if the instance x is not in the language, m remains hidden. The CDS protocols are presented in the two message setting, and can be thought of a lightweight alternative to zero-knowledge.

Another line of work, initiated by [GIKM98] studies CDS in the information theoretic setting, where the input x is divided among multiple senders that share common randomness (and a secret). Each sender is constrained to sending a single message to the receiver, who can then reconstruct the secret only if some relation R over x is satisfied. This setting has seen renewed interest, with recent works focusing on the communication complexity (for example, see [GKW15]). Due to the necessity of a common random string and the corruption model, this line of work is not relevant to our setting.

To the best of our knowledge, CDS in the *multiparty* setting was previously only considered in the work of [IKP10], where they present two separate notions. The first notion is reminiscent of the original notion in [GIKM98], which is a non-interactive protocol, where the parties that share a secret also share a common random string. The second notion, bearing slight similarity to ours, does not require the parties with the input to share randomness. But this second notion is only defined for a very special relation where the secret is revealed only if all the parties with inputs, have the same input (i.e. the relation on x is that all the divisions of x are the same). In this constrained setting, they in fact achieve information theoretic security in the dishonest majority for adversaries that have some additional "structural" requirements.

This work is the result of a merge of the works [CO19] and [CGJ19], and subsumes both these works.

1.3 Founding Secure Computation on Blockchains

Blockchain is an exciting new technology which is having a profound impact on the world of cryptography. Blockchains provide both: new applications of existing cryptographic primitives (such as hash function, or zero-knowledge proofs), as well as, novel foundations on which new cryptographic primitives can be realized (such as fair-secure computation [ADMM14, BK14, CGJ$^+$17], or, one-time programs [GG17]). In this work, we seek to examine the foundations of secure computation protocols in the context of blockchains. More concretely, we study what we call the *blockchain-hybrid model* and examine constructions of zero-knowledge and secure computation in this model.

The Blockchain-Hybrid Model. In order to facilitate the use of blockchains in secure computation, we study the blockchain-hybrid model, where the blockchain – modeled as a *global* ledger functionality – is available to all the participants of a cryptographic protocol. The parties can access the blockchain by posting and reading content, but no single party has any control over the blockchain. Our modeling follows previous elegant works on formalizing the blockchain functionality [KZZ16, BMTZ17, BGK$^+$18]. In particular, our model is based on the global blockchain ledger model from Badertscher et. al [BMTZ17].

We study simulation-based security in the blockchain-hybrid model. In our model, *the simulator does not have any control over the blockchain*, and simply treats it as an oracle just like protocol participants. Thus, unlike traditional trusted setup models such as common reference string, the blockchain-hybrid model does not provide any new "power" to the simulator. In particular, the simulator is restricted to its plain model capabilities such as resetting the adversary or using knowledge of its code. Thus, in our model, the blockchain can be *global*, in that it can be used by multiple different protocols at the same time. This is reminiscent of simulation in the global UC framework [CDPW07, CJS14, HPV16]. A related model is the global Random Oracle model [CJS14] where the simulator can only observe the queries made by the adversary to the random oracle, but cannot program the random oracle (since it is global and therefore shared across many protocols).

Secure Computation based on Blockchains. We study the foundations of secure computation in the presence of the global blockchain functionality. Interestingly, we demonstrate both destructive and constructive applications of blockchains to cryptography. Primitives which were earlier possible to realize now become impossible. At the same, working in this model allows us to overcome previously established deep impossibility results in cryptography. Interestingly, we also utilize mining delays – typically viewed as a negative feature of blockchains – for constructive purposes in this work. Our main results as discussed next.

1.3.1 Our Results

Simulation Failure in the Presence of Blockchains. We consider a new class of adversaries that we refer to as *blockchain-active adversaries*. These adversaries are similar to usual cryptographic adversaries, except that they have user access to a blockchain, i.e., they can post on the blockchain and read its state at any point.

We observe that such adversaries can foil many existing simulation techniques that are used for proving security of standard cryptographic schemes. To illustrate the main idea, let us consider rewinding-based black-box simulation techniques that are used, e.g., in zero-knowledge (ZK) proofs [GMR85], secure multiparty computation [Yao82a, GMW87a], and signature schemes in the random oracle model constructed via the Fiat-Shamir heuristic [FS87]. A crucial requirement for the success of rewinding-based simulation is that the adversary should be *oblivious* to the rewinding. Usually, this requirement can be easily met since the simulator can simply "reset" the code of the adversary, which prevents it from keeping state across the rewindings.

A blockchain-active adversary, however, can periodically post on the blockchain and use it to maintain state across rewindings, and therefore detect that it is being rewound. In this case, the adversary can simply abort and therefore fail the simulation process.[4] It is not too difficult to turn the above idea into a formal impossibility result for ZK proofs against blockchain-active adversaries, when the simulation is required to be *black-box*.

Theorem 6 (Informal). *There does not exist an interactive argument in the plain model which is zero-knowledge w.r.t. black-box simulation against blockchain-active adversaries.*

The above impossibility result extends to secure multiparty computation and other natural cryptographic primitives whose security is proven via a rewinding simulator.

Constructing Zero-Knowledge Protocols. To overcome the above problems posed by blockchains, we look towards blockchains for a solution as well. Our idea is to make the *protocol* blockchain active as well. That is, in addition to the adversary, the honest parties would have access to the blockchain as well.

Our first positive result is an $\omega(1)$-round ZK proof system in the blockchain-hybrid model whose security is proven w.r.t. black-box simulation.

Theorem 7 (Informal). *Assuming collision-resistant hash functions, there exists an $\omega(1)$-round ZK proof system in the blockchain-hybrid model w.r.t. black-box simulation.*

Interestingly, in our construction, *the honest parties do not post any message on the blockchains*. Instead, they only keep a "tab" on the current state of the blockchain in order to decide whether or not to continue the protocol.

We also show that the above result is tight. Namely, we show that using black-box simulation, constant-round ZK is impossible in the blockchain-hybrid model.

Theorem 8 (Informal). *Assuming one-way functions, there does not exist an $O(1)$-round ZK argument system in the blockchain-hybrid model w.r.t. a (expected probabilistic polynomial time) black-box simulator.*

This is in sharp contrast to the plain model where there are a number of classical constant round zero-knowledge protocols that are proven secure w.r.t. a black-box simulator [GK96b, FS90, BJY97].

Concurrent Secure Computation using Blockchains. Classical secure computation protocols such as [Yao82a, GMW87a] only achieve "stand-alone" security, and fail in the setting of *concurrent self-composition*, where multiple copies of a protocol may be executed concurrently, under the control

[4]This is reminiscent to the problems that arise in the context of UC security, where the adversary cannot be rewound since it can communicate with an external environment, leading to broad impossibility results for zero-knowledge and secure computation [Can01, CF01, CKL03].

of an adversary. In fact, achieving concurrent secure computation in the plain model has been shown to be impossible [CKL03, Lin03, Lin04, Lin08, BPS06, Goy12, AGJ+12, GKOV12]. The above impossibility results are far reaching and rule out secure computation for a large class of functionalities in a variety of settings.

Interestingly, we show that concurrent self-composition is possible in the blockchain-hybrid model w.r.t. standard real/ideal model notion of security with a PPT simulator. Thus, our results (put together) show that designing cryptographic primitives in the blockchain-hybrid model is, in some sense, harder and easier at the same time.

Theorem 9 (Informal). *Assuming collision-resistant hash functions and oblivious transfer, there exists a concurrent self-composable secure computation protocol for all polynomial-time functionalities in the blockchain-hybrid model.*

In our protocol, each party is required to post an initial message (which corresponds to a commitment to its input and randomness) on the blockchain. However, an honest party can simply perform this posting in an "offline" phase prior to the start of the protocol. In particular, once the protocol starts, an honest party is not required to post any additional message on the blockchain.

A number of prior beautiful works have constructed concurrent (and universally composable) secure computation in various setup models such as the trusted common reference string model [CLOS02], the registered public-key model [BCNP04], the tamper-proof hardware model [Kat07, CGS08, GIS+10], and the physically uncloneable functions model [BFSK11, DFK+14, BKOV17]. We believe that the blockchain model provides an appealing *decentralized* alternative to these models since there are no physical assumptions or centralized trusted parties involved. Moreover, it allows for basing concurrent security on an already existing and widely used infrastructure. Further, it is possible to obtain strong guarantees of the following form: an adversary who can break our construction can also break the security of the underlying blockchain (potentially allowing it to gain large amounts of cryptocurrency), or the underlying cryptographic assumptions (oblivious transfer and collision-resistant hash functions in our case).

Impossibility of UC Security. While Theorem 9 establishes the feasibility of concurrent self-composition, we show that universal composition security [Can01] is impossible in the blockchain-hybrid model:

Theorem 10 (Informal). *Universally composable commitments are impossible in the blockchain-hybrid model.*

We prove the above result via a simple adaptation of the impossibility result of [CF01] to the blockchain-hybrid model. The main intuition behind this result is that a simulator in the blockchain-hybrid model has the same capabilities as in the plain model, namely, the ability to rewind the adversary or using knowledge of its code. Crucially, (unlike the non-programmable random oracle model [CJS14]) the ability to see the queries made to the blockchain do not constitute a new capability for the simulator since *everyone* can see those queries.

1.3.2 Related Work

Blockchains and Cryptography. In a recent work, [GG17] used blockchains to construct non-interactive zero-knowledge (NIZK) arguments and selectively-secure one-time programs. Their model, however, is fundamentally different from ours in that they rely on a much stronger notion of simulation where the simulator controls all the honest miners in the blockchain. Intuitively, this is somewhat similar to the honest majority model used to design (universally composable) secure multiparty computation protocols. Due to the power given to the simulator, their model necessitates the blockchain to be "local" (i.e., private) to the protocol. In contrast, our model allows for the blockchain to be a "global" setup since the simulator has no extra power over the blockchain compared to the adversary. This is similar to the difference between universal composability framework [Can01] and global universal composability framework [CDPW07], where in the former model, a setup (such as a common reference string) cannot be reused by different protocols, whereas in the latter model, a common setup can be used across multiple protocols. Indeed, since the simulator has no additional power except the ability to reset the adversary or use knowledge of its code, NIZKs are impossible in our model, similar to the plain model. Unlike our work, [GG17] do not consider interactive ZK proofs or any notion of secure multiparty computation.

In another recent work, [CGJ+17] study the problem of fair multiparty computation in a "bulletin-board" model that can be implemented with blockchains. Similar to [GG17], however, their model provides the simulator the ability to control the blockchain. Prior to their work, multiple works [ADMM14, BK14] studied the problem of fairness with penalties using cryptocurrencies.

Several elegant works have conducted a formal study of various properties of blockchains [GKL15, PSs17, GKL17, KRDO17, BMTZ17]. Most relevant to our work is that of Badertscher et. al [BMTZ17] whose modeling of the blockchain ideal functionality we closely follow.

Concurrent Security. The study of concurrent security for cryptographic protocols was initiated by Dwork et al. [DNS98] who also introduced a timing model for constructing concurrent ZK. In this model, the parties have synchronized clocks and are required to insert "delays" at appropriate points in the protocol. A refined version of their model was later considered in [KLP05] for the problem of concurrent secure computation. We note that while our approach to concurrent secure computation in the blockchain-hybrid model appears to bear some similarity to the timing model, there are fundamental differences that separate these models. For example, the simulator can fully control the clock of the adversary in the timing model, while this is not possible in our setting since the blockchains provide an unforgeable clock to the adversary. More importantly, in the timing model, there are no "unsafe" points, and the simulator can rewind anywhere. For this reason, the timing model does not require developing new concurrent extraction techniques, and instead standard rewinding techniques for the stand-alone setting are applicable there. Finally, in the timing model, honest parties insert artificial delays in the protocol based on their clocks, while in our constructions, an honest party responds immediately to messages received from the other (possibly adversarial) party.

1.4 Outline of The Thesis

We provide an outline of this thesis below.

1.4.1 Organization

We start with our model of the blockchain in section 2.14, and all subsequent results are in this model. In section 5.3 we describe a $\omega(1)$ round black-box zero-knowledge protocol. We describe our concurrently extractable commitment scheme in section 5.4 and use our constructed commitment scheme to achieve a concurrently secure two-party computation protocol described in section 5.4.3. We move on to our impossibility results starting with a lower bound on the round complexity of black-box zero-knowledge in section 5.5. In section 5.6 we show that allowing only the adversary access to the blockchain rules out zero knowledge. Finally, we show that UC commitments are impossible in section 5.7.

Chapter 2

Preliminaries

We describe in this section notation that we will across this work. We also define some necessary cryptographic primitives. There are some primitives (not defined) that are closely tied to the work they appear in, and those we define in their corresponding chapter.

2.1 General

2.1.1 Basic Notation

Sets. We use $[k]$ to denote the set $\{1, \cdots, k\}$ and the shorthand $\{0,1\}^k$ to denote the set of all k bit strings. Finally, $\{0,1\}^*$ will be used to denote the infinite set $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \cdots\}$ where ε is the empty string i.e. $\{0,1\}^*$ denotes the set of *all* possible bit strings.

Probability distributions. We say X is a distribution over some universe $\mathcal{U}$ if it assigns a probability p_u to the element $u \in \mathcal{U}$ such that $\sum_{u \in \mathcal{U}} p_u = 1$. We denote by $x \leftarrow X$ the process of sampling an element x from the distribution X. When we sample *uniformly* at random from binary strings of length ℓ, we denote this by $x \leftarrow\!\!\$ \{0,1\}^\ell$.

Turing machines. In this work, all the participants are modeled as Turing machines, specifically they are modeled as probabilistic polynomial-time (PPT) Turning machines, where the Turing machine is equipped with a random tape and runs in polynomial time. We will also consider the non-uniform model of Turing machines, which incorporates "auxiliary" information for each input length. Formally, a non-uniform PPT $\mathsf{M} = \{\mathsf{M}_\lambda\}_\lambda$ is a family of probabilistic Turing machines (one for each λ), where there exists a polynomial poly, such that the description size $|\mathsf{M}_\lambda|$ and the running time of M_λ are bounded by $\mathsf{poly}(\lambda)$.

Randomized Algorithms. When we want to explicitly specify the randomness used by an algorithm A (resp. Turing machine M) we denote this by $y := A(x; r)$ (resp. $y := \mathsf{M}(x; r)$) where x is the input, y the output and r the random coins with ":=" the assignment operator. If the randomness is implicit, we write the same as $y \leftarrow A(x)$.

NP **Languages and Relations.** A *language* $\mathcal{L} \subseteq \{0,1\}^*$ is said to be in the class NP (i.e. $\mathcal{L} \in \mathsf{NP}$) if there exists a Boolean *relation* $R_{\mathcal{L}} \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $\mathsf{p}(\cdot)$ such that $R_{\mathcal{L}}$ can be recognized in (deterministic) polynomial time, and $x \in \mathcal{L}$ if and only if there exists a w such that $|w| \leq (\mathsf{p}|x|)$ and $(x, w) \in R_{\mathcal{L}}$. Such a w is called a witness for membership of $x \in \mathcal{L}$. We will overload notation and denote w to be a *valid* witness for x if $R_{\mathcal{L}}(x, w) = 1$. Further, we use the notation $R_{\mathcal{L}}(x)$ to denote the set of all witnesses w such that $R_{\mathcal{L}}(x, w) = 1$.

The complement of a language $\mathcal{L}$, is defined to be $\overline{\mathcal{L}} \coloneqq \{0,1\}^* \setminus \mathcal{L}$. We say $\mathcal{L} \in \mathsf{co\text{-}NP}$ if $\overline{\mathcal{L}} \in \mathsf{NP}$, i.e. there is an efficiently verifiable witness for the non-membership in $\mathcal{L}$.

Negligible functions. In this work we will often use the term *negligible* to describe a function that diminishes faster than any inverse polynomial. Specifically, we denote such functions by $\mathsf{negl}(\cdot)$ if $\forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$, $\mathsf{negl}(\lambda) < \frac{1}{n^c}$.

2.1.2 Indistinguishability of Ensembles

In this work, we shall often compare probability distributions. Specifically, we shall talk of the *closeness* of multiple distribution *ensembles*. Specifically, let $S \in \{0,1\}^*$ be a set of strings. A *probability ensemble* indexed by S is a sequence of random variables indexed by S. For instance $X = \{X_\alpha\}_{\alpha \in S}$ is ensemble indexed by S.

Often we will find it convenient to index the random variables in the ensemble by the set of natural numbers $\mathbb{N}$.

2.1.2.1 Computational Indistinguishability

Throughout this work, we will talk about computational indistinguishability with respect to non-uniform distinguishers.

Definition 1 (Computational Indistinguishability). *Two ensembles $X = \{X_\alpha\}_{\alpha \in S}$ and $Y = \{Y_\alpha\}_{\alpha \in S}$ are said to be computationally indistinguishable, denoted by $X \approx_c Y$, if for every non-uniform* PPT *distinguisher $\mathcal{D}$, every polynomial* p*, all sufficiently large λ and every $\alpha \in \{0,1\}^{\mathsf{poly}(\lambda)} \cap S$*

$$\left| \Pr\left[\mathcal{D}(1^\lambda, X_\alpha) = 1\right] - \Pr\left[\mathcal{D}(1^\lambda, Y_\alpha) = 1\right] \right| < \frac{1}{\mathsf{p}(\lambda)} \enspace ,$$

where the probability are taken over the samples of X_α, Y_α and coin tosses of $\mathcal{D}$.

2.1.2.2 Statistical Indistinguishability

We shall sometimes find it convenient to talk about the stronger notion of statistical indistinguishability, defined below. But first, we define the statistical distance between two distributions.

Definition 2 (Statistical Distance Δ). *If X and Y are probability distributions on a discrete universe $\mathcal{U}$, then the statistical distance between X and Y is defined to be*

$$\Delta(X, Y) \coloneqq \max_{T \subset \mathcal{U}} |\Pr[X \in T] - \Pr[Y \in T]| \enspace .$$

Definition 3 (Statistical Indistinguishability). *Two ensembles $X = \{X_\alpha\}_{\alpha \in S}$ and $Y = \{Y_\alpha\}_{\alpha \in S}$ are said to be statistically indistinguishable, denoted by $X \approx_s Y$, if for every polynomial p, all sufficiently large λ and every $\alpha \in \{0,1\}^{\mathsf{poly}(\lambda)} \cap S$*

$$\Delta(X_\alpha, Y_\alpha) < \frac{1}{\mathsf{p}(\lambda)} \quad ,$$

where $\Delta(X_\alpha, Y_\alpha)$ corresponds to the statistical distance between X_α and Y_α as described earlier.

When $\Delta(X_\alpha, Y_\alpha) = 0$ for *all* α, we say the ensembles are identical.

2.2 One-way Functions

Although we do not directly use one-way functions in our work, many of the primitives that we will use are known to exist assuming the existence of one-way functions (and its variants). We define it below for completeness.

Definition 4 (One-way Function). *A function $f : \{0,1\}^* \mapsto \{0,1\}^*$ is a one way function if it satisfies the following two conditions:*

1. Easy to compute*: There is a* PPT *algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$\Pr[r \leftarrow\!\!\$\ \{0,1\}^m \ : \ C(x;r) = f(x)] = 1.$$

2. Hard to invert*: For every non-uniform* PPT *adversary $\mathcal{A}$,*

$$\Pr\big[x \leftarrow\!\!\$\ \{0,1\}^\lambda, \widetilde{x} \leftarrow \mathcal{A}(1^\lambda, f(x)) : f(\widetilde{x}) = f(x)\big] \leq \mathsf{negl}(\lambda)$$

The above definition extends to injective one-way functions (resp. one-way permutations), if f is an injective function (resp. permutation). While the existence of injective one-way functions or one-way permutation clearly implies the existence of one-way functions, the opposite is not true. In fact there are black-box separation results between one-way functions and one-way permutations [Rud88, KSS11], i.e. the existence of one-way functions do not imply the existence of one-way functions in a black-box manner. Therefore the above notions are treated as *different* assumptions, and will be stated as such across the thesis.

2.3 Commitment Schemes

We will consider various flavors of commitments schemes that vary depending on the security properties desired.

2.3.1 Non-interactive Commitment Schemes

We define below bit commitment schemes

Definition 5 (Non-interactive Bit Commitment Schemes). *A polynomial time computable function:* $\mathsf{Com} : \{0,1\} \times \{0,1\}^\lambda \mapsto \{0,1\}^{\ell(\lambda)}$ *is a bit commitment if it satisfies the properties below:*

Binding: *For any* $r, r' \in \{0,1\}^\lambda, b, b' \in \{0,1\}$*, if* $\mathsf{Com}(b; r) = \mathsf{Com}(b'; r')$ *then* $b = b'$*.*

Computational Hiding: *The following holds:*

$$\left\{\mathsf{Com}(0; r) : r \leftarrow\$ \{0,1\}^\lambda\right\} \approx_c \left\{\mathsf{Com}(1; r) : r \leftarrow\$ \{0,1\}^\lambda\right\} .$$

where computational indistinguishability is with respect to arbitrary non-uniform PPT *distinguisher.*

We note that the above scheme can be extended to commit to strings, rather than just bits, by committing to each bit independently and we will use the syntax interchangeably. Such commitments can be constructed assuming injective one-way functions [Blu81, Yao82b, GL89]. Looking ahead, our construction of deterministic prover-zero knowledge protocol requires that the underlying string that is committed can be extracted in quasi-polynomial time. Such commitments can be constructed from subexponentiall-secure injective one-way functions (which in turn can be constructed from subexponential IO and one-way functions).

Two Round Statistically Binding Commitments. We also consider a variant of the above definition in the two message setting, where the commitment receiver computes the first message rec_1, and the commit function Com in addition to the input bit (or string) and randomness now also takes as input the receiver message rec_1. *Statistical binding* guarantees that binding condition in the above definition holds with all but negligible probability, where the probability is taken over the random coins used to compute rec_1. Unlike non-interactive commitments, two round statistically binding commitments can be built from the weaker assumption of one-way functions [HILL99, Nao91].

2.3.2 Statistically Hiding Commitment Schemes

We consider a variant of the commitment scheme, (two round) statistically hiding commitment scheme[NOVY98, HHK+05] where the hiding property is required to be statistical whereas the binding is only required to be computational.

Since this will be a two round protocol, we shall introduce some notation here. The protocol is specified by (S, R, V) between a sender S and the receiver R, with V corresponding to the verifier for the decommitment. The output of the interaction $\langle S(1^\lambda, b), R(1^\lambda)\rangle$ is $(\mathsf{decom}, \mathsf{com})$ where decom is the *decommitment* output by the sender S and com is the state information output by the receiver R. The verifier V on input $(\mathsf{decom}, \mathsf{com})$ outputs either 0 or 1 to indicate the committed bit.

Definition 6 (Statistically Hiding Commitment Schemes). *(S, R, V) is a statistically hiding commitment scheme if it satisfies the properties below:*

Completeness *For any* $\lambda \in \mathbb{N}, b \in \{0,1\}$*,*

$$\Pr\Big[V\Big(\mathsf{Out}_S\langle S(1^\lambda, b), R(1^\lambda)\rangle, \mathsf{Out}_R\langle S(1^\lambda, b), R(1^\lambda)\rangle\Big) = b\Big] = 1$$

Computational Binding: *For all* PPT *senders* S^**,*

$$\Pr\left[\begin{array}{l} V(\mathsf{decom}, \mathsf{com}), V(\mathsf{decom}', \mathsf{com}) \in \{0,1\} \\ V(\mathsf{decom}, \mathsf{com}) \neq V(\mathsf{decom}', \mathsf{com}) \end{array} : ((\mathsf{decom}, \mathsf{decom}'), \mathsf{com}) \leftarrow \langle S^*(1^\lambda), R(1^\lambda)\rangle\right] = 1$$

Statistical Hiding: *The following holds for every* PPT *algorithm* R^**:*

$$\left\{\mathsf{View}_{R^*}\langle S(1^\lambda,0),R^*(1^\lambda)\rangle\right\}\approx_s\left\{\mathsf{View}_{R^*}\langle S(1^\lambda,1),R^*(1^\lambda)\rangle\right\}\ .$$

.

Such two round schemes can be constructed based on collision resistant hash functions [NY89, HM96].

2.4 Indistinguishability Obfuscation (IO)

We now give a definition of indistinguishability obfuscator for Turing Machines, which can be constructed from indistinguishability obfuscators for circuits [KLW15, BCG$^+$18, GS18a].

Definition 7 (Indistinguishability Obfuscator for Turing Machines). *A succinct indistinguishability obfuscator for Turing machines consists of a* PPT *machine* iOM *that works as follows:*

- iOM *takes as input the security parameter* 1^λ*, the Turing machine* M *to obfuscate, an input length* n*, and time bound* t*.*
- iOM *outputs a Turing machine* $\widetilde{\mathsf{M}}$ *which is an obfuscation of* M *corresponding to input length* n *and time bound* t*.* $\widetilde{\mathsf{M}}$ *takes as input* $x\in\{0,1\}^n$*.*

The scheme should satisfy the following requirements:

Correctness *For all* $\lambda\in\mathbb{N}$*, for all* $\mathsf{M}\in\mathcal{M}_\lambda$*, for all inputs* $x\in\{0,1\}^n$*, time bounds* t' *such that* $t'\le t$*, let* y *be the output of* $\mathsf{M}(x)$ *after at most* t *steps, then*

$$\Pr\left[\widetilde{\mathsf{M}}\leftarrow\mathsf{iOM}(1^\lambda,1^n,1^{\log t},\mathsf{M})\ :\ \widetilde{\mathsf{M}}(x)=y\right]=1\ .$$

Security *It holds that*

$$\left\{\mathsf{iOM}(1^\lambda,1^n,1^{\log t},\mathsf{M}_0)\right\}_{\substack{\lambda,t,n,\\ \mathsf{M}_0,\mathsf{M}_1}}\approx_c\left\{\mathsf{iOM}(1^\lambda,1^n,1^{\log t},\mathsf{M}_1)\right\}_{\substack{\lambda,t,n,\\ \mathsf{M}_0,\mathsf{M}_1}}\ ,$$

where $\lambda\in\mathbb{N}$*,* $n\le t\le 2^\lambda$*, and* $\mathsf{M}_0,\mathsf{M}_1$ *are any pair of machines of the same size such that for any input* $x\in\{0,1\}^n$ *both halt after the same number of steps with the same output.*

Efficiency and Succinctness *We require that the running time of* iOM *and the length of its output, namely the obfuscated machine* $\widetilde{\mathsf{M}}$*, is* $\mathsf{poly}(|\mathsf{M}|,\log t,n,\lambda)$*. We also require that the running time* $\tilde{t}_x$ *of* $\widetilde{\mathsf{M}}(x)$ *is* $\mathsf{poly}(t_x,|\mathsf{M}|,n,\lambda)$*, where* t_x *is the running time of* $\mathsf{M}(x)$*.*

2.5 Witness Encryption

The following definition of witness encryption is taken from [GGSW13].

Definition 8. *A witness encryption scheme for an* NP *language* $\mathcal{L}$*, with corresponding witness relation* $R_\mathcal{L}$*, consists of the following two polynomial-time algorithms:*

Encryption. *The probabilistic algorithm* $\mathsf{WE.Enc}(1^\lambda, x, m)$ *takes as input a security parameter* 1^λ*, a string* $x \in \{0,1\}^*$*, and a message* $m \in \{0,1\}$*. It outputs a ciphertext* ct.

Decryption. *The algorithm* $\mathsf{WE.dec}(\mathsf{ct}, w)$ *takes as input a ciphertext* ct*, a string* $w \in \{0,1\}^*$*. It outputs either a message* $m \in \{0,1\}$.

The above algorithms satisfy the following conditions:

– ***Correctness.*** *For any security parameter* λ*, for any* $m \in \{0,1\}$*, and for any* $(x,w) \in R_{\mathcal{L}}$*, we have that*

$$\Pr\big[\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, x, m) \;:\; \mathsf{WE.dec}(\mathsf{ct}, w) = m\big] = 1 \enspace .$$

– ***Security.*** *For any non-uniform* PPT*adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any* $\lambda \in \mathbb{N}$*, and any* $x \notin \mathcal{L}$*, we have that*

$$\big\{\mathsf{WE.Enc}(1^\lambda, x, 0)\big\}_{\lambda \in \mathbb{N}, x \notin \mathcal{L}} \approx_c \big\{\mathsf{WE.Enc}(1^\lambda, x, 1)\big\}_{\lambda \in \mathbb{N}, x \notin \mathcal{L}} \enspace .$$

We note that the above scheme can be extended to encrypt strings, rather than just bits, by encrypting each bit independently. Witness encryption for all of NP can be constructed from IO for circuits [GGSW13].

2.6 Witness Indistinguishable Arguments

We define here delayed-input Interactive Arguments. We ignore the input security parameter 1^λ to each of the protocols in our description below and assume it is implicit.

Definition 9 (Delayed-Input Interactive Arguments). *An* n*-round delayed-input interactive protocol* (P, V) *for deciding a language* $\mathcal{L}$ *is an argument system for* $\mathcal{L}$ *that satisfies the following properties:*

– **Delayed-Input Completeness.** *For every security parameter* $\lambda \in \mathbb{N}$*, and any* $(x,w) \in R_{\mathcal{L}}$ *such that* $|x| \leq 2^\lambda$*,*

$$\Pr[\mathsf{Out}_{\mathsf{V}}\langle \mathsf{P}(x,w), \mathsf{V}(x)\rangle = 1] = 1 - \mathsf{negl}(\lambda). \enspace .$$

where the probability is over the randomness of P *and* V*. Moreover, the prover's algorithm initially takes as input only* 1^λ*, and the pair* (x,w) *is given to* P *only* in the beginning of the n'th round.

– **Delayed-Input Soundness.** *For any* PPT *cheating prover* P^* *that chooses* x^* *(adaptively)* after *the first* $n-1$ *messages, it holds that if* $x^* \notin \mathcal{L}$ *then*

$$\Pr[\mathsf{Out}_{\mathsf{V}}\langle \mathsf{P}^*(x^*), \mathsf{V}(x^*)\rangle = 1] = \mathsf{negl}(\lambda) \enspace .$$

where the probability is over the random coins of V.

We now define what it means for an argument to be witness indistinguishable (WI).

Definition 10 (Witness Indistinguishability). *A delayed-input interactive argument* (P, V) *for a language* $\mathcal{L}$ *is said to be witness indistinguishable if for every* PPT *algorithm* V^* *and every pair* (w_1, w_2) *such that* $R_{\mathcal{L}}(x, w_1) = 1$ *and* $R_{\mathcal{L}}(x, w_2) = 1$*,*

$$\left\{\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x, w_1), \mathsf{V}^*(x)\rangle\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w_1\in R_{\mathcal{L}}(x)}} \approx_c \left\{\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x, w_2), \mathsf{V}^*(x)\rangle\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w_2\in R_{\mathcal{L}}(x)}} .$$

where $\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x, w), \mathsf{V}^*(x)\rangle$ *denotes the view of the verifier during the execution of the protocol.*

Imported Theorem 1 ([LS91]). *Assuming non-interactive commitments there exists 3 round delayed-input witness indistinguishable proof systems.*

As in prior works, we rely on the public coin nature of the protocol in [LS91], i.e. the verifier messages in the protocol of [LS91] are simply random coins. See Section A.4 for a description of the protocol along with a high level overview of the properties.
Below we consider a few extensions of the above notion of WI.

Proofs and Statistical WI. WI can be strengthened in two ways: (i) *proofs:* by requiring that the soundness holds against *unbounded* cheating provers; and (ii) *statistical WI:* the views in Definition 10 are *statistically* indistinguishable.

Argument/Proof of Knowledge. Soundness of an argument/proof only implies that the verifier isn't convinced of a false statement. Proof of knowledge strengthens this requirement to state that if the verifier accepts, then it must be the case that the prover is in possession of the corresponding NP witness. We do not formally define the notion here, but note that it is formalized by the existence of a PPT extractor that has only oracle access (black-box access) to any convincing prover, and is able to "extract" a witness.

2.7 Non-interactive Witness Indistinguishability (NIWI)

We consider the non-interactive variant of the definition in Section 2.6, with the stronger property that it a *proof*, i.e. soundness holds against computationally unbounded cheating provers.

Definition 11 ([BOV03]). *A non-interactive witness-indistinguishable proof system* $\mathsf{NIWI} = (\mathsf{NIWI.Prov}, \mathsf{NIWI.Ver})$ *for an* NP *relation* $R_{\mathcal{L}}$ *consists of two polynomial-time algorithms:*

– *a probabilistic prover* $\mathsf{NIWI.Prov}(x, w, 1^\lambda)$ *that given an instance* x*, witness* w*, and security parameter* 1^λ*, produces a proof* π*.*

– *a deterministic verifier* $\mathsf{NIWI.Ver}(x, \pi)$ *that verifies the proof.*

We make the following requirements:

Completeness *for every* $\lambda \in \mathbb{N}, (x, w) \in R_{\mathcal{L}}$*,*

$$\Pr\left[\pi \leftarrow \mathsf{NIWI.Prov}(x, w, 1^\lambda) \ : \ \mathsf{NIWI.Ver}(x, \pi) = 1\right] = 1$$

Soundness *for every* $x \notin \mathcal{L}$ *and* $\pi \in \{0,1\}^*$*,*

$$\mathsf{NIWI.Ver}(x, \pi) = 0 \ .$$

Witness Indistinguishability *It holds that*

$$\left\{\mathsf{NIWI.Prov}(x, w_0, 1^\lambda)\right\}_{\substack{\lambda,x,\\ w_0,w_1}} \approx_c \left\{\mathsf{NIWI.Prov}(x, w_1, 1^\lambda)\right\}_{\substack{\lambda,x,\\ w_0,w_1}} ,$$

where $\lambda \in \mathbb{N}, x \in \{0,1\}^\lambda, w_0, w_1 \in R_{\mathcal{L}}(x)$.

As stated in the introduction, indistinguishability obfuscation implies non-interactive witness indistinguishable proofs, but with a randomized verifier [BP15], which is insufficient for our purpose. The verifier can be derandomized under a worst-case Nisan-Wigderson [NW94] type derandomization assumption [BV17]. Non-interactive witness indistinguishable proofs with a deterministic verifier are also known from standard assumptions on bilinear maps [GOS06].

2.8 Collision Resistance against Bounded Non-uniform Adversaries

We describe here the notion of keyless collision resistance against quasi-polynomial b-non-uniform adversaries, extending the definition in [BP04].

Syntax. A keyless collision resistance hash function is associated with an input function $\ell(\lambda) > \lambda$ and a polynomial time algorithm H such that $\mathsf{H}(1^\lambda, X)$ is a deterministic algorithm that takes as input an $X \in \{0,1\}^{\ell(\lambda)}$ and outputs a hash $Y \in \{0,1\}^\lambda$.

Definition 12. *We say that* H *is collision-resistant against quasi-polynomial adversaries if for any* b*-non-uniform probabilistic* $2^{\mathsf{poly}(\log\lambda)}$*-time* $\mathcal{A}$*, there exists a negligible function* negl*, such that for any* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[(x_1, x_2) \leftarrow \mathcal{A}(1^\lambda) \ : \ x_1 \neq x_2, \mathsf{H}(1^\lambda, x_1) = \mathsf{H}(1^\lambda, x_2)\right] \leq \mathsf{negl}(\lambda) \ .$$

2.9 Pseudorandom Generators

Definition 13 (Psedudorandom Generators). *A deterministic function* $\mathsf{PRG} : \{0,1\}^\lambda \rightarrow \{0,1\}^{p(\lambda)}$ *is called a pseudorandom generator (PRG) if:*

1. *(efficiency):* PRG *can be computed in polynomial time,*
2. *(expansion):* $p(\lambda) > \lambda$*,*
3. $\left\{x \leftarrow\$ \{0,1\}^\lambda : \mathsf{PRG}(x)\right\} \approx_c \left\{U_{p(\lambda)}\right\}$ *, where* $U_{p(\lambda)}$ *is the uniform distribution over* $p(\lambda)$ *bits.*

Pseudorandom generators with polynomial stretch as defined above can be constructed assuming one-way functions [HILL99].

2.10 Signature Scheme

An signature scheme [GMR88] consists of three polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$.

- Gen is PPT algorithm that takes as input 1^λ and generates a key and verification key. $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
- Sign is a PPT algorithm that computes the signature on a message m. $\sigma := \mathsf{Sign}(\mathsf{sk}, m)$.
- Vf is a deterministic algorithm verifies the signature using the verification key. $\mathsf{Vf}(\mathsf{vk}, m, \sigma)$ returns 0 or 1.

Definition 14. *A scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ *is a signature scheme that is existentially unforgeable against chosen message attacks if the following hold.*

Correctness *For every message $m \in \mathcal{M}$ (message space),*

$$\Pr\Big[\mathsf{Vf}(\mathsf{vk}, m, \sigma) = 1 \;:\; (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda), \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)\Big] = 1$$

Security *For any* PPT *adversary* $\mathcal{A}$

$$\Pr\left[\begin{array}{l} \mathcal{A} \text{ did not query } m \\ \mathsf{Vf}(\mathsf{vk}, m, \sigma) = 1 \end{array} : \begin{array}{l} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{vk}) \end{array}\right] < \mathsf{negl}(\lambda)$$

where $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}$ indicates that $\mathcal{A}$ has access to an oracle that returns the signature on the queried message m.

Digital signatures can be constructed assuming only one-way functions [GMR88, Rom90].

2.11 Secure Multiparty Computation

We provide the definition of MPC against malicious adversaries as well as (delayed) semi-malicious adversaries. Parts of this section have been taken verbatim from [Gol04].

A multi-party protocol is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality. The security of a protocol is defined with respect to a functionality f. In particular, let n denote the number of parties. An n-party functionality f is a (possibly randomized) mapping of n inputs to n outputs. A multi-party protocol with security parameter λ for computing a functionality f is a protocol running in time $\mathsf{poly}(\lambda)$ and satisfying the following correctness requirement: if parties $P_1, \ldots, P_n$ with inputs $(x_1, \ldots, x_n)$ respectively, all run an honest execution of the protocol, then the joint distribution of the outputs $y_1, \ldots, y_n$ of the parties is statistically close to $f(x_1, \ldots, x_n)$.

Defining Security. We assume that readers are familiar with standard simulation-based definitions of secure multi-party computation in the standalone setting. We provide a self-contained definition for completeness and refer to [Gol04] for a more complete description. The security of a protocol (with respect to a functionality f) is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of f by a trusted party. More concretely, it is required that for every adversary $\mathcal{A}$, which attacks the real execution of the protocol, there exist an adversary Sim, also referred to as a simulator, which can *achieve the same effect* in the ideal-world. Let's denote $\vec{x} = (x_1, \ldots, x_n)$.

The real execution In the real execution of the n-party protocol π for computing f is executed in the presence of an adversary $\mathcal{A}$. The honest parties follow the instructions of π. The adversary

$\mathcal{A}$ takes as input the security parameter k, the set $I \subset [n]$ of corrupted parties, the inputs of the corrupted parties, and an auxiliary input z. $\mathcal{A}$ sends all messages in place of corrupted parties and may follow an arbitrary polynomial-time strategy.

The interaction of $\mathcal{A}$ with a protocol π defines a random variable $\mathsf{REAL}_{\pi,\mathcal{A}(z),I}(k, \overrightarrow{x})$ whose value is determined by the coin tosses of the adversary and the honest players. This random variable contains the output of the adversary (which may be an arbitrary function of its view) as well as the outputs of the uncorrupted parties. We let $\mathsf{REAL}_{\pi,\mathcal{A}(z),I}$ denote the distribution ensemble $\{\mathsf{REAL}_{\pi,\mathcal{A}(z),I}(k, \overrightarrow{x})\}_{k\in\mathsf{N},\langle\overrightarrow{x},z\rangle\in\{0,1\}^*}$.

The ideal execution – security with abort . An ideal execution for a function f proceeds as follows:

– **Send inputs to the trusted party:** As before, the parties send their inputs to the trusted party, and we let x'_i denote the value sent by P_i. Once again, for a semi-honest adversary we require $x'_i = x_i$ for all $i \in I$.

– **Trusted party sends output to the adversary:** The trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i\in I}$ to the adversary.

– **Adversary instructs trust party to abort or continue:** This is formalized by having the adversary send either a continue or abort message to the trusted party. (A semi-honest adversary never aborts.) In the latter case, the trusted party sends to each uncorrupted party P_i its output value y_i. In the former case, the trusted party sends the special symbol $\perp$ to each uncorrupted party.

– **Outputs:** Sim outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

The interaction of Sim with the trusted party defines a random variable $\mathsf{IDEAL}_{f_\perp,\mathcal{A}(z)}(k, \overrightarrow{x})$ as above,and we let $\{\mathsf{IDEAL}_{f_\perp,\mathcal{A}(z),I}(k, \overrightarrow{x})\}_{k\in\mathsf{N},\langle\overrightarrow{x},z\rangle\in\{0,1\}^*}$ where the subscript "$\perp$" indicates that the adversary can abort computation of f.

Having defined the real and the ideal worlds, we now proceed to define our notion of security.

Definition 15. *Let k be the security parameter. Let f be an n-party randomized functionality, and π be an n-party protocol for $n \in \mathsf{N}$.*

1. *We say that π t-securely computes f in the presence of malicious (resp., semi-honest) adversaries if for every* PPT *adversary (resp., semi-honest adversary) $\mathcal{A}$ there exists a* PPT *adversary (resp., semi-honest adversary)* Sim *such that for any $I \subset [n]$ with $|I| \le t$ the following quantity is negligible:*
$$|Pr[\mathsf{REAL}_{\pi,\mathcal{A}(z),I}(k, \overrightarrow{x}) = 1] - Pr[\mathsf{IDEAL}_{f,\mathcal{A}(z),I}(k, \overrightarrow{x}) = 1]|$$
where $\overrightarrow{x} = \{x_i\}_{i\in[n]} \in \{0,1\}^*$ and $z \in \{0,1\}^*$.

2. *Similarly, π t-securely computes f* with abort *in the presence of malicious adversaries if for every* PPT *adversary $\mathcal{A}$ there exists a polynomial time adversary* Sim *such that for any $I \subset [n]$ with $|I| \le t$ the following quantity is negligible:*
$$|Pr[\mathsf{REAL}_{\pi,\mathcal{A}(z),I}(k, \overrightarrow{x}) = 1] - Pr[\mathsf{IDEAL}_{f_\perp,\mathcal{A}(z),I}(k, \overrightarrow{x}) = 1]|.$$

Security Against (Delayed) Semi-Malicious Adversaries We also define security against semi-malicious adversaries that are stronger than semi-honest adversaries. A semi-malicious adversary is modeled as an interactive Turing machine (ITM) which, in addition to the standard tapes, has a special witness tape. In each round of the protocol, whenever the adversary produces a new protocol message msg on behalf of some party P_k, it must also write to its special witness tape some pair (x, r) of input x and randomness r that explains its behavior. More specifically, all of the protocol messages sent by the adversary on behalf of P_k up to that point, including the new message m, must exactly match the honest protocol specification for P_k when executed with input x and randomness r. Note that the witnesses given in different rounds need not be consistent. Also, we assume that the attacker is rushing and hence may choose the message m and the witness (x, r) in each round adaptively, after seeing the protocol messages of the honest parties in that round (and all prior rounds). Lastly, the adversary may also choose to abort the execution on behalf of P_k in any step of the interaction.

A delayed semi-malicious adversary [BL18] is similar to semi-malicious adversary, except that it only needs to output the witness (i.e., a defense of honest behavior) in the second last round of the protocol. We refer the reader to [BL18] for a more detailed discussion.

Definition 16. *We say that a protocol π securely realizes f for (delayed) semi-malicious adversaries if it satisfies Definition 15 when we only quantify over all (delayed) semi-malicious adversaries A.*

2.12 Garbled Circuits

Definition 17 (Garbling Scheme). *A garbling scheme for circuits is a tuple of* PPT *algorithms* $\mathsf{GC} := (\mathsf{Gen}, \mathsf{Garble}, \mathsf{Eval})$ *such that"*

- $(\{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}}) \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{n})$*:* Garble *takes the security parameter* 1^λ *and length of input for the circuit as input and outputs a set of input labels* $\{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}}$.
- $\overline{C} \leftarrow \mathsf{Garble}(C, \{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}})$*:* Garble *takes as input a circuit* $C : \{0,1\}^{\mathsf{n}} \to \{0,1\}^{\mathsf{m}}$ *and a set of input labels* $\{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}}$ *and outputs the garbled circuit* $\overline{C}$.
- $y \leftarrow \mathsf{Eval}(\overline{C}, \mathsf{lab}^x)$*:* Eval *takes as input the garbled circuit* $\overline{C}$*, input labels* lab^x *corresponding to the input* $x \in \{0,1\}^{\mathsf{n}}$ *and outputs* $y \in \{0,1\}^{\mathsf{m}}$.

This garbling scheme satisfies the following properties:

1. ***Correctness:*** *For any circuit* C *and input* $x \in \{0,1\}^{\mathsf{n}}$*,*

$$\Pr[C(x) = \mathsf{Eval}(\overline{C}, \mathsf{lab}^x)] = 1$$

where $(\{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}}) \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{n})$ *and* $\overline{C} \leftarrow \mathsf{Garble}(C, \{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}})$.

2. ***Selective Security:*** *There exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{GC}}$ *such that, for any* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* $\mu(.)$ *such that,*

$$|\Pr[\mathsf{Experiment}_{\mathcal{A},\mathsf{Sim}_{\mathsf{GC}}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Experiment}_{\mathcal{A},\mathsf{Sim}_{\mathsf{GC}}}(1^\lambda, 1) = 1]| \leq \mu(\lambda)$$

where the experiment $\mathsf{Experiment}_{\mathcal{A},\mathsf{Sim}_{\mathsf{GC}}}(1^\lambda, b)$ *is defined as follows:*

(a) *The adversary $\mathcal{A}$ specifies the circuit C and an input $x \in \{0,1\}^{\mathsf{n}}$ and gets $\overline{C}$ and $\widehat{\mathsf{lab}}$, which are computed as follows:*

- ***If*** $b = 0$*:*
  - $(\{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}}) \leftarrow \mathsf{Gen}(1^\lambda, \mathsf{n})$
  - $\overline{C} \leftarrow \mathsf{Garble}(C, \{\mathsf{lab}^{w,b}\}_{w\in[\mathsf{n}],b\in\{0,1\}})$
- ***If*** $b = 1$*:*
  - $(\overline{C}, \widehat{\mathsf{lab}}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, C(x))$

(b) *The adversary outputs a bit b', which is the output of the experiment.*

Garbled circuits can be constructed from any symmetric key encryption scheme[Yao82a].

2.13 Oblivious Transfer

Here we follow [ORS15]. Oblivious Transfer (OT) is a two-party functionality $F_{\mathcal{OT}}$, in which a sender S holds a pair of strings (l_0, l_1), and a receiver R holds a bit b, and wants to obtain the string l_b. The security requirement for the $F_{\mathcal{OT}}$ functionality is that any malicious receiver does not learn anything about the string l_{1-b} and any malicious sender does not learn which string has been transferred. This security requirement is formalized via the ideal/real world paradigm. In the ideal world, the functionality is implemented by a trusted party that takes the inputs from S and R and provides the output to R and is therefore secure by definition. A real world protocol Π securely realizes the ideal $F_{\mathcal{OT}}$ functionalities, if the following two conditions hold. (a) Security against a malicious receiver: the output of any malicious receiver $R^\star$ running one execution of Π with an honest sender S can be simulated by a PPT simulator $\mathcal{S}$ that has only access to the ideal world functionality $F_{\mathcal{OT}}$ and oracle access to $R^\star$. (b) Security against a malicious sender. The joint view of the output of any malicious sender $S^\star$ running one execution of Π with R and the output of R can be simulated by a PPT simulator $\mathcal{S}$ that has only access to the ideal world functionality $F_{\mathcal{OT}}$ and oracle access to $S^\star$. We consider the weaker definition of OT introduced in [ORS15] which is referred as *one-sided simulatable OT*. In this we do not demand the existence of a simulator against a malicious sender, but we only require that a malicious sender cannot distinguish whether the honest receiver is playing with bit 0 or 1. That is, we require that for any PPT malicious sender $S^\star$ the view of $S^\star$ executing Π with the receiver R playing with bit 0 is computationally indistinguishable from the view of $S^\star$ where R is playing with the bit 1.

Definition 18 ([ORS15]). *Let $F_{\mathcal{OT}}$ be the Oblivious Transfer functionality as described previously. We say that a protocol Π securely computes $F_{\mathcal{OT}}$ with one-sided simulation if the following holds:*

1. *For every non-uniform* PPT *adversary $R^\star$ controlling the receiver in the real model, there exists a non-uniform* PPT *adversary $\mathcal{S}$ for the ideal model such that:* $\{\mathsf{REAL}_{\Pi,R^\star(z)}(1^\lambda)\}_{z\in\{0,1\}^\lambda} \approx \mathsf{IDEAL}_{F_{\mathcal{OT}},\mathcal{S}(z)}(1^\lambda)\}_{z\in\{0,1\}^\lambda}$*,*

   where $\mathsf{REAL}_{\Pi,R^\star(z)}(1^\lambda)$ *denotes the distribution of the output of the adversary $R^\star$ (controlling the receiver) after a real execution of protocol Π, where the sender S has inputs l_0, l_1 and the receiver has input b.* $\mathsf{IDEAL}_{f,\mathcal{S}(z)}(1^\lambda)$ *denotes the analogous distribution in an ideal execution with a trusted party that computes $F_{\mathcal{OT}}$ for the parties and hands the output to the receiver.*

2. *For every non-uniform* PPT *adversary $S^\star$ controlling the sender it holds that:*
   $\{\mathsf{View}^R_{\Pi,S^\star(z)}(l_0, l_1, 0)\}_{z\in\{0,1\}^\star} \approx \{\mathsf{View}^R_{\Pi,S^\star(z)}(l_0, l_1, 1)\}_{z\in\{0,1\}^\star}$
   where $\mathsf{View}^R_{\Pi,S^\star(z)}$ *denotes the view of adversary $S^\star$ after a real execution of protocol Π with the honest receiver R.*

2.14 Blockchain Model

In this section, we describe the model for the Blockchain, that will be used in Chapter 5.

Blockchains. In a blockchain protocol, the goal of all parties is to maintain a global ordered set of records that are referred to as *blocks*. New blocks can only be added using a special mining procedure that simulates a puzzle-solving race between participants and can be run by any party (called miner) executing the blockchain protocol. Presently, two broad categories of puzzles are used: Proof-of-Work (PoW) and Proof-of-Stake (PoS).

Following the works of [KZZ16, BMTZ17, BGK+18], we model the blockchain as a global ledger $\mathcal{G}_{\mathsf{ledger}}$ that internally keeps a state state which is the sequence of all the blocks in the ledger. Parties interact with the ledger by making one of many queries described by the functionality.

We reproduce here the ledger functionality described in [BMTZ17] with a few minor modifications to be described subsequent to the description.

The ledger maintains a central and unique permanent state denoted by state. When data/transactions are sent to $\mathcal{G}_{\mathsf{ledger}}$, they are validated using a Validate predicate and added to a buffer buffer. The buffer is meant to indicate those transactions that are not sufficiently deep to become permanent. The Blockify function creates a block including some transactions from buffer and extends state. The decision of when the state is extended is left to the adversary. The adversary proposes a next block candidate NxtBC containing the transactions from the buffer it wants included in the state. An empty NxtBC is used to indicate that the adversary does not want the state to be updated at the current clock tick. To restrict the behavior of the adversary, there is a ledger algorithm ExtendPolicy that enforces a state-update policy restriction. See appendix A.1 for further discussion on the ExtendPolicy.

Each registered party can see the state, but is guaranteed only a sufficiently long prefix of it. This is implemented by monotonically increasing pointers pt_i, defining the prefix $\mathsf{state}|_{\mathsf{pt}_i}$, for each party that the adversary can manipulate with some restrictions. This can be viewed as a sliding window over the state, wherein the adversary can only set pointers to be within this window starting from the head of state. The size of the sliding window is denoted by windowSize. It should be noted that the prefix view guarantees that the value at position k will appear in position i in every party's state.

A party is said to be desynchronized if the party recently registered or recently got de-registered from the clock. At this point, due to network delays, the adversary can make the parties believe in any value of the state up until the party gets messages from the network. This time period is denoted by the parameter Delay, wherein the desynchronized parties are practically under the control of the adversary. A timed honest input sequence $\overrightarrow{\mathcal{I}}^T_H$, is a vector of the form $((x_1, P_1, \tau_1), \cdots, (x_m, P_m, \tau_m))$, used to denote the inputs received by the parties from the environment, where P_i is the player that received the input and τ_i was the time of the clock when the environment handed the input to P_i. The ledger uses the function predict-time to ensure that the ideal world execution advances with the same pace (relative to the clock) as the protocol does. $\overrightarrow{\tau}_{\mathsf{state}}$ denotes the block-insertion times vector, which lists the times each block was inserted into state.

Functionality $\mathcal{G}_{\mathsf{ledger}}$

$\mathcal{G}_{\mathsf{ledger}}$ is parameterized by found algorithms, Validate, ExtendPolicy, Blockify, and predict-time: windowSize, Delay$\in \mathbb{N}$. The functionality manages variables state, NxtBCbuffer, τ_L, and $\overrightarrow{\tau}_{\mathsf{state}}$ as described above. The variables are initialized as follows: state $:= \overrightarrow{\tau}_{\mathsf{state}} :=$ NxtBC $:= \varepsilon$, buffer $:= \emptyset$, $\tau_L = 0$.

The functionality maintains the set of registered parties $\mathcal{P}$, the subset of honest parties $\mathcal{H} \subseteq \mathcal{P}$ and the subset of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$. The sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered at the ledger, if it is registered with the clock already then it added to the party sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded if the current time $\tau_L > 0$, it is also added to $\mathcal{P}_{DS}$. Similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ (and therefore also from $\mathcal{P}_{DS}$ or $\mathcal{H}$). The ledger maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$.

For each party $P_i \in \mathcal{P}$ the functionality maintains a pointer pt_i (initially set to 1) and a current-state view $\mathsf{state}_i := \varepsilon$ (initially set to empty). The functionality also keeps track of the timed honest-input sequence in a vector $\overrightarrow{\mathcal{I}}_H^T$ (initially $\overrightarrow{\mathcal{I}}_H^T := \varepsilon$)

Upon receiving any input I from any party or from the adversary, send $(\mathsf{CLOCK\text{-}READ}, \mathsf{sid}_C)$ to $\mathcal{G}_{\mathsf{clock}}$ and upon receiving the response $(\mathsf{CLOCK\text{-}READ}, \mathsf{sid}_C, \tau)$, set $\tau_L := \tau$ and do the following:

1. Let $\widehat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered continuously since time $\tau' < \tau_L - \mathsf{Delay}$ (to both ledger and clock). Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \widehat{\mathcal{P}}$.

2. If I was received from an honest party $P_i \in \mathcal{P}$:

   (a) Set $\overrightarrow{\mathcal{I}}_H^T := \overrightarrow{\mathcal{I}}_H^T || (I, P_i, \tau_L)$;

   (b) Compute

   $$\overrightarrow{N} = (\overrightarrow{N}_1, \cdots, \overrightarrow{N}_\ell) := \mathsf{ExtendPolicy}\left(\overrightarrow{\mathcal{I}}_H^T, \mathsf{state}, \mathsf{NxtBC}, \mathsf{buffer}, \overrightarrow{\tau}_{\mathsf{state}}\right)$$

   and if $\overrightarrow{N} \neq \varepsilon$ set state $:=$ state$||\mathsf{Blockify}(\overrightarrow{N}_1)|| \cdots ||\mathsf{Blockify}(\overrightarrow{N}_\ell)$ and $\overrightarrow{\tau}_{\mathsf{state}} := \overrightarrow{\tau}_{\mathsf{state}} || \tau_L^\ell$ where $\tau_L^\ell = \tau_L || \cdots || \tau_L$.

   (c) If there exists $P_j \in \mathcal{H} \setminus \mathcal{P}_{DS}$ such that $|\mathsf{state}| - \mathsf{pt}_j >$ windowSize or $\mathsf{pt}_j < |\mathsf{state}_j|$, then set $\mathsf{pt}_k := |\mathsf{state}|$ for all $P_k \in \mathcal{H} \setminus \mathcal{P}_{DS}$.

   (d) If $\overrightarrow{N} \neq \varepsilon$, send (state) to $\mathcal{A}$; else send (I, P_i, τ_L) to $\mathcal{A}$

3. Depending on the above input I and its sender's ID, $\mathcal{G}_{\mathsf{ledger}}$ executes the corresponding code from the following list:

   – *Submitting data:*
   If $I = (\mathsf{SUBMIT}, \mathsf{sid}, x)$ and is received from a party $P_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of corrupted party P_i) do the following

   (a) Choose a unique identifier uid and set $y := (x, \mathsf{uid}, \tau_L, P_i)$

(b) $\mathsf{buffer} := \mathsf{buffer} \cup \{y\}$.

(c) Send (SUBMIT, y) to $\mathcal{A}$ if not received from $\mathcal{A}$.

– *Reading the state:*
If $I = (\mathsf{READ}, \mathsf{sid})$ is received from a party $P_i \in \mathcal{P}$ then set $\mathsf{state}_i := \mathsf{state}|_{\min\{\mathsf{pt}_i, |\mathsf{state}|\}}$ and return $(\mathsf{READ}, \mathsf{sid}, \mathsf{state}_i)$ to the requestor. If the the requestor is $\mathcal{A}$ then send $(\mathsf{state}, \mathsf{buffer})$.

– *Maintain the ledger state:*
If $I = (\mathsf{MAINTAIN\text{-}LEDGER}, \mathsf{sid})$ is received by an honest $P_i \in \mathcal{P}$ and $\mathsf{predict\text{-}time}(\overrightarrow{\mathcal{I}}_H^T) = \widetilde{\tau} > \tau_L$ then send $(\mathsf{CLOCK\text{-}UPDATE}, \mathsf{sid}_C)$ to $\mathcal{G}_{\mathsf{clock}}$. Else send I to $\mathcal{A}$.

– *The adversary proposing the next block:*
If $I = (\mathsf{NEXT\text{-}BLOCK}, \mathsf{hflag}, (\mathsf{uid}_1, \cdots, \mathsf{uid}_\ell))$ is sent from the adversary, update NxtBC as follows:

(a) Set $\mathsf{listOfUid} \leftarrow \varepsilon$

(b) For $i \in [\ell]$, if there exists $y := (x, \mathsf{uid}, \tau_L, P_i) \in \mathsf{buffer}$ with ID $\mathsf{uid} = \mathsf{uid}_i$ then set $\mathsf{listOfUid} := \mathsf{listOfUid}||\mathsf{uid}_i$.

(c) Finally, set $\mathsf{NxtBC} := \mathsf{NxtBC}||(\mathsf{hflag}, \mathsf{listOfUid})$.

– *The adversary setting state-slackness:*
If $I = (\mathsf{SET\text{-}SLACK}, (P_{i_1}, \widehat{\mathsf{pt}_{i_1}}), \cdots, (P_{i_\ell}, \widehat{\mathsf{pt}_{i_\ell}}))$ with $\{P_{i_1}, \cdots, P_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary, do the following:

(a) If $\forall j \in [\ell] : |\mathsf{state}| - \widehat{\mathsf{pt}}_{i_j} \leq \mathsf{windowSize}$ and $\widehat{\mathsf{pt}}_{i_1} \geq |\mathsf{state}_{i_j}|$, set $\mathsf{pt}_{i_j} := \widehat{\mathsf{pt}}_{i_j}$ for every $j \in [\ell]$.

(b) Otherwise set $\mathsf{pt}_{i_j} := |\mathsf{state}|$ for all $j \in [\ell]$

– *The adversary setting the state for desynchronized parties:*
If $I = (\mathsf{DESYNC\text{-}STATE}, (P_{i_1}, \mathsf{state}'_{i_1}), \cdots, (P_{i_1}, \mathsf{state}'_{i_\ell}))$ with $\{P_{i_1}, \cdots, P_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary, set set $\mathsf{state}_{i_j} := \mathsf{state}'_{i_j}$ for every $j \in [\ell]$.

The work of Badertscher et al [BMTZ17] show that under appropriate assumptions, Bitcoin realizes the ledger functionality described enforcing the ExtendPolicy described in appendix A.1. For convenience we've made a few syntactic changes to the $\mathcal{G}_{\mathsf{ledger}}$ functionality as described in [BMTZ17]:

– Firstly, the Validate predicate is not relevant in our setting since parties will use ledger to post data, and these should be trivially validated. Hence, we've abstracted out the Validate predicate from the description of the model.

– We require that the adversary cannot invalidate data sent by other parties, thereby denying data from ever making it on to the ledger. For transactions, the adversary can invalidate honest transactions. This can be remedied using a strong variant of $\mathcal{G}_{\mathsf{ledger}}$ described in [BMTZ17].

– Every time that the size of the state increases, the adversary is notified of the new state by $\mathcal{G}_{\mathsf{ledger}}$.

The changed functionality the same properties of the ideal $\mathcal{G}_{\mathsf{ledger}}$ functionality as described in [BMTZ17].

Remarks. We point out a few properties of the $\mathcal{G}_{\mathsf{ledger}}$ functionality and its use case in our setting.

- As described in [BMTZ17], we can achieve a strong liveness guarantee by slightly modifying the above ledger functionality which guarantees that posted information will make it on to the view of other parties within $\Delta := 4 \cdot \mathsf{windowSize}$ number of blocks (relative to the view of the submitting party).
- There are occasions wherein we will run parallel executions of the adversary, and one thread will be assigned to be the main execution thread while the others will be denoted as 'look-ahead" threads. In an effort to make the adversary oblivious to rewinding, we cannot allow messages from these "look-ahead" threads to make its way to $\mathcal{G}_{\mathsf{ledger}}$. Drop messages sent by the adversary to $\mathcal{G}_{\mathsf{ledger}}$ and will have to abort the thread if $\mathcal{G}_{\mathsf{ledger}}$ sends a state with an increased size.
- We require that for a READ query, buffer is efficiently simulatable, while state is not. This is a reasonable assumption to make given that the state indicates the permanent component of the blockchain, and simulating this would requiring forging the state. On the other hand, the buffer consists of outstanding queries from both honest and adversarial parties. From the description of $\mathcal{G}_{\mathsf{ledger}}$, each time a SUBMIT query is made to $\mathcal{G}_{\mathsf{ledger}}$, the information is passed along to the adversary, and the adversary's own outstanding queries are known. Looking ahead, a READ query can be answered without making a query to $\mathcal{G}_{\mathsf{ledger}}$. The honest outstanding queries are replayed on each thread since they could not have changed across threads, while the adversarial queries local to that thread are known to the simulator.
- We wait for Delay time before the start of any protocol to ensure all parties are synchronized. Moving ahead, for simplicity of exposition, the notion of de-synchronised parties is ignored.
- While the works of [KZZ16, BMTZ17, BGK$^+$18] use $\mathcal{G}_{\mathsf{clock}}$ functionality, we do not require parties to have access to a clock and can consider this to be local to $\mathcal{G}_{\mathsf{ledger}}$. In fact our positive results do not rely on parties having access to a clock.
- Additionally, we require that a locally initialized $\mathcal{G}_{\mathsf{ledger}}$ is efficiently simulatable to any adversary that does not have additional access to the global $\mathcal{G}_{\mathsf{ledger}}$. These local $\mathcal{G}_{\mathsf{ledger}}$ will be useful in establishing certain properties of our protocol.

Blockchain active (BCA) adversaries. Consider an adversary that has access to $\mathcal{G}_{\mathsf{ledger}}$, and thus can post to and access the state (the entire blockchain) at any time. In fact its strategy in any protocol may be a function of the state. We refer to any such adversary that actively uses the $\mathcal{G}_{\mathsf{ledger}}$ as a *blockchain active adversary (BCA)*.

Simulation in the Blockchain-hybrid model. Moving ahead, we interchangeably use blockchain-hybrid and $\mathcal{G}_{\mathsf{ledger}}$-hybrid, while preferring the later for our formal descriptions. A simulator has the same power as other parties while accessing the global functionality $\mathcal{G}_{\mathsf{ledger}}$. In addition, it acts as an interface between the party and $\mathcal{G}_{\mathsf{ledger}}$, and thus can choose what messages between the party and the functionality it wants delivered. This is unlike the setting considered in [CGJ$^+$17, GG17] where the simulator has control of the blockchain, and thus can "rewind" the blockchain by discarding and re-creating blocks. This is reminiscent of the difference between simulation in Universal Composability (UC) framework [Can01] and simulation in the global UC framework [CDPW07, CJS14, HPV16].

Our simulator can use arbitrary polynomial amount of parallelism. Although arbitrary, the polynomial is fixed in advance. We will use this modeling to run parallel invocations of the adversary by making copies.

At this point we would like to emphasize the need for considering this model for the simulator. We start off by mentioning that any party can use the state obtained from $\mathcal{G}_{\mathsf{ledger}}$ as the basis for its execution. Importantly, the adversary's view is now no longer determined solely by the message it receives from the simulator since the $\mathcal{G}_{\mathsf{ledger}}$ state gives it an additional auxiliary input. In the plain model, if we wanted to rewind the adversary back to a specific point in the execution, we could restart the adversary and send the same messages up to the specific point. And we were guaranteed that the adversary's responses would be identical. But now since the adversary has access to $\mathcal{G}_{\mathsf{ledger}}$, its responses could depend on the state of $\mathcal{G}_{\mathsf{ledger}}$.

Let us consider such an adversary. Now when the simulator tries to restart the adversary, suppose the state has expanded since. Even if the simulator provides the same messages as a previous execution, the adversary's behavior now may be drastically different and of potentially no use to the simulator. The simulator could ensure identical behavior by providing it the earlier truncated view of the state, but moving forward with this execution would be problematic since any message that the adversary wants to post will no longer appear on the state within the promised time period, and thus the adversary will notice that the $\mathcal{G}_{\mathsf{ledger}}$ no longer follows the model specified. Thus it is imperative that executions are run in parallel to ensure that views across multiple threads are identical if the same inputs are provided.

The above modeling is crucial for rewinding when we prove security of our protocols. We will work with this modeling unless otherwise specified. Looking ahead, our construction of the zero-knowledge proof in the non-black-box setting will use a modified variant of this model.

Security. Since the distinguisher attempting to distinguish between views of the adversary in the real and simulated setting has access to $\mathcal{G}_{\mathsf{ledger}}$, the simulator cannot create an isolated view of $\mathcal{G}_{\mathsf{ledger}}$ for the adversary. But as it turns out, the ability to initialize a local $\mathcal{G}_{\mathsf{ledger}}$ is a useful property useful in certain situations that we will leverage in our work.

Protocols in the plain model are a reference to any protocol that does not require its participants to interact with $\mathcal{G}_{\mathsf{ledger}}$ in any form. These protocols are proven secure without considering the presence of $\mathcal{G}_{\mathsf{ledger}}$. Given such a protocol, a blockchain active adversary may try to leverage access to this global functionality $\mathcal{G}_{\mathsf{ledger}}$ to gain undue advantage over the setting where it did not have such access. We are interested in such adversaries since we want to see how the security of known protocols or primitives fare when the adversary has access to the $\mathcal{G}_{\mathsf{ledger}}$.

Chapter 3

Deterministic Prover Zero-Knowledge

3.1 Overview

We now give an overview of the main ideas and techniques behind our results. For providing context to our results, we provide a high level overview of the [GO94] impossibility in Section A.3.

The Deterministic-Prover Zero-Knowledge Protocol. Our starting point is the protocol against honest verifiers based on witness encryption [FNV17]. In their protocol, the verifier simply sends a witness encryption of a random message u with respect to the statement $x \in \mathcal{L}$ to be proven, and expects to get u back from the prover. Witness encryption guarantees that a prover that has a corresponding witness w, can obtain u and convince the verifier. However, if the statement is false, namely $x \notin \mathcal{L}$, u is hidden, and soundness is guaranteed.

While honest verifiers are easy to simulate in this scheme, it is not clear how to simulate malicious verifiers. For this purpose, we aim to add to the protocol *a trapdoor way of obtaining* u. A simulator that has the code of the verifier should be able to extract the message u. In contrast, a malicious prover who doesn't have the code (specifically, the verifier's randomness) should still fail to find u when $x \notin \mathcal{L}$.

Explainable Verifiers. To explain the idea behind the protocol in its simplest form, let us start by assuming that the first message v sent by verifier to the prover is always *explainable* [BKP19]. That is, there exist honest verifier coins r that explain this message as an honest verifier message $\mathsf{v} = \mathsf{V}(x; r)$. The difference between this setting and the honest verifier setting is that the explaining coins r may be distributed arbitrarily and also computationally hard to find.

Our basic idea is for the verifier to send the prover yet another witness encryption of u where *the witness is basically the malicious verifier code* V^*. Our realization of this idea is inspired by Barak's uniform simulation technique [Bar01]. Let b be the given bound on the description size of the verifier including its (bounded) auxiliary input hardwired. Then, the honest verifier samples a long random string $R \leftarrow \{0,1\}^{b+2\lambda}$. Then in addition to the witness encryption of u under the statement $x \in \mathcal{L}$, it sends a witness encryption of u under the statement:

"There exists a program Π of size $b + \lambda$ (namely short) that outputs R."

To argue that the protocol remains sound, we note that except with negligible probability $2^{-\lambda}$ over the choice of r, such a short program does not exist. In this case, witness encryption will guarantee that u remains hidden and soundness is preserved. Furthermore, a simulator in possession of the b-size code V^* of the malicious verifier can now use it to simulate. Specifically, let ℓ be the amount of coins r^* used by V^*, then the simulator will sample r^* using a pseudorandom generator that stretches a seed s^* of length $\approx \lambda$ to a pseudorandom r^* of length ℓ. Looking at the string R that $\mathsf{V}^*(x; r^*)$ outputs, the simulator now possesses a size-$(b + \lambda)$ program Π that computes R — the code of V^* with the seed s^* hardwired. This in turn leads to valid simulation.

Witness Encryption for Unbounded NP Relations and IO. One thing to notice about the latter protocol is that in fact the existence of program Π that outputs R is not an NP statement, unless we restrict the running time of Π to some specific polynomial. However, while the non-uniform description size (equivalently, auxiliary input size) of the malicious verifier V^* is a-priori bounded, its running time is not bounded by any specific polynomial.

Accordingly, we need a strong notion of witness encryption for unbounded non-deterministic relations. Specifically, encryption under a statement x should take time polynomial in $|x|$ (and the security parameter), and not depend on the time required to verify a witness for x. In contrast, decrypting with a witness w should take time proportional to the time required to verify w. Such witness encryption schemes directly follow from known indistinguishability obfuscation (IO) schemes for Turing Machines, which are in turn constructed from subexponentially-secure IO for circuits [KLW15, BCG+18, GS18a].

Malicious Verifiers. Having constructed a protocol against explainable verifiers, we use compilers from the literature to turn it into a protocol against arbitrary verifiers. These compilers use non-interactive witness-indistinguishable proofs (NIWIs) in order to enforce explainable behavior on the verifier's side. Being non-interactive verifying, these proofs require no randomness from the honest zero-knowledge prover.

The first such compiler [BKP19] works for $\mathsf{NP} \cap \mathsf{co}\text{-}\mathsf{NP}$ and requires no additional hardness assumptions. The second compiler is taken from [BP04] (where it was used in a different context) and relies in addition on keyless hash functions that are collision resistant against attackers with bounded auxiliary input and quasipolynomial running time, as well as subexponentially secure commitments (which in turn follow from subexponentially secure IO and one-way functions). In the body, we reanalyze these compilers to show that they can be used to enforce *robust explainability*, which roughly means that the verifier's messages are almost always explainable on any efficiently samplable distribution on its coins, a property required for our simulation strategy. See more details in Section 3.3.

From Deterministic-Prover Zero Knowledge to Predictable Arguments. We now explain how deterministic-prover zero knowledge implies predictable arguments, which in turn imply witness encryption (as well as the additional properties stated in Corollary 2). We start with an oversimplified transformation that captures the main idea, but does not fully work, and then explain how to augment it. This oversimplified transformation, in fact, starts from deterministic-prover *honest-verifier* zero knowledge.

Let (P, V) be our argument, and let Sim be the honest-verifier simulator. We consider a new verifier V' that works as follows. It applies the simulator $\mathsf{Sim}(x)$ to obtain simulated randomness $\tilde{r}$

for the honest verifier along with simulated prover messages $\tilde{\mathsf{p}}_1, \ldots, \tilde{\mathsf{p}}_k$. The verifier V' then certifies that the prover messages lead to an accepting transcript with respect to the verifier coins r. If they do not lead to an accepting transcript, V' automatically rejects; otherwise, it interacts with the prover, and rejects the moment it receives a message $\mathsf{p}_i \neq \tilde{\mathsf{p}}_i$. The described protocol is predictable by construction. Also, since we do not change the honest prover, it is zero knowledge against the same class of verifiers as the original protocol. We now turn to argue that the protocol is complete and sound.

To see that the protocol has almost perfect completeness, consider a distinguisher that has the witness w hardwired. Given a transcript $\mathsf{p}_1, \ldots, \mathsf{p}_k$ and verifier coins r, it can perfectly emulate a conversation between the deterministic prover $\mathsf{P}(x, w)$ and honest verifier $\mathsf{V}(x; r)$ and check whether the produced prover messages are consistent with the input transcript $\mathsf{p}_1, \ldots, \mathsf{p}_k$, and that the transcript is accepting. We deduce that with overwhelming probability the simulator produces simulated messages $\tilde{\mathsf{p}}_1, \ldots, \tilde{\mathsf{p}}_k$, and randomness r, such that the honest prover would produce the same messages, and the transcript will be accepting. To see soundness, notice that if the simulated coins r are pseudorandom and the simulated prover messages $\tilde{\mathsf{p}}_1, \ldots, \tilde{\mathsf{p}}_k$ are accepting, then by the soundness of the original protocol (P, V), it should be hard for an efficient prover to produce messages consistent with $\tilde{\mathsf{p}}_1, \ldots, \tilde{\mathsf{p}}_k$ (or with any accepting transcript).

Above, when proving soundness we actually made the implicit assumption that the honest verifier simulator $\mathsf{Sim}(x)$ produces pseudorandom verifier coins, even when given a no instance $x \notin \mathcal{L}$. Indeed, with respect to random, or pseudorandom, coins, we can argue that it is hard to find accepting transcripts. While this is a natural property, it does not follow directly from honest verifier zero knowledge. To circumvent this difficulty, we slightly augment the above transformation, while relying on zero-knowledge against (not necessarily honest) bounded-auxiliary-input verifiers.

Specifically, the verifier V' uses a pseudorandom generator to sample coins r for the honest verifier V, using a short seed s. It then applies the same procedure as above, except that it runs the simulator $\mathsf{Sim}(\mathsf{V}_s, x)$ for the deterministic verifier V_s that first derives the coins r from the seed s, and then applies V. By choosing an appropriate pseudorandom generator, we can guarantee that the non-uniform description of V_s is short enough. This transformation guarantees that the simulated coins are pseudorandom, even for a no instance, and allows the above proof to go through. The necessity of zero-knowledge to hold even for verifiers that are not necessarily honest comes from the fact that our description of V_s deviates from the honest verifier strategy. We give another construction of predictable arguments from deterministic-prover arguments that are only honest-verifier zero knowledge, provided that the arguments supports expressive enough languages. See Section 3.6 for details.

A Word on Two-Message Laconic Arguments. As stated in Corollary 2, we use the implication to predictable arguments to also derive that any deterministic-prover zero knowledge argument for bounded-auxiliary-input verifiers can be made two message and laconic. This corollary is obtained by applying as is general transformations on predictable arguments [FNV17]. The only thing we need to prove is that these transformations preserve zero knowledge. The only hurdle here is that the mentioned transformations involve parallel repetition for the sake of soundness amplification. We observe that (unlike many-round zero knowledge) two-message zero knowledge against bounded-auxiliary-input verifiers is closed under parallel repetition.

On Deterministic Prover Zero-Knowledge *Proofs*. While our results (in conjunction with prior works) provide a complete picture of deterministic zero-knowledge arguments, our results do not have any bearing on deterministic zero-knowledge *proofs*, where soundness is required to hold against unbounded provers. Completing the picture for *proofs* remains an interesting open problem.

3.2 Definitions

In this work, we will consider PPT machines with both, bounded and unbounded non-uniform auxiliary input. For simplicity of notation, rather than considering explicit auxiliary input in our definitions, we consider two basic notions of non-uniformity. The corresponding zero knowledge definition will in particular capture the auxiliary input setting. See Remark 1. The first notion, of *non-uniform* PPT has already been described in Chapter 2. Here we focus on the second notion, *b-non-uniform* PPT.

b**-non-uniform** PPT: These are PPT machines with non-uniform description of size $b(\lambda)$ and arbitrary polynomial running time (possibly larger than $b(\lambda)$). Formally, a b-non-uniform PPT $\mathsf{M} = \{\mathsf{M}_\lambda\}_\lambda$ is a family of probabilistic Turing machines (one for each λ), where $|\mathsf{M}_\lambda| \leq b(\lambda)$ and there exists a polynomial poly, such that the running time of M_λ is bounded by $\mathsf{poly}(\lambda)$.

In both of the notions, we often omit from M_λ the subscript λ when it is clear from the context. If we simply say a PPT machine, we mean a uniform one.

3.2.1 Deterministic-Prover Zero Knowledge Against Bounded-Auxiliary-Input Verifiers

We define the notion of deterministic-prover zero-knowledge arguments against verifiers with bounded auxiliary-input (DPZK). We shall denote by $\mathsf{Out}_A\langle A(a), B(b)\rangle$ the output of party A on execution of the protocol between A with input a, and B with input b. By $\mathsf{View}_A\langle A(a), B(b)\rangle$, we denote the view of party A consisting of the protocol transcript along with its random tape.

Definition 19. *An interactive protocol* (P, V) *between a* ***deterministic*** *polynomial time prover* P *and* PPT *verifier* V*, for a language* $\mathcal{L}$ *is a deterministic prover* b*-bounded-auxiliary-input zero knowledge argument if the following holds.*

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

$$\Pr[\mathsf{Out}_\mathsf{V}\langle \mathsf{P}(x, w), \mathsf{V}(x)\rangle = 1] = 1 \ .$$

Soundness: *For any non-uniform* PPT P^**, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $x \in \{0,1\}^\lambda \setminus \mathcal{L}$*,*

$$\Pr[\mathsf{Out}_\mathsf{V}\langle \mathsf{P}^*, \mathsf{V}(x)\rangle = 1] \leq \mathsf{negl}(\lambda) \ .$$

Zero Knowledge: *There exists a* PPT *simulator* Sim*, such that for every* b*-non-uniform* PPT *verifier* V^* *of running time at most* $t(\lambda)$*,*

$$\left\{\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x, w), \mathsf{V}^*\rangle\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_\mathcal{L}(x)}} \approx_c \left\{\mathsf{Sim}(\mathsf{V}^*, 1^t, x)\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_\mathcal{L}(x)}} \ .$$

Remark 1 (Universal Simulation). *In the above definition, there exists one universal simulator* Sim *that gets the code of the verifier as input. We note that this definition is known [GO94] to imply the alternative definition of (bounded) auxiliary-input zero knowledge that requires that any for any* t*-time* V^* *there is a* PPT *simulator* $\mathsf{Sim}_{\mathsf{V}^*}$ *such that given (bounded) auxiliary input* z*,* $\mathsf{Sim}_{\mathsf{V}^*}(x, z, 1^t)$ *simulates* $\mathsf{V}^*(z)$*.*

See Section A.2 for a discussion about the various notions of zero-knowledge.

3.2.2 Explainable Verifiers

We define here the a variant of the notion of explainable verifiers [BKP19] called robustly-explainable verifiers. Roughly speaking, explainable verifiers are ones whose messages almost always lie in the support of the honest verifier messages (or are abort). Robustly-explainable verifiers are such where this occurs when they use random coins sampled from an arbitrary efficient sampler (and not necessarily the uniform distribution).

Definition 20 (Explainable Message). *Let* $\langle \mathsf{P}, \mathsf{V} \rangle$ *be a two-message protocol. We say that a given message* m *is explainable with respect to* x*, if there exist honest verifier coins* r *such that* $m \in \{V(x; r), \perp\}$*.*

Definition 21 (Robustly-Explainable Verifier). *Let* $\langle \mathsf{P}, \mathsf{V} \rangle$ *be a protocol. A* b*-non-uniform* PPT *verifier* V^* *using* $\ell(\lambda)$ *random coins is robustly-explainable if for any* PPT *sampler* R *on* $\ell(\lambda)$ *bits, there exists a negligible* $\mathsf{negl}(\lambda)$ *such that for any* $\lambda \in \mathbb{N}$ *and* $x \in \lambda$*,*

$$\Pr\left[r \leftarrow R(1^\lambda), m = \mathsf{V}^*(x; r) : m \text{ is explainable}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

3.3 A Deterministic-Prover Zero-Knowledge Protocol

In this section we present our deterministic prover zero knowledge (DPZK) protocol. As explained in the introduction, we start by describing the protocol for robustly-explainable verifiers. We then show how to compile this protocol to one that is secure against malicious verifiers.

3.3.1 DPZK for Robustly-Explainable Verifiers

We use the following components for the deterministic prover zero knowledge (DPZK) protocol for an NP language $\mathcal{L}$ against b-non-uniform explainable verifiers.

– A witness encryption scheme $(\mathsf{WE.Enc}, \mathsf{WE.dec})$ for language $\mathcal{L}$.

– An indistinguishability obfuscation (IO) scheme iOM for Turing Machines (TM).

Additionally, we will use the machine described below that outputs the hardcoded secret u given as input the description of a "short" Turing machine that outputs a hardcoded public value R.

```
Machine: Prog

Hardcoded: $R, u$
Input: $\mathsf{M} \in \{0,1\}^{\rho(\lambda)}$

if M outputs $R$
        output $u$
else
        output $\perp$
```

In what follows, let $\rho(\lambda) = b(\lambda) + \lambda + \omega(1)$, $\ell(\lambda) = \rho(\lambda) + \lambda$. The protocol is described in Figure 3.1. We prove the properties of the protocol below.

Completeness. Completeness follows from the correctness of witness encryption.

Soundness. We now prove that the above protocol is sound against computationally bounded provers.

Proposition 1. *Assuming security of the indistinguishability obfuscation scheme and the witness encryption scheme, the protocol is sound.*

Proof. We consider a sequence of hybrids transitioning from the real protocol to an ideal protocol where the probability that the prover convinces the verifier of accepting is clearly negligible.

Hyb_0: This is the real protocol.

Hyb_1: In this hybrid, we modify the program Prog to Prog$'$ that always output $\perp$.

By our choice of parameters and a union bound, the probability that there exists a machine $\mathsf{M} \in \{0,1\}^{\rho}$ that outputs R is at most $2^{\rho-\ell} = 2^{-\lambda}$. Therefore, except with negligible probability Prog and Prog$'$ are functionally equivalent. The indistinguishability of Hyb_1 and Hyb_0 then follows from the indistinguishability of the IO scheme.

Hyb_2: In this hybrid, we additionally change the ciphertext ct of the witness encryption scheme to be the encryption of 0.

Since $x \notin \mathcal{L}$, the indistinguishability between Hyb_2 and Hyb_1 follows from the security of the witness encryption scheme.

It is left to observe that in Hyb_2 the prover obtains no information about u, and thus convinces the verifier with probability at most $2^{-\lambda}$. □

Zero Knowledge. We prove

Proposition 2. *Assuming the existence of pseudorandom generators, the protocol is zero knowledge against b-non-uniform verifiers.*

Protocol: DPZK for robustly-explainable verifiers

Common input: Input $x \in \mathcal{L}$, security parameter 1^λ

P**'s auxiliary input:** witness w such that $(x, w) \in R_{\mathcal{L}}$

1. Verifier V computes the first message as
   (a) $R \leftarrow\!\!\$ \{0,1\}^{\ell(\lambda)}$
   (b) $t := \lambda^{\log \lambda}$
   (c) $u \leftarrow\!\!\$ \{0,1\}^{\lambda}$
   (d) $\widetilde{\mathsf{Prog}} \leftarrow \mathsf{iOM}\left(1^\lambda, 1^\rho, 1^{\log t}, \mathsf{Prog}[R, u]\right)$
   (e) $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(u, x)$

   send $(R, \mathsf{ct}, \widetilde{\mathsf{Prog}})$ to the prover P.

2. Prover P computes the second message as
   (a) $\widetilde{u} := \mathsf{WE.dec}(\mathsf{ct}, x, w)$

   send $\widetilde{u}$ to the verifier V.

3. Verifier V performs the check
   (a) if $\widetilde{u} = u$, accept. Else, reject.

Figure 3.1: Deterministic prover zero-knowledge for robustly-explainable verifiers.

Proof. We describe the simulation strategy below. In what follows V^* is a b-non-uniform malicious verifier of polynomial running time at most $t(\lambda)$. Additionally, let k be the amount of random coins r^* used by V^*. The simulator Sim will use a PRG $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^k$.

$\underline{\mathsf{Sim}(\mathsf{V}^*, 1^t, x)}$:

1. Construct verifier V^*_s that has the seed s hardwired. V^*_s computes $\mathsf{PRG}(s)$ and uses it as random coins for V^*. Additionally, V^*_s truncates V^*'s output to R.

2. Initialize V^* with random coins $\mathsf{PRG}(s)$.

3. Given $\widetilde{\mathsf{Prog}}$ from V^*, use the description of V^*_s as input to $\widetilde{\mathsf{Prog}}$ and obtain u.

4. u is then used as the simulated prover message, along with verifier randomness $\mathsf{PRG}(s)$.

First, consider an execution between the prover and augmented verifier $\langle \mathsf{P}(x,w), \mathsf{V}^*_s \rangle$, and let v and p denote the verifier and prover messages in such an execution. Then by pseudorandomness of PRG,

$$\mathsf{View}_{\mathsf{V}^*} \langle \mathsf{P}(x,w), \mathsf{V}^* \rangle \approx_c \mathsf{p}, \mathsf{PRG}(s) \ .$$

Next, by the fact that V^* is robustly explainable, we know that except with negligible probability, $\mathsf{v} = (R, \mathsf{ct}, \widetilde{\mathsf{Prog}})$ is explainable; namely, has the structure prescribed by the honest verifier algorithm. Noting that V^*_s is a program of length $b + \lambda + O(1) < \rho(\lambda)$ and running time at most $t(\lambda)$ that outputs R. By the fact that v is explainable, $\widetilde{\mathsf{Prog}}(\mathsf{V}^*_s) = \mathsf{WE.dec}(\mathsf{ct}, x, w)$. It follows that

$$\mathsf{p}, \mathsf{PRG}(s) \approx_s \mathsf{Sim}(\mathsf{V}^*, 1^t, x) \ ,$$

and overall

$$\mathsf{View}_{\mathsf{V}^*} \langle \mathsf{P}(x,w), \mathsf{V}^* \rangle \approx_c \mathsf{Sim}(\mathsf{V}^*, 1^t, x) \ ,$$

as required.

□

3.3.2 From Explainable to Malicious Verifiers

In this section we give generic compilers going from robust-explainable to malicious verifiers. These compilers were constructed in [BKP19] where they were used to enforce explainability and in [BP04] where they were used in a different context. We prove that these compilers, in fact, enforce robust explainability. The statements, and correspondingly the underlying assumptions, change based on whether we want a DPZK for $\mathsf{NP} \cap \mathsf{co\text{-}NP}$, or for all of NP. We discuss the two cases separately.

3.3.2.1 DPZK for $\mathsf{NP} \cap \mathsf{co\text{-}NP}$

We consider languages $\mathcal{L} \in \mathsf{NP} \cap \mathsf{co\text{-}NP}$, which in turn means that in addition to relation $R_{\mathcal{L}}$, there is also a NP-relation $R_{\overline{\mathcal{L}}}$ to certify that a statement $x \notin \mathcal{L}$.
We use the following primitives in our construction:

- A two-message deterministic-prover zero-knowledge (DPZK) protocol $(e\mathsf{P}, e\mathsf{V})$ secure against robustly-explainable verifiers. Let the verifier and prover messages be denoted by v and p, respectively.
- A non-interactive witness indistinguishable proof (NIWI) $(\mathsf{NIWI.Prov}, \mathsf{NIWI.Ver})$ for the language

$$\mathcal{L}_{\mathsf{NIWI}} = \left\{ (\mathsf{v}, x) \;\middle|\; \exists (r, \bar{w}) \text{ s.t. } \mathsf{v} = e\mathsf{V}(x; r) \text{ OR } R_{\overline{\mathcal{L}}}(x, \bar{w}) = 1 \right\} \ ,$$

  namely, either the verifier's message is explainable, or the statement is not in the language. Henceforth, we shall refer to the second half of the 'OR' statement, that the statement is not in the language, to be the *trapdoor statement*.

Protocol: (P,V) for $\mathcal{L} \in \mathsf{NP} \cap \mathsf{co\text{-}NP}$

Common input: Input $x \in \mathcal{L}$, security parameter 1^λ

P**'s auxiliary input:** witness w such that $(x, w) \in R_{\mathcal{L}}$

1. Verifier V computes the first message as
   (a) $r \leftarrow\$ \{0,1\}^{p(n)}$
   (b) $\mathsf{v} := e\mathsf{V}(x; r)$
   (c) $x_{\mathsf{NIWI}} := (\mathsf{v}, x)$
   (d) $w_{\mathsf{NIWI}} := (r, \perp)$
   (e) $\mathsf{wi} \leftarrow \mathsf{NIWI.Prov}(x_{\mathsf{NIWI}}, w_{\mathsf{NIWI}})$

   send $(\mathsf{v}, \mathsf{wi})$ to the prover P.

2. Prover P computes the second message as
   (a) $\widetilde{x}_{\mathsf{NIWI}} := (\mathsf{v}, x)$
   (b) if $\mathsf{NIWI.Ver}(\widetilde{x}_{\mathsf{NIWI}}, \mathsf{wi}) \neq 1$, output $\perp$.
   (c) $\mathsf{p} := e\mathsf{P}(x, w, \mathsf{v})$.

   send p to the verifier V.

3. Verifier V performs the check
   (a) if $e\mathsf{V}(x, \mathsf{p}; r) = 1$, accept. Else, reject.

Figure 3.2: Deterministic-prover zero knowledge for $\mathcal{L} \in \mathsf{NP} \cap \mathsf{co\text{-}NP}$.

The protocol is presented in Figure 3.2.

Completeness. Completeness follows directly from the completeness of the underlying protocol and the NIWI proof.

Zero Knowledge. We show how any b-non-uniform malicious verifier V^* for the above protocol can be converted to a robustly-explainable $b + O(1)$-non-uniform verifier against the original protocol.

Claim 1. *There exist an efficient simulator* S *and a verifier* $e\mathsf{V}^*$ *such that*

1. *$e\mathsf{V}^*$ is a robustly explainable verifier against $\langle e\mathsf{P}, e\mathsf{V}\rangle$.*

2. *$e\mathsf{V}^*$ is $(b + O(1))$-non-uniform and efficiently constructable from $e\mathsf{V}^*$.*

3. *For every $x \in \mathcal{L}$,*
$$\mathsf{View}_{\mathsf{V}^*}\langle\mathsf{P}(x,w),\mathsf{V}^*\rangle \equiv \mathsf{S}(\mathsf{View}_{e\mathsf{V}^*}\langle e\mathsf{P}(x,w), e\mathsf{V}^*\rangle) \ .$$

Proof. We construct $\mathsf{S}, e\mathsf{V}^*$.

$e\mathsf{V}^*$:

1. Emulates V^* and obtains (v,wi).
2. If wi is not a valid proof for the statement (v, x), send $e\mathsf{P}$ the message $\perp$.
3. Else, send $e\mathsf{P}$ v, and get p.
4. Complete emulation of V^* with message p.

S:

1. Outputs the randomness of the emulated V^* (can be derived from the randomness of $e\mathsf{V}^*$,
2. as well as the received prover message p (possibly $\perp$).

The third property asserted in the claim follows by construction of $\mathsf{S}, e\mathsf{V}^*$ and the fact that the prover P checks on its own whether the verifier's proof is accepting. It is left to see that $e\mathsf{V}^*$ is robustly explainable, $(b + O(1))$-non-uniform, and efficiently constructable from V^*. Robust explainability follows directly by the (unconditional) soundness of the NIWI — $e\mathsf{V}^*$ either outputs an explainable message or $\perp$. $(b+O(1))$-non-uniformity and efficient construction follow from the fact that V^* is b-non-uniform and $e\mathsf{V}^*$ uses it as a black box and described by the four code lines above.

□

Claim 1 directly gives rise to a zero knowledge Sim for the protocol (P,V). In what follows, let $e\mathsf{Sim}$ be the simulator of the underlying DPZK protocol against robustly-explainable verifiers.

$\mathsf{Sim}(\mathsf{V}^*, 1^t, x)$:

1. Construct the explainable verifier $e\mathsf{V}^*$.
2. Output $\mathsf{S}(e\mathsf{Sim}(e\mathsf{V}^*, 1^t, x)$.

The validity of the simulator Sim follows directly from that of $e\mathsf{Sim}$ and Claim 1.

Soundness. For soundness, we show that any cheating prover P^* breaking the soundness of the above protocol, can be converted into a prover $e\mathsf{P}^*$ that breaks the soundness of the underlying protocol. $e\mathsf{P}^*$ will have the witness $\bar{w}$ for $x \notin \mathcal{L}$ hardwired.

$e\mathsf{P}^*$:

1. Obtain message v from $e\mathsf{V}$.

2. Use $\bar{w}$ as the witness to compute the NIWI proof wi.
3. Emulate P^* with $(\mathsf{v}, \mathsf{wi})$ and obtain p.
4. Send p to the verifier $e\mathsf{V}$.

First note that since $\mathcal{L} \in \mathsf{NP} \cap \mathsf{co\text{-}NP}$, the statement $x \notin \mathcal{L}$ has a witness $\bar{w}$ as required. The only difference in the views of P^* and its emulated version in $e\mathsf{P}^*$ is in the NIWI proof. From the witness indistinguishability of the NIWI, P^*'s success probability does not change by more than a negligible amount.

3.3.2.2 DPZK for all of NP

As mentioned to in the introduction, for the case of NP, we require stronger primitives. Specifically, we use the following primitives for our construction:

– A two round deterministic prover zero knowledge (DPZK) protocol $(e\mathsf{P}, e\mathsf{V})$ secure against robustly-explainable verifiers. Let the verifier and prover messages be denoted by v and p, respectively.

– A non-interactive commitment scheme Com with perfect binding and computational hiding. Additionally, as mentioned earlier, we require that the plaintext underlying a commitment can be extracted in quasi-polynomial time. Such commitments can be constructed from subexponentially-secure injective one-way functions (which in turn can be constructed from subexponential IO and one-way functions).

– A keyless collision-resistant hash function H secure against $(b + O(1))$-non-uniform quasi-polynomial time adversaries.

– A non-interactive witness-indistinguishable proof (NIWI) $(\mathsf{NIWI.Prov}, \mathsf{NIWI.Ver})$ for the language

$$\mathcal{L}_{\mathsf{NIWI}} = \Big\{ (\mathsf{v}, x, c) \;\Big|\; \exists (r, r_{\mathsf{Com}}, x_1, x_2) \text{ s.t. } \mathsf{v} = e\mathsf{V}(x; r) \text{ OR } \big(c = \mathsf{Com}((x_1, x_2); r_{\mathsf{Com}}) \wedge x_1 \neq x_2 \wedge \mathsf{H}(1^\lambda, x_1) = \mathsf{H}(1^\lambda, x_2)\big) \Big\} ,$$

namely, either the verifier's message is explainable, or the commitment sent by the verifier contains a collision in H. As before, we shall refer to the second half of the 'OR' statement as the *trapdoor statement*.

The protocol is presented in Figure 3.3.

Completeness. Follows directly from the completeness of the underlying protocol and the NIWI.

Zero Knowledge. For zero knowledge, we follow the same strategy as in the previous subsection and show how any b-non-uniform verifier V^* for the above protocol can be converted into a robustly-explainable $(b + O(1))$-non-uniform verifier against the original protocol.

We argue that Claim 1 also holds for this protocol with the exact same S and $e\mathsf{V}^*$. The only difference is in the proof of robust explainability of the verifier $e\mathsf{V}^*$, which is based on complexity leveraging.

Protocol: (P,V) **for** $\mathcal{L} \in \mathsf{NP}$

Common input: Input $x \in \mathcal{L}$, security parameter 1^λ

P**'s auxiliary input:** witness w such that $(x, w) \in R_{\mathcal{L}}$

1. Verifier V compute the first message as
   (a) $r \leftarrow\$ \{0,1\}^{p(n)}$
   (b) $c := \mathsf{Com}(0; r_{\mathsf{Com}})$
   (c) $\mathsf{v} := e\mathsf{V}(x; r)$
   (d) $x_{\mathsf{NIWI}} := (x, \mathsf{v}, c, \mathsf{H})$
   (e) $w_{\mathsf{NIWI}} := (r, \perp, \perp)$
   (f) $\mathsf{wi} \leftarrow \mathsf{NIWI.Prov}(x_{\mathsf{NIWI}}, w_{\mathsf{NIWI}})$.

   send $(\mathsf{v}, \mathsf{wi}, c)$ to the prover P.

2. Prover P computes the second message as
   (a) $\widetilde{x}_{\mathsf{NIWI}} := (x, \mathsf{v}, c, \mathsf{H})$
   (b) if $\mathsf{NIWI.Ver}(\widetilde{x}_{\mathsf{NIWI}}, \mathsf{wi}) \neq 1$, output $\perp$.
   (c) $\mathsf{p} := e\mathsf{P}(x, w, \mathsf{v})$.

   send p to the verifier V.

3. Verifier V performs the check
   (a) if $e\mathsf{V}(x, \mathsf{p}; r) = 1$, accept. Else, reject.

Figure 3.3: Deterministic prover zero-knowledge for $\mathcal{L} \in \mathsf{NP}$.

Robust Explainability of $e\mathsf{V}^*$. Fix some PPT sampler R for coins for $e\mathsf{V}^*$ and assume toward contradiction that with noticeable probability it outputs a message v that is not explainable when initialized with random coins sampled using R. We show that there exists a $(b + O(1))$-non-uniform quasi-polynomial time attacker that finds a collision in H. Recall the $e\mathsf{V}^*$ only outputs a non-$\perp$ message provided that the emulated V^* produces a valid NIWI. By the unconditional soundness of the NIWI, it follows that whenever $e\mathsf{V}^*$ outputs a non-explainable message, it must be that c is a valid commitment to a collision in H. This collision is then be extracted from the commitment in quasi-polynomial time. Note that the corresponding collision finder can be described by $e\mathsf{V}^*$ and R, which have non-uniform description of size $b + O(1)$. □

Zero knowledge of (P, V) now follows from that of $(e\mathsf{P}, e\mathsf{V})$ and the existence of S and $e\mathsf{V}^*$, exactly as in the previous subsection.

Soundness. We show that any cheating prover P^* breaking the soundness of the above protocol, can be converted into a prover $e\mathsf{P}^*$ that breaks the soundness of the underlying robustly-explainable protocol. The reduction is similar to that in the previous subsection with some required changed. $e\mathsf{P}^*$ will have a collision (x_1, x_2) as (part of the) witness for the *trapdoor statement* hardwired in its code.

$\underline{e\mathsf{P}^*:}$

1. Obtain message v from $e\mathsf{V}$.
2. Compute $c = \mathsf{Com}(x_1, x_2; r_{\mathsf{Com}})$.
3. Use $(x_1, x_2, r_{\mathsf{Com}})$ as the witness to compute the NIWI proof wi.
4. Emulate P^* with $(\mathsf{v}, \mathsf{wi})$ and obtain p.
5. Send p to the verifier $e\mathsf{V}$.

The difference in the views of P^* and its emulated version in $e\mathsf{P}^*$ is the commitment to (x_1, x_2) rather than zero, and in the witness used for the NIWI proof. Using the hiding of the commitment (against non-uniform PPT attackers) and the witness indistinguishability of the NIWI, P^*'s success probability does not change by more than a negligible amount.

Remark 2. *We emphasize that for soundness, we require that all the underlying primitives to are secure against non-uniform adversaries since our soundness reduction is non-uniform.*

3.4 Predictable Arguments and DPZK

In this section, we show that any deterministic-prover zero-knowledge (DPZK) argument against bounded-non-uniform verifier can be made *predictable*. The notion of predictable arguments was introduced in [FNV17], where it is in particular shown to imply witness encryption. In the next section, we address additional properties of DPZK that follow from this connection.

We start by recalling the definition of predictable arguments (PA) [FNV17]. While they also address predictable argument of knowledge, we restrict attention to predictable arguments that are only sound.

Definition 22 (Predictable Argument). *A ρ-round predictable argument is an argument specified by a tuple of algorithms* $(\mathsf{Chal}, \mathsf{Resp})$ *as described below:*

1. *The verifier* $\mathsf{PA.V}$ *samples* $\left(\overrightarrow{c}, \overrightarrow{b}\right) \leftarrow \mathsf{Chal}(1^\lambda, x)$*, where* $\overrightarrow{c} := (c_1, \cdots, c_\rho)$ *and* $\overrightarrow{b} := (b_1, \cdots, b_\rho)$.
2. *For all $i \in [\rho]$ in increasing sequence:*
   (a) $\mathsf{PA.V}$ *sends c_i to the* $\mathsf{PA.P}$*;*

(b) *The prover* PA.P *computes* $a_i := \mathsf{Resp}(1^\lambda, x, w, c_1, \cdots, c_i)$ *and sends* a_i *to* PA.V.

(c) PA.V *checks if* $a_i = b_i$*, and returns 0 otherwise.*

3. *If all challenges are answered correctly,* PA.V *returns 1.*

The protocol is required to satisfy:

Correctness. *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $x \in \mathcal{L}$ *such that* $R_L(x, w) = 1$*, we have*

$$\Pr[\mathsf{Out}_{\mathsf{PA.V}}\langle \mathsf{PA.P}(x, w), \mathsf{PA.V}(x)\rangle = 1] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Soundness. *For any non-uniform* PPT *prover* P^**, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $x \notin \mathcal{L}$*,*

$$\Pr[\langle \mathsf{PA.P}^*, \mathsf{PA.V}(x)\rangle = 1] \leq \mathsf{negl}(\lambda) \ .$$

A **deterministic-prover zero-knowledge predictable argument (PA-DPZK)** is a deterministic-prover zero-knowledge argument that is also a predictable argument.

We prove the following:

Theorem 11. *Let* (P, V) *be a deterministic-prover zero-knowledge argument for* $\mathcal{L}$ *against bounded-non-uniform verifiers. There exists a verifier* V' *such that* $(\mathsf{P}, \mathsf{V}')$ *is a predictable argument.*

Note that since we do not change the honest prover P it follows that $(\mathsf{P}, \mathsf{V}')$ is also deterministic-prover zero knowledge against the same class of verifiers.

Relying on the following result by Faonio, Nielsen, and Venturi,

Theorem 12 ([FNV17]). *If there exists a Predictable Argument (PA) for a language* $\mathcal{L}$*, then there exists a witness encryption scheme for* $\mathcal{L}$*.*

our theorem holds for all $\lambda^{\Omega(1)}$-non-uniform verifiers, and we deduce

Corollary 3. *If there exists a deterministic-prover zero-knowledge argument for* $\mathcal{L}$ *against* $\lambda^{\Omega(1)}$*-non-uniform verifiers, then there exists a witness encryption scheme for* $\mathcal{L}$*.*

We now proceed with the proof.

Proof of Theorem 11. Let (P, V) be a ρ-round DPZK argument for $\mathcal{L}$ against b-non-uniform verifiers, for $b(\lambda) \geq 2\lambda + \omega(1)$. Let $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^\ell$ be a pseudorandom generator, where $\ell(\lambda)$ is the amount of coins used by V. For a given seed $s \in \{0,1\}^\lambda$, we define the deterministic verifier $\mathsf{V}_s(x)$ that derives coins $r = \mathsf{PRG}(s)$ for V then emulates $\mathsf{V}(x; r)$.

The transformed verifier V' is presented in Figure 3.4.
First, note that the protocol satisfies the structural requirement of a predictable argument. We now move to prove completeness and soundness with respect to the new verifier V'.

Completeness. We show that $(\mathsf{P}, \mathsf{V}')$ is complete based on (a) the completeness of $(\mathsf{P}, \mathsf{V}')$; (b) zero knowledge of $(\mathsf{P}, \mathsf{V}')$; and (c) pseudorandomness of PRG.

Fix any statement $x \in \mathcal{L}$ and corresponding prover witness w. We need to show that in an interaction $\langle \mathsf{P}(x, w), \mathsf{V}'(x)\rangle$, V' rejects with negligible probability. First, by the completeness of (P, V) and the pseudorandomness of PRG, an interaction $\langle \mathsf{P}(x, w), \mathsf{V}_s(x))\rangle$ is accepting except with negligible

The New Verifier V'

Input: x, security parameter 1^λ

1. Sample $s \leftarrow\$ \{0,1\}^\lambda$ and construct V_s.
2. Sample $\{\widetilde{\mathsf{p}}_i\}_{i=1}^{\rho} \leftarrow \mathsf{Sim}(\mathsf{V}_s, 1^t, x)$, where t is the running time of V_s.
3. Emulate an execution of $\mathsf{V}_s(x)$ with prover messages $\{\widetilde{\mathsf{p}}_i\}_{i=1}^{\rho}$; let $\{\widetilde{\mathsf{v}}_i\}_{i=1}^{\rho}$ be the resulting verifier messages.
4. If the verifier V_s rejects in the above execution, reject.
5. Proceed interacting with the prover P: at each round $i \in [\rho]$:
   - send $\mathsf{v_i}(= \widetilde{\mathsf{v}}_i)$ to P,
   - if the prover answers with $\mathsf{p}_i = \widetilde{\mathsf{p}}_i$, proceed to the next round,
   - else, reject.
6. Accept.

Figure 3.4: The Verifier in the Predictable Protocol

probability over the choice of s. Noting that $\mathsf{V}_s(x)$ is b-non-uniform, we can invoke zero knowledge, to deduce that the simulated prover messages $\{\widetilde{\mathsf{p}}_i\}_{i=1}^{\rho}$ make V_s accept with overwhelming probability over the choice of s.

We next argue that the deterministic prover $\mathsf{P}(x, w)$ produces messages $\{\mathsf{p}_i = \widetilde{\mathsf{p}}_i\}_{i=1}^{\rho}$ with overwhelming probability (over the coins of Sim that sampled them). This again follows from zero knowledge. Indeed, we can consider a zero-knowledge distinguisher that has (x, w, s) hardwired, and given messages p_i emulates a conversation of the deterministic $\mathsf{P}(x, w)$ with $\mathsf{V}_s(x)$, and outputs "real" if the corresponding prover messages coincide with p_i, or "simulated" otherwise. If the simulated messages $\widetilde{\mathsf{p}}_i$ are inconsistent with the real prover messages p_i, the distinguisher will tell them apart.

Soundness. We show that $(\mathsf{P}, \mathsf{V}')$ is sound based on (a) the pseudorandomness of PRG; and (b) the soundness of (P, V).

First, note that by pseudorandomness the protocol $(\mathsf{P}, \mathsf{V}_s)$ where s is chosen at random is also sound, since otherwise a cheating prover can be directly used to distinguish real verifier coins form pseudorandom ones. Next, note that any cheating prover against V' directly implies a cheating prover against V_s (for a random s) by construction. Indeed, V' emulates V_s and accepts only when the prover is consistent with a simulated strategy $\widetilde{\mathsf{p}}_i$ that convinces V_s.[1] Soundness follows.

[1]Here we implicitly rely on the fact that the simulator produces an accepting transcript for the deterministic verifier V_s.

$\square$

3.5 Round Reduction and Laconicity

Faonio, Nielsen, and Venturi [FNV17] proved that the round complexity of any predictable argument can be collapsed to one (two messages overall) and that any predictable argument can be made laconic — namely, the prover message is a single bit (or more generally ℓ bits to achieve soundness $\approx 2^{-\ell}$). In this section, we review their transformations and show that they preserve zero knowledge against bounded-non-uniform verifiers. As a corollary of this and the previous section, we deduce that any deterministic-prover zero knowledge argument against bounded-non-uniform verifiers can be collapsed to one round and made laconic.

3.5.1 Round Reduction

We start by recalling the round-collapsing transformation from [FNV17]. In what follows, let $(\mathsf{P}', \mathsf{V}')$ be a ρ-round predictable argument, the following transformation provides a one round predictable argument (P, V) with a large soundness error (to be dealt with later on). Roughly, the verifier randomly chooses a "cut-off" point i^* for the underlying protocol, and sends all the verifier messages up to, and including, the i^*-th round verifier message to the prover. Being a predictable argument, the verifier is able to do so without requiring the corresponding intermediate prover messages. The prover then iteratively computes the response for each round of the underlying protocol and send over all the prover messages with the verifier accepting if and only if each prover messages corresponds to the predicted prover message.

In [FNV17], it is proven that this protocol has soundness error at most $1 - \rho^{-1} + \mathsf{negl}(\lambda)$. The protocol is then repeated $\omega(\rho \log \lambda)$ times to achieve negligible soundness, using a parallel repetition theorem for one round arguments [BIN97].

Proposition 3. *The round collapsing transformation preserves zero knowledge against b-non-uniform verifiers.*

Proof. We prove the proposition in two steps. First, we show that the transformation in Figure 3.5 preserves zero-knowledge. Then we show that two-message zero-knowledge against bounded-non-uniform adversaries is closed under parallel repetition.

To prove the first part, let V^* be a b-non-uniform verifier. We show the following claim.

Claim 2. *There exist an efficient simulator S and a verifier V'^* against $\langle \mathsf{P}', \mathsf{V}' \rangle$ such that*

1. *V'^* is $(b + O(1))$-non-uniform and efficiently constructable from V^*.*

2. *For every $x \in \mathcal{L}$,*
$$\mathsf{View}_{\mathsf{V}^*} \langle \mathsf{P}(x, w), \mathsf{V}^* \rangle \equiv \mathsf{S}(\mathsf{View}_{\mathsf{V}'^*} \langle \mathsf{P}'(x, w), \mathsf{V}'^* \rangle) \ .$$

This claim gives rise to a simulator Sim for (P, V), which simply invokes Sim' of (P, V) on V'^* and then invokes S.

The deterministic nature of the verifier ensures that the simulator cannot manipulate the verifier's randomness and therefore must produce an accepting transcript is consistent with $\mathsf{V}(\cdot; \mathsf{PRG}(s))$.

Protocol: One Round (P, V)

Common input: Input $x \in \mathcal{L}$, security parameter 1^λ

P**'s auxiliary input:** witness w such that $(x, w) \in R_{\mathcal{L}}$

1. Verifier V
   (a) Samples $i^* \leftarrow\$ [\rho]$,
   (b) Samples $(\mathsf{v}_i, \mathsf{b}_i)_{i\in[\rho]} \leftarrow\$ \mathsf{V}(x)$.
   (c) Sends $\mathsf{v}_1, \cdots, \mathsf{v}_{i^*}$ to the prover P.
2. Prover P
   (a) For each $i \in [i^*]$, compute $\mathsf{p}_i := \mathsf{P}(x, w, \{\mathsf{v}_j\}_{j\in[i]})$.
   (b) Send $\mathsf{p}_1, \cdots, \mathsf{p}_{i^*}$ to the verifier V.
3. Verifier V accepts if and only if for all $j \in [i^*]$, $\mathsf{p}_j = \mathsf{b}_j$.

Figure 3.5: Round collapsing transformation.

Proof of Claim. We construct $\mathsf{S}, \mathsf{V}'^*$.

<u>V'^*:</u>

1. Emulates V^* and obtains $(\mathsf{v}_1, \ldots, \mathsf{v}_{i^*})$.
2. At each round $i \in [i^*]$, forward v_i to P'.
3. Abort after round i^*.

<u>S:</u>

1. Outputs the randomness of the emulated V^* (can be derived from the randomness of V'^*),
2. as well as the received prover messages $\mathsf{p}_1, \ldots, \mathsf{p}_{i^*}$.

The second property asserted in the claim follows by construction of $\mathsf{S}, \mathsf{V}'^*$ and the construction of P from P' in Figure 3.5. It is left to see that V'^* is $(b + O(1))$-non-uniform and efficiently constructable from V^*. $(b + O(1))$-non-uniformity and efficient construction follow from the fact that V^* is b-non-uniform and V'^* uses it as a black box and described by the three code lines above. □

We now prove that closure under parallel repetition.

Claim 3. *For any two-message zero knowledge system* (P, V) *against* b*-non-uniform verifiers and a any polynomial* ℓ*, the* ℓ*-fold parallel repetition* $(\mathsf{P}_{\otimes\ell}, \mathsf{V}_{\otimes\ell})$ *is zero knowledge against* $(b - O(\log\lambda))$*-non-uniform verifiers.*

Proof. In what follows, let Sim be the simulator for the original argument (P, V), and let $\mathsf{V}^*_{\otimes\ell}$ be any $(b - \lambda - O(\log\lambda))$-non-uniform verifier of polynomial running time $t(\lambda)$. We now describe the simulator $\mathsf{Sim}_{\otimes\ell}$ for $(\mathsf{P}_{\otimes\ell}, \mathsf{V}_{\otimes\ell})$. The simulator will use a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^k$, where k is the amount of coins used by $\mathsf{V}^*_{\otimes\ell}$.

$\underline{\mathsf{Sim}_{\otimes\ell}(\mathsf{V}^*_{\otimes\ell}, 1^t, x)}$:

1. Sample a $s \leftarrow\$ \{0,1\}^\lambda$.
2. For each $i \in [\ell]$:
   (a) Construct the deterministic verifier $\mathsf{V}^*_{s,i}$ that first derives coins $\mathsf{PRG}(s)$, uses them to emulate $\mathsf{V}^*_{\otimes\ell}$, obtains $\mathsf{v}_1, \dots, \mathsf{v}_\ell$, and outputs v_i. Let $t' = t + \mathsf{poly}(\lambda)$ be a bound on its running time.
   (b) Sample $\widetilde{\mathsf{p}}_i \leftarrow\$ \mathsf{Sim}(\mathsf{V}^*_{s,i}, 1^{t'}, x)$.
3. Output $\widetilde{\mathsf{p}}_1, \dots, \widetilde{\mathsf{p}}_\ell, \mathsf{PRG}(s)$.

We now prove the validity of $\mathsf{Sim}_{\otimes\ell}$. First, consider an execution between the prover $\mathsf{P}(x, w)$ and verifier $\mathsf{V}^*_s = (\mathsf{V}^*_{s,1}, \dots, \mathsf{V}^*_{s,\ell})$, and let $\mathsf{p}_1, \dots, \mathsf{p}_\ell$ denote the prover messages in such an execution. Then by pseudorandomness of PRG,

$$\mathsf{View}_{\mathsf{V}^*_{\otimes\ell}}\langle \mathsf{P}(x,w), \mathsf{V}^*_{\otimes\ell}\rangle \approx_c \mathsf{p}_1, \dots, \mathsf{p}_\ell, \mathsf{PRG}(s) \enspace .$$

Noting that $\mathsf{V}^*_{s,i}$ is a program of length at most b and running time at most $t'(\lambda)$, we can invoke the simulation guarantee (P, V). Specifically, we can deduce that

$$\mathsf{p}_1, \dots, \mathsf{p}_\ell, \mathsf{PRG}(s) \approx_c \widetilde{\mathsf{p}}_1, \dots, \widetilde{\mathsf{p}}_\ell, \mathsf{PRG}(s) \enspace .$$

This can be shown by a standard hybrid argument and follows from the fact that $\mathsf{p}_i \approx_c \widetilde{\mathsf{p}}_i = \mathsf{Sim}(\mathsf{V}^*_{s,i}, 1^{t'}, x)$ and that the distinguisher can have (x, w, s) hardwired in order to simulate any other p_j or $\widetilde{\mathsf{p}}_i$. Overall

$$\mathsf{View}_{\mathsf{V}^*_{\otimes\ell}}\langle \mathsf{P}(x,w), \mathsf{V}^*_{\otimes\ell}\rangle \approx_c \mathsf{Sim}_{\otimes\ell}(\mathsf{V}^*_{\otimes\ell}, 1^t, x) \enspace .$$

□

This complete the proof of Proposition 3. □

3.5.2 Laconic Prover Messages

As in the previous section, we start by recalling the laconic prover transformation from [FNV17]. In what follows, let $(\mathsf{P}', \mathsf{V}')$ be a one round predictable argument, the following transformation provides a laconic prover predictable argument (P, V) with a soundness error negligibly close to $1/2$, where the prover sends only a single bit. Roughly, the verifier samples a sufficiently large random string γ and sends it to the prover along with the verifier message. The prover responds with a single bit corresponding to the inner product of γ and its own response to the verifier message, with the verifier accepting if only if the bit matches its own computed inner product of γ with the predicted prover message.

Protocol: Laconic Prover (P, V)

Common input: Input $x \in \mathcal{L}$, security parameter 1^λ

P**'s auxiliary input:** witness w such that $(x, w) \in R_{\mathcal{L}}$

1. Verifier V
   (a) Sample $(\mathsf{v}, \mathsf{b}) \leftarrow\$ \mathsf{V}'(x)$.
   (b) Sample $\gamma \leftarrow\$ \{0,1\}^{|\mathsf{b}|}$.
   (c) Send v, γ to the prover P.
2. Prover P
   (a) Compute $\mathsf{p} := \mathsf{P}'(x, w, \mathsf{v})$.
   (b) Send $\mathsf{q} := \langle \mathsf{p}, \gamma \rangle$ to the verifier V.
3. Verifier V accepts if and only if $\mathsf{q} = \langle \mathsf{b}, \gamma \rangle$.

Figure 3.6: Laconic prover transformation.

In [FNV17], it is proven that this protocol has soundness error at most $\frac{1}{2} + \mathsf{negl}(\lambda)$. As we have seen in the previous subsection (Claim 3), the soundness can be amplified in a manner that preserves zero knowledge. Specifically, ℓ repetitions yields a protocol with soundness error at most $2^{-\ell} + \mathsf{negl}(\lambda)$[2]. Therefore, we focus on proving that a single instance of the above transformation preserves zero knowledge.

Proposition 4. *The round collapsing transformation preserves zero knowledge against b-non-uniform verifiers.*

Proof. Let V^* be a b-non-uniform verifier. We show the following claim.

Claim 4. *There exist an efficient simulator* S *and a verifier* V'^* *against* $\langle \mathsf{P}', \mathsf{V}' \rangle$ *such that*

[2]It should be noted that due to technical reasons as explained in [BIN97, FNV17], the error rate goes down exponentially up to *any* inverse polynomial, which suffices to achieve negligible soundness by setting $\ell = \omega(\log \lambda)$. We encourage the reader to see the aforementioned works for details.

1. V'^* *is* $(b + O(1))$*-non-uniform and efficiently constructable from* V^*.

2. *For every* $x \in \mathcal{L}$,
$$\mathsf{View}_{\mathsf{V}^*} \langle \mathsf{P}(x, w), \mathsf{V}^* \rangle \equiv \mathsf{S}(\mathsf{View}_{\mathsf{V}'^*} \langle \mathsf{P}'(x, w), \mathsf{V}'^* \rangle) \ .$$

This claim gives rise to a simulator Sim for (P, V), which simply invokes Sim' of $(\mathsf{P}', \mathsf{V}')$ on V'^* and then invokes S.

Proof of Claim. We construct $\mathsf{S}, \mathsf{V}'^*$.

$\underline{\mathsf{V}'^*}$:

1. Emulate V^* and obtains (v, γ).
2. Forward v to P'.

$\underline{\mathsf{S}}$:

1. Outputs the randomness of the emulated V^* (can be derived from the randomness of V'^*),
2. as well as $\langle \mathsf{p}, \gamma \rangle$, where p is the received prover message and γ is derived from the randomness of V^*.

The proof is similar to that of Claim 2 in the previous subsection. The second property asserted in the claim follows by construction of $\mathsf{S}, \mathsf{V}'^*$ and the construction of P from P'. It is left to see that V'^* is $(b + O(1))$-non-uniform and efficiently constructable from V^*. $(b + O(1))$-non-uniformity and efficient construction follow from the fact that V^* is b-non-uniform and V'^* uses it as a black box and described by the two code lines above. $\square$

This completes the proof of Proposition 4. $\square$

3.6 Predictable Arguments from Honest-Verifier ZK

In Section 3.4, we showed how to transform any deterministic-prover zero-knowledge (DPZK) protocol into one that is also a predictable argument (PA). In this section, we show that if we start with a weaker notion of deterministic-prover honest verifier zero-knowledge (DP-HVZK) [3] and the existence of an appropriate hard language, we can transform the DP-HVZK protocol into a predictable argument. One caveat of this transformation is that the languages of the DP-HVZK and PA in our transformation will be related, but not identical. As long as the DP-HVZK we start from is for an expressive enough class of languages (e.g. for $\mathsf{NP} \cap \mathsf{co\text{-}NP}$), we will get a PA for the same class.

Definition 23 (Hard-on-Average Language). *A language* $\mathcal{L}$ *is hard-on-average if there exist two* PPT *samplers* $Y_{\mathcal{L}}, N_{\mathcal{L}}$ *where the support of the first is* $\mathcal{L}$ *and of the second is* $\{0,1\}^* \setminus \mathcal{L}$ *such that*
$$\left\{x : x \leftarrow Y_{\mathcal{L}}(1^\lambda)\right\}_{\lambda \in \mathbb{N}} \approx_c \left\{x : x \leftarrow N_{\mathcal{L}}(1^\lambda)\right\}_{\lambda \in \mathbb{N}} \ .$$

We establish the following theorem.

[3] Only zero-knowledge against honestly behaving verifiers.

Theorem 13. *If there exists a deterministic-prover honest-verifier zero-knowledge argument (DP-HVZK) for $\mathcal{L} \vee \mathcal{L}_{\mathsf{hard}}$, where $\mathcal{L}_{\mathsf{hard}}$ is a hard-on-average language, then there exists a predictable argument (PA) for $\mathcal{L}$.*

By the fact that both NP and $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ are closed under OR, we deduce the following corollaries.

Corollary 4. *Assuming DP-HVZK for all of* NP *and hard-on-average languages in* NP*, there is a witness encryption scheme for all of* NP.

Corollary 5. *Assuming DP-HVZK for all of* $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ *and hard-on-average languages in* $\mathsf{NP} \cap \mathsf{co\text{-}NP}$*, there is a witness encryption scheme for all of* $\mathsf{NP} \cap \mathsf{co\text{-}NP}$.

We note that hard-on-average languages in NP are known to follow from one-way functions, and hard-on-average languages in $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ are known to follow from one-way permutations.

We now proceed with the proof.

Proof of Theorem 13. To build a predictable argument for $\mathcal{L}$, we use the following primitives:

- A hard language $\mathcal{L}_{\mathsf{hard}}$ given by samplers $(Y_{\mathcal{L}_{\mathsf{hard}}}, N_{\mathcal{L}_{\mathsf{hard}}})$.
- A ρ-round DP-HVZK protocol $\langle \mathsf{P}', \mathsf{V}' \rangle$ for the language $\mathcal{L}_{OR}$ defined below, where the verifier V' sends messages v_i in round i, and the prover P' sends message p_i in round i. We denote by Sim' the corresponding honest-verifier simulator. The language $\mathcal{L}_{OR}$ is defined below,

  $$\mathcal{L}_{OR} = \left\{ (x, \widetilde{x}) \;\middle|\; \exists (w, \widetilde{w}) \text{ s.t. } R_{\mathcal{L}}(x, w) = 1 \text{ OR } R_{\mathcal{L}_{\mathsf{hard}}}(\widetilde{x}, \widetilde{w}) = 1 \right\} \;,$$

  namely, either the statement x is in $\mathcal{L}$, or $\widetilde{x}$ is in $\mathcal{L}_{\mathsf{hard}}$.

The transformation is presented in Figure 3.7.
Before we proceed with the completeness and soundness, we note that the protocol structure follows that of a predictable argument.

Completeness. We show that (P, V) is complete based on the honest verifier zero-knowledge property of $(\mathsf{P}', \mathsf{V}')$.

Fix any $x \in \mathcal{L}$ and the corresponding witness w, a yes-instance $\widetilde{x} \in \mathcal{L}_{\mathsf{hard}}$, and let $x' = (x, \widetilde{x})$. Let $\widetilde{\mathsf{p}}_1, \ldots, \widetilde{\mathsf{p}}_\rho$ denote the messages and $\widetilde{r}$ denote the verifier randomness simulated by $\mathsf{Sim}'(x')$. We argue that the deterministic prover $\mathsf{P}(x, w)$ produces messages $\{\mathsf{p}_i = \widetilde{\mathsf{p}}_i\}_{i=1}^{\rho}$ with overwhelming probability (over the coins of Sim'). This follows from zero knowledge. Consider a distinguisher that has (x, w) hardwired, and given messages p_i and verifier randomness $\widetilde{r}$ emulates a conversation of the deterministic $\mathsf{P}'(x, w)$ with $\mathsf{V}'(x; \widetilde{r})$, and outputs "real" if the corresponding prover messages coincide with p_i, or "simulated" otherwise. If the simulated messages $\widetilde{\mathsf{p}}_i$ are inconsistent with the real prover messages p_i, the distinguisher will tell them apart.

Soundness. We show that (P, V) is sound based on the completeness, soundness and zero knowledge of $(\mathsf{P}', \mathsf{V}')$, as well as the hardness of $\mathcal{L}_{\mathsf{hard}}$.

Fix any $x \notin \mathcal{L}$ and cheating prover P^*. We prove that P^* fails to convince $\mathsf{V}(x)$ of accepting, except with negligible probability. We consider several hybrid experiments transitioning from a real interaction to an ideal interaction. We will show that when moving from one hybrid to the next

Protocol: PA (P, V)

Common input: Input $x \in \mathcal{L}$, security parameter 1^λ

P**'s auxiliary input:** witness w such that $(x, w) \in R_{\mathcal{L}}$

Verifier V computes

1. $\widetilde{x} \leftarrow Y_{\mathcal{L}_{\mathsf{hard}}}(1^\lambda)$
2. $x' := (x, \widetilde{x})$.
3. $\left(\{(\mathsf{v}_i, \widetilde{\mathsf{p}}_i)\}_{i=1}^{\rho}, \widetilde{r}\right) \leftarrow \mathsf{Sim}'(x')$.

sends $\widetilde{x}$ to the prover P in the first message.

In each round $i \in [\rho]$,

1. Verifier V sends $\mathsf{v_i}$ to the prover P.
2. Prover P computes

   (a) $x' := (x, \widetilde{x})$
   (b) $w' := (w, \perp)$
   (c) $\mathsf{p}_i := \mathsf{P}'(x', w', \{\mathsf{v}_j\}_{j=1}^{i})$

   sends p_i to the verifier V.
3. If for any $i \in [\rho]$, $\mathsf{p}_i \neq \widetilde{\mathsf{p}}_i$, V rejects.

If verifier V has not rejected in all rounds, accept.

Figure 3.7: Transforming DP-HVZK to PA

the prover's chance of convincing the verifier does not decrease by more than a negligible amount. Then we will show that the chance that $\mathsf{V}(x)$ is convinced the final (ideal interaction) hybrid is negligible.

Hyb_0: This is a real interaction between P^* and $\mathsf{V}(x)$.

Hyb_1: In this hybrid, once V samples a simulated transcript $\widetilde{\mathsf{p}}_1, \ldots, \widetilde{\mathsf{p}}_\rho, \widetilde{r} \leftarrow\!\!\$\ \mathsf{Sim}(x')$, it emulates an execution of $\mathsf{V}'(x'; \widetilde{r})$ with the simulated prover messages and checks whether it is accepting. If it is not, V rejects immediately.

We argue that the probability that P^* convinces $\mathsf{V}(x)$ to accept in this hybrid is negligibly close to that in Hyb_0. For this purpose, we argue that with overwhelming probability $\mathsf{Sim}(x')$

samples an accepting transcript. This is shown based on completeness and zero knowledge of $(\mathsf{P}', \mathsf{V}')$. Specifically, recall that $\mathsf{V}(x)$ samples $\widetilde{x} \in \mathcal{L}_{\mathsf{hard}}$ and thus $x' = (x, \widetilde{x}) \in \mathcal{L}_{OR}$. By the completeness of $(\mathsf{P}', \mathsf{V}')$, in an interaction between $\mathsf{V}'(x')$ and $\mathsf{P}'(x', w')$ where $w' = (\perp, \widetilde{w})$ and $\widetilde{w}$ is a witness for $\widetilde{x}$, the prover convince V' with overwhelming probability. It then follows from zero knowledge of $(\mathsf{P}', \mathsf{V}')$ that $\mathsf{Sim}(x')$ also generates an accepting transcript with overwhelming probability; otherwise, we can non-uniformly fix $\widetilde{x}, \widetilde{w}$ and construct a distinguisher that violates zero knowledge.

Hyb_2: In this hybrid, the verifier V does not insist that the prover P^* is consistent with the simulated messages $\widetilde{\mathsf{p}}_1, \ldots, \widetilde{\mathsf{p}}_\rho$. Instead, it emulates $\mathsf{V}'(x'; \widetilde{r})$, and accepts if the messages sent by P^* convince V'.

The probability that V accepts in this hybrid is at least as large as the probability it accepts in Hyb_1. Indeed, any execution that would have been accepted in the previous hybrid Hyb_1 is in particular an execution in which $\mathsf{V}'(x'; \widetilde{r})$ is convinced and thus is also accepted in the current Hyb_2.

Hyb_3: In this hybrid, the verifier V does not check that the simulated $\widetilde{\mathsf{p}}_1, \ldots, \widetilde{\mathsf{p}}_\rho, \widetilde{r}$ make $\mathsf{V}'(x'; \widetilde{r})$ accept. (In particular, the simulated prover messages $\widetilde{\mathsf{p}}_1, \ldots, \widetilde{\mathsf{p}}_\rho$ are ignored altogether, and only the simulated coins $\widetilde{r}$ are used).

The probability that $\mathsf{V}(x)$ accepts in this hybrid is at least as large as the probability it accepts in the previous hybrid, as we have only removed a verifier test.

Hyb_4: In this hybrid, instead of sampling simulated coins $\widetilde{r}$ using $\mathsf{Sim}'(x')$, V samples truly random coins r.

The probability that $\mathsf{V}(x)$ accepts in this hybrids is negligibly close to that in the previous hybrid. This follows from zero knowledge of $(\mathsf{P}', \mathsf{V}')$. Indeed, since $x' \in \mathcal{L}_{OR}$, the simulated honest verifier coins $\widetilde{r}$ are pseudorandom.

Hyb_5: In this hybrid, $\mathsf{V}(x)$ samples a no-instance $\widetilde{x} \leftarrow N_{\mathcal{L}_{\mathsf{hard}}}$ instead of a yes-instance. By the indistinguishability of $Y_{\mathcal{L}_{\mathsf{hard}}}$ and $N_{\mathcal{L}_{\mathsf{hard}}}$, the probability that P^* convinces $\mathsf{V}(x)$ to accept in this hybrid is negligibly close to that in Hyb_4.

We now argue that the probability that P^* convinces $\mathsf{V}(x)$ to accept in Hyb_5 is negligible. Note that in Hyb_5 it holds that both $x \notin \mathcal{L}$ and $\widetilde{x} \notin \mathcal{L}_{\mathsf{hard}}$ and thus $x' = (x, \widetilde{x}) \notin \mathcal{L}_{OR}$. For P^* to convince $\mathsf{V}(x)$ of accepting in Hyb_5, it must convince $\mathsf{V}'(x'; r)$ of accepting, when V' uses truly random coins. By the soundness of $(\mathsf{P}', \mathsf{V}')$ this occurs with negligible probability. Soundness follows. □

3.7 Open Problems

As noted earlier, Goldreich-Oren [GO94] establishes lower bound for both arguments and *proofs*. Further, the lower bounds also extend to the notions of statistical and perfect zero-knowledge. While our work provides a complete characterization for deterministic *computational* zero-knowledge *arguments*, a full characterization of deterministic zero-knowledge remains an interesting open problem.

Chapter 4

Round Optimal Multiparty Computation

4.1 Overview

Before we dive into the technical contributions of our work, for the uninitiated reader, we provide a brief summary of the key challenges that arise in the design of a four round MPC protocol and the high-level strategies adopted in prior works for addressing them. We group these challenges into three broad categories, and will follow the same structure in the remainder of the section.

Enforcing honest behavior. A natural idea, adopted in prior works, is to start with a protocol that achieves security against semi-malicious[1] adversaries and compile it using zero-knowledge (ZK) proofs [GMR89a] à la GMW compiler [GMW87b] to achieve security against malicious adversaries. This is not easy, however, since we are constrained by the number of rounds. As observed in prior works, when the underlying protocol is *delayed* semi-malicious[2] [ACJ17, BL18], we can forego establishing honest behavior in the first two rounds. In particular, it suffices to establish honest behavior in the third and fourth rounds. The main challenge that still persists, however, is that ZK proofs – the standard tool for enforcing honest behavior – are impossible in three rounds w.r.t. black-box simulation [GK96b]. Thus, an alternative mechanism is required for establishing honest behavior in the third round.

Need for rewind security. Due to the constraint on the number of rounds, all prior works utilize design templates where multiple sub-protocols are executed in *parallel*. This creates a challenge when devising a black-box simulation strategy that works by rewinding the adversary. In particular, if the simulator rewinds the adversary (say) during second and third round of the protocol, e.g., to extract its input, we can no longer rely on stand-alone security of sub-protocols used in those

[1]Roughly speaking, such adversaries behave like semi-honest adversaries, except that they may choose arbitrary random tapes.

[2]Roughly speaking, a delayed semi-malicious adversary is similar to semi-malicious adversary, except that in the second last round of a k-round protocol, it is required to output (on a special tape) a witness (namely, its input and randomness) that establishes its honest behavior in all the rounds so far.

rounds. This motivates the use of sub-protocols that retain their security even in the presence of some number of rewinds. Indeed, much work is done in all prior works to address this challenge.

Non-malleability. For similar reasons as above, we can no longer rely on standard soundness guarantee of ZK proofs (which only hold in the stand-alone setting). All prior works address this challenge via a careful use of some non-malleable primitive such as non-malleable commitments [DDN91] in order to "bootstrap non-malleability" in the entire protocol. This leads to an involved security analysis.

Our primary technical contribution is in addressing the first two issues. We largely follow the template of prior works in addressing non-malleability challenges. As such, in the remainder of this technical overview, we focus on the first two issues, and defer discussion on non-malleability to Section 4.2.5.

Organization. We describe our key ideas for tackling the first and second issues in Sections 4.1.1 and 4.1.2, respectively. We conclude by providing a summary of our protocol in Section 4.1.3.

4.1.1 Enforcing Honest Behavior

In any four round protocol, a rushing adversary may always choose to *abort* after receiving the messages of honest parties in the last round. At this point, the adversary has already received enough information to obtain the output of the function being computed. This suggests that we must enforce "honest behavior" on the protocol participants *within the first three rounds* in order to achieve security against malicious adversaries. Indeed, without any such safeguard, a malicious adversary may be able learn the inputs of the honest parties, e.g., by acting maliciously so as to change the functionality being computed to the identity function.

Since three-round ZK proofs with black-box simulation are known to be impossible, all recent works on four round MPC devise non-trivial strategies that only utilize weaker notions of ZK (that are achievable in three or less rounds) to enforce honest behavior within the first three rounds of the MPC protocol. However, all of these approaches end up relying on assumptions that are far from optimal: [ACJ17] and [BHP17] use super-polynomial-time hardness assumptions, [HHPV18] use Zaps [DN00] and affine-homomorphic encryption schemes, and [BGJ$^+$18] use a new notion of promise ZK together with three round strong WI [JKKR17], both of which require specific number-theoretic assumptions.

A Deferred Verification Approach. We use a different approach to address the above challenge. We do not require the parties to give an explicit proof of honest behavior within the first three rounds. Of course, this immediately opens up the possibility for an adversary to cheat in the first three rounds in such a manner that by observing the messages of the honest parties in the fourth round, it can completely break privacy. To prevent such an attack, we require the parties to "encrypt" their last round message in such a manner that it can only be decrypted by using a "witness" that establishes honest behavior in the first three rounds. In other words, the verification check for honest behavior is deferred to the fourth round.

In the literature, the above idea is referred to as conditional disclosure of secrets (CDS) [AIR01]. Typically, however, CDS is defined and constructed as a two-party protocol involving a single encryptor – who encrypts a secret message w.r.t. some statement – and a single decryptor who presumably holds a witness that allows for decryption. [3]

[3]There are some exceptions; we refer the reader to Section 1.2.2 for discussion on other models.

This does not suffice in the multiparty setting due to the following challenges:

– The multiparty setting involves multiple decryptors as opposed to a single decryptor. A naive way to address this would be to simply run multiple executions of two-party CDS in parallel, each involving a different decryptor, such that the i^{th} execution allows party i to decrypt by using a witness that establishes its own honest behavior earlier in the protocol. However, consider the case where the adversary corrupts at least two parties. In the above implementation, a corrupted party who behaved honestly during the first three rounds would be able to decrypt the honest party message in the last round even if another corrupted party behaved maliciously. This would clearly violate security. As such, we need a mechanism to *jointly* certify honest behavior of *all* the parties (as opposed to a single party).

– In the two-party setting, the input and randomness of the decryptor constitutes a natural witness for attesting its honest behavior. In the multiparty setting, however, it is not clear how an individual decryptor can obtain such a witness that establishes honest behavior of all the parties without trivially violating privacy of other parties.

We address these challenges by implementing a *multiparty conditional disclosure of secrets* (MCDS) mechanism. Informally speaking, an MCDS scheme can be viewed as a tuple of (possibly interactive) algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$: (a) Gen takes as input an instance and witness pair (x, w) and outputs a "public" witness π. (b) Enc takes as input n statements $(x_1, \ldots, x_n)$ and a message m and outputs an encryption c of m. (c) Dec takes as input a ciphertext c and tuples $(x_1, \pi_1), \ldots, (x_n, \pi_n)$ and outputs m or $\perp$. We require the following properties:

– **Correctness:** If all the instances $(x_1, \ldots, x_n)$ are true, then dec outputs m.

– **Message Privacy:** If at least one instance is false, then c is semantically secure.

– **Witness Privacy:** There exists a simulator algorithm that can simulate the output π of Gen without using the private witness w.

The security properties of MCDS allow us to overcome the aforementioned challenges. In particular, the witness privacy guarantee allows the parties to publicly release the witnesses $(\pi_1, \ldots, \pi_n)$ while maintaining privacy of their inputs and randomness.

In order to construct MCDS with witness privacy guarantee, we look towards ZK proof systems. As a first attempt, we could implement public witnesses via a delayed-input[4] four round ZK proof system. Specifically, each party i is required to give a ZK proof for x_i such that the last round of the proof constitutes a public witness π_i. Further, a simple, non-interactive method to implement the encryption and the decryption mechanism is witness encryption [GGSW13]. However, presently witness encryption is only known from non-standard assumptions (let alone OT).

To achieve our result from minimal assumptions, we instead use garbled circuits [Yao86] and four round OT to implement MCDS. Namely, each party i garbles a circuit that contains hardwired the entire transcript of the first three rounds of the underlying MPC, as well the fourth round message of the MPC of party i. Upon receiving as input a witness $\pi_1, \ldots, \pi_n$, where π_j is a witness for honest behavior of party j, the garbled circuit outputs the fourth round message. Each party j can encode its witness π_j in the OT receiver messages, where the corresponding sender inputs will be the wire labels of the garbled circuit. Party j then release its private randomness used inside OT

[4]A proof system is said to be delayed input if the instance is only required for computing the last round of the proof.

in the fourth round so that any other party j' can use it to compute the output of the OT, thereby learning the necessary wire labels for evaluating the garbled circuit sent by party i. For security, it is imperative the witness π_j remains hidden until the randomness is revealed in the fourth round.

A problem with the above strategy is that in a four round OT, the receiver's input must be fixed by the third round. This means that we can no longer use four round ZK proofs, and instead must use *three round* proofs to create public witnesses of honest behavior. But which three round proofs must we use? Towards this, we look to the weaker notion of *promise* ZK introduced by [BGJ+18]. Roughly, promise ZK relaxes the standard notion of ZK by guaranteeing security only against malicious verifiers who do *not* abort. Importantly, unlike standard ZK, distributional[5] promise ZK can be achieved in only three rounds with black-box simulation in the bidirectional message model. This raises two questions – is promise ZK sufficient for our purposes, and what assumptions are required for three round promise ZK?

Promise ZK Under the Hood. Let us start with the first question. An immediate challenge with using promise ZK is that it provides no security in the case where the verifier always aborts. In application to four round MPC, this corresponds to the case where the (rushing) adversary always aborts in the third round. Since the partial transcript at the end of third round (necessarily) contains inputs of honest parties, we still need to argue security in this case. The work of [BGJ+18] addressed this problem by using a "hybrid" ZK protocol that achieves the promise ZK property when the adversary is non-aborting, and strong witness-indistinguishability (WI) property against aborting adversaries. The idea is that by relying on strong WI property (only in the case where adversary aborts in the third round), we can switch from using real inputs of honest parties to input 0. However, three round strong WI is only known based on specific number-theoretic assumptions [JKKR17].

To minimize our use of assumptions, we do *not* use strong WI, and instead devise a hybrid argument strategy – similar to that achieved via strong WI – by using promise ZK *under the hood*. Recall that since we use the third round prover message of promise ZK as a witness for conditional decryption, it is not given in the clear, but is instead "encrypted" inside the OT receiver messages in the third round. This has the positive effect of shielding promise ZK from the case where the adversary always aborts in the third round.[6] In particular, we can use the following strategy for arguing security against aborting adversaries: we first switch from using promise ZK third round prover message to simply using 0's as the OT receiver's inputs. Now, we can replace the honest parties' inputs with 0 inputs by relying on the security of the sub-protocols used within the first three rounds. Next, we can switch back to using honestly computed promise ZK third round prover message as the OT receiver's inputs.

Let us now consider the second question, namely, the assumptions required for three round promise ZK. The work of [BGJ+18] used specific number-theoretic assumptions to construct three round (distributional) promise ZK. However, we only wish to rely on the use of four round OT. Towards this, we note that the main ingredient in the construction of promise ZK by [BGJ+18] that necessitated the use of number-theoretic assumptions is a three round WI proof system that achieves "bounded-rewind-security." Roughly, this means that the WI property holds even against verifiers who can rewind the prover an a priori bounded number of times.

Towards minimizing assumptions, we note that a very recent work of [GR19] provides a construction of such a WI based only on non-interactive commitments. By using their result, we can

[5]That is, where the instances are sampled from a public distribution.

[6]Note that if the protocol does progress to the fourth round, then we do not need to shield promise ZK anymore.

obtain three round promise ZK based on non-interactive commitments, which in turn can be obtained from four round OT using the recent observation of Lombardi and Schaeffer [LS19].

4.1.2 Rewinding Related Challenges

While the above ideas form the basis of our approach, we run into several obstacles during implementation due to rewinding-related issues that we mentioned earlier. In order to explain these challenges and our solution ideas, we first describe a high-level template of our four round MPC protocol based on the ideas discussed so far. To narrow the focus of the discussion on the challenges unique to the present work, we ignore some details for now and discuss them later.

An Initial Protocol Template. We devise a compiler from four round delayed semi-malicious MPC protocols of a special form to a four round malicious-secure MPC protocol. Specifically, we use a four round delayed semi-malicious protocol Π obtained by plugging in a four-round malicious-secure (which implies delayed semi-malicious security) OT in the k-round semi-malicious MPC protocol of [GS18b, BL18] based on k-round semi-malicious OT. An important property of this protocol that we rely upon is that it consists only of OT messages in the first $k-2$ rounds. Further, we also rely upon the random self-reducibility of OT, which implies that the first two rounds do not depend on the OT receiver's input, and the first three rounds do not depend on the sender's input.[7]

To achieve malicious security, similar to prior works, our compiler uses several building blocks (see Section 4.1.3 for a detailed discussion). One prominent building block is a three-round extractable commitment scheme that is executed in *parallel* with the first three rounds of the delayed semi-malicious MPC. The extractable commitment scheme is used by the parties to commit to their inputs and randomness. This allows the simulator for our protocol to extract the adversary's inputs and randomness *by rewinding the second and third rounds*, and then use them to simulate the delayed semi-malicious MPC.

Bounded-Rewind-Secure OT. The above template poses an immediate challenge in proving security of the protocol. Since the simulator rewinds the second and third rounds in order to extract the adversary's inputs, this means that the second and third round messages of the delayed semi-malicious MPC also get rewound. For this reason, we cannot simply rely upon delayed semi-malicious security of the MPC. Instead, we need the MPC protocol to remain secure *even when it is being rewound*. More specifically, since we are using an MPC protocol where the first two rounds only consist of OT messages, we need a *four round rewind-secure OT protocol*. Since the third round of a four round OT only contains a message from the OT receiver, we need the following form of rewind security property: an adversarial sender cannot determine the input bit used by the receiver even if it can rewind the receiver during the second and third round.

Clearly, an OT protocol with black-box simulation cannot be secure against an arbitrary number of rewinds. In particular, the best we can hope for is security against an a priori *bounded* number of rewinds. Following observations from [BGJ+18], we note that bounded-rewind security of OT is, in fact, *sufficient* for our purposes. Roughly, the main idea is that the rewind-security of OT is invoked to argue indistinguishability of two consecutive hybrids inside our security proof. In order to establish indistinguishability by contradiction, it suffices to build an adversary that breaks OT security with some non-negligible probability (as opposed to overwhelming probability). This, in turn means that the reduction only needs to extract the adversary's input required for generating

[7]We note that this property was also used by [BL18] in their construction of k-round malicious-secure MPC.

its view with non-negligible probability. By using a specific extractable commitment scheme, we can ensure that the number of rewinds necessary for this task are a priori bounded.

Standard OT protocols, however, do not guarantee any form of bounded-rewind security. Towards this, we provide a generic construction of a four round bounded-rewind secure OT starting from any four round OT, which may be of independent interest. Our transformation is in fact more general and works for any $k \geq 4$ round OT, when rewinding is restricted to rounds $k-2$ and $k-1$. For simplicity, we describe our ideas for the case where we need security against *one* rewind; our transformation easily extends to handle more rewinds.

A natural idea to achieve one-rewind security for receivers, previously considered in [BL18], is the following: run two copies of an OT protocol in parallel for the first $k-2$ rounds. In round $k-1$, the receiver randomly chooses one of the two copies and only continues that OT execution, while the sender continues both the OT executions. In the last round, the parties only complete the OT execution that was selected by the receiver in round $k-1$. Now, suppose that an adversarial sender rewinds the receiver in rounds $k-2$ and $k-1$. Then, if the receiver selects different OT copies on the "main" execution thread and the "rewound" execution thread, we can easily reduce one-rewind security of this protocol to stand-alone security of the underlying OT.

The above idea suffers from a subtle issue. Note that the above strategy for dealing with rewinds is inherently *biased*, namely, the choice made by the receiver on the rewound thread is *not* random, and is instead correlated with its choice on the main thread. If we use this protocol in the design of our MPC protocol, it leads to the following issue during simulation: consider an adversary who chooses a random z and then always aborts if the receiver selects the z-th OT copy. Clearly, this adversary only aborts with probability $1/2$ in an honest execution. Now, consider the high-level simulation strategy for our MPC protocol discussed earlier, where the simulator rewinds the second and third rounds to extract the adversary's inputs. In order to ensure rewind security of the OT, this simulator, with overall probability $1/2$, will select the z-th OT copy on *all* the rewound execution threads. However, in this case, the simulator will always *fail* in extracting the adversary's inputs no matter how many times it rewinds.

We address the above problem via a secret-sharing approach to eliminate the bias. Instead of simply running two copies of OT, we run $\ell \cdot n$ copies in parallel during the first $k-2$ rounds. These $\ell \cdot n$ copies can be divided into n tuples, each consisting of ℓ copies. In round $k-1$, the receiver selects a single copy from each of the n tuples at random. It then uses n-out-of-n secret sharing to divide its input bit b into n shares $b_1, \ldots, b_n$, and then uses share b_i in the OT copy selected from the i-th tuple. In the last round, sender now additionally sends a garbled circuit (GC) that contains its input (x_0, x_1) hardwired. The GC takes as input all the bits $b_1, \ldots, b_n$, reconstructs b and then outputs x_b. The sender uses the labels of the GC as its inputs in the OT executions. Intuitively, by setting ℓ appropriately, we can ensure that for at least one tuple i, the OT copies randomly selected by the receiver on the main thread and the rewound threads are different, which ensures that b_i (and thereby, b) remains hidden. We refer the reader to the technical section for more details.

Proofs Of Proofs. We now describe another challenge in implementing our template of four round MPC. As discussed earlier, we use a three round extractable commitment scheme to enable extraction of the adversary's inputs and randomness. For reasons similar to those as for the case of OT, we actually use an extractable commitment scheme that achieves bounded-rewind security. Specifically, we use a simplified variant of the three-round commitment scheme constructed by [BGJ$^+$18].[8]

[8]The commitment scheme of [BGJ$^+$18] also achieves some security properties, in addition to bounded rewind security,

A specific property of this commitment scheme is that in order to achieve rewind security, it is designed such that the third round message of the committer is not "verifiable." This means that the committer may be able to send a malformed message without being detected by the receiver. For this reason, we require each party to prove the "well-formedness" of its commitment via promise ZK. This, however, poses the following challenge during simulation: since the third round prover message of promise ZK is encrypted inside OT receiver message, the simulator doesn't know whether the adversary's commitment is well-formed or not. In particular, if the adversary's commitment is not well-formed, the simulator may end up running *forever*, in its attempt to extract the adversary's input via rewinding.

One natural idea to deal with this issue is to first extract adversary's promise ZK message from the OT executions via rewinding, and then decide whether or not to attempt extracting the adversary's input. However, since we are using an *arbitrary* (malicious-secure) OT, we do *not* know in advance the number of rewinds required for extracting the receiver's input. This in turn means that we cannot correctly set the rewind security of the sub-protocols used in our final MPC protocol appropriately in advance.

We address this issue via the following strategy. We use *another* three round (delayed-input) extractable commitment scheme [PRS02] (ecom) as well as another copy of promise ZK. This copy of promise ZK proves honest behavior in the first three rounds, and its third message is committed inside the extractable commitment. Further, the third round message of this extractable commitment is such that it allows for polynomial-time extraction (with the possibility of "over-extraction"[9]). This, however, comes at the cost that this extractable commitment does not achieve any rewind security. Interestingly, stand-alone security of this scheme suffices for our purposes since we only use it in the case where the adversary always aborts in the third round (and therefore, no rewinds are performed).

The main idea is that by using such a special-purpose extractable commitment scheme, we can ensure that an a priori fixed constant number of rewinds are sufficient for extracting the committed value, namely, the promise ZK third round prover message, with noticeable probability. This, in turn, allows us to set the rewind security of other sub-protocols used in our MPC protocol in advance to specific constants.

Of course, the adversary may always choose to commit to malformed promise ZK messages within the extractable commitment scheme. In this case, our simulator may always decide not to extract adversary's input, *even if the adversary was behaving honestly otherwise*. This obviously would lead to a view that is distinguishable from the real world. To address this issue, we use a *proofs of proofs* strategy. Namely, we require the first copy of promise ZK, which is encrypted inside OT, to prove that the second copy of promise ZK is "accepting". In this case, if the adversary commits malformed promise ZK messages within the extractable commitment, the promise ZK message inside OT will not be accepting. This, in turn, means that due to the security of garbled circuits, the fourth round messages of the parties will become "opaque".

Finally, we remark that for technical reasons, we do extract the promise ZK encrypted inside the OT receiver message in our *final* hybrid. However, in this particular hybrid, the number of rewinds required for extraction do not matter since in this hybrid, we only make change inside a *non-interactive* primitive (specifically, garbled circuit) that is trivially secure against an *unbounded* polynomial number of rewinds.

that are not required by our compiler. Hence, we use a simplified variant of their scheme.

[9]This means the extractor can output a non $\perp$ value if the commitment has no valid opening.

4.1.3 Protocol Design Summary

Putting all the various pieces together, we describe the overall structure of the protocol at a high level to demonstrate the purpose of its various components in the context of the protocol.

For simplicity we consider the messages sent from P_i to P_j. Note that even though P_j is the intended recipient for the messages in a two party sub-protocol, the messages are broadcast to all parties. The overview of the protocol messages is presented in Figure 4.1.

Delayed semi-malicious MPC (blue). P_i uses input x and randomness r to compute the messages msg_k for the bounded rewind secure four-round delayed semi malicious protocol Π.

Multiparty Conditional Disclosure of Secrets (red). As discussed earlier, the last message of Π is not sent in the clear but instead sent inside a garbled circuit GC used to implement MCDS. We use a four-round oblivious transfer protocol ot_k to allow the parties to obtain garbled circuit wire labels corresponding to their witnesses. We defer the discussion on the witness for MCDS below.

Rewind Secure Extractable commitment (green). The same input and randomness used to compute messages for Π is committed via an extractable commitment recom_k. This is done to enable the simulator to extract the inputs and randomness of the adversary for simulation. As discussed earlier, we use a three round extractable commitment that achieves bounded rewind security.

Promise ZK (purple). We use promise ZK in a non-black box manner in our protocol. Specifically, it consists of a trapdoor generation phase td_k, and a bounded rewind secure witness indistinguishable proof rwi_k. As discussed in our *proofs of proofs* strategy, we actually use two copies of the promise ZK (indexed by subscripts a and b in the figure), but both of these copies will share a single instance of the trapdoor generation. At a high level, both rwis prove that either the claim is true or "I committed to the trapdoor in the non-malleable commitment" (see below). We also note that one of the rwi copies, specifically, the copy indexed with subscript b is used as a witness for the MCDS mechanism.

Witness Indistinguishable Proof (orange). We also use a regular witness indistinguishable proof wi (without any rewind security) to establish honest behavior of the parties in the last round of the protocol. This effectively involves proving that either the last round message was computed honestly or "I committed to the trapdoor in the non-malleable commitment" (see below).

Extractable commitment (brown). As discussed earlier, we use an extractable commitment ecom (without rewind security) to implement our *proofs of proofs* strategy to enable simulation.

Non-malleability (dark blue). We bootstrap non-malleability in our protocol using non-malleable commitments ncom in a similar manner to prior works [ACJ17, BGJ$^+$18][10]. Specifically, in the honest execution of the protocol, the parties simply commit to a random value $\widetilde{\mathsf{r}}$. We rely on specific properties of the ncom, which we do not discuss here and refer the reader to the technical sections.

Finally, we note that our protocol design uses multiple sub-protocols with bounded rewind security. We do not discuss how the bounds for the sub-protocols are set here, and instead defer this discussion to Section 4.4.

[10]see Section 4.2.5 for discussion on *bootstrapping non-malleability using* ncom as used in prior works to construct constant round protocols

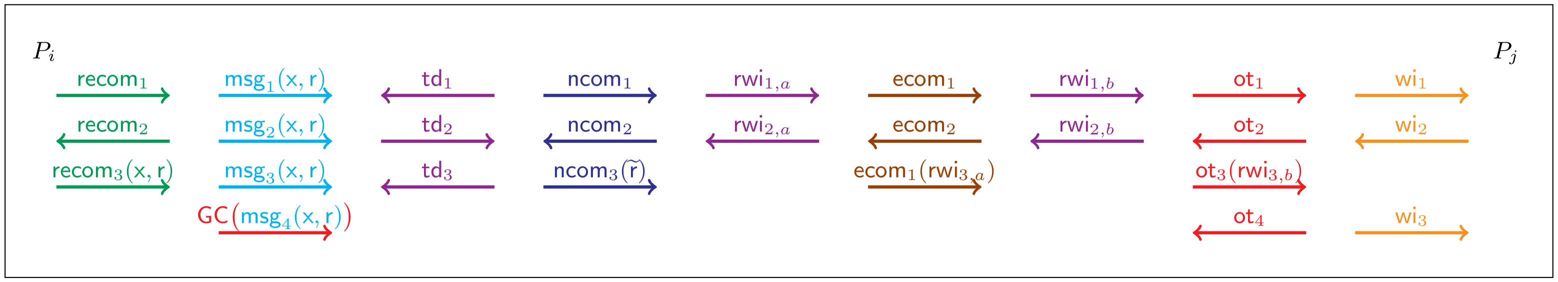


Figure 4.1: Overall structure of the protocol

Complexity of the protocol description. One might wonder why our construction is so involved and whether there is a simpler construction. This is an important question that needs to be addressed. Unfortunately, our current understanding of the problem does not allow for a protocol that is easier to describe, but we believe that our solution is less complex than the prior state-of-the-art solutions [BGJ$^+$18, HHPV18].

4.2 Preliminaries

4.2.1 Extractable Commitment Scheme

We will use a variant of a simple challenge-response based extractable statistically-binding string commitment scheme $\langle C, R\rangle$ that has been used in several prior works, most notably [PRS02, Ros04]. We note that in contrast to [PRS02] where a multi-slot protocol was used, here (similar to [Ros04]), we only need a one-slot protocol.

Protocol $\langle C, R\rangle$**.** Let $\mathsf{Com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding string commitment scheme which requires the assumption of injective one-way functions for its construction. Let n denote the security parameter. The commitment scheme $\langle C, R\rangle$ is described as follows.

COMMIT PHASE:

1. To commit to a string $\mathtt{str}$, C chooses $k = \omega(\log(n))$ independent random pairs $\{\alpha_i^0, \alpha_i^1\}_{i=1}^k$ of strings such that $\forall i \in [k]$, $\alpha_i^0 \oplus \alpha_i^1 = \mathtt{str}$; and commits to all of them to R using Com. Let $B \leftarrow \mathsf{Com}(\mathtt{str})$, and $A_i^0 \leftarrow \mathsf{Com}(\alpha_i^0)$, $A_i^1 \leftarrow \mathsf{Com}(\alpha_i^1)$ for every $i \in [k]$.

2. R sends k uniformly random bits $v_1, \ldots, v_n$.

3. For every $i \in [k]$, if $v_i = 0$, C opens A_i^0, otherwise it opens A_i^1 to R by sending the appropriate decommitment information.

OPEN PHASE: C opens all the commitments by sending the decommitment information for each one of them.

For our construction, we require a modified extractor for the extractable commitment scheme. The standard extractor returns the value $\mathtt{str}$ that was committed to in the scheme. Instead, we require that the extractor return i, and the openings of A_i^0 and A_i^1. This extractor can be constructed easily, akin to the standard extractor for the extractable commitment scheme.

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

We say that commit phase between C' and R' is *well formed* with respect to a value $\hat{\mathtt{str}}$ if there exist values $\{\hat\alpha_i^0, \hat\alpha_i^1\}_{i=1}^k$ such that:

1. For all $i \in [k]$, $\hat\alpha_i^0 \oplus \hat\alpha_i^1 = \hat{\mathtt{str}}$, and

2. Commitments B, $\{A_i^0, A_i^1\}_{i=1}^k$ can be decommitted to $\hat{\mathtt{str}}$, $\{\hat\alpha_i^0, \hat\alpha_i^1\}_{i=1}^k$ respectively.

3. Let $\bar\alpha_1^{v_1}, \ldots, \bar\alpha_k^{v_k}$ denote the secret shares revealed by C' in the commit phase. Then, for all $i \in [k]$, $\bar\alpha_i^{v_i} = \hat\alpha_i^{v_i}$.

We now state the following simple lemma about extraction from commitments when they are well formed.

Lemma 1. *There exists a* PPT *extractor algorithm* Ext *such that, given a set of* 2 *"well-formed" execution transcripts of* Com *where each transcript consists of the same first round sender message, the extractor successfully extracts the value committed in each transcript, except with negligible probability.*

It is easy to see that two random challenge strings will differ in at least a single position other than with negligible probability. From the description of the protocol, if the commit phase was well formed, both commitments at a single position suffice to extract the value. In the sequel, we refer to this as *2-extractable* property of the extractable commitment scheme.

4.2.2 Extractable Commitments with Bounded Rewinding Security

In this section, we describe an extractable commitment protocol that is additionally secure against a bounded number of rewinds. Since we are interested in the three round protocol, we limit our discussion in this section to this setting. A simple extractable commitment is a commitment protocol between a sender (with input x) and a receiver which allows an extractor, with the ability to rewind the sender via the second and third round of the protocol, to extract the sender's committed value. Several constructions of three round extractable commitment schemes are known in the literature (see, e.g., [PRS02, Ros04] and Section 4.2.1 for an example construction).

When we additionally require bounded rewind security, we shall parameterize this bound by B_{recom}. Roughly this means that the value committed by a sender in an execution of the commitment protocol remains hidden even if a malicious receiver can rewind the sender back to the start of the second round of the protocol an a priori bounded B_{recom} number of times. Extraction will then necessarily require strictly larger than B_{recom} rewinds.

In the remainder of the section, we describe a construction of a three round extractable commitment protocol with bounded rewind security $\mathsf{RECom} = (S, R)$. The construction is adapted from the construction presented in [BGJ+18], and simplified for our setting since we do not require the stronger notion of "reusability", as defined in their work.

In our application, we set $B_{\mathsf{recom}} = 4$; however, our construction also supports larger values of B_{recom}. For technical reasons, we don't define or prove B_{recom}-rewinding security property and reusability property for our extractable commitment protocol. Instead, this is done inline in our four round MPC protocol.

Construction. Let Com denote a non-interactive perfectly binding commitment scheme based on injective one-way functions. Let N and B_{recom} be positive integers such that $N - B_{\mathsf{recom}} - 1 \geq \frac{N}{2} + 1$. For $B_{\mathsf{recom}} = 4$, it suffices to set $N = 12$. The three round extractable commitment protocol RECom is described in Figure 4.2.

Protocol (S, R): Extractable Commitments with Bounded Rewinding Security

Sender S has input x.

Commitment Phase:

1. **Round 1:** S does the following:
   - Pick N random degree B_{recom} polynomials $\mathsf{p}_1, \ldots, \mathsf{p}_N$ over $\mathbb{Z}_q$, where q is a prime larger than 2^λ.
   - Compute $\mathsf{recom}_{1,\ell}^{S\to R} \leftarrow \mathsf{Com}(\mathsf{p}_\ell; r_\ell)$ using a random string r_ℓ, for every $\ell \in [N]$.
   - Send $\mathsf{recom}_1^{S\to R} = (\mathsf{recom}_{1,1}^{S\to R}, \ldots, \mathsf{recom}_{1,N}^{S\to R})$ to R.
2. **Round 2:** R does the following:
   - Pick random values $\mathsf{z}_\ell \leftarrow\$ \mathbb{Z}_q$ for every $\ell \in [N]$.
   - Send $\mathsf{recom}_2^{R\to S} = (\mathsf{z}_1, \ldots, \mathsf{z}_N)$ to S.
3. **Round 3:** S does the following:
   - Compute $\mathsf{recom}_{3,\ell}^{S\to R} \leftarrow (x \oplus \mathsf{p}_\ell(0), \mathsf{p}_\ell(\mathsf{z}_\ell))$ for all $\ell \in [N]$.
   - Send $\mathsf{recom}_3^{S\to R} = (\mathsf{recom}_{3,1}^{S\to R}, \ldots, \mathsf{recom}_{3,N}^{S\to R})$ to R.

Decommitment Phase:

1. S outputs $\mathsf{p}_1, \ldots, \mathsf{p}_N$ together with the randomness $r_1, \ldots, r_N$ used in the first round commitments.
2. R first verifies the following:
   - For each $\ell \in [N]$, $\mathsf{recom}_{1,\ell}^{S\to R} = \mathsf{Com}(\mathsf{p}_\ell; r_\ell)$.
   - Parse $\mathsf{recom}_{3,\ell}^{S\to R} = (\alpha_\ell, \beta_\ell)$. Verify that $\beta_\ell = \mathsf{p}_\ell(\mathsf{z}_\ell)$.
   - For each $\ell \in [N]$, compute $x_\ell = \mathsf{p}_\ell(0) \oplus \alpha_\ell$. Verify that all the x_ℓ values are equal.

   If any of the above verifications fail, R outputs $\perp$. Otherwise, R outputs x.

Figure 4.2: Extractable Commitment Scheme recom.

Well-Formedness of recom **Transcripts.** We now define a "well-formedness" property of an execution transcript of RECom. Roughly, we say that a transcript $(\mathsf{recom}_1^{S\to R}, \mathsf{recom}_2^{R\to S}, \mathsf{recom}_3^{S\to R})$ is well-formed w.r.t. an input x and randomness r if:

– $N-1$ out of the N tuples $\mathsf{recom}_{3,\ell}^{S\to R} = (\alpha_\ell, \beta_\ell)$ (where $\ell \in [N]$) are "honestly" computed using randomness $r = \left(\{\mathsf{p}_i\}_{i=1}^N, \{r_i\}_{i=1}^N\right)$ in the sense that: each α_ℓ is a one-time pad of x w.r.t. the key $\mathsf{p}_\ell(0)$ where p_ℓ is a polynomial committed (using randomness r_ℓ) in the first round message $\mathsf{recom}_1^{S\to R}$, and each β_ℓ is a correct evaluation of the polynomial p_ℓ over the "challenge" value z_ℓ contained in $\mathsf{recom}_2^{R\to S}$.

We now proceed to formally define the well-formedness property. For any set T, let $T[i]$ denote the i^{th} element of T.

Definition 24 (Well-Formed Transcripts). *An execution transcript* $(\mathsf{recom}_1^{S\to R}, \mathsf{recom}_2^{R\to S}, \mathsf{recom}_3^{S\to R})$ *of* recom *is said to be* well-formed *with respect to an input x and randomness $r = \left(\{\mathsf{p}_i\}_{i=1}^{N}, \{r_i\}_{i=1}^{N}\right)$ if there exists an index set $\mathcal{I}$ of size $N-1$ such that the following holds:*

- *For every $j \in |\mathcal{I}|$,* $\mathsf{recom}_{1,\mathcal{I}[j]}^{S\to R} = \mathsf{Com}(\mathsf{p}_{\mathcal{I}[j]}; r_{\mathcal{I}[j]})$ *(AND)*
- *For every $j \in |\mathcal{I}|$,* $\mathsf{recom}_{3,\mathcal{I}[j]}^{S\to R} = (x \oplus \mathsf{p}_{\mathcal{I}[j]}(0), \mathsf{p}_{\mathcal{I}[j]}(\mathsf{z}_{\mathcal{I}[j]}))$*, where* $\mathsf{recom}_2^{R\to S} = (\mathsf{z}_1, \ldots, \mathsf{z}_N)$

We remark that the above well-formedness property is "weak" in the sense that we only require $N-1$ out of the N tuples $\mathsf{recom}_{3,\ell}^{S\to R} = (\alpha_\ell, \beta_\ell)$ to be honestly generated (instead of requiring that all N tuples are honestly generated). This relaxation is crucial to establishing the B_{recom}-rewinding-security property for recom.

We now define an "admissibility" property for any input to the extractor.

Definition 25 (Admissible Inputs). *An input set* $(\mathsf{recom}_1, \{\mathsf{recom}_2^i, \mathsf{recom}_3^i\}_{i=1}^{B_{\mathsf{recom}}+1})$ *is said to be* admissible *if for every $i, j \in [B_{\mathsf{recom}}+1]$ s.t. $i \neq j$ and every $\ell \in [N]$, we have that $\mathsf{z}_\ell^i \neq \mathsf{z}_\ell^j$, where* $\mathsf{recom}_2^t = (\mathsf{z}_1^t, \ldots, \mathsf{z}_N^t)$.

Extractor $\mathsf{Ext}_{\mathsf{recom}}$**.** The extractor algorithm $\mathsf{Ext}_{\mathsf{recom}}$ is described in Figure 4.3.[11]

Lemma 2. *There exists a* PPT *extractor algorithm* $\mathsf{Ext}_{\mathsf{recom}}$ *such that, given a set of $(B_{\mathsf{recom}}+1)$ "well-formed" and "admissible" execution transcripts of* RECom *where each transcript consists of the same first round sender message, the extractor successfully extracts the value committed in each transcript, except with negligible probability.*

Input: An admissible set $(\mathsf{recom}_1, \{\mathsf{recom}_2^i, \mathsf{recom}_3^i\}_{i=1}^{B_{\mathsf{recom}}+1})$ where $\forall i$, $(\mathsf{recom}_1, \mathsf{recom}_2^i, \mathsf{recom}_3^i)$ is well-formed w.r.t. some value x_i.

1. For every $i \in [B_{\mathsf{recom}}+1]$, parse $\mathsf{recom}_2^i = (\mathsf{z}_1^i, \ldots, \mathsf{z}_N^i)$ and $\mathsf{recom}_3^i = (\mathsf{recom}_{3,1}^i, \ldots, \mathsf{recom}_{3,N+2}^i)$.
2. For each $\ell \in [N]$:
   - Parse $\mathsf{recom}_{3,\ell}^i = (\alpha_\ell^i, \beta_\ell^i)$.
   - Using polynomial interpolation, compute a degree B_{recom} polynomial p_ℓ over $\mathbb{Z}_q$ such that on point z_ℓ^i, $\mathsf{p}_\ell(\mathsf{z}_\ell^i) = \beta_\ell^i$.
   - Compute $x_\ell^i = (\alpha_\ell^i \oplus \mathsf{p}_\ell(0))$.
3. For every $i \in [B_{\mathsf{recom}}]$, let x^i be the value that equals a majority of the values in the set $\{x_1^i, \ldots, x_N^i\}$. If no such x^i value exists, set $x_i = \bot$.
4. Output $(x_1, \ldots, x_{B_{\mathsf{recom}}})$.

[11] An admissible input set consisting of $(B_{\mathsf{recom}}+1)$ "well-formed" execution transcripts of recom that share the same first round sender message can be obtained from a malicious sender via an expected PPT rewinding procedure. The expected PPT simulator in our application performs the necessary rewindings to obtain such transcripts and then feeds them to the extractor $\mathsf{Ext}_{\mathsf{recom}}$.

Figure 4.3: Strategy of algorithm $\mathsf{Ext}_{\mathsf{recom}}$.

Proof. We now analyze the extraction algorithm. Recall that for every $i \in [B_{\mathsf{recom}}+1]$, the transcript $(\mathsf{recom}_1, \mathsf{recom}_2^i, \mathsf{recom}_3^i)$ is well-formed w.r.t. some value x_i. By the definition of well-formedness, we have that for every i, there exists at most one $j \in [N]$ such that $\mathsf{recom}_{3,j}^i$ was not computed correctly and consistently with the other $\mathsf{recom}_{3,j'}^i$. This means that overall, across all $i \in [B_{\mathsf{recom}}+1]$ execution transcripts, there exists at most $(B_{\mathsf{recom}}+1)$ values of $\mathsf{recom}_{3,j}^i$ that were not computed correctly. This implies that for at least $(N - B_{\mathsf{recom}} - 1)$ values of j, the values $\mathsf{recom}_{3,j}^i$ were computed correctly in all $B_{\mathsf{recom}}+1$ transcripts. This means that for every $i \in [B_{\mathsf{recom}}+1]$, $(N - B_{\mathsf{recom}} - 1)$ out of N values $\{k_1^i, \ldots, k_N^i\}$ computed by the extractor are the same. Then, since $N - B_{\mathsf{recom}} - 1 \geq \frac{N}{2} + 1$, we have that the extractor computes the correct values k^i and x_i for every $i \in [B_{\mathsf{recom}}]$. $\square$

4.2.3 Trapdoor Generation Protocol with Bounded Rewind Security

In this section, taken largely verbatim from [BGJ+18], we discuss the syntax, definition and construction for a Trapdoor Generation Protocol with Bounded Rewind Security.

In a Trapdoor Generation Protocol, without bounded rewind security, a sender S (a.k.a. trapdoor generator) communicates with a receiver R. The protocol itself has no output, and the receiver has no input. The goal is for the sender to establish a trapdoor upon completion. On the one hand, the trapdoor can be extracted via a special extraction algorithm that has the ability to rewind the sender. On the other hand, no cheating receiver should be able to recover the trapdoor.

Syntax. A trapdoor generation protocol $\mathsf{TDGen} = (\mathsf{TDGen}_1, \mathsf{TDGen}_2, \mathsf{TDGen}_3, \mathsf{TDOut}, \mathsf{TDValid}, \mathsf{TDExt})$ is a three round protocol between two parties - a sender (trapdoor generator) S and receiver R that proceeds as below.

1. **Round 1 -** $\mathsf{TDGen}_1(\cdot)$**:**
   S computes and sends $\mathsf{td}_1^{S\to R} \leftarrow \mathsf{TDGen}_1(\mathsf{r}_S)$ using a random string r_S.

2. **Round 2 -** $\mathsf{TDGen}_2(\cdot)$**:**
   R computes and sends $\mathsf{td}_2^{R\to S} \leftarrow \mathsf{TDGen}_2(\mathsf{td}_1^{S\to R}; \mathsf{r}_R)$ using randomness r_R.

3. **Round 3 -** $\mathsf{TDGen}_3(\cdot)$**:**
   S computes and sends $\mathsf{td}_3^{S\to R} \leftarrow \mathsf{TDGen}_3(\mathsf{td}_2^{R\to S}; \mathsf{r}_S)$

4. **Output -** $\mathsf{TDOut}(\cdot)$
   The receiver R outputs $\mathsf{TDOut}(\mathsf{td}_1^{S\to R}, \mathsf{td}_2^{R\to S}, \mathsf{td}_3^{S\to R})$.

5. **Trapdoor Validation Algorithm -** $\mathsf{TDValid}(\cdot)$**:**
   Given input $(\mathsf{t}, \mathsf{td}_1^{S\to R})$, output a single bit 0 or 1 that determines whether the value t is a valid trapdoor corresponding to the message td_1 sent in the first round of the trapdoor generation protocol.

In what follows, for brevity, we set td_1 to be $\mathsf{td}_1^{S\to R}$. Similarly we use td_2 and td_3 instead of $\mathsf{td}_2^{R\to S}$ and $\mathsf{td}_3^{S\to R}$, respectively. Note that the algorithm $\mathsf{TDValid}$ does not form a part of the interaction between the trapdoor generator and the receiver. It is, in fact, a public algorithm that enables public verification of whether a value t is a valid trapdoor for a first round message td_1.

The protocol satisfies two properties: (i) Sender security, i.e., no cheating PPT receiver can learn a valid trapdoor, and (ii) Extraction, i.e., there exists an expected PPT algorithm (a.k.a. extractor) that can extract a trapdoor from an adversarial sender via rewinding.

Extraction. There exists a PPT extractor algorithm TDExt that, given a set of values[12] $(\mathsf{td}_1, \{\mathsf{td}_2^i, \mathsf{td}_3^i\}_{i=1}^3)$ such that $\mathsf{td}_2^1, \mathsf{td}_2^2, \mathsf{td}_2^3$ are distinct and $\mathsf{TDOut}(\mathsf{td}_1, \mathsf{td}_2^i, \mathsf{td}_3^i) = 1$ for all $i \in [3]$, outputs a trapdoor t such that $\mathsf{TDValid}(\mathsf{t}, \mathsf{td}_1) = 1$.

1-Rewinding Security. We define the notion of *1-rewinding security* for a trapdoor generation protocol TDGen. Intuitively, a Trapdoor Generation protocol is 1-rewind secure if it protects a sender against a (possibly cheating) receiver that has the ability to rewind it once. Specifically, the receiver is allowed to query the sender on two (possibly adaptive) different second round messages, thereby receiving two different third round responses from the sender. It should be the case that the trapdoor still remains hidden to the receiver.

Consider the following experiment between a sender S and any (possibly cheating) receiver R^*.
Experiment E:

- R^* interacts with S and completes one execution of the protocol TDGen. R^* receives values $(\mathsf{td}_1, \mathsf{td}_3)$ in rounds 1 and 3 respectively.
- Then, R^* rewinds S to the beginning of round 2.
- R^* sends S a new second round message td_2^* and receives a message td_3^* in the third round.
- At the end of the experiment, R^* outputs a value t^*.

Definition 26 (1-Rewinding Security). *A trapdoor generation protocol* $\mathsf{TDGen} = (\mathsf{TDGen}_1, \mathsf{TDGen}_2, \mathsf{TDGen}_3, \mathsf{TDOut}, \mathsf{TDValid})$ *achieves 1-rewinding security if, for every non-uniform* PPT *receiver* R^* *in the above experiment E,*

$$\Pr\left[\mathsf{TDValid}(\mathsf{t}^*, \mathsf{td}_1) = 1\right] = \mathsf{negl}(\lambda),$$

where the probability is over the random coins of S*, and where* t^* *is the output of* R^* *in the experiment E, and* td_1 *is the message from* S *in round 1.*

4.2.3.1 Construction

We now describe a three round trapdoor generation protocol based on one way functions. We first sketch the simple construction before providing a formal description. In the first round, the sender samples a signing key pair and sends the verification key to the receiver. The receiver queries a random message in the second round, and the sender responds with the corresponding signature in the third. The trapdoor is defined to be 3 distinct (message,signature) pairs. It is easy to see that both extraction and 1-rewind security are satisfied for this construction. Now, we formally describe the construction below.

[12]These values can be obtained from the malicious sender via an expected PPT rewinding procedure. The expected PPT simulator in our applications performs the necessary rewindings and then feeds these values to the extractor TDExt.

Let S and R denote the sender and the receiver, respectively. Let λ denote the security parameter. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ be a signature scheme that is existentially unforgeable against chosen-message attacks. Such schemes are known based on one-way functions [GMR88].

Protocol: Trapdoor Generation Protocol with Bounded Rewind Security

1. **Round 1 -** $\mathsf{TDGen}_1(\mathsf{r}_S)$**:**
   S does the following:
   - Generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(\mathsf{r}_S)$.
   - Send $\mathsf{td}_1^{S\to R} = \mathsf{vk}$ to R.
2. **Round 2 -** $\mathsf{TDGen}_2(\mathsf{td}_1^{S\to R})$**:**
   R sends a random string m as the message $\mathsf{td}_2^{R\to S}$ to S.
3. **Round 3 -** $\mathsf{TDGen3}(\mathsf{td}_1^{S\to R}, \mathsf{td}_2^{R\to S}; \mathsf{r}_S)$**:**
   S computes and sends $\mathsf{td}_3^{S\to R} = \mathsf{Sign}(\mathsf{sk}, \mathsf{m}; \mathsf{r}_\mathsf{m})$ where r_m is randomly chosen.
4. **Output: -** $\mathsf{TDOut}(\mathsf{td}_1^{S\to R}, \mathsf{td}_2^{R\to S}, \mathsf{td}_3^{S\to R})$
   The receiver R outputs 1 if $\mathsf{Vf}(\mathsf{td}_1^{S\to R}, \mathsf{m}, \mathsf{td}_3^{S\to R}) = 1$.
5. **Trapdoor Validation Algorithm -** $\mathsf{TDValid}(\mathsf{t}, \mathsf{td}_1)$**:**
   Given input $(\mathsf{t}, \mathsf{td}_1)$, the algorithm does the following:
   - Let $\mathsf{t} = \{\mathsf{m}_i, \sigma_i\}_{i=1}^{3}$.
   - Output 1 if $\mathsf{m}_1, \mathsf{m}_2, \mathsf{m}_3$ are distinct and $\mathsf{Vf}(\mathsf{td}_1, \mathsf{m}_i, \sigma_i) = 1$ for all $i \in [3]$.

Figure 4.4: Trapdoor Generation Protocol Π^{TD}.

Theorem 14. *Assuming the existence of one way functions, the protocol Π^{TD} described in Figure 4.4 is a 1-rewinding secure trapdoor generation protocol.*

We refer the reader to [BGJ+18] for the proof.

Extractor $\mathsf{TDExt}(\cdot)$**.** The extractor works as follows. It receives a verification key $\mathsf{vk} = \mathsf{td}_1$, and a set of values $\{\mathsf{m}_i, \sigma_i\}_{i=1}^{3}$ such that m_i are all distinct and $\mathsf{Vf}(\mathsf{vk}, \mathsf{m}_i, \sigma_i) = 1$ for every $i \in [3]$. Then, TDExt outputs $\mathsf{t} = \{\mathsf{m}_i, \sigma_i\}_{i=1}^{3}$ as a valid trapdoor. Correctness of the extraction is easy to see by inspection.

4.2.4 Witness Indistinguishable Proofs with Bounded Rewinding Security

We have previously defined delayed input witness indistinguishable arguments (WI) in Section 2.6. We now consider the notion of a WI that additionally satisfy B_{rwi}-bounded rewinding security, where the same statement is proven across all the rewinds. We refer to such primitives as B_{rwi}-bounded rewind secure WI.

The intuition for the definition is similar to that of the trapdoor generation protocol as described in the previous section. Here, for the three round delayed-input witness indistinguishable argument we want witness indistinguishability to be preserved as long as the verifier is restricted to rewinding the prover B_{rwi}-1 times. Specifically, the prover sends its first round message to the verifier, who then chooses (i) a triple consisting of a statement, and any two corresponding witnesses w_0 and w_1; (ii) B_{rwi}-1 second round verifier messages for the single first round prover message, along with the corresponding statement and witness for that rewind thread. The prover then completes the protocol, responding to each of the B_{rwi}-1 verifier messages, and using either witness w_0 or w_1 for the main thread.

Definition 27 (3-Round Delayed-Input WI with Non-Adaptive Bounded Rewinding Security). *[BGJ+18] Fix a positive integer B_{rwi}. A delayed-input 3-round interactive argument (as defined in Definition 9) for an* NP *language $\mathcal{L}$, with an* NP *relation $R_{\mathcal{L}}$ is said to be WI with B_{rwi}-*Rewinding Security *if for every non-uniform* PPT *interactive Turing Machine V^*, it holds that $\{\mathsf{REAL}_0^{V^*}(1^\lambda)\}_\lambda$ and $\{\mathsf{REAL}_1^{V^*}(1^\lambda)\}_\lambda$ are computationally indistinguishable, where for $b \in \{0,1\}$ the random variable $\mathsf{REAL}_b^{V^*}(1^\lambda)$ is defined via the following experiment. In what follows we denote by P_1 the prover's algorithm in the first round, and similarly we denote by P_3 its algorithm in the third round. We now define it formally below.*

***Experiment** $\mathsf{REAL}_b^{V^*}(1^\lambda)$:*

1. *Run $\mathsf{P}_1(1^\lambda)$ and denote its output by (rwi_1, σ), where σ is its secret state, and rwi_1 is the message to be sent to the verifier.*
2. *Run the verifier $V^*(1^\lambda, \mathsf{rwi}_1)$, who outputs $\{(x^i, w^i)\}_{i\in[B_{\mathsf{rwi}}-1]}$, $x^{B_{\mathsf{rwi}}}, w_0^{B_{\mathsf{rwi}}}, w_1^{B_{\mathsf{rwi}}}$ and a set of messages $\{\mathsf{rwi}_2^i\}_{i\in[B_{\mathsf{rwi}}]}$.*
3. *For each $i \in [B_{\mathsf{rwi}} - 1]$, run $\mathsf{P}_3(\sigma, \mathsf{rwi}_2^i, x^i, w^i)$, and for $i = B_{\mathsf{rwi}}$, run $\mathsf{P}_3(\sigma, \mathsf{rwi}_2^i, x^i, w_b^i)$ where P_3 is the (honest) prover's algorithm for generating the third message of the WI protocol. Send the resulting messages $\{\mathsf{rwi}_3^i\}_{i\in[B_{\mathsf{rwi}}]}$ to V^*.*
4. *The output of the experiment is the output of V^*.*

The following theorem is proven in [GR19].

Imported Theorem 2 ([GR19]). *Assuming non-interactive commitments, for every (polynomial) rewinding parameter B, there exists a three round delayed-input witness-indistinguishable argument system with B-rewinding security.*

4.2.5 Non-Malleable Commitments

In this section, we recall the definition of non-malleable commitments (NMCOM) and describe some additional properties that are relevant to our use case. In particular, we define *special non-malleable commitments* that capture the exact requirements that we need from a three round NMCOM for our application to MPC. We then provide an instantiation of such special NMCOM via the scheme of Goyal et al. [GPR16].

We start by proving some background on how NMCOMs are used in many prior works for bootstrapping non-malleability in a constant-round MPC protocol. We then briefly discuss the issue of "over-extraction" in NMCOMs and how it guides some of our requirements from special NMCOMs.

Bootstrapping Non-Malleability via NMCOMs. As noted in many prior works, standard soundness guarantees of ZK proofs do not suffice in the design of constant-round MPC protocols. In particular, since the proofs given by various parties are executed in parallel, we need to ensure that the proofs given by adversarial parties remain sound even when the honest party proofs are simulated [Sah99].

Many prior works use the following template to ensure the above property: the parties are required to send a non-malleable commitment (NMCOM) to a random value, and then prove that either they are behaving "honestly" or the NMCOM commits to a "trapdoor" string, which is determined via a separate "trapdoor generation" sub-protocol. Such a use of NMCOM intuitively suffices to bootstrap non-malleability throughout the protocol. Indeed, the main idea is that in order to ensure "simulation soundness" across the hybrids in the security proof, it suffices to prove an *invariant* that the adversary never commits to the trapdoor in its NMCOM. If the NMCOM scheme supports extraction of the committed value, then it is indeed possible to prove that the invariant holds:

– First, the invariant is established in the real world, i.e., the first hybrid, by simply extracting the value inside adversary's NMCOM and invoking the security of the trapdoor generation protocol.

– In subsequent hybrids, we continue to argue that the invariant holds via one of the following two strategies: (i) In all but one of the hybrids, we use the NMCOM extractor to argue that the value inside adversary's NMCOM does not change from the previous hybrid. (ii) In one specific hybrid – referred to as the "NMCOM-hybrid" – where we switch the value inside the honest party NMCOM, we simply rely on the non-malleability property of NMCOM to argue that the value committed by the adversary did not change.

In the design of four-round MPC, due to aborting adversaries, it is imperative to use a *three* round NMCOM to implement the above strategy. Towards this end, we rely upon the three round NMCOM scheme of Goyal et al. [GPR16] in order to minimize the use of assumptions in our protocol. An important weakeness, however, of their NMCOM scheme is that it suffers from "over-extraction", namely, the extractor can output a valid (non-$\perp$) value even if the adversary's committed to $\perp$ (i.e., its commitment was not valid). This, unfortunately, leads to a failure in the implementation of the above strategy as the extractor could output the trapdoor even when the adversary was committing to $\perp$ in the NMCOM.

We crucially observe that a weak "split-state" extractor used inside the security proof of Goyal et al's NMCOM scheme satisfies useful properties that suffice for our application. Specifically, it guarantees the following two properties: (1) If we switch the honest party commitment from m_0 to m_1, the value extracted from adversary's NMCOM does not change, (2) If the adversary sends a well-formed commitment to some value m, then with noticeable probability, the output of the extractor is m. Using these properties, we can establish simulation-soundness as follows. We first strengthen the above invariant to claim that a particular extractor, when applied on the adversary's NMCOM, does not output the trapdoor. Then, throughout the hybrids, we first use the above extractor to argue that the value extracted from adversary's NMCOM is not the trapdoor. Then, using the second property, we can argue that the adversary must not be committing to the trapdoor.

4.2.5.1 Definitions

We start with the definition of non-malleable commitments by Pass and Rosen [PR05] and further refined by Lin et al [LPV08] and Goyal [Goy11]. (All of these definitions build upon the original definition of Dwork et al. [DDN91]).

In the real experiment, a man-in-the-middle adversary MIM interacts with a committer C in the left session, and with a receiver R in the right session. Without loss of generality, we assume that each session has identities or tags, and require non-malleability only when the tag for the left session is different from the tag for the right session.

At the start of the experiment, the committer C receives an input val and MIM receives an auxiliary input z, which might contain a priori information about val. Let $\mathsf{MIM}_{\langle C,R\rangle}(\mathsf{val}, z)$ be a random variable that describes the value $\widetilde{\mathsf{val}}$ committed by MIM in the right session, jointly with the view of MIM in the real experiment.

In the ideal experiment, a PPT simulator $\mathcal{S}$ directly interacts with MIM. Let $\mathsf{Sim}_{\langle C,R\rangle}(1^\lambda, z)$ denote the random variable describing the value $\widetilde{\mathsf{val}}$ committed to by $\mathcal{S}$ and the output view of $\mathcal{S}$.

In either of the two experiments, if the tags in the left and right interaction are equal, then the value $\widetilde{\mathsf{val}}$ committed in the right interaction, is defined to be $\perp$.

Definition 28 (Synchronous Non-malleable Commitments). *A 3-round commitment scheme $\langle C, R\rangle$ is said to be synchronous non-malleable if for every* synchronizing[13] PPT MIM*, there exists a* PPT *simulator $\mathcal{S}$ such that the following ensembles are computationally indistinguishable:*

$$\{\mathsf{MIM}_{\langle C,R\rangle}(\mathsf{val}, z)\}_{\lambda\in\mathbb{N},\mathsf{val}\in\{0,1\}^\lambda,z\in\{0,1\}^*} \text{ and } \{\mathsf{Sim}_{\langle C,R\rangle}(1^\lambda, z)\}_{\lambda\in\mathbb{N},\mathsf{val}\in\{0,1\}^\lambda,z\in\{0,1\}^*}$$

Non-malleability with Respect to Extraction. In this section we consider also a different notion of non-malleability that we call *non-malleability with respect to extraction*. Consider the experiment in which the adversary interacts with an honest committer C in the left session, and with an extractor $\mathsf{Ext}_{\mathsf{NMCom}}$ in the right session which guarantees the extraction of the committed value only when the adversary computes a well-formed commitment (if the commitment is ill-formed that it is not guaranteed that the extractor outputs $\perp$). Without loss of generality, we assume that each session has identities or tags, and require our non-malleability property to hold only when the tag for the left session is different from the tag for the right session.

At the start of the experiment, the committer C receives an input m and $\mathcal{A}^{\mathsf{NMCom}}$ receives an auxiliary input z, which might contain a priori information about m. Let $\mathsf{MIM}^{\mathtt{Ext}}_{\langle C,\mathsf{Ext}_{\mathsf{NMCom}}\rangle}(\ m\ , z)$ be a random variable that describes the value $\widetilde{\mathsf{val}}$ output by $\mathsf{Ext}_{\mathsf{NMCom}}$ jointly with the view of $\mathcal{A}^{\mathsf{NMCom}}$ in the real experiment.

In either of the two experiments, if the tags in the left and right interaction are equal, then the value $\widetilde{\mathsf{val}}$ committed in the right interaction, is defined to be $\perp$.

Definition 29. *A 3-round commitment scheme $\langle C, R\rangle$ is said to be* non-malleable with respect to extraction *if for every* synchronizing PPT $\mathcal{A}^{\mathsf{NMCom}}$ *there exists an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$ *such that the following ensembles are computationally indistinguishable:*

$$\{\mathsf{MIM}^{\mathtt{Ext}}_{\langle C,\mathsf{Ext}_{\mathsf{NMCom}}\rangle}(\ m_0\ , z)\}_{\lambda\in\mathbb{N},m_0\in\{0,1\}^\lambda,z\in\{0,1\}^*} \text{ and } \{\mathsf{MIM}^{\mathtt{Ext}}_{\langle C,\mathsf{Ext}_{\mathsf{NMCom}}\rangle}(\ m_1\ , z)\}_{\lambda\in\mathbb{N},m_1\in\{0,1\}^\lambda,z\in\{0,1\}^*}$$

[13]A synchronizing adversary is one that sends its message for every round before obtaining the honest party's message for the next round.

Delayed-input non-malleability. In a delayed-input non-malleable commitment scheme $\langle C, R\rangle$, the sender C can specify the message to commit to in the last round of the protocol. Additionally, we require $\langle C, R\rangle$ to be secure even against an adversary that picks val adaptively on the first round received from C. More formally, consider the following two experiments.

1) C and R interact which MIM, and in the second last round MIM sends val to C. C then commits to val by completing the last round of the protocol. Let $\mathsf{MIM}^0_{\langle C,R\rangle}(z)$ be a random variable that describes the value $\widetilde{\mathsf{val}}$ committed by MIM in the right session, jointly with the view of MIM in the real experiment.

2) C and R interact which MIM, and in the second last round MIM sends val to C. C then picks a random string r and commits to r by completing the last round of the protocol. Let $\mathsf{MIM}^1_{\langle C,R\rangle}(z)$ be a random variable that describes the value $\widetilde{\mathsf{val}}$ committed by MIM in the right session, jointly with the view of MIM in the real experiment.

Definition 30 (Delayed-Input Non-malleable Commitments). *A 3-round commitment scheme $\langle C, R\rangle$ is said to be delayed-input non-malleable if for every* PPT MIM*, there exists a* PPT *simulator $\mathcal{S}$ such that the following ensembles are computationally indistinguishable:*

$$\{\mathsf{MIM}^0_{\langle C,R\rangle}(z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \textit{ and } \{\mathsf{MIM}^1_{\langle C,R\rangle}(z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

In [COSV16] the authors, in order to construct their non-malleable commitment scheme, need to implicitly show how to transform a standard 3-round non-malleable commitment scheme $\langle C, R\rangle$ into a 3-delayed-input non-malleable commitment scheme $\langle C', R'\rangle$. The idea is the following.

1. To compute the first round C' samples a random message m_0 and runs C on input m_0.
2. R' simply runs R, obtains the second round and sends it to C'
3. C', on input the message to be committed m computes $m_0 \oplus m = m_1$, run C to obtain the last message, and send it thogeter with m_1.

The opening of $\langle C', R'\rangle$ corresponds to the opening of $\langle C, R\rangle$, where R' needs to compute the exclusive-or of the opened message with m_1 to reconstruct the committed message.

It should be easy to see that the above scheme is correct. The proof that the scheme is delayed-non malleable follow form a simple reduction to the non-malleability of $\langle C, R\rangle$. We refer to Lemma 3 of [COSV16] for the formal proof. We remark that in [COSV16] the authors do not make a stand-alone claim, but this is implicit in their proof.

Special Non-Malleable Commitments. We are now ready to define the notion of non-malleability required for our construction.

Definition 31 (Special Non-malleable Commitments). *A three round commitment scheme $\langle C, R\rangle$ is said to be special non-malleable if:*

– *$\langle C, R\rangle$ is synchronous non-malleable and non-malleable with respect to extraction.*

– *$\langle C, R\rangle$ is delayed-input.*

– *$\langle C, R\rangle$ satisfies* last-message pseudorandomness*, that is, for every non-uniform* PPT *receiver R^*, it holds that $\{\mathsf{REAL}_0^{R^*}(1^\lambda)\}_\lambda$ and $\{\mathsf{REAL}_1^{R^*}(1^\lambda)\}_\lambda$ are computationally indistinguishable, where for $b \in \{0, 1\}$, the random variable $\mathsf{REAL}_b^{R^*}(1^\lambda)$ is defined via the following experiment.*

1. *Run $C(1^\lambda)$ and denote its output by (com_1, σ), where σ is its secret state, and com_1 is the message to be sent to the receiver.*
2. *Run the receiver $R^*(1^\lambda, \mathsf{com}_1)$, who outputs a message com_2.*
3. *If $b = 0$, run $C(\sigma, \mathsf{com}_2)$ and send its message com_3 to R^*. Otherwise, if $b = 1$, compute $\mathsf{com}_3 \leftarrow\!\!\$ \{0,1\}^m$ and send it to R^*. Here $m = m(\lambda)$ denotes $|\mathsf{com}_3|$.*
4. *The output of the experiment is the output of R^*.*

Goyal et al. [GPR16] construct three-round special non-malleable commitments satisfying Definition 31 based on non-interactive commitments.

Imported Theorem 3 ([GPR16]). *Assuming non-interactive commitments, there exists a three round non-malleable commitment satisfying Definition 31.*

For completeness, we propose a proof sketch of the above theorem.

4.2.5.2 Proof of Special Non-Malleable Commitments

Let $(\mathsf{Com}, \mathsf{Dec})$ be a non-interactive statistically binding commitment scheme, and (E, D) be a split-state non-malleable code that splits the input into two codewords L and R. The scheme $\mathsf{NMCom} = (\mathsf{Sen}, \mathsf{Rec})$ proposed in [GPR16] can be described as follows.
Commitment phase. Let m be the message to be committed.

$\mathsf{Sen} \to \mathsf{Rec}$: Sen chooses $(L, R) \leftarrow \mathsf{Enc}(m)$ where L is viewed as a field element in $\mathbb{Z}_q$; Sen also draws $r \leftarrow \mathbb{Z}_q$ at random, compute $\mathsf{Com}, \mathsf{dec} \leftarrow \mathsf{Com}(L||r)$ and sends Com to Rec.

$\mathsf{Rec} \to \mathsf{Sen}$: Rec chooses a random $\alpha \leftarrow \mathbb{Z}_q^\star$ and sends it to Sen.

$\mathsf{Sen} \to \mathsf{Rec}$: Sen sends $a = r\alpha + L$ and R to Rec.

Decommitment phase To decommit, Sen sends dec to Rec.

Intuitively, Sen commits to a polynomial-based 2-out-of-2 secret sharing of L in the first round, and in the third round sends R along with one share. We now give an intuition about why this commitment scheme is special non-malleable. We refer the reader to [GPR16] for the formal proof.

Non-malleability against synchronizing PPT adversary. In [GPR16] the authors show how to use an adversary $\mathcal{A}^{\mathsf{NMCom}}$ that breaks the non-malleability of $(\mathsf{Sen}, \mathsf{Rec})$ to construct two tampering functions (f, g) that break the security of the underling split-state non-malleable code. The functions f and g share a partial transcript consisting of the first two messages of an interaction of $(\mathsf{Sen}, \mathsf{Rec})$ with $\mathcal{A}^{\mathsf{NMCom}}$ and the value a. Note that g contains a non-interactive commitment of L and this could be an issue given that the non-malleable code is split-state (and therefore g should not contain information about L). However, this does not represent a problem since L is committed and from the hiding of the non-interactive commitment L can be replaced with another value without the adversary noticing that. The output of g simply consists of the value $\tilde{R}$ that is sent from $\mathcal{A}^{\mathsf{NMCom}}$ to the receiver in the last round (more details on how g works are given later in this section).

The function f does not contain any information about R, but in this case the challenging part is to compute its output since the left part ($\tilde{L}$) of the non-malleable code is committed. However, f can extract $\tilde{L}$ by rewinding $\mathcal{A}^{\mathsf{NMCom}}$. In more details, f on input L chooses a random value $R_\$$

and sends $(a, R_\$)$ to $\mathcal{A}^{\mathsf{NMCom}}$. Upon receiving $(\tilde{a}_\$, \tilde{R}_\$)$ from $\mathcal{A}^{\mathsf{NMCom}}$, f rewinds the adversary and sends a freshly generated second round $\tilde{\beta}$ on the right and upon receiving β on the left f computes and sends (b, R) where $b = (a - L)(\beta/\alpha) + L$. At this point f receives $(\tilde{b}, \cdot)$ on the right from the adversary and computes its output, which consists of the constant term on the line spanned by $\{(\tilde{a}_\$, \tilde{\alpha}), (\tilde{b}, \tilde{\beta})\}$.

We are now ready to complete the description of the function g. This function also shares the random value $R_\$$ and therefore it can compute $\tilde{a}_\$$. At this point $g(R)$ rewinds $\mathcal{A}^{\mathsf{NMCom}}$ and sends (a, R) on the left and receives $(\tilde{a}, \tilde{R})$ on the right. If $\tilde{a} = \tilde{a}_\$$ then $g(R)$ outputs $\tilde{R}$, otherwise it outputs $\perp$.

Note that for (f, g) to succeed in extracting $(\tilde{L}, \tilde{R})$, it must be that the answer $\tilde{a}_\$$ $\mathcal{A}^{\mathsf{NMCom}}$ provides when given the random $R_\$$ is equal to the $\tilde{a}$ he provides given R. This will follow from and additional property that the authors of [GPR16] require on the underling non-malleable code. Given this property the authors show that the chance that $\mathcal{A}^{\mathsf{NMCom}}$ answers correctly (i.e., consistently with the linear map he committed to in the first round) given $R_\$$ is about the same as the chance he answers correctly given R. So either both are incorrect with high probability, in which case $\mathcal{A}^{\mathsf{NMCom}}$ is always committing to $\perp$ and so cannot be mauling; or is it possible to show that f and g extract the correct value.

Delayed-input property. As we have argued in Section 4.2.5.1 any three round non-malleable commitment scheme can be turn into a delayed-input non-malleable commitment scheme. It is worth nothing that the transformation preserves all the property of the original commitment scheme in this case.

Last-message pseudorandomness. This property comes immediately from the hiding of the non-interactive commitment Com and from the fact that R is the right state of a split-state non-malleable code, which is also a 2-out-of-2 secret sharing (like any split-state non-malleable code).

Non-malleability with respect to extraction. To show that this property holds we first need to construct an extractor $\mathsf{Ext}_{\mathsf{NMCom}}$. $\mathsf{Ext}_{\mathsf{NMCom}}$ interacts with the the adversarial sender using random coins α acting as the honest receiver in the right session. Let $\tau = (\mathsf{Com}, \alpha, a, R, \tilde{\mathsf{Com}}, \tilde{\alpha}, \tilde{a}, \tilde{R})$ be the transcript of $\mathcal{A}^{\mathsf{NMCom}}$'s view, $\mathsf{Ext}_{\mathsf{NMCom}}$ extracts $\tilde{L}$ and $\tilde{R}$ in two steps.

- To extract $\tilde{L}$ $\mathsf{Ext}_{\mathsf{NMCom}}$ chooses a random value $R_\$$ and sends $(a, R_\$)$ to $\mathcal{A}^{\mathsf{NMCom}}$. Upon receiving $\tilde{a}_\$, \tilde{R}_\$$ from $\mathcal{A}^{\mathsf{NMCom}}$, f rewinds the adversary and sends a freshly generated second round $\tilde{\beta}$ on the right and upon receiving β computes and sends (b, R) where $b = (a - L)(\beta/\alpha) + L$. Upon receiving $(\tilde{b}, \cdot)$ on the right by the adversary, $\mathsf{Ext}_{\mathsf{NMCom}}$ computes $\tilde{L}$, which consists of the constant term on the line spanned by $\{(\tilde{a}_\$, \tilde{\alpha}), (\tilde{b}, \tilde{\beta})\}$.

- To extract $\tilde{R}$ $\mathsf{Ext}_{\mathsf{NMCom}}$ checks if $\tilde{a} = \tilde{a}_\$$. If it is the case then $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs $\mathsf{D}(\tilde{L}, \tilde{R})$, otherwise he outputs $\perp$.

In summary, $\mathsf{Ext}_{\mathsf{NMCom}}$ simply runs the extraction procedures described by the function f and g defined in the non-malleability proof of [GPR16] (that we have also sketched above). We now note that an adversary attacking the property of non-malleability with respect to extraction yields to an adversary for the non-malleable code. The only difference with the non-malleability proof of [GPR16] is that we do not need to check whether the extracted values actually corresponds to the committed value. That is, the adversary could compute an ill-formed commitment that yields to the extraction of a message $m \neq \perp$. We note, however, that if the commitment is well formed then $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs the actual committed value (we refer the reader to [GPR16, Claim 8] for the formal proof).

4.3 Oblivious Transfer with Bounded Rewind Security

In this section we define, and then construct, a strengthening of regular oblivious transfer. We construct a rewinding secure Oblivious Transfer (OT) assuming the existence of four round OT protocol. For an OT protocol to be rewind secure, we require security against an adversary who is allowed to re-execute the second and third round of the protocol multiple times. But the first and fourth round are executed only once.

4.3.1 Definition

We start by formalizing the notion of a rewind secure oblivious transfer protocol. We shall denote by $\mathsf{Out}_R\langle S(x), R(y)\rangle$ the output of the receiver R on execution of the protocol between R with input y, and sender S with input x. The four round oblivious transfer protocol is specified by four algorithms OT_j for $j \in [4]$; and the corresponding output protocol message is denoted by ot_j. We consider a delayed receiver input notion of the protocol where the receiver input is only required for the computation of ot_3.

Definition 32. *An interactive protocol (S, R) between a polynomial time sender S with inputs s_0, s_1 and polynomial time receiver R with input b, is a four round bounded rewind secure oblivious transfer (OT) if the following properties hold:*

Correctness *For any selection bit b, for any messages $s_0, s_1 \in \{0,1\}$, it holds that*

$$\Pr\left[\ \mathsf{Out}_R\langle S(s_0, s_1), R(b)\rangle = s_b\right] = 1$$

where the probability is over the random coins of the sender S and receiver R.

Security against Malicious Sender with B rewinds. *Here, we require indistinguishability security against a malicious sender where the receiver uses input $b[k]$ in the k-th rewound execution of the second and third round. Specifically, consider the experiment described below. $\forall \left\{b^0[k], b^1[k]\right\}_{k\in[B]} \in \{0,1\}$ where **Experiment** E^σ:*

1. *Run OT_1 to obtain ot_1 which is independent of the receiver input. Send to $\mathcal{A}$.*
2. *$\mathcal{A}$ then returns $\{\mathsf{ot}_2[j]\}_{j\in[B]}$ messages.*
3. *For each $j \in [B]$, run OT_3 on $(\mathsf{ot}_1, \mathsf{ot}_2[j], b^\sigma[j])$ and send the response to $\mathcal{A}$.*
4. *The output of the experiment is the entire transcript.*

We say that the scheme is secure against malicious senders with B rewinds if the experiments E^0 and E^1 are indistinguishable.

4.3.2 Construction

We describe below the protocol Π^{R} which achieves rewind security against malicious senders. The Sender S's input is $s_0, s_1 \in \{0,1\}$ while the receiver R's input is $b \in \{0,1\}$.

Components. We require the following two components:

- $n \cdot \mathsf{B}_{\mathsf{OT}}$ instances of a 4 round OT protocol which achieves indistinguishability security against malicious senders.
- $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$ is a secure garbling scheme (see Section 2.12).

Protocol. The basic idea is to split the receiver input across multiple different OT executions such that during any rewind, a different set of OTs will be selected to proceed with the execution thereby preserving the security of the receiver's input. The sender constructs a garbled circuit which is used to internally recombine the various inputs shares and only return the appropriate output. The protocol is described below.

Protocol (S, R): Oblivious Transfer Protocol with Bounded Rewind Security

---

Round 1. **(Π^{R}_1)** : The receiver R computes the first round message of all the OTs. $\forall i \in [n], k \in [\mathsf{B}_{\mathsf{OT}}]$, $\mathsf{ot}_1^{i,k} := \mathsf{OT}_1\left(1^\lambda; \mathsf{r}_R\right)$ and send $\left\{\mathsf{ot}_1^{i,k}\right\}_{i\in[n],k\in[\mathsf{B}_{\mathsf{OT}}]}$ to S. We refer to index i as the outer index, and k as the inner index.

Round 2. **(Π^{R}_2)** : The sender S responds to all of the OT messages. $\forall i \in [n], k \in [\mathsf{B}_{\mathsf{OT}}]$, compute $\mathsf{ot}_2^{i,k} := \mathsf{OT}_2\left(\mathsf{ot}_1^{i,k}; \mathsf{r}_S\right)$ and sends $\left\{\mathsf{ot}_2^{i,k}\right\}_{i\in[n],k\in[\mathsf{B}_{\mathsf{OT}}]}$ to R.

Round 3. **(Π^{R}_3)** : The receiver now selects only a single OT to continue for i. It then encodes its input b by computing n additive shares and using each share as an input to a separate OT. Specifically, receiver R does the following:

- Compute n additive shares of b. Specifically, sample the first $n-1$ shares at random $\forall \ell \in [n-1]$ $b_\ell \leftarrow\!\$ \{0,1\}$ and set the last share $b_n := b \bigoplus_{\ell=1}^{n-1} b_j$.
- Sample within each tuple, the index for which to continue the OT. $\forall i \in [n]$, $\sigma_i \leftarrow\!\$ [\mathsf{B}_{\mathsf{OT}}]$.
- Use input b_i to compute the receiver message for $\mathsf{ot}_3^{i,\sigma_i}$. The other OTs are discontinued. Specifically, $\forall i \in [n]$, compute $\mathsf{ot}_3^{i,\sigma_i} \leftarrow \mathsf{OT}_3\left(b_i, \mathsf{ot}_1^{i,\sigma_i}, \mathsf{ot}_2^{i,\sigma_i};\ \mathsf{r}_R\right)$ and send $\left\{\mathsf{ot}_3^{i,\sigma_i},\ \sigma_i\right\}_{i\in[n]}$ to S.

Round 4. **(Π^{R}_4)** : The sender encodes its inputs (s_0, s_1) in a garbled circuit and uses the corresponding labels to complete the OT protocol.

- Compute garbled circuit: $\left(\overline{\mathsf{C}}_{\mathsf{ot}}, \overline{\mathsf{lab}}\right) := \mathsf{Garble}\left(\mathsf{C}_{\mathsf{ot}}\left[s_0, s_1\right]; \mathsf{r}_{\mathsf{gc},i}\right)$, where Circuit $\mathsf{C}_{\mathsf{ot}}[s_0, s_1]$ on input $b_1, \ldots, b_n$ outputs s_b where $b := \bigoplus_{i=1}^{n} b_i$.

– For $i \in [n]$, compute $\mathsf{ot}_4^{i,\sigma_i} := \mathsf{OT}_4\left(\mathsf{lab}_{i,0}, \mathsf{lab}_{i,1}, \mathsf{ot}_1^{i,\sigma_i}, \mathsf{ot}_2^{i,\sigma_i}, \mathsf{ot}_3^{i,\sigma_i}; \mathsf{r}_S\right)$ and send $\left\{\mathsf{ot}_4^{i,\sigma_i}\right\}_{i\in[n]}$ to R.

Evaluation. (OTEval') : The receiver R now evaluates the OT protocol to obtain labels needed to evaluate the output of the garbled circuit.

– For $i \in [n]$, compute $\widehat{\mathsf{lab}}_i := \mathsf{OTEval}\left(b_i, \mathsf{ot}_1^{i,\sigma_i}, \mathsf{ot}_2^{i,\sigma_i}, \mathsf{ot}_3^{i,\sigma_i}, \mathsf{ot}_4^{i,\sigma_i};\ \mathsf{r}_R\right)$

– Output $s' := \mathsf{Eval}\left(\overline{\mathsf{C}}_{\mathsf{ot}}, \left\{\widehat{\mathsf{lab}}_i\right\}_{i\in[n]}\right)$

Security. We prove security of our constructed protocol below.

Lemma 3. *Assuming receiver indistinguishability of* OT *against malicious senders, the receiver input in* Π^{R} *remains indistinguishable under* B_{OT}*-rewinds.*

Proof. Suppose the B_{OT} inputs used by the receiver are

$$b^0[1], \cdots, b^0[\mathsf{B}_{\mathsf{OT}}] \text{ and } b^1[1], \cdots, b^1[\mathsf{B}_{\mathsf{OT}}]$$

in experiment 0 and 1 respectively, where $b[j]$ is the receiver input in the j-th rewind. We want to show that an adversarial rewinding sender's view is indistinguishable in both experiments.

We do this by via a sequence of hybrids, where in hybrid ℓ we change the input of the ℓ-th rewind. Consider two adjacent hybrids, $\mathsf{Hyb}_{\ell-1}$ and Hyb_ℓ which use inputs

$$b^1[1], \cdots, b^1[\ell-1], b^0[\ell], \cdots, b^0[\mathsf{B}_{\mathsf{OT}}] \text{ and } b^1[1], \cdots, b^1[\ell-1], b^1[\ell], \cdots, b^0[\mathsf{B}_{\mathsf{OT}}]$$

respectively.

Suppose there is an adversarial sender $\mathcal{A}$ that can distinguish $\mathsf{Hyb}_{\ell-1}$ and Hyb_ℓ, then we construct an adversary $\mathcal{A}_{\mathsf{OT}}$ that breaks the indistinguishability security of OT. We now describe the working of $\mathcal{A}_{\mathsf{OT}}$.

To rely on the security of OT, we need to find an instance of OT that is not rewound during the experiment. Since the OT indices are sampled independently and uniformly, with non-negligible probability, any given outer index i will have inner indices in each of the B_{OT} rewinds to be distinct. The probability being non-negligible follows from the fact that B_{OT} is a constant.

We sample an outer index $\widetilde{i}$ randomly from $[n]$. We will expose one of the OTs from this tuple to an external OT receiver. To determine the index of the exposed OT, $\forall i \in [n], \ell \in [\mathsf{B}_{\mathsf{OT}}]$, sample

$$\sigma_i[\ell] \leftarrow\!\!\$\ [\mathsf{B}_{\mathsf{OT}}].$$

Here we denote by $\sigma_i[\ell]$, the inner index picked for the ℓ-th rewind. If for $\widetilde{i}$, $\{\sigma_{\widetilde{i}}[\ell]\}_{\ell\in[\mathsf{B}_{\mathsf{OT}}]}$ are not distinct, we sample again. Thus, the OT we will expose externally is the one with outer index $\widetilde{i}$, and inner index $\sigma_{\widetilde{i}}[\ell]$.

Specifically, on receiving ot_1 message from the external challenger set

$$\mathsf{ot}_1^{\widetilde{i},\sigma_{\widetilde{i}}[\ell]} = \mathsf{ot}_1.$$

All other $\mathsf{ot}_1^{i,k}$ messages are computed honestly, using fresh randomness, by $\mathcal{A}_{\mathsf{OT}}$. All first round messages are sent to $\mathcal{A}$.

$\mathcal{A}$ responds with $\left\{\mathsf{ot}_2^{i,k}[j]\right\}_{i\in[n],k\in[\mathsf{B_{OT}}],j\in[\mathsf{B_{OT}}]}$ where $\mathsf{ot}_2^{i,k}[j]$ corresponds to the sender message to be used in the j-the thread.

From our assumption of distinct indices for outer index $\widetilde{i}$, $\forall \ell \neq \ell'$, $\sigma_{\widetilde{i}}[\ell] \neq \sigma_{\widetilde{i}}[\ell']$. This means that $\mathsf{ot}_2^{\widetilde{i},\sigma_{\widetilde{i}}[\ell]}$ is only going to be picked once across rewinds. Thus $\mathsf{ot}_2^{\widetilde{i},\sigma_{\widetilde{i}}[\ell]}$ can be forwarded to the external challenger without any fear of rewinding. But we also need to send it two challenge receiver bits, which we compute below.

For receiver inputs to the OT, we need to generate additive shares: $\forall i \in [n-1], \ell' \in [\mathsf{B_{OT}}] \setminus \{\ell\}$

$$b_i[\ell'] \leftarrow\$ \{0,1\}$$

Now to complete the sharing, we need to set the last share bit appropriately. This is done as follows: $\forall \ell'$,

– if $\ell' < \ell$,

$$b_n[\ell'] := b^1[\ell'] \bigoplus_{i=1}^{n-1} b_i[\ell']$$

– if $\ell' > \ell$,

$$b_n[\ell'] := b^0[\ell'] \bigoplus_{i=1}^{n-1} b_i[\ell']$$

Now for ℓ, we want the $\widetilde{i}$-th share to differ, but all others to be the same. With this in mind, sample $\forall i \in [n] \setminus \left\{\widetilde{i}\right\}$

$$b_i[\ell] \leftarrow\$ \{0,1\}$$

We set two special shares below:

$$b^{*,0} := b^0[\ell] \bigoplus_{\substack{i=1\\ i\neq\widetilde{i}}}^{n} b_i[\ell] \text{ and } b^{*,1} := b^1[\ell] \bigoplus_{\substack{i=1\\ i\neq\widetilde{i}}}^{n} b_i[\ell]$$

Now if we set the challenge to be $(b^{*,0}, b^{*,1})$ then depending on the receiver bit chosen by the external challenger, we are either in hybrid $\mathsf{Hyb}_{\ell-1}$ and Hyb_{ℓ}.

Once we send the challenge, we get as response the 3round OT message corresponding to the choice bit sampled by the challenger. The remaining OT messages can be answered internally using the shares computed. The collected third round messages are now sent to $\mathcal{A}$. Thus, if $\mathcal{A}$ can distinguish the two hybrids with non-negligible probability, then $\mathcal{A}_{\mathsf{OT}}$ wins the challenge game with non-negligible probability. The only loss in advantage comes from the probability of sampling $\mathsf{B_{OT}}$ inner indices from the set $[\mathsf{B_{OT}}]$ such that the indices are all distinct. Since $\mathsf{B_{OT}}$ is a constant, this still leaves the advantage to be non-negligible.

□

Remark 3. *We note that while our construction is proved against malicious senders, for our application it suffices to have the following two properties:*

– *bounded rewind security against semi malicious senders.*

– *standalone security against receivers.*

Remark 4. *While not relevant to the bounded rewind security of the scheme, we note that in our applications, a malicious sender might compute the garbled circuit incorrectly. This stems from the fact that there will be multiple participants evaluating the garbled circuit to compute the OT output. We will therefore have to prove that the messages of the protocol were in fact computed correctly.*

4.3.3 Four Round Delayed Input Multiparty Computation with Bounded Rewind Security

Looking ahead, for our main result, we will compile an underlying semi-malicious protocol to achieve malicious security. In order to use the underlying semi-malicious protocol in a black-box manner, we will require the protocol to satisfy bounded rewind security. We start with an intuitive definition which we follow by formalizing the intuition.

To start with, we consider a four round delayed input semi-malicious protocols satisfying the following additional properties, where we denote by msg_k the messages of all parties output in the k-th round by Π.

1. **Property 1:** msg_1 and msg_2 of Π contain only messages of OT instances.

2. **Property 2:** msg_1 and msg_2 of Π do not depend on the input. The input is used only in the computation of msg_3 and msg_4.

3. **Property 3:** The simulator $\mathcal{S}$ simulates the honest parties' messages msg_1 and msg_2 via $\mathcal{S}_1$ and $\mathcal{S}_2$ by simply running the honest OT sender and receiver algorithms.

4. **Property 4:** msg_3 can be divided into two parts: (i) components independent of the OT messages; and (ii) OT messages.

Here we clarify what it means for a component of a message to be independent of OT messages. We say a component of msg_3 is independent of OT messages if its computation in the third round is independent of the both the private and public state of OT.

The recent works of [GS18b, BL18] construct two round semi-malicious protocols. Both protocols when instantiated with a four round OT protocol, satisfy the above structure. This follows from the fact that when their protocols are instantiated with a four round OT protocol, the non-OT components of their protocol are executed only in round 3.

The bounded rewind security notion follows in similar vein to the bounded rewind secure primitives we have previously defined. Note that the primary difference here stems from the fact that the protocol we consider is in the simultaneous message model. We say that a protocol satisfying the above properties is bounded rewind secure if the protocol remains secure in the presence of an that adversary is able to rewind the honest parties in the second and third round of the execution. Specifically, an adversary is allowed to: (a) initially query $B-1$ many distinct second round messages and receive third round messages in response; (b) in the last (B-th) query, the adversary also includes inputs for the honest parties. The adversary should then be unable to distinguish between the case that the protocol completes from the B-th query onward, where the last round was either completed with honest inputs provided by the adversary, or simulated.

We consider the bounded rewind security of protocols satisfying the structure defined above, where only the second and third rounds of the protocol can be rewound. For clarity of exposition, we will refer to protocols satisfying the properties to be *special four round delayed input semi-malicious MPC* protocols.

Definition 33 (Bounded rewind secure special four round delayed input semi-malicious MPC). *A special four round delayed input semi-malicious MPC protocol is said to be secure against* B *rewinds against a semi malicious adversary if the outputs of the experiments* E^0 *and* E^1 *are indistinguishable. The experiments are parameterized by the total number of parties* n *and the total number of corrupted parties* t. *We denote the set of honest parties as* $\mathcal{H}$, *and correspondingly the set of adversarial parties as* $\overline{\mathcal{H}}$. Trans_k *denotes the transcript of the first* k *rounds, and by extension* $\mathsf{Trans}_{k,\ell}$ *is the transcript of the first* k *rounds on rewind* ℓ. *The experiment* E^σ *with* $\sigma \in \{0,1\}$ *is defined as follows.*

1. *Compute* $\forall i \in \mathcal{H},\ \mathsf{msg}_{1,i} \coloneqq \Pi(\mathsf{r}_i)$ *and send to* $\mathcal{A}$.
2. *Receive* $\left\{\mathsf{msg}_{1,i}\right\}_{i\in\overline{\mathcal{H}}}$ *from* $\mathcal{A}$.
3. *Compute* $\forall i \in \mathcal{H},\ \mathsf{msg}_{2,i} \coloneqq \Pi(\mathsf{Trans}_1, \mathsf{r}_i)$ *and send to* $\mathcal{A}$.
4. *Receive* $\left\{\mathsf{msg}_{2,i,\ell}\right\}_{i\in\overline{\mathcal{H}},\ell\in[\mathsf{B}-1]}$ *from* $\mathcal{A}$
5. *Compute responses to the queries as follows.* $\forall \ell \in [\mathsf{B}]$, *compute third round messages as:* $\forall i \in \mathcal{H},\ \mathsf{msg}_{3,i,\ell} \leftarrow \Pi(0, \mathsf{Trans}_{2,\ell}, \mathsf{r}_i)$. *Send* $\left\{\mathsf{msg}_{3,i,\ell}\right\}_{i\in\mathcal{H},\ell\in[\mathsf{B}]}$ *to* $\mathcal{A}$.
6. *Receive* $\left(\{x_i\}_{i\in[n]}, \{\mathsf{r}_i\}_{i\in\overline{\mathcal{H}}}\right)$ *and* $\left\{\mathsf{msg}_{2,i}\right\}_{i\in\overline{\mathcal{H}}}$ *from the* $\mathcal{A}$.
7. *Based on the value of* σ, *the the messages are computed as follows:*
   - *if* $\sigma = 0$, *compute the third and fourth round messages of the last query using the inputs provided. Specifically, compute* $\forall i \in \mathcal{H},\ \mathsf{msg}_{3,i} \leftarrow \Pi(x_i, \mathsf{Trans}_2, \mathsf{r}_i)$, *and send* $\left\{\mathsf{msg}_{3,i}\right\}_{i\in\mathcal{H}}$ *to* $\mathcal{A}$. *On receiving,* $\left\{\mathsf{msg}_{3,i}\right\}_{i\in\overline{\mathcal{H}}}$, *compute* $\forall i \in \mathcal{H},\ \mathsf{msg}_{4,i} \leftarrow \Pi(x_i, \mathsf{Trans}_3, \mathsf{r}_i)$, *and send to* $\mathcal{A}$.
   - *if* $\sigma = 1$, *simulate the third and fourth round messages of the last query. Specifically, compute* $\left\{\mathsf{msg}_{3,i}\right\}_{i\in\mathcal{H}} \leftarrow \mathcal{S}_3\left(\mathsf{Trans}_2, \{\mathsf{r}_i\}_{i\in\mathcal{H}}\right)$, *and send* $\left\{\mathsf{msg}_{3,i}\right\}_{i\in\mathcal{H}}$ *to* $\mathcal{A}$. *On receiving,* $\left\{\mathsf{msg}_{3,i}\right\}_{i\in\overline{\mathcal{H}}}$, *compute* $\left\{\mathsf{msg}_{4,i}\right\}_{i\in\mathcal{H}} \leftarrow \mathcal{S}_4\left(\mathsf{Trans}_3, \{x_i\}_{i\in\overline{\mathcal{H}}}, \{\mathsf{r}_i\}_{i\in[n]}\right)$, *and send to* $\mathcal{A}$.
8. *The output of the experiment is the view of the adversary* $\mathcal{A}$.

Lemma 4. *The semi malicious protocols of [GS18b, BL18], when instantiated with our constructed 4 round OT with bounded rewind security, satisfies the above definition. The rewind security parameter of the resultant protocol is identical to that of the rewind secure parameter of the OT with bounded rewind security.*

We refer the reader to Remark 3 for the sufficient properties from the underlying oblivious transfer (OT) with bounded rewind security.

Proof sketch. We briefly describe why the resultant protocol is rewind secure. This primarily follows from the structure of the protocols and the bounded rewind security of the OT scheme.

To argue security, consider augmenting the protocol to allow additional threads that execute only the second and third round of the protocol multiple times. The adversary has control over what messages to send in each of the threads. On these threads, the honest inputs used are always going to be 0, with fresh randomness sampled for each thread. From the structure of the protocol, other than the OT, all components of the protocol are oblivious to rewinds in the second and third round. This follows from the fact that the components have messages no earlier than the third round.

Note that since fresh randomness is sampled to compute the third round of the protocol, this is akin to restarting the components (other than OT) with fresh randomness. Thus, when we have to rely on the bounded rewind security of OT, the other components of the third round can be computed without knowledge of the private state of the OT challenger.

4.4 Four Round MPC

Building Blocks. We list below all the building blocks of our protocol.

- **Trapdoor Generation Protocol:** $\mathsf{TDGen} = (\mathsf{TDGen}_1, \mathsf{TDGen}_2, \mathsf{TDGen}_3, \mathsf{TDOut}, \mathsf{TDValid}, \mathsf{TDExt})$ is a three round B_{td}-rewind secure trapdoor generation protocol based on one-way functions (see Section 4.2.3). We set B_{td} to be 2.
  In our MPC construction, we use a "multi-receiver" version of TDGen that works as follows: whenever a sender party i sends its first round message td_1, *all* of the other $(n-1)$ parties send a second round receiver message $\mathsf{td}_{2,i}$. The sender now prepares $\mathsf{td}_2 = (\mathsf{td}_{2,1}||\ldots||\mathsf{td}_{2,n-1})$, and then uses it to compute td_3. All the $(n-1)$ receivers individually verify the validity of td_3.
- **Delayed-Input WI Argument:** $\mathsf{WI} = (\mathsf{WI}_1, \mathsf{WI}_2, \mathsf{WI}_3, \mathsf{WI}_4)$ is a three round delayed-input witness indistinguishable proof system (see Section 4.2.4), where WI_4 is used to compute the decision of the verifier.
- **Bounded-Rewind Secure WI Argument:** $\mathsf{RWI} = (\mathsf{RWI}_1, \mathsf{RWI}_2, \mathsf{RWI}_3, \mathsf{RWI}_4)$ is a three round delayed-input witness-indistinguishable proof with B_{rwi_a}-rewind security (see Section 4.2.4). RWI_4 is used to compute the decision of the verifier. We will use two different instances of RWI that we will refer to as RWI_a and RWI_b, where the subscripts a and b denote the different instances. We set their respective rewind security parameters B_{rwi_a} and B_{rwi_b} to be some fixed polynomial.
- **Special Non-malleable Commitment:** $\mathsf{NMCom} = (\mathsf{NMCom}_1, \mathsf{NMCom}_2, \mathsf{NMCom}_3)$ is a three round special non-malleable commitment scheme (see Section 4.2.5). Let $\mathsf{Ext}_{\mathsf{NMCom}}$ denote the extractor associated with NMCom.
- **Bounded-Rewind Secure Extractable Commitment:** $\mathsf{RECom} = (\mathsf{RECom}_1, \mathsf{RECom}_2, \mathsf{RECom}_3)$ is the three round B_{recom}-rewind secure delayed-input extractable commitment based on non-interactive commitments (see Section 4.2.2). We set rewinding security parameter B_{recom} to be 4. $\mathsf{Ext}_{\mathsf{RECom}}$ is the extractor associated with RECom.
- **Extractable Commitment:** $\mathsf{Ecom} = (\mathsf{Ecom}_1, \mathsf{Ecom}_2, \mathsf{Ecom}_3, \mathsf{Ext}_{\mathsf{Ecom}})$ is the three round delayed-input extractable commitment scheme based on statistically binding commitment schemes (see Section 4.2.1). They satisfy the 2-extraction property.

- **Delayed Semi-Malicious MPC:** Π is a four round B_Π-bounded rewind secure delayed input MPC protocol based on oblivious transfer (see Section 4.3.3). We set B_Π to be 9.
- **Garbled Circuits:** $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$ is a secure garbling scheme (see Section 2.12). We denote the labels $\{\mathsf{lab}_{i,0}, \mathsf{lab}_{i,1}\}_{i\in[L]}$ by $\overline{\mathsf{lab}}$. We will often partition the labels of the garbled circuit to indicate the party providing the input corresponding to the label indices, and denote this by $\overline{\mathsf{lab}}_{|_j}$ for party j.
- **Oblivious Transfer:** $\mathsf{OT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3, \mathsf{OT}_4)$ is a four round oblivious transfer protocol. We abuse notation slightly and use this as implementing parallel OT executions where the receiver's input is a string of length ℓ and the sender now has ℓ pairs of inputs. We require regular indistinguishability security against a malicious sender. In addition, we require extraction of the receiver's input bit.

Levels of rewind security. We recall the notion of bounded-rewind security and the need for levels of rewind security. Bounded-rewind security, as in [BGJ+18], is used in the security proof to argue indistinguishability in intermediate hybrids. The main idea is that when arguing indistinguishability of two hybrids, to derive a contradiction it suffices to build an adversary with non-negligible success probability. As such, as long as the adversary does not abort with some non-negligible probability (which is indeed true), a small constant number of rewinds are sufficient for extracting with non-negligible probability. The exact bounded-rewind security constants for various primitives are carefully set to establish various "levels" of security.

For primitives with bounded rewind security, we require

$$B_{\mathsf{rwi}_a}, B_{\mathsf{rwi}_b}, B_\Pi > B_{\mathsf{recom}} > B_{\mathsf{td}}$$

where they denote the total number of rewinds (including the main thread) that they are secure against. In addition, we require all of them to be larger than the number of threads required to extract from NMCom and Ecom. For the above primitives, we have $B_{\mathsf{rwi}_a} = B_{\mathsf{rwi}_b} = \mathsf{poly}(\lambda)$ (for some fixed polynomial), $B_\Pi = 9$, $B_{\mathsf{recom}} = 4$ and $B_{\mathsf{td}} = 2$ thus satisfying our requirements.

Notation for Transcripts. We introduce a common notation that we shall use to denote *partial* transcripts of an execution of different protocols that we use in our MPC construction. For any execution of protocol X, we use $\mathbf{T}_X[\ell]$ to denote the transcript of the first ℓ rounds.

NP languages. We define the NP languages used for the three different proof systems that we use in our protocol. We denote statements and witnesses as st and w, respectively.

1. RWI_a: We use RWI_a for language $\mathcal{L}_a$, which is characterized by the following relation R_a:

$\mathsf{st} := \left(\mathbf{T}_\Pi[2], \left\{\mathbf{T}^j_{\mathsf{recom}}[3]\right\}_{j\in[n]}, \{\mathsf{msg}_\ell\}_{\ell\in[3]}, \mathbf{T}_{\mathsf{ncom}}[3], \mathsf{td}_1\right)$

$\mathsf{w} := \left(\mathsf{n}, \mathsf{r}, \left\{\mathsf{r}^j_{\mathsf{recom}}\right\}_{j\in[n]}, \mathsf{t}, \mathsf{r}_{\mathsf{ncom}}\right)$

$R_a(\mathsf{st}, \mathsf{w}) = 1$ if *either* of the following conditions is satisfied:

(a) **Honest:** *all* of the following conditions hold:

- $\forall j$, $\mathbf{T}^j_{\mathsf{recom}}[3]$ is a well-formed transcript of RECom w.r.t. input (n, r) and randomness $\mathsf{r}^j_{\mathsf{recom}}$.

– for every $\ell \leq 3$, msg_ℓ is an honestly computed ℓ^{th} round message in the protocol Π w.r.t. input n, randomness r and the first $(\ell-1)$ round protocol transcript $\mathbf{T}_\Pi[\ell-1]$.

(b) **Trapdoor:** $\mathbf{T}_{\mathsf{ncom}}[3]$ is an honest transcript of NMCom w.r.t. input t and randomness $\mathsf{r}_{\mathsf{ncom}}$ (AND) t is a valid trapdoor w.r.t. td_1

2. RWI_b: We use RWI_b for language $\mathcal{L}_b$, which is characterized by the following relation R_b:

$$\mathsf{st} := \left(\left\{\mathbf{T}^j_{\mathsf{rwi}_a}[2], \mathsf{st}^j_a, \mathbf{T}^j_{\mathsf{ecom}}[3]\right\}_{j\in[n]}, \mathbf{T}_{\mathsf{ncom}}[3], \mathsf{td}_1\right)$$
$$\mathsf{w} := \left(\left\{\mathsf{r}^j_{\mathsf{rwi}_a}, \mathsf{w}^j_a, \mathsf{rwi}^j_{3,a}, \mathsf{r}^j_{\mathsf{ecom}}\right\}_{j\in[n]}, \mathsf{t}, \mathsf{r}_{\mathsf{ncom}}\right).$$

$R_b(\mathsf{st},\mathsf{w}) = 1$ if *either* of the following conditions is satisfied:

(a) **Honest:** *all* of the following conditions hold:

– $\forall j$, $\mathbf{T}^j_{\mathsf{ecom}}[3]$ is a well-formed transcript of Ecom w.r.t. input $\left\{\mathsf{rwi}^k_{3,a}\right\}_{k\in[n]}$ and randomness $\mathsf{r}^j_{\mathsf{ecom}}$.

– $\forall j$, $\mathbf{T}^j_{\mathsf{rwi}_a}[2]||\mathsf{rwi}^j_{3,a}$ is an honestly computed transcript of RWI_a for $\mathcal{L}_a$ with statement st^j_a, witness w^j_a and randomness $\mathsf{r}^j_{\mathsf{rwi}_a}$. [a]

(b) **Trapdoor:** $\mathbf{T}_{\mathsf{ncom}}[3]$ is an honest transcript of NMCom w.r.t. input t and randomness $\mathsf{r}_{\mathsf{ncom}}$ (AND) t is a valid trapdoor w.r.t. td_1

[a]Since RWI is not publicly verifiable, the relation establishes that the RWI prover messages were computed honestly w.r.t. the witness and randomness for the statement.

3. WI: We use WI for language $\mathcal{L}_c$, which is characterized by the following relation R_c:

$$\mathsf{st} := \left(\mathbf{T}_\Pi[3], \left\{\mathbf{T}^j_{\mathsf{recom}}[3], \mathbf{T}^j_{\mathsf{rwi}}[2], \mathsf{st}^j_b, \mathbf{T}^j_{\mathsf{ot}}[4]\right\}_{j\in[n]}, \overline{\mathsf{C}}, \mathbf{T}_{\mathsf{ncom}}[3], \mathsf{td}_1\right)$$
$$\mathsf{w} := \left(\mathsf{n}, \mathsf{r}, \left\{\mathsf{r}^j_{\mathsf{recom}}, \mathsf{r}^j_{\mathsf{ot}}, \mathsf{r}^j_{\mathsf{rwi}}\right\}_{j\in[n]}, \mathsf{msg}_4, \mathsf{r}_{\mathsf{gc}}, \mathsf{t}, \mathsf{r}_{\mathsf{ncom}}\right)$$

$R_c(\mathsf{st},\mathsf{w}) = 1$ if *either* of the following conditions is satisfied:

(a) **Honest:** For every j, *all* of the following conditions hold:

– $\mathbf{T}^j_{\mathsf{recom}}[3]$ is a well-formed transcript of RECom w.r.t. input (n,r) and randomness $\mathsf{r}^j_{\mathsf{recom}}$.

– msg_4 is honestly computed round 4 message of Π w.r.t. n, randomness r and transcript $\mathbf{T}_\Pi[3]$.

– $(\overline{\mathsf{C}}, \overline{\mathsf{lab}})$ is honest garbling of C that contains hardwired values $\mathsf{msg}_4, \left\{\mathbf{T}^j_{\mathsf{rwi}}[2], \mathsf{st}^j_b, \mathsf{r}^j_{\mathsf{rwi}}\right\}_{j\in[n]}$, using randomness r_{gc}. (See Figure 4.5.)

- ot_4^j is honestly computed using $\overline{\mathsf{lab}}_{|j}$, randomness $\mathsf{r}_{\mathsf{ot}}^j$ and transcript $\mathbf{T}_{\mathsf{ot}}^j[3]$. $(\mathbf{T}_{\mathsf{ot}}^j[4] = \mathbf{T}_{\mathsf{ot}}^j[3] \| \mathsf{ot}_4^j)$.

(b) **Trapdoor:** $\mathbf{T}_{\mathsf{ncom}}[3]$ is an honest transcript of NMCom w.r.t. input t and randomness $\mathsf{r}_{\mathsf{ncom}}$ (AND) t is a valid trapdoor w.r.t. td_1

$$\mathsf{C}\left[\mathsf{msg}_4, \left\{\mathbf{T}_{\mathsf{rwi}_b}^j[2], \mathsf{st}_b^j, \mathsf{r}_{\mathsf{rwi}_b}^j\right\}_{j\in[n]}\right]$$

Input: $\{\mathsf{rwi}_{3,b}^j\}_{j\in[n]}$

- If for every $j \neq i$, $\mathsf{RWI}_4\left(\mathsf{st}_b^j, \mathbf{T}_{\mathsf{rwi}_b}^j[2] \| \mathsf{rwi}_{3,b}^j; \mathsf{r}_{\mathsf{rwi}_b}^j\right) = 1$, **output** msg_4;
- Else, **output** $\perp$.

Figure 4.5: Circuit C

4.4.1 The Protocol

In this section, we describe our four round MPC protocol between n players $P_1, \cdots, P_n$. Let x_i denote the input of party P_i. At the start of the protocol, each party samples a sufficiently long random tape to use in the various sub-protocols; let r_X denote the randomness used in sub-protocol X.

Notational Conventions. We establish some conventions for simplifying notation in the protocol description. We only indicate randomness as an explicit input for computing the first round message of a sub-protocol; for subsequent computations, we assume it to be an implicit input. Similarly, we assume that any next-message of a sub-protocol takes as input a partial transcript of the "previous" rounds, and do not write it explicitly. Whenever necessary, we augment our notation with superscript $i \to j$ to indicate the a instance of an execution of a sub-protocol between a "sender" i and "receiver" j (where sometimes, the sender is a prover and receiver is a verifier). When the specific instance is clear from context, we shall drop the superscript. When we wish to refer to multiple instances involving a party i, we will use the shorthand superscript $i \to \bullet$ or $\bullet \to i$, depending upon whether i is the sender or the receiver. For example, $\mathbf{T}_X^{i\to\bullet}[\ell]$ will be a shorthand to indicate $\left\{\mathbf{T}_X^{i\to j}[\ell]\right\}_{j\in[n]}$.

We will sometimes use explanatory comments within the protocol description, denoted as //-comment. Finally, we note that all messages in the protocol are broadcast; if any party aborts during the first three rounds of the protocol, it broadcasts an abort in the subsequent round. We do not write this explicitly in the protocol, and assume it to be implicit. We now proceed to describe the protocol.

Protocol: Four Round MPC

Round 1: P_i computes and broadcasts the *first* round messages of the following protocols:

1. Delayed semi-malicious MPC Π: $\mathsf{msg}_{1,i} \leftarrow \Pi_1(\mathsf{r}_i)$.
2. Sender message of TDGen: $\mathsf{td}_{1,i} \leftarrow \mathsf{TDGen}_1(\mathsf{r}_{\mathsf{td},i})$.

For every $j \neq i$:

3. Prover message of the three delayed-input WI argument systems
   - WI: $\mathsf{wi}_1^{i\to j} \leftarrow \mathsf{WI}_1(\mathsf{r}_{\mathsf{wi}}^{i\to j})$.
   - RWI_a: $\mathsf{rwi}_{a,1}^{i\to j} \leftarrow \mathsf{RWI}_1(\mathsf{r}_{\mathsf{rwi}_a}^{i\to j})$.
   - RWI_b: $\mathsf{rwi}_{b,1}^{i\to j} \leftarrow \mathsf{RWI}_1(\mathsf{r}_{\mathsf{rwi}_b}^{i\to j})$.
4. Sender message of the three delayed-input commitment schemes
   - Ecom: $\mathsf{ecom}_1^{i\to j} \leftarrow \mathsf{Ecom}_1(\mathsf{r}_{\mathsf{ecom}}^{i\to j})$.
   - RECom: $\mathsf{recom}_1^{i\to j} \leftarrow \mathsf{RECom}_1(\mathsf{r}_{\mathsf{recom}}^{i\to j})$.
   - NMCom: $\mathsf{ncom}_1^{i\to j} \leftarrow \mathsf{NMCom}_1(\mathsf{r}_{\mathsf{ncom}}^{i\to j})$.
5. Receiver message of OT: $\mathsf{ot}_1^{j\to i} \leftarrow \mathsf{OT}_1\left(\mathsf{r}_{\mathsf{ot}}^{j\to i}\right)$.

Round 2: P_i computes and broadcasts the *second* round messages of the following protocols:

1. Delayed semi-malicious MPC Π: $\mathsf{msg}_{2,i} \leftarrow \Pi_2$.

For every $j \neq i$:

2. Receiver message of TDGen: $\mathsf{td}_2^{i\to j} \leftarrow \mathsf{TDGen}_2$.
3. Verifier message of the three delayed-input WI argument systems
   - WI: $\mathsf{wi}_2^{j\to i} \leftarrow \mathsf{WI}_2$
   - RWI_a: $\mathsf{rwi}_{a,2}^{j\to i} \leftarrow \mathsf{RWI}_2$
   - RWI_b: $\mathsf{rwi}_{b,2}^{j\to i} \leftarrow \mathsf{RWI}_2$
4. Receiver message of the three delayed-input commitment schemes
   - Ecom: $\mathsf{ecom}_2^{j\to i} \leftarrow \mathsf{Ecom}_2$.
   - RECom: $\mathsf{recom}_2^{j\to i} \leftarrow \mathsf{RECom}_2$.
   - NMCom: $\mathsf{ncom}_2^{j\to i} \leftarrow \mathsf{NMCom}_2$.
5. Sender message of OT: $\mathsf{ot}_2^{i\to j} \leftarrow \mathsf{OT}_2$.

Round 3: P_i computes and broadcasts the *third* round messages of the following protocols:

1. Delayed semi-malicious Π: $\mathsf{msg}_{3,i} \leftarrow \Pi_3(\mathsf{x}_i)$ using input x_i. //First step where P_i is using its input.
2. TDGen: $\mathsf{td}_{3,i} \leftarrow \mathsf{TDGen}_3$.

For every $j \neq i$:

3. NMCom: $\mathsf{ncom}_3^{i\to j} \leftarrow \mathsf{NMCom}_3(\widetilde{\mathsf{r}}_j)$ to commit to a random $\widetilde{\mathsf{r}}_j$.
4. RECom: $\mathsf{recom}_3^{i\to j} \leftarrow \mathsf{RECom}_3(\mathsf{x}_i, \mathsf{r}_i)$ to commit to $(\mathsf{x}_i, \mathsf{r}_i)$.

5. RWI: $\mathsf{rwi}_{a,3}^{i\to j} \leftarrow \mathsf{RWI}_3\left(\mathsf{st}_a^{i\to j}, \mathsf{w}_a^{i\to j}\right)$ to prove that $R_a(\mathsf{st}_a^{i\to j}, \mathsf{w}_a^{i\to j}) = 1$, where
   Statement $\mathsf{st}_a^{i\to j} := (\mathbf{T}_\Pi[2], \mathbf{T}_{\mathsf{recom}}^{i\to\bullet}[3], \{\mathsf{msg}_{\ell,i}\}_{\ell\in[3]}, \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$
   "Honest" witness $\mathsf{w}_a^{i\to j} := (\mathsf{x}_i, \mathsf{r}_i, \mathsf{r}_{\mathsf{recom}}^{i\to\bullet})$
6. Ecom: $\mathsf{ecom}_3^{i\to j} \leftarrow \mathsf{Ecom}_3\left(\mathsf{rwi}_{a,3}^{i\to\bullet}\right)$ to commit to $\mathsf{rwi}_{a,3}^{i\to\bullet}$.
7. RWI_b: $\mathsf{rwi}_{b,3}^{i\to j} \leftarrow \mathsf{RWI}_3(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j})$ to prove that $R_b(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j}) = 1$, where
   Statement $\mathsf{st}_b^{i\to j} := (\mathbf{T}_{\mathsf{rwi}_a}^{i\to\bullet}[2], \mathsf{st}_a^{i\to\bullet}, \mathbf{T}_{\mathsf{ecom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$
   "Honest" witness $\mathsf{w}_b^{i\to j} := (\mathsf{r}_{\mathsf{rwi}_a}^{i\to\bullet}, \mathsf{w}_a^{i\to\bullet}, \mathsf{rwi}_{a,3}^{i\to\bullet}, \mathsf{r}_{\mathsf{ecom}}^{i\to\bullet})$
8. OT: Receiver message $\mathsf{ot}_3^{j\to i} \leftarrow \mathsf{OT}_3(\mathsf{rwi}_{b,3}^{i\to j})$ using input $\mathsf{rwi}_{b,3}^{i\to j}$.

Round 4: P_i computes and broadcasts the following messages:

1. If $\exists j \neq i$ such that $\mathsf{TDValid}(\mathsf{td}_{1,j}, \mathsf{td}_{2,j}, \mathsf{td}_{3,j}) \neq 1$, **abort**.
   //where $\mathsf{td}_{2,j} := (\mathsf{td}_2^{1\to j}||\cdots||\mathsf{td}_2^{n\to j})$.
2. Delayed semi-malicious MPC Π: Fourth round message $\mathsf{msg}_{4,i} \leftarrow \Pi_4$.
3. Garbled Circuit: $\overline{\mathsf{C}}_i$, where $(\overline{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(\mathsf{C}[\mathsf{msg}_{4,i}, \mathbf{T}_{\mathsf{rwi}_b}^{\bullet\to i}[2], \mathsf{st}_b^{\bullet\to i}, \mathsf{r}_{\mathsf{rwi}_b}^{\bullet\to i}]; \mathsf{r}_{\mathsf{gc},i})$.
   Circuit C is defined in Figure 4.5.

For every $j \neq i$:

4. OT: Fourth round sender message $\mathsf{ot}_4^{i\to j} \leftarrow \mathsf{OT}_4\left(\overline{\mathsf{lab}}_{i|_j}\right)$ using input $\overline{\mathsf{lab}}_{i|_j}$
   //$\overline{\mathsf{lab}}_{i|_j}$ denotes labels corresponding to the input wires for P_j's input.
5. OT: Receiver randomness $\mathsf{r}_{\mathsf{ot}}^{j\to i}$. //This is used by other parties to compute OT output.
6. WI: $\mathsf{wi}_3^{i\to j} \leftarrow \mathsf{WI}_3\left(\mathsf{st}_c^{i\to j}, \mathsf{w}_c^{i\to j}\right)$, to prove that $R_c(\mathsf{st}_c^{i\to j}, \mathsf{w}_c^{i\to j}) = 1$, where
   Statement $\mathsf{st}_c^{i\to j} := (\mathbf{T}_\Pi[3], \mathbf{T}_{\mathsf{recom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{rwi}_b}^{\bullet\to i}[2], \mathsf{st}_b^{\bullet\to i}, \mathbf{T}_{\mathsf{ot}}^{i\to\bullet}[4], \overline{\mathsf{C}}_i, \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$
   "Honest" witness $\mathsf{w}_c^{i\to j} := (\mathsf{x}_i, \mathsf{r}_i, \mathsf{r}_{\mathsf{recom}}^{i\to\bullet}, \mathsf{r}_{\mathsf{ot}}^{i\to\bullet}, \mathsf{r}_{\mathsf{rwi}_b}^{\bullet\to i}, \mathsf{msg}_{4,i}, \mathsf{r}_{\mathsf{gc},i})$

Output Computation: P_i computes the following:

1. If $\exists j \neq i$, s.t. $\mathsf{WI}_4(\mathsf{st}_c^{j\to i}, \mathbf{T}_{\mathsf{wi}}^{j\to i}[3]) \neq 1$, output $\perp$ and **abort**.
2. Compute OT outputs: $\forall j \neq i, \forall k \neq \{i,j\}$,
   $\widehat{\mathsf{lab}}_{j|_k} \leftarrow \mathsf{OTEval}(\mathbf{T}_{\mathsf{ot}}^{j\to k}[4]; \mathsf{r}_{\mathsf{ot}}^{j\to k})$
3. Evaluate garbled circuits: $\forall j \neq i$, $\widehat{\mathsf{msg}}_{4,j} \leftarrow \mathsf{Eval}(\overline{\mathsf{C}}_j, \widehat{\mathsf{lab}}_j)$, where $\widehat{\mathsf{lab}}_j := (\widehat{\mathsf{lab}}_{j|_1}||\cdots||\widehat{\mathsf{lab}}_{j|_n})$.
   If any evaluation returns $\perp$, then output $\perp$ and **abort**.
4. Output $y_i \leftarrow \mathsf{OUT}(\mathsf{x}_i, \mathbf{T}_\Pi[4]; \mathsf{r}_i)$, where $\mathbf{T}_\Pi[4]$ includes $\mathbf{T}_\Pi[3]$ and $\widehat{\mathsf{msg}}_{4,j}$ for every j.

Our main result is stated in the following theorem.

Theorem 15. *Assuming the hiding property of oblivious transfer, the hiding property of extractable commitment, the hiding property of extractable commitment with bounded rewind security, delayed* *semi malicious protocol with bounded rewind security computing any function $\mathcal{F}$, special non-malleable commitments, witness indistinguishable proofs with bounded rewind security, security of garbled circuits, trapdoor generation protocol with bounded rewind security, in addition to the correctness of these primitives, then the presented protocol is a four round protocol for $\mathcal{F}$ secure against a malicious dishonest majority.*

Remark 5. *All the above primitives can be based on one-way functions, non-interactive commitments and oblivious trasnfer (OT). In a recent note by Lombardi and Schaeffer [LS19], they give a construction of a perfectly binding non-interactive commitment based on perfectly correct key agreement. As they point out, such key agreement schemes can be based on perfectly correct oblivious transfer [GKM+00]. This gives us both a non-interactive commitment schemes, and one-way functions, based on perfectly correct oblivious transfer. Thus it suffices to instantiate all our primitives using just oblivious transfer.*

We thus have the following corollary.

Corollary 6. *Assuming polynomially secure oblivious transfer with perfect correctness, our constructed protocol is a four round multiparty computation protocol for any function $\mathcal{F}$.*

The complete security analysis of the above protocol is presented in the Section 4.5. Below we first present a high level description of the main ideas of the proof and how the bounded rewind-security parameters are set.

4.4.1.1 Overview of Security Proof

The discussion below is informal, and not a complete picture of the simulator and hybrids. Our intent is to give an outline of the key hybrids and simulation steps to convey the main ideas. This will already highlight the need for various levels of rewind security, one of the main challenges in proving security. There are lots of other challenges that we do not discuss here, and similar to prior works, the full security analysis is much more complex and we refer the reader to Section 4.5 for the analysis.

One particular challenge that we ignore is that of an aborting adversary, either implicitly or explicitly, in the first three rounds of the protocol. The case of an explicitly aborting adversary is dealt with in a similar manner to [BGJ+18, GK96a] by initially sampling a partial transcript, using dummy inputs, to determine if the adversary aborts, and then re-sampling the transcript in case the adversary does not abort. For an implicitly aborting adversary, the simulator (via extraction) can determine if the adversary aborted, but honest parties are not aware of this in the first three rounds of the protocol. This case relies on the security of the multi-party CDS (via OT and garbled circuits) to deal with the implicit aborts. Stepping around these challenges, the main steps in the simulation involve (a) rewinding the adversary to extract the trapdoor and inputs; (b) completing the witness indistinguishable arguments using the extracted trapdoor; (c) simulating the underlying protocol using the output obtained from the ideal functionality.

Key Hybrid Components. We give below a high level overview of some key hybrids in keeping with our simplified description of the simulator above. This will allow us to discuss our specific choices for the level of rewinds.

– The first hybrid is identical to the real protocol execution. Each witness indistinguishable (WI) argument in our protocol allows for a *trapdoor witness*, arising from the trapdoor generation protocol and the non-malleable commitment (NMCom). We would like it to be the case that a simulator is able to derive the trapdoor and produce a simulated transcript via the *trapdoor witness*, an adversary should not be in possession of a *trapdoor witness* thereby forcing honest behavior if the witness indistinguishable argument is accepting.

In order to argue that the adversary is not in possession of the *trapdoor witness*, we need to ensure the following *invariant*: the adversary does not commit to the trapdoor inside of the NMCom.

In order to do so in this hybrid, we rely on the rewind security of the trapdoor generation protocol. Specifically, we extract from the NMCom by rewinding the adversary once in the second and third round (two total executions of the second and third round). If indeed the adversary was committing to the trapdoor, the extraction is successful with some noticeable probability and thereby breaking the rewind security of the trapdoor generation protocol. Note, as observed in [BGJ+18], to arrive at a contradiction via reduction it is sufficient to extract with noticeable (as opposed to overwhelming) probability. This explains why we require $B_{\mathsf{td}} \geq 2$.

For each change that we subsequently make through the various primitives, we will bootstrap the above technique, and argue that this invariant continues to hold. Specifically, in order to arrive at a contradiction, we will extract from the NMCom to break the security property of the corresponding primitive if the *invariant* ceases to hold. This already gives us a flavor for primitives to be secure against (at least) two rewinds needed for the extraction from the NMCom.

– In this hybrid, the simulator creates sufficient rewind execution threads in order to extract the adversary's input and the trapdoors needed to prove the WI using the *trapdoor witness*. These rewind threads have the same first round messages as the "main" execution thread, but the second and third round messages are computed in each rewind thread with fresh randomness. The rewind threads terminate on completion of the third round of the protocol.

– In the previous hybrid, the simulator is still using the honest inputs in the rewind threads. In this hybrid the rewind threads are switched from using the honest party's inputs, to an honest execution with input 0. Note that these threads finish by the end of the third round.

  While the changes made in this hybrid are done in a sequence of steps, and needs to be argued carefully, the sequence closely resembles the changes that will be made in the main execution thread below. Therefore, we primarily focus on the hybrids pertaining to the main execution thread.

– In this hybrid, the simulator uses the trapdoors extracted from the rewind threads to commit to the trapdoor inside the NMCom on the main execution thread. In order to argue indistinguishability, we perform a reduction to an external NMCom challenger. In order to generate the transcript internally, and complete the reduction, we need to rewind the adversary to get the trapdoor and inputs. But this causes a problem since the rewind threads might require responses to challenges that are meant for the external challenger. Here, we rely on the fact that the third round of our instantiated NMCom has pseudorandom messages, allowing us to respond to adversarial queries in the third round, that cannot be forwarded to the external NMCom challenger. This prevents the need for bounded rewind security from the NMCom.

– In a sequence of sub-hybrids, the simulator uses the extracted trapdoor to complete both the bounded rewind secure witness indistinguishable arguments using the *trapdoor witness*.

As seen above, for the reduction we will need to rewind the adversary to extract, thereby rewinding the external challenger. Since we require extraction of the adversary's inputs, the parameter for the bounded rewind secure witness indistinguishable argument needs to satisfy $B_{\mathsf{rwi}} > B_{\mathsf{recom}}$.

– In this hybrid, the simulator uses the extracted trapdoor to complete the witness indistinguishable argument. Since the third round of this protocol is completed in the fourth round of our compiled protocol, rewinding the adversary to extract the trapdoor and input in the second and third round circumvents issues discussed above. Therefore, we don't require this primitive to be rewind secure.

– In this hybrid, the simulator switches to committing to 0 inside the rewind secure extractable commitment (RECom). Unlike the previous cases, this is potentially circularity since the arguments above do not directly extend. This is because it cannot be the case that the external challenger remains secure if we rewind the adversary B_{recom} times to extract its input.

  Instead, this is argued carefully where initially we argue that switching to a commitment of a "junk" value in the third round of the RECom doesn't affect our ability to extract from the adversary. This "junk" commitment can be made without knowledge of any randomness of the specific RECom instance. To argue this, we rely on the bounded rewind security of the extractable commitment, while still extracting the trapdoor to complete the transcript. This gives us the requirement that $B_{\mathsf{recom}} > B_{\mathsf{td}}$. This then allows for extraction of input in the reduction without violating rewinding circularity since, on the look ahead threads to extract, we can commit to junk without affecting input extraction.

– In this hybrid, the simulator simulates the transcript of the underlying bounded rewind secure protocol Π. Here too, we require extracting the inputs in order to send it to the ideal functionality. Therefore, we require $B_{\Pi} > B_{\mathsf{recom}}$.

4.5 Full Security Proof

We now present the complete security analysis of our constructed protocol. Consider a malicious non-uniform PPT adversary $\mathcal{A}$ who corrupts $t < n$ parties.

4.5.1 Overview of the Simulation

Before providing a formal description of the simulator, we provide a high level overview of the various steps in our simulation strategy:

Step 1: Check Adversary Abort The first thing our simulator does is to determine if the adversary aborts in the first three rounds of the protocol. If so, the adversary can simulate the first three rounds using input 0. But there is a small subtlety here. By the end of the third round, none of the proofs are sent in the clear. It is possible that the adversary is implicitly aborting by sending incorrect messages, and hence the proofs will fail, but the honest parties are unaware of this.

Since they both constitute as aborts, we want to treat them identically. But in the latter case, we're still required to send the fourth round messages of the honest parties, since as mentioned earlier, they aren't aware of an implicit abort until the fourth round.

- if the adversary aborts in a manner that is identifiable by the honest parties, i.e. by not sending the protocol message or an identifiable incorrect message (such as failed trapdoor validity), the simulator just outputs an aborted transcript.
- if the adversary aborts implicitly, then we set all the garbled circuits to output $\perp$.

This step is performed so as to ensure that if an adversary aborts with a disproportionately high probability, we don't have to bother attempting to simulate all the other components in the protocol. In the case where there the adversary explicitly aborts, the simulation ends here.

Step 2: Rewinding If the adversary has not aborted, we need to produce a non aborting transcript. To enable us to do so, we need to first extract relevant information. This is done by creating multiple "look-ahead" threads that share a common first round prefix with the main thread. On the look ahead threads, we're using input 0, as in the previous step, to compute the first three rounds honestly (with respect to input 0).

These threads also help us estimate the probability that an adversary does not abort. With sufficiently many look-ahead threads, we can extract all the relevant information.

Step 3: Input and Trapdoor Extraction With sufficiently many look-ahead threads from the rewinding step above, we can extract all the relevant information.

Step 4: Abort Probability Estimation Depending on the number of threads created to in the rewinding step to have sufficiently many threads to extract, we can estimate the probability of abort.

Step 5: Re-sampling Main Thread Now that we have extracted the trapdoor information and input, we need to sample the "main thread", which corresponds to the actual view of the adversary. Note we are at this point because the adversary did not abort, and thus to avoid skewing the distribution of aborting transcripts, we must force a non-aborting transcript on to the adversary. We use the earlier estimate of the non-aborting probability of the adversary to repeatedly try to force the transcript. By a careful analysis, this step will succeed other than with negligible probability.

Step 6: Query the Ideal Functionality Given that we have managed to force a non-aborting transcript, corresponding to the first three rounds, on the adversary, we need to simulate the last round of the protocol. This is done by first querying the ideal functionality using the extracted inputs.

Step 7: Extract Proofs from OT It is still possible that the adversary has put in non-accepting proofs as the OT receiver input even though it did not implicitly abort. We want to rely on the "opaqueness" of the garbled circuits in such a situation. To do so, we must extract from the oblivious transfer to determine which circuits to set to output $\perp$.

Step 8: Finishing the Main Thread Given the output received from the ideal functionality, and knowledge of whether the adversary implicitly aborted, or sent incorrect proofs in the OT, we simulate the last round of the protocol and appropriately compute the garbled circuits.

While the above suffices for a high level overview of our strategy, the proof is quite delicate involving the security levels of the various primitives.

4.5.2 Simulator Sim

We provide a full description of the simulator Sim below. We note that Sim also performs simple checks akin to the protocol description in order to send an abort message if it receives one. For simplicity, we have not explicitly stated these checks in the below description. Also, as in our protocol description, we shall assume that the protocols have partial state and we do not specify the state as input when we make a protocol call.

Step 1 - Check Adversary Abort: In this step, Sim checks if the adversary aborts prior to the completion of the third round. This can be via either an implicit or explicit abort.

1. **Round 1:**

   Compute the first round message of all honest parties of the underlying protocol Π,

   – $\{\mathsf{msg}_{1,i}\}_{P_i \in \mathcal{H}} \leftarrow \mathcal{S}_1\left(1^\lambda; \mathsf{r}_{\mathcal{S}}\right)$

   where $\mathcal{H}$ denotes the set of honest parties. Recall that this is done as just the honest execution of the first round on behalf of the honest players P_i using randomness $\mathsf{r}_{\mathcal{S}} := \{\mathsf{r}_i\}_{P_i \in \mathcal{H}}$. This is sent to $\mathcal{A}$ along with the messages computed below.

   For each honest party P_i, Sim follows the honest party protocol in the first round of the following protocols and sends the messages to $\mathcal{A}$:

   (a) Sender message of TDGen: $\mathsf{td}_{1,i} \leftarrow \mathsf{TDGen}_1\left(\mathsf{r}_{\mathsf{td},i}\right)$.

   For every $j \neq i$:

   (b) Prover message of the three delayed-input WI argument systems
   - WI: $\mathsf{wi}_1^{i \to j} \leftarrow \mathsf{WI}_1(\mathsf{r}_{\mathsf{wi}}^{i \to j})$.
   - RWI_a: $\mathsf{rwi}_{a,1}^{i \to j} \leftarrow \mathsf{RWI}_1(\mathsf{r}_{\mathsf{rwi}_a}^{i \to j})$.
   - RWI_b: $\mathsf{rwi}_{b,1}^{i \to j} \leftarrow \mathsf{RWI}_1(\mathsf{r}_{\mathsf{rwi}_b}^{i \to j})$.

   (c) Sender message of the three delayed-input commitment schemes
   - Ecom: $\mathsf{ecom}_1^{i \to j} \leftarrow \mathsf{Ecom}_1(\mathsf{r}_{\mathsf{ecom}}^{i \to j})$.
   - RECom: $\mathsf{recom}_1^{i \to j} \leftarrow \mathsf{RECom}_1(\mathsf{r}_{\mathsf{recom}}^{i \to j})$.
   - NMCom: $\mathsf{ncom}_1^{i \to j} \leftarrow \mathsf{NMCom}_1(\mathsf{r}_{\mathsf{ncom}}^{i \to j})$.

   (d) Receiver message of OT: $\mathsf{ot}_1^{j \to i} \leftarrow \mathsf{OT}_1\left(\mathsf{r}_{\mathsf{ot}}^{j \to i}\right)$.

2. **Round 2:**

   For the second round, Sim follows the honest strategy since the inputs of the honest parties are not required up until the third round of the protocol. Compute the second round message of all honest parties of the delayed semi-malicious Π:

   – $\{\mathsf{msg}_{2,i}\}_{P_i \in \mathcal{H}} \leftarrow \mathcal{S}_2$

using the transcript obtained so far. Recall that this is done as just the honest execution of the second round on behalf of the honest players P_i using the randomness sampled as a part of the first round.

For each honest party P_i, Sim follows the honest party protocol in the second round of the following protocols and sends the messages to $\mathcal{A}$:

For every $j \neq i$:

(e) Receiver message of TDGen: $\mathsf{td}_2^{i \to j} \leftarrow \mathsf{TDGen}_2$.

(f) Verifier message of the three delayed-input WI argument systems
- WI: $\mathsf{wi}_2^{j \to i} \leftarrow \mathsf{WI}_2$
- RWI_a: $\mathsf{rwi}_{a,2}^{j \to i} \leftarrow \mathsf{RWI}_2$
- RWI_b: $\mathsf{rwi}_{b,2}^{j \to i} \leftarrow \mathsf{RWI}_2$

(g) Receiver message of the three delayed-input commitment schemes
- Ecom: $\mathsf{ecom}_2^{j \to i} \leftarrow \mathsf{Ecom}_2$.
- RECom: $\mathsf{recom}_2^{j \to i} \leftarrow \mathsf{RECom}_2$.
- NMCom: $\mathsf{ncom}_2^{j \to i} \leftarrow \mathsf{NMCom}_2$.

(h) Sender message of OT: $\mathsf{ot}_2^{i \to j} \leftarrow \mathsf{OT}_2$.

3. **Round 3:**

For the third round, Sim will set 0 to be the input each honest party.

For each honest party P_i, Sim follows the honest party protocol in the third round of the following protocols, *using input 0*, and sends the messages to $\mathcal{A}$:

(a) Compute the third round message of the delayed semi-malicious Π:
- $\mathsf{msg}_{3,i} \leftarrow \Pi_3(0,)$

using input 0, randomness r_i and the transcript obtained so far. Recall that we are able to do this since the "simulation" of the first two rounds of the underlying protocol were honest computations.

(b) TDGen: $\mathsf{td}_{3,i} \leftarrow \mathsf{TDGen}_3$.

For every $j \neq i$:

(c) NMCom: $\mathsf{ncom}_3^{i \to j} \leftarrow \mathsf{NMCom}_3(\widetilde{\mathsf{r}}_j)$ to commit to a random $\widetilde{\mathsf{r}}_j$.

(d) RECom: $\mathsf{recom}_3^{i \to j} \leftarrow \mathsf{RECom}_3(0, \mathsf{r}_i)$ to commit to $(0, \mathsf{r}_i)$.

(e) RWI: $\mathsf{rwi}_{a,3}^{i \to j} \leftarrow \mathsf{RWI}_3\left(\mathsf{st}_a^{i \to j}, \mathsf{w}_a^{i \to j}\right)$ to prove that $R_a(\mathsf{st}_a^{i \to j}, \mathsf{w}_a^{i \to j}) = 1$, where

$$\text{Statement } \mathsf{st}_a^{i \to j} := (\mathbf{T}_\Pi[2], \mathbf{T}_{\mathsf{recom}}^{i \to \bullet}[3], \{\mathsf{msg}_{\ell,i}\}_{\ell \in [3]}, \mathbf{T}_{\mathsf{ncom}}^{i \to j}[3], \mathsf{td}_{1,j})$$

$$\text{"Honest" witness } \mathsf{w}_a^{i \to j} := (0, \mathsf{r}_i, \mathsf{r}_{\mathsf{recom}}^{i \to \bullet})$$

(f) Ecom: $\mathsf{ecom}_3^{i \to j} \leftarrow \mathsf{Ecom}_3\left(\mathsf{rwi}_{a,3}^{i \to \bullet}\right)$ to commit to $\mathsf{rwi}_{a,3}^{i \to \bullet}$.

(g) RWI_b: $\mathsf{rwi}_{b,3}^{i\to j} \leftarrow \mathsf{RWI}_3(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j})$ to prove that $R_b(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j}) = 1$, where

Statement $\mathsf{st}_b^{i\to j} := (\mathbf{T}_{\mathsf{rwi}_a}^{i\to\bullet}[2], \mathsf{st}_a^{i\to\bullet}, \mathbf{T}_{\mathsf{ecom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$

"Honest" witness $\mathsf{w}_b^{i\to j} := (\mathsf{r}_{\mathsf{rwi}_a}^{i\to\bullet}, \mathsf{w}_a^{i\to\bullet}, \mathsf{rwi}_{a,3}^{i\to\bullet}, \mathsf{r}_{\mathsf{ecom}}^{i\to\bullet})$

(h) OT: Receiver message $\mathsf{ot}_3^{j\to i} \leftarrow \mathsf{OT}_3(\mathsf{rwi}_{b,3}^{i\to j})$ using input $\mathsf{rwi}_{b,3}^{i\to j}$.

4. **Check Abort Condition:**

   Sim now checks whether $\mathcal{A}$ explicitly aborted in the third round. This happens if $\mathcal{A}$ doesn't send its third round messages, or if every honest party aborts when the trapdoor condition does not verify. Check if $\exists P_j \in \mathcal{A}$, such that $\mathsf{TDOut}(\mathsf{td}_{1,j}, \mathsf{td}_{2,j}, \mathsf{td}_{3,j}) \neq 1$. If so, the Sim outputs the partial view generated so far and stops. Otherwise, we say that "Check Abort" succeeded and we proceed.

5. **Check Implicit Abort:**

   We run a look ahead threads to extract from Ecom the RWI proofs for $\mathcal{L}_a$ from each malicious party P_j. We then check if all the extracted RWI proofs verify. This ensures that on the given thread, the malicious parties exhibit honest behavior. If for even a single malicious party P_j the proofs don't verify, then we take evasive action as mentioned in Step 1.5 below.

Remark 6. *We use a specific property of* $\mathsf{Ext}_{\mathsf{ecom}}$*, namely that since it's input delayed, the commitment in the first round is to a mask* mask *and the input delayed property is achieved by masking the input with* mask*. In fact,* mask *is statistically determined by the first round of* Ecom*. Thus, to extract from multiple instances of the input-delayed extractable commitment with a single shared first message that potentially commit to different inputs, it suffices to extract* mask *in a single instance and using* mask *to unmask, and thus retrieve, other inputs. Since the* mask *is extracted via decommittment information, it's easy to verify that the extracted value* mask *is indeed correct.*

Step 1.5 - Evasive Action for Implicit Abort: We run this step only if there is an implicit abort. Since we cannot do an explicit abort on behalf of the honest parties, we want to continue the main thread from Step 1 but garble the $\mathsf{C}_\perp$ circuit[14] in the fourth round, since we are sure that adversary will not be able to evaluate the garbled circuit to produce any other output. But in order to do this, we will need to extract the trapdoor to prove the WI statement for $\mathcal{L}_c$ claiming that the garbled circuit was computed honestly. We can do this because the adversary did not cause an explicit abort, and the extracted trapdoor can be publicly checked. But recall that this trapdoor must be committed inside the ncom to use the "trapdoor witness" for $\mathcal{L}_c$. To do so, we must re-sample the main thread making a change only in the third round to commit to the trapdoor, while at the same time ensuring that the resultant thread still causes an implicit abort. To do so, we follow a similar analysis as that of [GK96a]. Since some of these steps are similar to the case where there are no aborts, we defer the description to the relevant steps indicating whether we are in the case of an **implicit abort** or **no abort**.

[14]Circuit that outputs $\perp$ independent of input.

Step 2 - Rewinding: Since the adversary has not aborted explicitly, we will need to start simulation the underlying protocol to produce an appropriate transcript. As the first step, the simulator will rewind $\mathcal{A}$.

1. Sim now rewinds $\mathcal{A}$ to the end of round 1 and freezes the main thread at this point. Then, Sim creates a set of T (to be determined later) look-ahead threads, where on each thread, only rounds 2 and 3 of the protocol are executed in the following manner:

   (a) **Round 2:**
   In every look-ahead thread, for each honest party P_i and for each $j \neq i$, Sim executes the same strategy as in round 2 of step 1, using fresh randomness each time(for each primitive).

   (b) **Round 3:**
   In every look-ahead thread, for each honest party P_i and for each $j \neq i$, Sim executes the same strategy as in round 3 of step 1, using fresh randomness each time.

2. **No abort case:**

   (a) For each look-ahead thread, define a thread to be GOOD with respect to P_{i^*} if for all malicious parties P_j:
      – P_j does send its third round messages.
      – $\mathsf{TDOut}\left(\mathsf{td}_{1,j}, \mathsf{td}_{2,j}, \mathsf{td}_{3,j}\right) = 1$ where $\mathsf{td}_{2,j}$ is as computed in round 3.
      – The extracted RWI proofs for $\mathcal{L}_a$ are all accepting where P_j is the prover, and P_{i^*} is the verifier. *We use* mask *obtained in Step 1 to do the extractions by simply unmasking the commitment in* Ecom.

   (b) The number of threads T created is such that at least $(12 \cdot \lambda)$ GOOD threads exists. That is, Sim keeps running till it obtains $(12 \cdot \lambda)$ GOOD threads.

3. **Implicit abort case:**

   (a) For each look-ahead thread, define a thread to be $\mathsf{IMPLICIT}$ if
      – every malicious party P_j does send its third round messages.
      – for every malicious party P_j, $\mathsf{TDOut}\left(\mathsf{td}_{1,j}, \mathsf{td}_{2,j}, \mathsf{td}_{3,j}\right) = 1$ where $\mathsf{td}_{2,j}$ is as computed in round 3.
      – For some malicious party P_j, the extracted RWI proofs for $\mathcal{L}_a$ where P_j is the prover are all accepting. *We use* mask *obtained in Step 1 to do the extractions by simply unmasking the commitment in* Ecom.

   (b) The number of threads T_{IMPLICIT} created is such that at least $(12 \cdot \lambda)$ $\mathsf{IMPLICIT}$ threads exists. That is, Sim keeps running till it obtains $(12 \cdot \lambda)$ $\mathsf{IMPLICIT}$ threads.

Remark 7. *We want to re-emphasize that only one of the two above cases are executed.*

Step 3 - Input and Trapdoor Extraction: Now, Sim extracts all relevant information. Note that all the relevant information can be extracted from sufficient number of GOOD threads with respect to a single honest party for the case of *no abort*. For the case of *implicit abort*, we extract only the trapdoor.

Sim does the following for the **no abort** case:

1. Select 5 threads that are GOOD with respect to some honest party P_{i^*}. In each GOOD thread, we know $\exists$ honest party P_i such that for all malicious parties P_j, the adversary does not cause P_i to abort. Since $(12 \cdot \lambda) > (5 \cdot n)$[a], there must exist one honest party P_{i^*} corresponding to a set of 5 GOOD threads.

2. **Trapdoor Extraction:** For every corrupted party P_j, extract a trapdoor t_j by running the trapdoor extractor TDExt on input the transcript of the trapdoor generation protocol with P_j playing the role of the trapdoor generator from any 3 GOOD threads. Specifically, compute
$$\mathsf{t}_j \leftarrow \mathsf{TDExt}\left(\mathsf{td}_1, \{\mathsf{td}_2^k, \mathsf{td}_3^k\}_{k\in[3]}\right)$$
where $\left(\mathsf{td}_1, \mathsf{td}_2^k, \mathsf{td}_3^k\right)$ denotes the transcript of the trapdoor generation protocol with P_j as the sender of the k-th GOOD thread.

3. **Input Extraction:** For every corrupted party P_j, extract the mask for the input and randomness pair $(\mathsf{x}_j, \mathsf{r}_j)$ by running the extractor $\mathsf{Ext}_{\mathsf{RECom}}$ on input the transcript of the extractable commitment protocol between P_j and P_{i^*} from the 5 GOOD threads picked above. That is, compute
$$\mathsf{mask}^{j\to i^*} \leftarrow \mathsf{Ext}_{\mathsf{RECom}}\left(\mathsf{recom}_1^{j\to i^*}, \{\mathsf{recom}_{2,k}^{j\to i^*}, \mathsf{recom}_{3,k}^{j\to i^*}\}_{k\in[5]}\right)$$
where $\mathsf{recom}_1^{j\to i^*}, \mathsf{recom}_{2,k}^{j\to i^*}, \mathsf{recom}_{3,k}^{j\to i^*}$ denotes the transcript of the extractable commitment protocol between P_j and P_{i^*} on the k-th GOOD thread.

4. **Proof Extraction:** Since we've already extracted the proofs in Step 1, by Remark 6 we can extract the proofs in each thread without having to rewind, by just unmasking with the extracted mask from Step 1.

5. Output $\perp_{\mathsf{extract}}$ if any of steps 2 or 3 fail.

Sim does the following for the **implicit abort** case:

1. **Trapdoor Extraction:** For every corrupted party P_j, extract a trapdoor t_j by running the trapdoor extractor TDExt on input the transcript of the trapdoor generation protocol with P_j playing the role of the trapdoor generator from any 3 IMPLICIT threads. Specifically, compute
$$\mathsf{t}_j \leftarrow \mathsf{TDExt}\left(\mathsf{td}_1, \{\mathsf{td}_2^k, \mathsf{td}_3^k\}_{k\in[3]}\right)$$
where $\left(\mathsf{td}_1, \mathsf{td}_2^k, \mathsf{td}_3^k\right)$ denotes the transcript of the trapdoor generation protocol with P_j as the sender of the k-th IMPLICIT thread.

2. **Proof Extraction:** Since we've already extracted the proofs in Step 1, by Remark 6 we can extract the proofs in each thread without having to rewind, by just unmasking with the extracted mask from Step 1.
3. Output $\bot_{\mathsf{extract}}$ if step 2 fails.

[a]without loss of generality, assume the number of parties $n = \lambda$

Step 4 - Abort Probability Estimation: Sim estimate below the probability with which $\mathcal{A}$ either does not abort, or implicitly aborts.

If **Implicit abort** case,

Set $\varepsilon' = \frac{12\cdot\lambda}{T_{\mathsf{IMPLICIT}}}$ as the probability that the adversary implicitly abort.

If **no abort** case,

Set $\varepsilon' = \frac{12\cdot\lambda}{T}$ as the probability that the adversary doesn't abort.

Step 5 - Re-sampling the Main Thread: Using the information extracted, Sim samples the main thread. It also needs to force this transcript, and uses the estimate obtained earlier to upper bound the number of attempts to do try this.

Sim sets a counter to value 0. Now Sim attempts to force the following transcript in the main thread until it accepts, or the counter reaches the cut-off point.

1. **Round 2** :

   Run exactly as done in Step 1.

2. **Round 3** :

   There are some key differences from the threads generated in the previous steps:

   - The non-malleable commitment from an honest party P_i to a malicious party P_j now contains the extracted trapdoor t_j.
   - The witness indistinguishable proofs use the "trapdoor witness".
   - If **implicit abort** case: The third round of the MPC is generated by using inputs 0.
   - If **no abort** case: The third round of the MPC is generated by the simulator for the underlying protocol.

   In more detail, Sim computes the third round of the delayed semi-malicious protocol Π for all the honest parties:

   - if **implicit abort** case, for each honest player P_i, $\mathsf{msg}_{3,i} \leftarrow \Pi_3(0)$
   - if **no abort** case, $\{\mathsf{msg}_{3,i}\}_{P_i\in\mathcal{H}} \leftarrow \mathcal{S}_3$ using the transcript obtained so far.

   For each honest party P_i, Sim computes the following, and sends the messages to $\mathcal{A}$:

   (a) TDGen: $\mathsf{td}_{3,i} \leftarrow \mathsf{TDGen}_3$.

   For every $j \neq i$:

(b) NMCom: $\mathsf{ncom}_3^{i\to j} \leftarrow \mathsf{NMCom}_3(\mathsf{t}_j)$ to commit to extracted trapdoor t_j.

(c) REContent: $\mathsf{recom}_3^{i\to j} \leftarrow \mathsf{RECom}_3(0)$ to commit to 0.

(d) RWI: $\mathsf{rwi}_{a,3}^{i\to j} \leftarrow \mathsf{RWI}_3\left(\mathsf{st}_a^{i\to j}, \mathsf{w}_a^{i\to j}\right)$ to prove that $R_a(\mathsf{st}_a^{i\to j}, \mathsf{w}_a^{i\to j}) = 1$, where

$$\text{Statement } \mathsf{st}_a^{i\to j} := (\mathbf{T}_\Pi[2], \mathbf{T}_{\mathsf{recom}}^{i\to\bullet}[3], \{\mathsf{msg}_{\ell,i}\}_{\ell\in[3]}, \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$

$$\text{"Trapdoor" witness } \mathsf{w}_a^{i\to j} := \left(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j}\right)$$

(e) Ecom: $\mathsf{ecom}_3^{i\to j} \leftarrow \mathsf{Ecom}_3\left(\mathsf{rwi}_{a,3}^{i\to\bullet}\right)$ to commit to $\mathsf{rwi}_{a,3}^{i\to\bullet}$.

(f) RWI$_b$: $\mathsf{rwi}_{b,3}^{i\to j} \leftarrow \mathsf{RWI}_3(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j})$ to prove that $R_b(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j}) = 1$, where

$$\text{Statement } \mathsf{st}_b^{i\to j} := (\mathbf{T}_{\mathsf{rwi}_a}^{i\to\bullet}[2], \mathsf{st}_a^{i\to\bullet}, \mathbf{T}_{\mathsf{ecom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$

$$\text{"Trapdoor" witness } \mathsf{w}_b^{i\to j} := \left(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j}\right)$$

(g) OT: Receiver message $\mathsf{ot}_3^{j\to i} \leftarrow \mathsf{OT}_3(\mathsf{rwi}_{b,3}^{i\to j})$ using input $\mathsf{rwi}_{b,3}^{i\to j}$.

3. **Abort Condition:**

   (a) **implicit abort** case: if the adversary doesn't send its third round message; or $\exists P_j \in \mathcal{A}$, such that $\mathsf{TDOut}(\mathsf{td}_{1,j}, \mathsf{td}_{2,j}, \mathsf{td}_{3,j}) = 1$ increment counter by 1.

   no abort case: if the adversary doesn't send its third round message; or $\exists P_j \in \mathcal{A}$, such that $\mathsf{TDOut}(\mathsf{td}_{1,j}, \mathsf{td}_{2,j}, \mathsf{td}_{3,j}) = 1$ or the extracted proofs for $\mathcal{L}_a$ from P_j do not accept, increment counter by 1.

   (b) If Sim's running time is 2^λ. Abort.

   (c) If the counter value was not increased, we can proceed to Step 7.

   (d) Else, if the counter value is less that $\frac{\lambda^2}{\varepsilon'}$ rewind back to the beginning of round 2 in Step 6 and re-sample the main thread with fresh randomness. Otherwise, Abort indicating failure.

Step 6 - Query the Ideal Functionality: The following is done only in the **no abort** case.

1. Sim queries the ideal functionality with the set of values $\{x_j\}$ where x_j is the input of adversarial party P_j that was extracted in the previous step using mask obtained through extraction by rewinding. *This is done in this manner since the adversary may use a different input in each thread, and we want to use the input it uses on the main thread.* Since the adversary commits to its input only on completion of the third round on the main thread.
2. Sim receives output y from the ideal functionality.

Step 7 - Extract proofs from OT: In order to determine whether we need to put in simulated messages into garbled circuits in the fourth round, we extract from all OT receiver messages in parallel by running sufficiently many look-ahead threads. Note that this is different from an implicit abort since if there is no implicit abort, it is guaranteed that the adversary behaved honestly in the underlying protocol. It is still possible that it doesn't put the correct proof inside of the OT receiver messages. We just need to ensure that the relevant garbled circuits then become "opaque". Let us denote this event as **opaque**, when there is at least one malicious party who's extracted proof is not accepting.

Step 8 - Finishing the Main Thread: Sim now finishes off the main thread by computing the last round of the protocol.

1. **Round 4:**

   If **no abort** case, compute the simulated fourth round message of the delayed semi-malicious protocol Π: $\{\mathsf{msg}_{4,i}\}_{P_i \in \mathcal{H}} \leftarrow \mathcal{S}_4\left(y, \{\mathsf{x}_j, \mathsf{r}_j\}_{P_j \in \mathcal{A}}\right)$. Note that $\mathcal{S}_4$ will not be called if there is an implicit or explicit abort.

   For each honest party P_i, Sim computes the following, and sends the messages to $\mathcal{A}$:

   (a) Garbled Circuit taking into account the extracted RWI proof for $\mathcal{L}_b$: $\overline{\mathsf{C}}_i$, where
      - if **implicit abort** or **opaque** case, then $\left(\overline{\mathsf{C}}_i, \overline{\mathsf{lab}}_i\right) \leftarrow \mathsf{Garble}\left(\mathsf{C}_\perp; \mathsf{r}_{\mathsf{gc},i}\right)$
      - else if **no abort** case, $(\overline{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(\mathsf{C}[\mathsf{msg}_{4,i}, \mathbf{T}^{\bullet\to i}_{\mathsf{rwi}_b}[2], \mathsf{st}^{\bullet\to i}_b, \mathsf{r}^{\bullet\to i}_{\mathsf{rwi}_b}]; \mathsf{r}_{\mathsf{gc},i})$

   For every $j \neq i$:

   (b) OT: Fourth round sender message $\mathsf{ot}^{i\to j}_4 \leftarrow \mathsf{OT}_4\left(\overline{\mathsf{lab}}_{i|_j}\right)$ using input $\overline{\mathsf{lab}}_{i|_j}$.

   (c) OT: Receiver randomness $\mathsf{r}^{j\to i}_{\mathsf{ot}}$.

   (d) WI: $\mathsf{wi}^{i\to j}_3 \leftarrow \mathsf{WI}_3\left(\mathsf{st}^{i\to j}_c, \mathsf{w}^{i\to j}_c\right)$, to prove that $R_c(\mathsf{st}^{i\to j}_c, \mathsf{w}^{i\to j}_c) = 1$, where

      Statement $\mathsf{st}^{i\to j}_c := (\mathbf{T}_\Pi[3], \mathbf{T}^{i\to\bullet}_{\mathsf{recom}}[3], \mathbf{T}^{\bullet\to i}_{\mathsf{rwi}_b}[2], \mathsf{st}^{\bullet\to i}_b, \mathbf{T}^{i\to\bullet}_{\mathsf{ot}}[4], \overline{\mathsf{C}}_i, \mathbf{T}^{i\to j}_{\mathsf{ncom}}[3], \mathsf{td}_{1,j})$

      "Trapdoor" witness $\mathsf{w}^{i\to j}_c := \left(\mathsf{t}_j, \mathsf{r}^{i\to j}_{\mathsf{ncom}}\right)$

2. **Output Computation:**

   If **no abort**:

   - *For each honest party P_i*, Sim does the following in the main thread,:

     (a) If $\exists j \neq i$, s.t. $\mathsf{WI}_4(\mathsf{st}^{j\to i}_c, \mathbf{T}^{j\to i}_{\mathsf{wi}}[3]) \neq 1$, abort.

   If there is no abort, instruct the ideal functionality to deliver output to the honest parties.

Remark 8. *We note that if any round, a subprotocol outputs $\perp$, P_i broadcast $\perp$, sets output to be $\perp$ and aborts. If P_i receives a $\perp$ from another party, it sets its output to be $\perp$ and aborts.*

Running Time of the Simulator: The simulator runs in expected time polynomial in λ. The analysis follows identically from that of [BGJ+18]. The only steps that the simulator can run in exponential time are:

1. Step 2, where Sim rewinds till it gets $12 \cdot \lambda$ implicitly-aborting/non-aborting transcripts. If ε denotes the probability with which Sim goes into Step 2 (i.e. implicit abort or did not abort in

Step 1), then the expected total number of threads created are $\frac{12\cdot\lambda}{\varepsilon}$, where each thread takes only $\mathsf{poly}(\lambda)$ time.

2. Step 5, where Sim resamples the main thread. If the probability estimate is correct, then it is easy to see that this step requires the creation of at most $\frac{\lambda^2}{\varepsilon}$ threads. This step might take time 2^λ, but that only happens with probability $\frac{1}{2^\lambda}$. See Section A.5 for further discussion.

This gives a total expected running time of

$$\mathsf{poly}(\lambda) + \mathsf{poly}(\lambda)\cdot\varepsilon\left(\frac{12\cdot\lambda}{\varepsilon} + \left(1-\frac{1}{2^\lambda}\right)\frac{\lambda^2}{\varepsilon} + 2^\lambda\left(\frac{1}{2^\lambda}\right)\right) \le \mathsf{poly}(\lambda)$$

4.5.2.1 Hybrids

Assume by contradiction that there is an adversary $\mathcal{A}$ that distinguishes the real and ideal worlds with some non-negligible probability μ. μ will be used to set certain parameters in the hybrids.

$\mathsf{Hyb}_{\mathsf{REAL}}$: **Real World:** The hybrid is the same as the real world execution. We consider a simulator $\mathsf{Sim}_{\mathsf{Hyb}}$ that plays the role of the honest parties.

Hyb_0: **Determining Abort in the 3rd Round and Extraction:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ makes the following changes:

1. $\mathsf{Sim}_{\mathsf{Hyb}}$ executes the first 3 rounds of the protocol using the honest parties' strategy. If the adversary causes an abort, $\mathsf{Sim}_{\mathsf{Hyb}}$ outputs only the view of the adversary and stops.
2. If the "Check Abort" step succeeds, $\mathsf{Sim}_{\mathsf{Hyb}}$ checks if there is an implicit abort by extracting the RWI proofs.
3. If there is either an implicit abort or no abort, $\mathsf{Sim}_{\mathsf{Hyb}}$ rewinds back to after the completion of round 1 of the protocol and freezes the main thread. $\mathsf{Sim}_{\mathsf{Hyb}}$ creates a set of $\frac{5\cdot n\cdot\lambda}{\mu}$ look ahead threads as described in Step 2 of Sim. Which is to say that in all the threads, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses the honest parties' inputs and follows the protocol. The look ahead threads are identical to the main thread.
4. If there is an implicit abort, the $\mathsf{Sim}_{\mathsf{Hyb}}$ now extracts the trapdoors and proofs from the created look-ahead threads. Specifically, it runs the "Input and Trapdoor Extraction" phase described in step 3 of the description of Sim using the first 3 look-ahead threads that are IMPLICIT.
5. If there is no abort, the $\mathsf{Sim}_{\mathsf{Hyb}}$ now extracts the input, trapdoors and proofs from the created look-ahead threads. Specifically, it runs the "Input and Trapdoor Extraction" phase described in step 3 of the description of Sim using the first 5 look-ahead threads that are GOOD with respect to some honest party P_{i^*}.
6. $\mathsf{Sim}_{\mathsf{Hyb}}$ outputs $\perp_{\mathsf{extract}}$ if either of the above two steps fails.
7. $\mathsf{Sim}_{\mathsf{Hyb}}$ continues the execution of the main thread it had previously frozen. It does this as in the honest execution of $\mathsf{Hyb}_{\mathsf{REAL}}$. If the adversary causes an abort, $\mathsf{Sim}_{\mathsf{Hyb}}$ rewinds to the end of round 1 and re-samples the main thread honestly. This process is repeated at most $\frac{\lambda}{\mu}$ times.

Since μ is noticeable, we are guaranteed that $\mathsf{Sim}_{\mathsf{Hyb}}$ will run in polynomial in this hybrid, and subsequent hybrids, when performing this check.

Hyb_1: **Using input 0 in the Aborting Step:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ does the "Check Abort" step using the input 0 instead of the real honest party inputs. If the adversary does cause an abort, then $\mathsf{Sim}_{\mathsf{Hyb}}$ just outputs the view of the adversary and stops. Else, it proceeds as in Hyb_0. This is done using a sequence of sub-hybrids. We only describe changes made in each sub-hybrid, with the remaining execution identical to the previous hybrid.

$\mathsf{Hyb}_{1,0}$: **Change** OT **receiver input to 0:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round to replace the OT receiver input for all honest parties with 0. In Hyb_0, the receiver input to the OT was the third message of the RWI proof for $\mathcal{L}_b$.

$\mathsf{Hyb}_{1,1}$: **Change** Ecom **input to 0:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round to replace the Ecom input for all honest parties with 0. In $\mathsf{Hyb}_{1,0}$, the input to Ecom was the third message of the RWI proof for $\mathcal{L}_a$.

$\mathsf{Hyb}_{1,2}$: **Change** RECom **input to 0:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round to replace the RECom input for all honest parties with $(0, \mathsf{r}_i)$. In $\mathsf{Hyb}_{1,1}$, the input to Ecom for an honest party P_i was its input and randomness $(\mathsf{x}_i, \mathsf{r}_i)$ for the underlying protocol Π.

$\mathsf{Hyb}_{1,3}$: **Change** Π **input to 0:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round to replace the Π input for all honest parties with 0. In $\mathsf{Hyb}_{1,2}$, the input to Π for an honest party P_i in the third round was x_i.

$\mathsf{Hyb}_{1,4}$: **Change** Ecom **input to** RWI**:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round to replace the Ecom input for all honest parties with the correctly computed third message of the RWI proof for $\mathcal{L}_a$ using input 0. In $\mathsf{Hyb}_{1,3}$, the input to Ecom was 0.

$\mathsf{Hyb}_{1,5}$: **Change** OT **receiver input to** RWI**:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round to replace the OT receiver input for all honest parties with the correctly computed third message of the RWI proof for $\mathcal{L}_b$ using. In Hyb_0, the receiver input to the OT was 0.

Note that $\mathsf{Hyb}_{1,5} \equiv \mathsf{Hyb}_1$

Note that if the adversary aborts in the first three rounds, then we can skip the remaining hybrids.

Hyb_2: **Using input 0 in the look-ahead threads:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies each look-ahead thread to follow the protocol but replacing the honest player inputs with 0. This is done in a sequence of hybrids, where in each sequence we only modify a single look ahead thread. Since the number of threads are T, we do the following:

$\forall k \in [T]$ the following changes are made *only to the k-th thread*:

$\mathsf{Hyb}_{2,k,0}$: **Change** NMCom **on k-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to commit in the NMCom to the trapdoor. In Hyb_1, NMCom was a commitment to a random value. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round NMCom message to be $\mathsf{ncom}_3^{i \to j} \leftarrow \mathsf{NMCom}_3(\mathsf{t}_j)$ where t_j is a valid trapdoor extracted from the *other look-ahead threads* as in Hyb_1.

$\mathsf{Hyb}_{2,k,1}$: **Switch** RWI **proofs for** $\mathcal{L}_b$ **on the** k**-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to switch to the "trapdoor witness" in the RWI proofs for $\mathcal{L}_b$. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes $\mathsf{rwi}_{b,3}^{i\to j} \leftarrow \mathsf{RWI}_3(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j})$, where

$$\text{Statement } \mathsf{st}_b^{i\to j} := (\mathbf{T}_{\mathsf{rwi}_a}^{i\to\bullet}[2], \mathsf{st}_a^{i\to\bullet}, \mathbf{T}_{\mathsf{ecom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$
$$\text{“Trapdoor” witness } \mathsf{w}_b^{i\to j} := \left(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j}\right)$$

$\mathsf{Hyb}_{2,k,2}$: **Switch** RWI **proofs for** $\mathcal{L}_a$ **on the** k**-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to switch to the "trapdoor witness" in the RWI proofs for $\mathcal{L}_a$. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes $\mathsf{rwi}_{a,3}^{i\to j} \leftarrow \mathsf{RWI}_3\left(\mathsf{st}_a^{i\to j}, \mathsf{w}_a^{i\to j}\right)$, where

$$\text{Statement } \mathsf{st}_a^{i\to j} := (\mathbf{T}_{\Pi}[2], \mathbf{T}_{\mathsf{recom}}^{i\to\bullet}[3], \{\mathsf{msg}_{\ell,i}\}_{\ell\in[3]}, \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$
$$\text{“Trapdoor” witness } \mathsf{w}_a^{i\to j} := \left(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j}\right)$$

$\mathsf{Hyb}_{2,k,3}$: **Change** RECom **input to 0 on the** k**-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to replace the RECom input for all honest parties with $(0, \mathsf{r}_i)$. In $\mathsf{Hyb}_{2,k,2}$, the input to Ecom for an honest party P_i was its input and randomness $(\mathsf{x}_i, \mathsf{r}_i)$ for the underlying protocol Π. This is done by a sequence of sub-hybrids given below.

$\mathsf{Hyb}_{2,k,3,0}$: **Change** Com **sender's message on main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ changes the Com commitment inside the RECom in the first round of the protocol. Specifically, for every honest party P_i and malicious party P_j and for all $\ell \in [N]$, compute $\mathsf{recom}_{1,\ell} \leftarrow \mathsf{Com}(0)$. This is done since all the look ahead threads share the same first round messages with the main thread.

$\mathsf{Hyb}_{2,k,3,1}$: **Change polynomial in third round:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ picks a new polynomial q to change the RECom third round messages. Specifically, for every honest party P_i and malicious party P_j do the following:

- for every $\ell \in [N]$, pick a new degree 4 polynomial q_ℓ such that $(\mathsf{x}_i \oplus \mathsf{p}_\ell(0)) = (0 \oplus \mathsf{q}_\ell(0))$.
- compute $\mathsf{recom}_{3,\ell}$ as $(0 \oplus \mathsf{q}_\ell(0), \mathsf{q}_\ell(\mathsf{z}_\ell))$.

$\mathsf{Hyb}_{2,k,3,2}$: **Commit to new polynomial:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ changes the Com commitment inside the RECom in the first round of the protocol. Specifically, for every honest party P_i and malicious party P_j and for all $\ell \in [N]$, compute $\mathsf{recom}_{1,\ell} \leftarrow \mathsf{Com}(q_\ell)$.

Note that $\mathsf{Hyb}_{2,k,3,2} \equiv \mathsf{Hyb}_{2,k,3}$

$\mathsf{Hyb}_{2,k,4}$: **Change** Π **input to 0 on the** k**-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to replace the Π input for all honest parties with 0. In $\mathsf{Hyb}_{2,3}$, the input to Π for an honest party P_i in the third round was x_i.

$\mathsf{Hyb}_{2,k,5}$: **Switch** RWI **proofs for** $\mathcal{L}_a$ **on the** k**-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to switch back to the "honest witness" in the RWI proofs for $\mathcal{L}_a$. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes $\mathsf{rwi}_{a,3}^{i\to j} \leftarrow \mathsf{RWI}_3\left(\mathsf{st}_a^{i\to j}, \mathsf{w}_a^{i\to j}\right)$, where

$$\text{Statement } \mathsf{st}_a^{i\to j} := (\mathbf{T}_\Pi[2], \mathbf{T}_{\mathsf{recom}}^{i\to\bullet}[3], \{\mathsf{msg}_{\ell,i}\}_{\ell\in[3]}, \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$
$$\text{“Honest” witness } \mathsf{w}_a^{i\to j} := (0, \mathsf{r}_i, \mathsf{r}_{\mathsf{recom}}^{i\to\bullet})$$

$\mathsf{Hyb}_{2,k,6}$: **Switch** RWI **proofs for $\mathcal{L}_b$ on the k-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to switch back to the “honest witness” in the RWI proofs for $\mathcal{L}_b$. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes $\mathsf{rwi}_{b,3}^{i\to j} \leftarrow \mathsf{RWI}_3(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j})$ to prove that $R_b(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j}) = 1$, where

$$\text{Statement } \mathsf{st}_b^{i\to j} := (\mathbf{T}_{\mathsf{rwi}_a}^{i\to\bullet}[2], \mathsf{st}_a^{i\to\bullet}, \mathbf{T}_{\mathsf{ecom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$
$$\text{“Honest” witness } \mathsf{w}_b^{i\to j} := (\mathsf{r}_{\mathsf{rwi}_a}^{i\to\bullet}, \mathsf{w}_a^{i\to\bullet}, \mathsf{rwi}_{a,3}^{i\to\bullet}, \mathsf{r}_{\mathsf{ecom}}^{i\to\bullet})$$

$\mathsf{Hyb}_{2,k,7}$: **Change** NMCom **on k-th thread:** In this sub-hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the k-th thread to commit in the NMCom to a random value. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round NMCom message to be $\mathsf{ncom}_3^{i\to j} \leftarrow \mathsf{NMCom}_3\left(\widetilde{\mathsf{r}}_j\right)$.

Note that $\mathsf{Hyb}_{2,T,7} \equiv \mathsf{Hyb}_2$

Hyb_3: **Change** NMCom **on main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the main thread to commit in the NMCom to the trapdoor. In Hyb_2, NMCom was a commitment to a random value. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round NMCom message to be $\mathsf{ncom}_3^{i\to j} \leftarrow \mathsf{NMCom}_3\left(\mathsf{t}_j\right)$ where t_j is a valid trapdoor extracted from the look-ahead threads as in Hyb_2.

Hyb_4: **Switch** RWI **proofs for $\mathcal{L}_b$ on the main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the main thread to switch to the “trapdoor witness” in the RWI proofs for $\mathcal{L}_b$. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes $\mathsf{rwi}_{b,3}^{i\to j} \leftarrow \mathsf{RWI}_3(\mathsf{st}_b^{i\to j}, \mathsf{w}_b^{i\to j})$, where

$$\text{Statement } \mathsf{st}_b^{i\to j} := (\mathbf{T}_{\mathsf{rwi}_a}^{i\to\bullet}[2], \mathsf{st}_a^{i\to\bullet}, \mathbf{T}_{\mathsf{ecom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$
$$\text{“Trapdoor” witness } \mathsf{w}_b^{i\to j} := \left(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j}\right)$$

Hyb_5: **Switch** RWI **proofs for $\mathcal{L}_a$ on the main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the main thread to switch to the “trapdoor witness” in the RWI proofs for $\mathcal{L}_a$. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes $\mathsf{rwi}_{a,3}^{i\to j} \leftarrow \mathsf{RWI}_3\left(\mathsf{st}_a^{i\to j}, \mathsf{w}_a^{i\to j}\right)$, where

$$\text{Statement } \mathsf{st}_a^{i\to j} := (\mathbf{T}_\Pi[2], \mathbf{T}_{\mathsf{recom}}^{i\to\bullet}[3], \{\mathsf{msg}_{\ell,i}\}_{\ell\in[3]}, \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$
$$\text{“Trapdoor” witness } \mathsf{w}_a^{i\to j} := \left(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j}\right)$$

Hyb_6: **Switch** WI **proofs for $\mathcal{L}_c$ on the main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the fourth round of the main thread to switch to the “trapdoor witness” in the WI proofs for $\mathcal{L}_c$. Specifically, for every honest party P_i and every party P_j, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes $\mathsf{wi}_3^{i\to j} \leftarrow \mathsf{WI}_3\left(\mathsf{st}_c^{i\to j}, \mathsf{w}_c^{i\to j}\right)$, to prove that $R_c(\mathsf{st}_c^{i\to j}, \mathsf{w}_c^{i\to j}) = 1$, where

$$\begin{aligned}\text{Statement } \mathsf{st}_c^{i\to j} &:= (\mathbf{T}_\Pi[3], \mathbf{T}_{\mathsf{recom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{rwi}_b}^{\bullet\to i}[2], \mathsf{st}_b^{\bullet\to i}, \mathbf{T}_{\mathsf{ot}}^{i\to\bullet}[4], \overline{\mathsf{C}}_i, \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})\\ \text{“Trapdoor” witness } \mathsf{w}_c^{i\to j} &:= \left(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j}\right)\end{aligned}$$

Hyb_7: **Change** RECom **input to 0 on the main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the third round of the main thread to replace the RECom input for all honest parties with 0. In Hyb_6, the input to RECom for an honest party P_i was its input and randomness $(\mathsf{x}_i, \mathsf{r}_i)$ for the underlying protocol Π.

Hyb_8: **Simulate** Π **on main thread:** In this hybrid $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the transcript of the underlying protocol Π.

Specifically, if there is an implicit abort $\mathsf{Sim}_{\mathsf{Hyb}}$ does the following:

1. Compute the third round message of each honest party using input 0.

Else, if there is no abort $\mathsf{Sim}_{\mathsf{Hyb}}$ does the following:

1. Due to the fact that the first two simulated rounds of Π are honest computations, we do not make any changes to the first two rounds but refer to the collective first round honest inputs as the output of $\mathcal{S}_1$ with randomness $\mathsf{r}_{\mathcal{S}} := \{\mathsf{r}_i\}_{P_i\in\mathcal{H}}$. Likewise for the second round messages.
2. Compute the third round messages of all honest parties in the delayed semi-malicious protocol Π:
   $\{\mathsf{msg}_{3,i}\}_{P_i\in\mathcal{H}} \leftarrow \mathcal{S}_3$ using the transcript obtained so far and randomness defined above.
3. Compute the third round messages of all honest parties in the underlying protocol Π:
   $\{\mathsf{msg}_{4,i}\}_{P_i\in\mathcal{H}} \leftarrow \mathcal{S}_4\left(y, \{\mathsf{x}_j, \mathsf{r}_j\}_{P_j\notin\mathcal{H}}\right).$

Hyb_9: **Extract Proofs from** OT**:** In this hybrid $\mathsf{Sim}_{\mathsf{Hyb}}$ only creates sufficiently many look ahead threads to extract the proofs for $\mathcal{L}_b$ that are used as receiver inputs to the OT. If there is proof from a malicious party that does not accept, we denote this event as **opaque**.

Hyb_{10}: **Change** GC **on main thread:** In this hybrid $\mathsf{Sim}_{\mathsf{Hyb}}$ only modifies the garbled circuits of honest parties P_i if there is an implicit abort in Step 1. Specifically, if there is an implicit abort, the garbled circuit for each honest party P_i is computed as:

$$\left(\mathsf{C}_i, \overline{\mathsf{lab}}_i\right) \leftarrow \mathsf{Garble}\left(\mathsf{C}_\perp\right)$$

where $\mathsf{C}_\perp$ is the circuit with the same topology as C but always outputs $\perp$. We note that even in the case of an implicit abort, we are able to extract the trapdoor, but not necessarily the witness.

For every honest party P_i,

– if **implicit abort** or **opaque** case, then $\left(\overline{\mathsf{C}}_i, \overline{\mathsf{lab}}_i\right) \leftarrow \mathsf{Garble}\left(\mathsf{C}_\perp; \mathsf{r}_{\mathsf{gc},i}\right)$

– else if **no abort** case, $(\overline{\mathsf{C}}_i, \overline{\mathsf{lab}}_i) \leftarrow \mathsf{Garble}(\mathsf{C}[\mathsf{msg}_{4,i}, \mathbf{T}_{\mathsf{rwi}_b}^{\bullet\to i}[2], \mathsf{st}_b^{\bullet\to i}, \mathsf{r}_{\mathsf{rwi}_b}^{\bullet\to i}]; \mathsf{r}_{\mathsf{gc},i})$

Remark 9. *We note that if there is an implicit abort, all honest parties will have a* $\perp$ *encoded in the circuit.*

$\mathsf{Hyb}_{\mathsf{IDEAL}}$: **Run the actual probability estimation:** In this hybrid, the number of look-ahead threads is increased from $\frac{5 \cdot n \cdot \lambda}{\mu}$ to as many as needed to estimate the probability of the adversary not aborting – ε'.

Additionally, at this point, $\mathsf{Sim}_{\mathsf{Hyb}}$ doesn't re-sample the main thread $\frac{\lambda}{\mu}$ times. Instead, $\mathsf{Sim}_{\mathsf{Hyb}}$ resamples the main thread for $\min\left(2^{\lambda}, \frac{\lambda^2}{\varepsilon'}\right)$ times as in the ideal world. This hybrid corresponds exactly to the ideal world.

4.5.2.2 Indistinguishability of Hybrids

We will maintain the following invariant across the hybrids.

Definition 34 (Invariant). *Consider any malicious party P_j and any honest party P_i. $\mathsf{td}_{1,i}$ denotes the first message of the trapdoor generation protocol with P_i as the trapdoor generator. $\mathbf{T}^{j \to i}_{\mathsf{ncom}}[3]$ denotes the messages of the non-malleable commitment with P_j as the committer and P_i is the receiver.*

This event E occurs if $\exists i, j$ such that

- $\mathsf{Ext}_{\mathsf{NMCom}}$ *outputs t_i from the non-malleable commitment $\mathbf{T}^{j \to i}_{\mathsf{ncom}}[3]$ (AND)*
- $\mathsf{TDValid}(\mathsf{td}_{1,i}, \mathsf{t}_i) = 1$

That is, the event E occurs if the extractor for the non-malleable commitment outputs a valid trapdoor t_i (corresponding to the trapdoor generation protocol where P_i was the trapdoor generator) from the non-malleable commitment from player P_j to P_i.

The invariant is

$$\Pr\left[\text{Event } \mathsf{E} \text{ occurs}\right] \leq \mathsf{negl}(\lambda)$$

Claim 5. *Assuming the "1-rewinding security" of the trapdoor generation protocol TDGen and the existence of an extractor $\mathsf{Ext}_{\mathsf{NMCom}}$ for the non-malleable commitment scheme NMCom, the invariant holds in $\mathsf{Hyb}_{\mathsf{REAL}}$.*

Proof. This is proven by contradiction. Assume that the invariant doesn't hold in $\mathsf{Hyb}_{\mathsf{REAL}}$. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ causes event E to occur with non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{TDGen}}$ that breaks the "1-rewinding security" of the trapdoor generation protocol TDGen with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{TDGen}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{TDGen}}$. $\mathcal{A}_{\mathsf{TDGen}}$ picks randomly, an honest party P_i, and a random malicious party P_j. All messages other than the trapdoor messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The trapdoor messages for P_i are exposed to the external challenger. Specifically, in round 1, set $\mathsf{td}_{1,i} = \mathsf{td}_1$ where td_1 is received from $\mathcal{C}_{\mathsf{TDGen}}$. On receiving all the values $\mathsf{td}_2^{1 \to i}, \cdots, \mathsf{td}_2^{n \to i}$, including the value $\mathsf{td}_2^{j \to i}$ from $\mathcal{A}$ in round 2, $\mathcal{A}_{\mathsf{TDGen}}$ sets $\mathsf{td}_{2,i} := \left(\mathsf{td}_2^{1 \to i} || \cdots || \mathsf{td}_2^{1 \to i}\right)$ and this is the value forwarded to $\mathcal{C}_{\mathsf{TDGen}}$ as the second round response. Set $\mathsf{td}_{3,i} = \mathsf{td}_3$ where td_3 is received from $\mathcal{C}_{\mathsf{TDGen}}$, and compute the rest of the third round messages for $\mathcal{A}$. At this point, $\mathcal{A}_{\mathsf{TDGen}}$ rewinds $\mathcal{A}$ back to the beginning of round 2 to enable extraction from the NMCom. Specifically, $\mathcal{A}_{\mathsf{TDGen}}$ creates a look ahead thread that runs only the second and third round. As in the main thread, the trapdoor messages are received from $\mathcal{C}_{\mathsf{TDGen}}$. Recall that the "1-rewinding" property of the trapdoor generation protocol allows for a second td_2 query to $\mathcal{C}_{\mathsf{TDGen}}$.

Now $\mathcal{A}_{\mathsf{TDGen}}$ runs the extractor $\mathsf{Ext}_{\mathsf{NMCom}}$ of the non-malleable commitment scheme using the message in both the threads that correspond to the non-malleable commitment from malicious party P_j to honest party P_i. Let the output of $\mathsf{Ext}_{\mathsf{NMCom}}$ be t^*. $\mathcal{A}_{\mathsf{TDGen}}$ outputs t^* as a valid trapdoor to $\mathcal{C}_{\mathsf{TDGen}}$.

By our assumption, the invariant doesn't hold. Thus $\mathsf{Ext}_{\mathsf{NMCom}}$, on adversary P_{j^*}'s commitment to P_i, outputs a valid trapdoor t_{i^*} for the trapdoor generation messages of the honest party P_{i^*} with non-negligible probability ε. With probability at least $\frac{1}{n^2}$, where n is the total number of players, this corresponds to honest party P_i and malicious party P_j picked randomly by $\mathcal{A}_{\mathsf{TDGen}}$. Therefore, with non-negligible probability $\frac{\varepsilon}{n^2}$, $\mathcal{A}_{\mathsf{TDGen}}$ outputs t^* as a valid trapdoor to $\mathcal{C}_{\mathsf{TDGen}}$ which breaks the 1-rewinding security of the trapdoor generation protocol TDGen. Thus, it must be the case that the invariant holds in $\mathsf{Hyb}_{\mathsf{REAL}}$. □

Remark 10. *We note that if the invariant holds, it must be the case that no adversary can commit to a valid trapdoor with a non-negligible probability. This in turn implies accepting witness indistinguishable proofs cannot use a "trapdoor witness" other than with negligible probability.*

Claim 6. *The invariant holds in* Hyb_0.

Proof. Since there is no difference in the main thread in the first 3 rounds between $\mathsf{Hyb}_{\mathsf{REAL}}$ and Hyb_0, the invariant continues to hold. □

Claim 7. Hyb_0 *is indistinguishable from* $\mathsf{Hyb}_{\mathsf{REAL}}$ *except with probability at most* $\frac{\mu}{4} + \mathsf{negl}(\lambda)$.

Proof. This is argued in two cases depending on the probability with which the adversary abort.

Case 1: $\Pr[\text{not abort}] \geq \frac{\mu}{4}$**:**

Suppose the adversary doesn't cause an abort with probability greater that $\frac{\mu}{4}$. Let us analyze the probability with which $\bot_{\mathsf{extract}}$ is output by $\mathsf{Sim}_{\mathsf{Hyb}}$. For simplicity, we present the argument only for the case there is no abort. The argument for implicit abort is identical.

By the Chernoff bound, in Hyb_0, except with negligible probability, in the set of $\frac{5 \cdot n \cdot \lambda}{\mu}$ threads, there will be at least 5 GOOD threads with respect to some honest party P_{i^*}. Now all that's left to argue is that $\mathsf{Ext}_{\mathsf{RECom}}$ and TDExt fail to extract with negligible probability.

From the definition of RECom, algorithm $\mathsf{Ext}_{\mathsf{RECom}}$ is successful except with negligible probability if given as input $(\mathsf{recom}_1, \{\mathsf{recom}_2^k, \mathsf{recom}_3^k\}_{k \in [5]})$ such that $(\mathsf{recom}_1, \mathsf{recom}_2^k, \mathsf{recom}_3^k)$ constitute "well-formed" and "admissible" rewinding secure extractable commitment messages. "Admissibility" follows trivially since $\mathsf{Sim}_{\mathsf{Hyb}}$ picks random challenges z for the extractable commitment. From the above claim, we've proved that the invariant holds in Hyb_0, and thus from the soundness of RWI and WI, in each GOOD thread with respect to some honest party P_{i^*}, the following holds: for every malicious P_j and every honest P_i, $\mathbf{T}_{\mathsf{recom}}^{j \to i}[3]$ is a "well formed" transcript of RECom. Thus $\mathsf{Ext}_{\mathsf{RECom}}$ fails only with negligible probability.

From the definition of TDGen, algorithm TDExt is successful except with negligible probability if given as input $\left(\mathsf{td}_1, \{\mathsf{td}_2^k, \mathsf{td}_3^k\}_{k \in [3]}\right)$ where td_1 is the first message of the protocol TDGen and $\mathsf{td}_2^k, \mathsf{td}_3^k$ denote the second and third round message of the k-th execution of TDGen using the same first round message. Since there are 5 GOOD threads, we can extract every malicious party's trapdoor except with negligible probability.

Finally, from the Chernoff bound, in the set of $\frac{\lambda}{\mu}$ re-sampled main threads, there will be at least one completed execution. Thus, the adversary's view in $\mathsf{Hyb}_{\mathsf{REAL}}$ and Hyb_0 is indistinguishable.

Case 2: $\Pr[\text{not abort}] < \frac{\mu}{4}$**:**

Suppose the adversary doesn't cause an abort with probability smaller than $\frac{\mu}{4}$. Then, in both hybrids, $\mathsf{Sim}_{\mathsf{Hyb}}$ aborts at the end of the "Check Abort" step except with probability $\frac{\mu}{4}$. Thus, in this case, the adversary's view in $\mathsf{Hyb}_{\mathsf{REAL}}$ and Hyb_0 is indistinguishable except with probability at most $\frac{\mu}{4} + \mathsf{negl}(\lambda)$.

□

Remark 11. *To avoid cluttering of the proof, we will assume the argument that if both adjacent hybrids have fewer than 5* GOOD *(or 3* IMPLICIT*) look-ahead threads with respect to all parties, the two hybrids are identical.*

*Unless otherwise stated, we shall present the indistinguishability arguments for the **no abort** case since this case requires additional steps. The **implicit abort** case arguments follow identically. We will indicate and argue the two cases separately when they are different.*

Claim 8. *The invariant holds in* $\mathsf{Hyb}_{1,0}$.

Proof. Since there is no difference in the main thread in the first 3 rounds between $\mathsf{Hyb}_{1,0}$ and Hyb_0, the invariant continues to hold. □

Claim 9. *Assuming the hiding property of* OT *against malicious senders,* $\mathsf{Hyb}_{1,0}$ *is indistinguishable from* Hyb_0.

Proof. The only difference between the two hybrids is when the "Check Abort" step doesn't succeed. In that case, in Hyb_0, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses as input to OT the third round message for the RWI proof for $\mathcal{L}_b$, while in $\mathsf{Hyb}_{1,0}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses input 0 for the third round of OT. This is in fact done by a sequence of hybrids, wherein only a single instance of the honest party's input to the OT is changed. There are $< n^2$ instances where an honest party is the receiver, and thus at most n^2 intermediate hybrids. Suppose there is an adversary $\mathcal{D}$ that can distinguish between any two adjacent hybrids, we will create an adversary $\mathcal{A}_{\mathsf{OT}}$ that breaks the hiding of the OT scheme. Recall that this is only in the setting that "Check Abort" doesn't succeed and hence the fourth round messages of the honest party are not sent.

We now describe the working of $\mathcal{A}_{\mathsf{OT}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{OT}}$. Let the change in these adjacent hybrids be made for an honest party $P_{\widehat{i}}$ to a party $P_{\widehat{j}}$. All messages other than those of the chosen OT are computed as in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. First, set $\mathsf{ot}_1^{\widehat{j}\to\widehat{i}} := \mathsf{ot}_1$ where ot_1 is sent by $\mathcal{C}_{\mathsf{OT}}$. On receiving/computing[15] message $\mathsf{ot}_1^{\widehat{j}\to\widehat{i}}$, send this along with $(\mathsf{rwi}_{b,3}^{\widehat{i}\to\widehat{j}}, 0)$ to $\mathcal{C}_{\mathsf{OT}}$. Where $\mathsf{rwi}_{b,3}^{\widehat{i}\to\widehat{j}}$ is computed as in the previous hybrid by $\mathsf{Sim}_{\mathsf{Hyb}}$. $\mathcal{C}_{\mathsf{OT}}$ then chooses as input one of the two values at random and sends ot_3. $\mathcal{A}_{\mathsf{OT}}$ sets $\mathsf{ot}_3^{\widehat{j}\to\widehat{i}} := \mathsf{ot}_3$. The view generated is then given to the adversary $\mathcal{D}$, wherein depending on the choice of $\mathcal{C}_{\mathsf{OT}}$, the view corresponds to one of the two adjacent hybrids. The output from $\mathcal{D}$ is set to be the output of $\mathcal{A}_{\mathsf{OT}}$.

By our assumption, views of adjacent hybrids are distinguishable with non-negligible probability ε. Therefore, with the same probability ε $\mathcal{A}_{\mathsf{OT}}$ can break the hiding property of OT. Thus, it must be the case that ε is negligible. Since there are at most n^2 intermediate hybrids, the two end hybrids, $\mathsf{Hyb}_{1,0}$ and Hyb_0, remain indistinguishable except with negligible probability. □

Claim 10. *The invariant holds in* $\mathsf{Hyb}_{1,1}$.

[15] Since the OT sender in question may in fact be an honest party.

Proof. Since there is no difference in the main thread in the first 3 rounds between $\mathsf{Hyb}_{1,1}$ and $\mathsf{Hyb}_{1,0}$, the invariant continues to hold. $\square$

Claim 11. *Assuming the hiding property of* Ecom, $\mathsf{Hyb}_{1,1}$ *is indistinguishable from* $\mathsf{Hyb}_{1,0}$.

Proof. The proof works in the same way as the proof in the previous claim. The only difference between the two hybrids is when the "Check Abort" step doesn't succeed. In that case, in $\mathsf{Hyb}_{1,0}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses as input to Ecom the third round message for the RWI proof for $\mathcal{L}_a$, while in $\mathsf{Hyb}_{1,1}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses input 0 for the third round of Ecom. This is in fact done by a sequence of hybrids, wherein only a single instance of the honest party's input to the Ecom is changed. There are $< n^2$ instances where an honest party is the committer, and thus at most n^2 intermediate hybrids. Suppose there is an adversary $\mathcal{D}$ that can distinguish between any two adjacent hybrids, we will create an adversary $\mathcal{A}_{\mathsf{Ecom}}$ that breaks the hiding of the Ecom scheme.

We now describe the working of $\mathcal{A}_{\mathsf{Ecom}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{Ecom}}$. Let the change in these adjacent hybrids be made for an honest party $P_{\widehat{i}}$ to a party $P_{\widehat{j}}$. All messages other than those of the chosen Ecom are computed as in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. First, set $\mathsf{ecom}_1^{\widehat{i}\to\widehat{j}} := \mathsf{recom}_1$ where ecom_1 is sent by $\mathcal{C}_{\mathsf{recom}}$. On receiving/computing message $\mathsf{ecom}_1^{\widehat{i}\to\widehat{j}}$, send this along with $(\mathsf{rwi}_{a,3}^{\widehat{i}\to\widehat{j}}, 0)$ to $\mathcal{C}_{\mathsf{Ecom}}$. Where $\mathsf{rwi}_{b,3}^{i\to j}$ is computed as in the previous hybrid by $\mathsf{Sim}_{\mathsf{Hyb}}$. $\mathcal{C}_{\mathsf{Ecom}}$ then commits to one of the two values at random and sends recom_3. $\mathcal{A}_{\mathsf{Ecom}}$ sets $\mathsf{ecom}_3^{\widehat{i}\to\widehat{j}} := \mathsf{ecom}_3$. The view generated is then given to the adversary $\mathcal{D}$, wherein depending on the choice of $\mathcal{C}_{\mathsf{Ecom}}$, the view corresponds to one of the two adjacent hybrids. The output from $\mathcal{D}$ is set to be the output of $\mathcal{A}_{\mathsf{Ecom}}$.

By our assumption, views of adjacent hybrids are distinguishable with non-negligible probability ε. Therefore, with the same probability ε $\mathcal{A}_{\mathsf{Ecom}}$ can break the hiding property of Ecom. Thus, it must be the case that ε is negligible. Since there are at most n^2 intermediate hybrids, the two end hybrids, $\mathsf{Hyb}_{1,1}$ and $\mathsf{Hyb}_{1,0}$, remain indistinguishable except with negligible probability. $\square$

Claim 12. *The invariant holds in* $\mathsf{Hyb}_{1,2}$.

Proof. Since there is no difference in the main thread in the first 3 rounds between $\mathsf{Hyb}_{1,2}$ and $\mathsf{Hyb}_{1,1}$, the invariant continues to hold. $\square$

Claim 13. *Assuming the hiding property of* RECom, $\mathsf{Hyb}_{1,2}$ *is indistinguishable from* $\mathsf{Hyb}_{1,1}$.

Proof. The only difference between the two hybrids is when the "Check Abort" step doesn't succeed. In that case, in $\mathsf{Hyb}_{1,1}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses as input to RECom $(\mathsf{x}_{\widehat{i}}, \mathsf{r}_{\widehat{i}})$, while in $\mathsf{Hyb}_{1,2}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses input 0 for the third round of RECom. This is in fact done by a sequence of hybrids, wherein only a single instance of the honest party's input to the RECom is changed. There are $< n^2$ instances where an honest party is the committer, and thus at most n^2 intermediate hybrids. Suppose there is an adversary $\mathcal{D}$ that can distinguish between any two adjacent hybrids, we will create an adversary $\mathcal{A}_{\mathsf{RECom}}$ that breaks the hiding of the RECom scheme.

We now describe the working of $\mathcal{A}_{\mathsf{RECom}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{RECom}}$. Let the change in these adjacent hybrids be made for an honest party $P_{\widehat{i}}$ to a party $P_{\widehat{j}}$. All messages other than those of the chosen RECom are computed as in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. First, set $\mathsf{recom}_1^{\widehat{i}\to\widehat{j}} := \mathsf{recom}_1$ where recom_1 is sent by $\mathcal{C}_{\mathsf{recom}}$. On receiving/computing message $\mathsf{recom}_1^{\widehat{i}\to\widehat{j}}$, send this along with $\big((\mathsf{x}_{\widehat{i}}, \mathsf{r}_{\widehat{i}}), 0\big)$ to $\mathcal{C}_{\mathsf{RECom}}$. Where $(\mathsf{x}_{\widehat{i}}, \mathsf{r}_{\widehat{i}})$ is the input and randomness of $P_{\widehat{i}}$ computed as in the previous hybrid by $\mathsf{Sim}_{\mathsf{Hyb}}$. $\mathcal{C}_{\mathsf{RECom}}$ then commits to one of the two values at random and sends recom_3. $\mathcal{A}_{\mathsf{REComm}}$ sets $\mathsf{recom}_3^{\widehat{i}\to\widehat{j}} := \mathsf{recom}_3$. The view generated is then given to the adversary $\mathcal{D}$, wherein depending on the choice of $\mathcal{C}_{\mathsf{RECom}}$, the view corresponds to one of the two adjacent hybrids. The output from $\mathcal{D}$ is set to be the output of $\mathcal{A}_{\mathsf{RECom}}$.

By our assumption, views of adjacent hybrids are distinguishable with non-negligible probability ε. Therefore, with the same probability ε, $\mathcal{A}_{\mathsf{RECom}}$ can break the hiding property of RECom. Thus, it must be the case that ε is negligible. Since there are at most n^2 intermediate hybrids, the two end hybrids, $\mathsf{Hyb}_{1,2}$ and $\mathsf{Hyb}_{1,1}$, remain indistinguishable except with negligible probability. □

Claim 14. *The invariant holds in* $\mathsf{Hyb}_{1,3}$.

Proof. Since there is no difference in the main thread in the first 3 rounds between $\mathsf{Hyb}_{1,3}$ and $\mathsf{Hyb}_{1,2}$, the invariant continues to hold. □

Claim 15. *Assuming the privacy of* Π, $\mathsf{Hyb}_{1,3}$ *is indistinguishable from* $\mathsf{Hyb}_{1,2}$.

Proof. The only difference between the two hybrids is when the "Check Abort" step doesn't succeed. In that case, in $\mathsf{Hyb}_{1,2}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses as input to the third round of Π [16] $(\mathsf{x}_i, \mathsf{r}_i)$ for all honest parties P_i, while in $\mathsf{Hyb}_{1,3}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ uses as input to the third round of Π $(0, \mathsf{r}_{\widehat{i}})$ for all honest parties P_i. This is in fact done by a sequence of hybrids, wherein only a single instance of the honest party's input to the Π is changed. There are $< n$ parties, and thus at most n^2 intermediate hybrids. Suppose there is an adversary $\mathcal{D}$ that can distinguish between any two adjacent hybrids, we will create an adversary $\mathcal{A}_\Pi$ that breaks the indistinguishability of Π.

We now describe the working of $\mathcal{A}_\Pi$ which interacts with the challenger $\mathcal{C}_\Pi$. Let the change in these adjacent hybrids be made for an honest party P_i. All messages other than those of the chosen Π are computed as in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. First, set $\mathsf{msg}_{1,i} := \mathsf{msg}_1$ where msg_1 is sent by $\mathcal{C}_{\mathsf{recom}}$. On receiving and computing message $\mathsf{msg}_{1,j}$ for all other parties P_j, send this to $\mathcal{C}_\Pi$. Set $\mathsf{msg}_{2,i} := \mathsf{msg}_2$ where msg_2 is sent by $\mathcal{C}_{\mathsf{recom}}$. On receiving and computing message $\mathsf{msg}_{2,j}$ for all other parties P_j, send this to $\mathcal{C}_\Pi$ along with $((\mathsf{x}_i, \mathsf{r}_i), (0, \mathsf{r}_i))$. Where $(\mathsf{x}_i, \mathsf{r}_i)$ is the input and randomness of P_i computed as in the previous hybrid by $\mathsf{Sim}_{\mathsf{Hyb}}$. $\mathcal{C}_\Pi$ then uses one of the two values at random and sends msg_3. $\mathcal{A}_\Pi$ sets $\mathsf{msg}_{3,i} := \mathsf{msg}_3$. The view generated is then given to the adversary $\mathcal{D}$, wherein depending on the choice of $\mathcal{C}_\Pi$, the view corresponds to one of the two adjacent hybrids. The output from $\mathcal{D}$ is set to be the output of $\mathcal{A}_\Pi$.

By our assumption, views of adjacent hybrids are distinguishable with non-negligible probability ε. Therefore, with the same probability ε $\mathcal{A}_\Pi$ can break the input indistinguishability property of Π. Thus, it must be the case that ε is negligible. Since there are at most n intermediate hybrids, the two end hybrids, $\mathsf{Hyb}_{1,3}$ and $\mathsf{Hyb}_{1,2}$, remain indistinguishable except with negligible probability. □

Claim 16. *The invariant holds in* $\mathsf{Hyb}_{1,4}$.

Proof. Since there is no difference in the main thread in the first 3 rounds between $\mathsf{Hyb}_{1,4}$ and $\mathsf{Hyb}_{1,3}$, the invariant continues to hold. □

Claim 17. *Assuming the hiding property of* Ecom, $\mathsf{Hyb}_{1,4}$ *is indistinguishable from* $\mathsf{Hyb}_{1,3}$.

Proof. This proof follows identically as in Claim 11. □

Claim 18. *The invariant holds in* $\mathsf{Hyb}_{1,5}$.

[16] This is the first round of Π that uses the input.

Proof. Since there is no difference in the main thread in the first 3 rounds between $\mathsf{Hyb}_{1,5}$ and $\mathsf{Hyb}_{1,4}$, the invariant continues to hold. $\square$

Claim 19. *Assuming the hiding property of* OT *against malicious senders,* $\mathsf{Hyb}_{1,5}$ *is indistinguishable from* $\mathsf{Hyb}_{1,4}$.

Proof. This proof follows identically as in Claim 9. $\square$

Note that $\mathsf{Hyb}_{1,5} \equiv \mathsf{Hyb}_1$. This gives us that Hyb_1 and Hyb_0 are indistinguishable other than with negligible probability.

We now prove claims for all $k \in [T]$, where we set $\mathsf{Hyb}_{2,0,7} \equiv \mathsf{Hyb}_1$

We note that we will argue that the invariant holds even in the look ahead thread that we are making changes in. Initially, since all the look ahead threads are identical to the main thread, by claim 5 we know that the invariant holds in each of them. The invariant is useful since we will argue that if the invariant holds true, the probability of the extracted RWI accepting cannot change with noticeable probability. From the soundness of RWI we are guarantees that, with the change, we are still successfully extracting from the adversary with the same probability.

Claim 20. *Assuming* NMCom *is a secure non-malleable commitment scheme with non-malleability with respect to extraction, the invariant holds in* $\mathsf{Hyb}_{2,k,0}$.

Proof. We know that the invariant holds $\mathsf{Hyb}_{2,k-1,7}$. The only difference between $\mathsf{Hyb}_{2,k-1,7}$ and $\mathsf{Hyb}_{2,k,0}$ is that the simulator commits to the trapdoor in the k-th look ahead thread. Assume, for the sake of contradiction, that the invariant doesn't hold in $\mathsf{Hyb}_{2,k,0}$. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ causes event E to occur with non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{NMCom}}$ that breaks the security of the non-malleable commitment scheme NMCom with non-negligible probability. Specifically, we will break the property of non-malleability with respect to extraction.

We now describe the working of $\mathcal{A}_{\mathsf{NMCom}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{NMCom}}$. $\mathcal{A}_{\mathsf{NMCom}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen NMCom messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The NMCom messages from P_i to P_j are exposed to the external challenger. Specifically, in round 1, set $\mathsf{ncom}_1^{i \to j} := \mathsf{ncom}_1^L$ where ncom_1^L is received from $\mathcal{C}_{\mathsf{NMCom}}$ for the left execution. On receiving $\mathsf{ncom}_1^{j \to i}$ from $\mathcal{A}$, $\mathcal{A}_{\mathsf{NMCom}}$ forwards this to $\mathcal{C}_{\mathsf{NMCom}}$ as its first round message on the right hand side.

$\mathcal{A}_{\mathsf{NMCom}}$ creates a set of 5 look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. In each look-ahead thread, $\mathcal{A}_{\mathsf{NMCom}}$ computes $\mathsf{ncom}_3^{i \to j}$ as a commitment to $\perp$. From the definition of the NMCom scheme, from the pseudorandomness property, $\mathcal{A}_{\mathsf{NMCom}}$ can do this even without knowing the randomness used to generate $\mathsf{ncom}_1^{i \to j}$.[17] These 5 threads are all GOOD with respect to some party H with noticeable probability. With the 5 threads, $\mathcal{A}_{\mathsf{NMCom}}$ can successfully run the input and trapdoor extraction phase.

On the k-th thread $\mathcal{A}_{\mathsf{NMCom}}$ receives ncom_2^R from $\mathcal{C}_{\mathsf{NMCom}}$ as the second round message on the right side which it sets as the value $\mathsf{ncom}_2^{j \to i}$. On receiving $\mathsf{ncom}_2^{i \to j}$ in the k-th thread, $\mathcal{A}_{\mathsf{NMCom}}$ sends this to $\mathcal{C}_{\mathsf{NMCom}}$ as its second round message on the left side along with the pair of values $(\widetilde{\mathsf{r}}, \mathsf{t}_j)$ where t_j was obtained during the extraction phase, and $\widetilde{\mathsf{r}}$ is a random value.

[17] While in the real execution, these are to random value (instead of $\perp$) by the hiding property these are indistinguishable.

$\mathcal{A}_{\mathsf{NMCom}}$ receives a third round message ncom_3^L which is either a commitment to $\perp$ or t_j. This is sent to $\mathcal{A}$ as the value $\mathsf{ncom}_3^{i\to j}$ in the k-th thread.

We note that $\mathcal{A}_{\mathsf{NMCom}}$ acts as an interface for the $\mathsf{Ext}_{\mathsf{NMCom}}$, rewinding $\mathcal{A}$ as necessary.

By our assumption, the invariant doesn't hold. Thus $\mathsf{Ext}_{\mathsf{NMCom}}$, on adversary P_{j^*}'s commitment to P_i, outputs a valid trapdoor t_{i^*} for the trapdoor generation messages of the honest party P_{i^*} with non-negligible probability ε. With probability at least $\frac{1}{n^2}$, where n is the total number of players, this corresponds to honest party P_i and malicious party P_j picked randomly by $\mathcal{A}_{\mathsf{NMCom}}$. Therefore, with non-negligible probability $\frac{\varepsilon}{n^2}$, $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^* as a valid trapdoor. Since the invariant holds in $\mathsf{Hyb}_{2,k-1,7}$, if $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^*, it must be the case that we are in $\mathsf{Hyb}_{2,k,0}$ with non-negligible probability. That is, when $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs a valid trapdoor, it must correspond to $\mathcal{A}_{\mathsf{NMCom}}$ receiving a commitment to 0. This breaks the security of NMCom, which is a contradiction. Thus the invariant must also hold for $\mathsf{Hyb}_{2,k,0}$.

□

Claim 21. *Assuming hiding of* NMCom*,* $\mathsf{Hyb}_{2,k-1,7}$ *is indistinguishable from* $\mathsf{Hyb}_{2,k,0}$

Proof. Since we are only making changes in a look-ahead thread, all we need to do is argue that the extraction continues to succeed. i.e. $\mathsf{Sim}_{\mathsf{Hyb}}$ does not output $\perp_{\mathsf{extract}}$ in the extraction phase of one hybrid but not the other. The only difference between $\mathsf{Hyb}_{2,k-1,7}$ and $\mathsf{Hyb}_{2,k,0}$ is that the simulator commits to the trapdoor in the k-th look ahead thread.

Since we have already established that the invariant holds in each look-ahead thread independently, we want to use the fact that the probability that the RWI proof for $\mathcal{L}_a$ is accepting cannot change with non-negligible probability if the invariant is true. If this were the case, the probability $\mathsf{Sim}_{\mathsf{Hyb}}$ outputs $\perp_{\mathsf{extract}}$ will not change in the extraction phase of the two hybrids, since $\mathcal{L}_a$ proves honest behavior of the first 3 rounds of the protocol.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ commits RWI proofs for $\mathcal{L}_a$ in Ecom such that the probability of accept in the two cases differs by a non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{NMCom}}$ that breaks the hiding of the non-malleable commitment scheme NMCom with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{NMCom}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{NMCom}}$. $\mathcal{A}_{\mathsf{NMCom}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen NMCom messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The NMCom messages from P_i to P_j are exposed to the external challenger. Specifically, in round 1, set $\mathsf{ncom}_1^{i\to j} := \mathsf{ncom}_1$ where ncom_1 is received from $\mathcal{C}_{\mathsf{NMCom}}$.

$\mathcal{A}_{\mathsf{NMCom}}$ creates a set of 5 look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. In each look-ahead thread, $\mathcal{A}_{\mathsf{NMCom}}$ computes $\mathsf{ncom}_3^{i\to j}$ as a commitment to $\perp$. From the definition of the NMCom scheme, $\mathcal{A}_{\mathsf{NMCom}}$ can do this even without knowing the randomness used to generate $\mathsf{ncom}_1^{i\to j}$. These 5 threads are all GOOD with respect to some party H with noticeable probability. With the 5 threads, $\mathcal{A}_{\mathsf{NMCom}}$ can successfully run the input and trapdoor extraction phase.

On receiving $\mathsf{ncom}_1^{i\to j}$, $\mathcal{A}_{\mathsf{NMCom}}$ forwards it to $\mathcal{C}_{\mathsf{NMCom}}$ along with pair of values $(\widetilde{\mathsf{r}}, \mathsf{t}_j)$ where t_j was obtained during the extraction phase, and $\widetilde{\mathsf{r}}$ is a random value.

$\mathcal{A}_{\mathsf{NMCom}}$ receives a third round message ncom_3^L which is either a commitment to $\widetilde{\mathsf{r}}$ or t_j. This is sent to $\mathcal{A}$ as the value $\mathsf{ncom}_3^{i\to j}$ on the k-th thread. On receiving the third round messages from $\mathcal{A}$, from 2 GOOD look ahead threads with respect to P_i, extract $\mathsf{rwi}_{a,3}^{j\to i}$ from Ecom[18]. From the definition of Ecom, the extracted value can be verified to be correctly extracted. $\mathcal{A}_{\mathsf{NMCom}}$ now checks if

$$\mathsf{RWI}_4\left(\mathsf{st}_a^{j\to i}, \mathbf{T}_{\mathsf{rwi}_a}^{j\to i}[3]; \mathsf{r}_{\mathsf{rwi},a}^{j\to i}\right) = 1.$$

If so, it guesses that the commitment was to $\widetilde{\mathsf{r}}$. Otherwise, it guesses that the commitment was to t_j. Let us define Trap as the event that the commitment was to the trapdoor and $\overline{\mathsf{Trap}}$ as the even that the commitment was to $\perp$. From the challenge game, we know $\Pr[\mathsf{Trap}] = \Pr\left[\overline{\mathsf{Trap}}\right] = \frac{1}{2}$

$$\begin{aligned}\Pr[\text{guess correct}] &= \Pr\left[\text{guess correct} \;\middle|\; \mathsf{Trap}\right] \cdot \Pr[\mathsf{Trap}] + \Pr\left[\text{guess correct} \;\middle|\; \overline{\mathsf{Trap}}\right] \cdot \Pr\left[\overline{\mathsf{Trap}}\right] \\ &= \Pr\left[\text{guess correct} \;\middle|\; \mathsf{Trap}\right] \cdot \frac{1}{2} + \Pr\left[\text{guess correct} \;\middle|\; \overline{\mathsf{Trap}}\right] \cdot \frac{1}{2} \\ &= \frac{1}{2} \cdot \left(\Pr\left[\mathsf{RWI\ proof\ accepts} \;\middle|\; \mathsf{Trap}\right] + \Pr\left[\mathsf{RWI\ proof\ rejects} \;\middle|\; \overline{\mathsf{Trap}}\right]\right) \\ &= \frac{1}{2} \cdot \left(\Pr\left[\mathsf{RWI\ proof\ accepts} \;\middle|\; \mathsf{Trap}\right] + 1 - \Pr\left[\mathsf{RWI\ proof\ accepts} \;\middle|\; \overline{\mathsf{Trap}}\right]\right) \\ &= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr\left[\mathsf{RWI\ proof\ accepts} \;\middle|\; \mathsf{Trap}\right] - \Pr\left[\mathsf{RWI\ proof\ accepts} \;\middle|\; \overline{\mathsf{Trap}}\right]\right)\end{aligned}$$

By our assumption, the adversary P_{j^*}'s acceptance probability of the RWI proof for $\mathcal{L}_a$ to P_{i^*} differs non-negligible probability ε. With probability at least $\frac{1}{n^2}$, where n is the total number of players, this corresponds to honest party P_i and malicious party P_j picked randomly by $\mathcal{A}_{\mathsf{NMCom}}$. Therefore, P_j's acceptance probability of the RWI proof for $\mathcal{L}_a$ to P_i differs non-negligible probability $\frac{\varepsilon}{n^2}$. Now, the extractor $\mathsf{Ext}_{\mathsf{Ecom}}$ is successful with some non-negligible probability ε'. Therefore, with non-negligible advantage $\frac{\varepsilon\cdot\varepsilon'}{2\cdot n^2}$, $\mathcal{A}_{\mathsf{NMCom}}$ wins the challenge game with $\mathcal{C}_{\mathsf{NMCom}}$ which breaks the hiding property of NMCom. Thus, ε must be negligible, and thus the views are indistinguishable. □

Claim 22. *Assuming Assuming that* RWI *is a bounded rewinding secure protocol, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{2,k,1}$.

Proof. We know that the invariant holds $\mathsf{Hyb}_{2,k,0}$. The only difference between $\mathsf{Hyb}_{2,k,0}$ and $\mathsf{Hyb}_{2,k,1}$ is that the simulator switches the witness in the RWI for $\mathcal{L}_b$. Assume, for the sake of contradiction, that the invariant doesn't hold in $\mathsf{Hyb}_{2,k,1}$. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ causes event E to occur with non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{RWI}}$ that breaks the bounded rewinding security of RWI with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{RWI}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{RWI}}$. $\mathcal{A}_{\mathsf{RWI}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen RWI messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The RWI messages from P_i to P_j are exposed to the external challenger. Specifically, in round 1, set $\mathsf{rwi}_{b,1}^{i\to j} := \mathsf{rwi}_1$ where rwi_1 is received from $\mathcal{C}_{\mathsf{RWI}}$.

After receiving $\mathsf{rwi}_{b,2}^{i\to j}$ from $\mathcal{A}$, $\mathcal{A}_{\mathsf{RWI}}$ creates a set of 5 look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. In each look-ahead thread, $\mathcal{A}_{\mathsf{RWI}}$ on receiving $\mathsf{rwi}_{b,1}^{i\to j}$

[18]The extraction in fact does not require further rewinds since mask already extracted in the "Check Abort" phase. But for simplicity, we ignore this point for now.

forwards it to $\mathcal{C}_{\mathsf{RWI}}$ as its second round message. For each thread, $\mathcal{A}_{\mathsf{RWI}}$ also sends the statement

$$\mathsf{st}_b^{i\to j} := (\mathbf{T}_{\mathsf{rwi}_a}^{i\to\bullet}[2], \mathsf{st}_a^{i\to\bullet}, \mathbf{T}_{\mathsf{ecom}}^{i\to\bullet}[3], \mathbf{T}_{\mathsf{ncom}}^{i\to j}[3], \mathsf{td}_{1,j})$$

where the other values are generated as in $\mathsf{Hyb}_{2,k,0}$.

In the main thread, $\mathcal{A}_{\mathsf{RWI}}$ also sends the pair of witnesses $(\mathsf{r}_{\mathsf{rwi}_a}^{i\to\bullet}, \mathsf{w}_a^{i\to\bullet}, \mathsf{rwi}_{a,3}^{i\to\bullet}, \mathsf{r}_{\mathsf{ecom}}^{i\to\bullet})$ and $(\mathsf{t}_j, \mathsf{r}_{\mathsf{ncom}}^{i\to j})$ where t_j is obtained in the input extraction phase, and $\mathsf{w}_a^{i\to k}$ is computed as defined. For each thread, $\mathcal{A}_{\mathsf{RWI}}$ receives rwi_3 which is set as $\mathsf{rwi}_{b,3}^{i\to j}$.

Recall that RWI is secure even in the presence of 6 total threads. Now $\mathcal{A}_{\mathsf{RWI}}$ runs the extractor $\mathsf{Ext}_{\mathsf{NMCom}}$ of the non-malleable commitment scheme using the message in both the threads that correspond to the non-malleable commitment from malicious party P_j to honest party P_i. Let the output of $\mathsf{Ext}_{\mathsf{NMCom}}$ be t^*. $\mathcal{A}_{\mathsf{RWI}}$ checks if $\mathsf{TDValid}(t^*, \mathsf{td}_{1,i}) = 1$. If so, it outputs 1 to indicate $\mathsf{Hyb}_{2,k,1}$ and 0 otherwise. Let us denote this output by $\widetilde{b}$, and let the challenge bit be b. Then,

$$\begin{aligned}
\Pr\left[\widetilde{b} = b\right] &= \Pr\left[\widetilde{b} = 0 \;\middle|\; b = 0\right] \cdot \Pr[b = 0] + \Pr\left[\widetilde{b} = 1 \;\middle|\; b = 1\right] \cdot \Pr[b = 1] \\
&= \Pr\left[\widetilde{b} = 0 \;\middle|\; b = 0\right] \cdot \frac{1}{2} + \Pr\left[\widetilde{b} = 1 \;\middle|\; b = 1\right] \cdot \frac{1}{2} \\
&= \frac{1}{2} \cdot \left(1 - \Pr\left[\widetilde{b} = 1 \;\middle|\; b = 0\right] + \Pr\left[\widetilde{b} = 1 \;\middle|\; b = 1\right]\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr\left[\widetilde{b} = 1 \;\middle|\; b = 1\right] - \Pr\left[\widetilde{b} = 1 \;\middle|\; b = 0\right]\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr\left[\mathsf{EXT} \;\middle|\; b = 1\right] - \Pr\left[\mathsf{EXT} \;\middle|\; b = 0\right]\right)
\end{aligned}$$

where EXT denotes the even that the extractor outputs a valid trapdoor. By our assumption, the invariant doesn't hold. Thus $\mathsf{Ext}_{\mathsf{NMCom}}$, on adversary P_{j^*}'s commitment to P_i, outputs a valid trapdoor t_{i^*} for the trapdoor generation messages of the honest party P_{i^*} with non-negligible probability ε. With probability at least $\frac{1}{n^2}$, where n is the total number of players, this corresponds to honest party P_i and malicious party P_j picked randomly by $\mathcal{A}_{\mathsf{RWI}}$. Therefore, with non-negligible probability $\frac{\varepsilon}{n^2}$, $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^* as a valid trapdoor. Since the invariant holds in $\mathsf{Hyb}_{2,k,0}$, if $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^*, it must be the case that we are in $\mathsf{Hyb}_{2,k,1}$ with non-negligible probability. That is, when $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs a valid trapdoor, it must correspond to $\mathcal{A}_{\mathsf{RWI}}$ receiving a proof using the trapdoor witness. This breaks the security of RWI, which is a contradiction. Thus the invariant must also hold for $\mathsf{Hyb}_{2,k,1}$. □

Claim 23. *Assuming the bounded rewinding witness indistinguishability* RWI*,* $\mathsf{Hyb}_{2,k,0}$ *is indistinguishable from* $\mathsf{Hyb}_{2,k,1}$

Proof. Since we're only making changes in a look-ahead thread, all we need to do is argue that the extraction continues to succeed. i.e. $\mathsf{Sim}_{\mathsf{Hyb}}$ does not output $\bot_{\mathsf{extract}}$ in the extraction phase of one hybrid and not the other. The only difference between $\mathsf{Hyb}_{2,k,0}$ and $\mathsf{Hyb}_{2,k,1}$ is that the simulator switches the witness in the RWI for $\mathcal{L}_b$.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ commits RWI proofs for $\mathcal{L}_b$ in Ecom such that the probability of accept in the two cases in non-negligible. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{RWI}}$ that breaks the bounded rewinding security of RWI with non-negligible probability.

The proof is similar to that of Claim 21 and Claim 22. We note that we use the fact that RWI is secure even in the presence of the 2 total threads used for extracting from Ecom. $\square$

Claim 24. *Assuming Assuming that* RWI *is a bounded rewinding secure protocol, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{2,k,2}$.

Proof. We know that the invariant holds $\mathsf{Hyb}_{2,k,1}$. The only difference between $\mathsf{Hyb}_{2,k,1}$ and $\mathsf{Hyb}_{2,k,2}$ is that the simulator switches the witness in the RWI for $\mathcal{L}_a$. Assume, for the sake of contradiction, that the invariant doesn't hold in $\mathsf{Hyb}_{2,k,2}$. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ causes event E to occur with non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{RWI}}$ that breaks the bounded rewinding security of RWI with non-negligible probability. The rest of the proof is similar to that of Claim 22. $\square$

Claim 25. *Assuming the bounded rewinding witness indistinguishability* RWI*,* $\mathsf{Hyb}_{2,k,1}$ *is indistinguishable from* $\mathsf{Hyb}_{2,k,2}$

Proof. Since we're only making changes in a look-ahead thread, all we need to do is argue that the extraction continues to succeed. i.e. $\mathsf{Sim}_{\mathsf{Hyb}}$ does not output $\perp_{\mathsf{extract}}$ in the extraction phase of one hybrid and not the other. The only difference between $\mathsf{Hyb}_{2,k,1}$ and $\mathsf{Hyb}_{2,k,2}$ is that the simulator switches the witness in the RWI for $\mathcal{L}_a$.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ commits RWI proofs for $\mathcal{L}_a$ in Ecom such that the probability of accept in the two cases in non-negligible. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{RWI}}$ that breaks the bounded rewinding security of RWI with non-negligible probability.

The proof is similar to that of Claim 21 and Claim 22. $\square$

Claim 26. *Assuming that* Com *is a secure commitment scheme, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{2,k,3}$.

Proof. We prove this by a sequence of sub-claims.

Sub-Claim 27. *Assuming that* Com *is a secure commitment scheme, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{2,k,3,0}$.

Proof. We know that the invariant holds $\mathsf{Hyb}_{2,k,2}$. The only difference between $\mathsf{Hyb}_{2,k,2}$ and $\mathsf{Hyb}_{2,k,3,0}$ is that the simulator switches the commitment in Com from polynomials p to 0. This is in fact done by a sequence of hybrids where only a single Com is changed at a time. For simplicity, we proceed with the assumption that in this hybrid, only a single commitment was changed. Assume, for the sake of contradiction, that the invariant doesn't hold in $\mathsf{Hyb}_{2,k,3}$. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ causes event E to occur with non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{Com}}$ that breaks the hiding property of Com with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{Com}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{Ecom}}$. $\mathcal{A}_{\mathsf{Com}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen Com messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The Com messages from P_i to P_j are exposed to the external challenger. Specifically, $\mathcal{A}_{\mathsf{Com}}$ sends two challenges

$(\mathsf{p}_\ell, 0)$ to $\mathcal{C}$. And sets $\mathsf{recom}_{1,\ell}^{i\to j} := \mathsf{Com}$ where Com is received from $\mathcal{C}_{\mathsf{Com}}$. Depending on the challenge used by $\mathcal{C}_{\mathsf{Com}}$, we are either in $\mathsf{Hyb}_{2,k,2}$ or $\mathsf{Hyb}_{2,k,3,0}$.

$\mathcal{A}_{\mathsf{Com}}$ creates sufficiently many look ahead threads where it runs rounds 2 and 3 of the protocol alone. Now $\mathcal{A}_{\mathsf{Com}}$ runs the extractor $\mathsf{Ext}_{\mathsf{NMCom}}$ of the non-malleable commitment scheme using the message in both the threads that correspond to the non-malleable commitment from malicious party P_j to honest party P_i. Let the output of $\mathsf{Ext}_{\mathsf{NMCom}}$ be t^*. $\mathcal{A}_{\mathsf{Com}}$ checks if $\mathsf{TDValid}(t^*, \mathsf{td}_{1,i}) = 1$. If so, it outputs 1 to indicate $\mathsf{Hyb}_{2,k,3}$ and 0 otherwise.

By our assumption, the invariant doesn't hold. Thus $\mathsf{Ext}_{\mathsf{NMCom}}$, on adversary P_{j^*}'s commitment to P_i, outputs a valid trapdoor t_{i^*} for the trapdoor generation messages of the honest party P_{i^*} with non-negligible probability ε. With probability at least $\frac{1}{n^2}$, where n is the total number of players, this corresponds to honest party P_i and malicious party P_j picked randomly by $\mathcal{A}_{\mathsf{Com}}$. Therefore, with non-negligible probability $\frac{\varepsilon}{n^2}$, $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^* as a valid trapdoor. Since the invariant holds in $\mathsf{Hyb}_{2,k,2}$, if $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^*, it must be the case that we're in $\mathsf{Hyb}_{2,k,3}$ with non-negligible probability. That is, when $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs a valid trapdoor, it must correspond to $\mathcal{A}_{\mathsf{Com}}$ receiving input 0. This breaks the security of Com, which is a contradiction. Thus the invariant must also hold for $\mathsf{Hyb}_{2,k,3}$.

□

Sub-Claim 28. *The invariant holds in* $\mathsf{Hyb}_{2,k,3,1}$.

Proof. The change from is statistical $\mathsf{Hyb}_{2,k,3,0}$ when there are fewer than $\mathsf{B}_{\mathsf{recom}}$ rewinds when extracting from the NMCom. This follows from the fact that the degree of the polynomial is set to be $\mathsf{B}_{\mathsf{recom}}$, and thus statistically undetermined by the number of rewinds $\leq \mathsf{B}_{\mathsf{recom}}$. By our setting of parameters, we know that number of rewinds $\leq \mathsf{B}_{\mathsf{recom}}$. Thus The invariant holds in $\mathsf{Hyb}_{2,k,3,1}$. □

Sub-Claim 29. *Assuming that* Com *is a secure commitment scheme, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{2,k,3,2}$.

Proof. The proof follows identically as in Sub-Claim 27. □

Thus we have that the invariant holds for $\mathsf{Hyb}_{2,k,3}$.

□

Claim 30. *Assuming that* Com *is a secure commitment scheme,* $\mathsf{Hyb}_{2,k,2}$ *is indistinguishable from* $\mathsf{Hyb}_{2,k,3}$

Proof. Since we're only making changes in a look-ahead thread, all we need to do is argue that the extraction continues to succeed. i.e. $\mathsf{Sim}_{\mathsf{Hyb}}$ does not output $\perp_{\mathsf{extract}}$ in the extraction phase of one hybrid and not the other. The only difference between $\mathsf{Hyb}_{2,k,2}$ and $\mathsf{Hyb}_{2,k,3}$ is that the simulator switches commitment in RECom to 0.

The proof is similar to that of Claim 21 and Claim 26. □

Claim 31. *Assuming that* Π *is a rewinding secure protocol for the first three rounds, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{2,k,4}$.

Proof. We know that the invariant holds $\mathsf{Hyb}_{2,k,3}$. The only difference between $\mathsf{Hyb}_{2,k,3}$ and $\mathsf{Hyb}_{2,k,4}$ is that the simulator switches the input in Π from (x, r) to 0 for each honest party $P_{\widehat{i}}$. This is in fact done by a sequence of hybrids where only a single party's input is changed at a time. For simplicity, we proceed with the assumption that in this hybrid, only a single party's input was changed. Assume, for the sake of contradiction, that the invariant doesn't hold in $\mathsf{Hyb}_{2,k,4}$. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ causes event E to occur with non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_\Pi$ that breaks the bounded rewinding security of the first three round of Π with non-negligible probability.

We now describe the working of $\mathcal{A}_\Pi$ which interacts with the challenger $\mathcal{C}_\Pi$. $\mathcal{A}_\Pi$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen Π messages for $P_{\widehat{i}}$ are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The Π messages for $P_{\widehat{i}}$ are exposed to the external challenger. Specifically, in round 1, set $\mathsf{msg}_{1,\widehat{i}} := \mathsf{msg}_1$ where msg_1 is received from $\mathcal{C}_\Pi$.

After generating/receiving $\mathbf{T}_\Pi[1]$ from $\mathcal{A}$, $\mathcal{A}_\Pi$ creates a set of 5 look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. In each look-ahead thread, $\mathcal{A}_\Pi$ on computing $\mathbf{T}_\Pi[2]$ forwards it to $\mathcal{C}_\Pi$.

In the main thread (k-th look-ahead thread), $\mathcal{A}_\Pi$ also sends the pair of inputs $(\mathsf{x}_i, \mathsf{r}_i)$ and 0. For the look-ahead threads for extraction, $\mathcal{A}_\Pi$ sends the input $(\mathsf{x}_i, \mathsf{r}_i)$. For each thread, $\mathcal{A}_\Pi$ receives msg_3 which is set as $\mathsf{msg}_{3,\widehat{i}}$. Depending on the input used by $\mathcal{C}_\Pi$, we are either in $\mathsf{Hyb}_{2,k,3}$ or $\mathsf{Hyb}_{2,k,4}$.

Recall that Π is secure even in the presence of 3 total threads. Now $\mathcal{A}_\Pi$ runs the extractor $\mathsf{Ext}_{\mathsf{NMCom}}$ of the non-malleable commitment scheme using the message in both the threads that correspond to the non-malleable commitment from malicious party P_j to honest party P_i. Let the output of $\mathsf{Ext}_{\mathsf{NMCom}}$ be t^*. $\mathcal{A}_\Pi$ checks if $\mathsf{TDValid}(t^*, \mathsf{td}_{1,i}) = 1$. If so, it outputs 1 to indicate $\mathsf{Hyb}_{2,k,4}$ and 0 otherwise.

By our assumption, the invariant doesn't hold. Thus $\mathsf{Ext}_{\mathsf{NMCom}}$, on adversary P_{j^*}'s commitment to P_i, outputs a valid trapdoor t_{i^*} for the trapdoor generation messages of the honest party P_{i^*} with non-negligible probability ε. With probability at least $\frac{1}{n^2}$, where n is the total number of players, this corresponds to honest party P_i and malicious party P_j picked randomly by $\mathcal{A}_\Pi$. Therefore, with non-negligible probability $\frac{\varepsilon}{n^2}$, $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^* as a valid trapdoor. Since the invariant holds in $\mathsf{Hyb}_{2,k,3}$, if $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^*, it must be the case that we're in $\mathsf{Hyb}_{2,k,4}$ with non-negligible probability. That is, when $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs a valid trapdoor, it must correspond to $\mathcal{A}_\Pi$ receiving the messages using input 0. This breaks the security of Π, which is a contradiction. Thus the invariant must also hold for $\mathsf{Hyb}_{2,k,4}$.

□

Claim 32. *Assuming that Π is a rewinding secure protocol for the first three rounds, $\mathsf{Hyb}_{2,k,3}$ is indistinguishable from $\mathsf{Hyb}_{2,k,4}$*

Proof. Since we're only making changes in a look-ahead thread, all we need to do is argue that the extraction continues to succeed. i.e. $\mathsf{Sim}_{\mathsf{Hyb}}$ does not output $\perp_{\mathsf{extract}}$ in the extraction phase of one hybrid and not the other. The only difference between $\mathsf{Hyb}_{2,k,3}$ and $\mathsf{Hyb}_{2,k,4}$ is that the simulator switches input to 0.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ commits RWI proofs for $\mathcal{L}_a$ in Ecom such that the probability of accept in the two cases in non-negligible. We will use this adversary to create an adversary $\mathcal{A}_\Pi$ that breaks the bounded rewinding security of Π with non-negligible probability.

The proof is similar to that of Claim 21 and Claim 31. □

Claim 33. *Assuming Assuming that RWI is a bounded rewinding secure protocol, and the existence of an extractor $\mathsf{Ext}_{\mathsf{NMCom}}$, the invariant holds in $\mathsf{Hyb}_{2,k,5}$.*

Proof. Proof is identical to that of Claim 24. □

Claim 34. *Assuming the bounded rewinding witness indistinguishability* RWI*,* $\mathsf{Hyb}_{2,k,4}$ *is indistinguishable from* $\mathsf{Hyb}_{2,k,5}$

Proof. Proof is identical to that of Claim 25. □

Claim 35. *Assuming Assuming that* RWI *is a bounded rewinding secure protocol, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{2,k,6}$.

Proof. Proof is identical to that of Claim 22. □

Claim 36. *Assuming the bounded rewinding witness indistinguishability* RWI*,* $\mathsf{Hyb}_{2,k,5}$ *is indistinguishable from* $\mathsf{Hyb}_{2,k,6}$

Proof. Proof is identical to that of Claim 23. □

Claim 37. *Assuming* NMCom *is a secure non-malleable commitment scheme with respect to extraction, the invariant holds in* $\mathsf{Hyb}_{2,k,7}$.

Proof. Proof is identical to that of Claim 20. □

Claim 38. *Assuming* NMCom *is a secure non-malleable commitment scheme,* $\mathsf{Hyb}_{2,k,6}$ *is indistinguishable from* $\mathsf{Hyb}_{2,k,7}$

Proof. Proof is identical to that of Claim 21. □

Claim 39. *Assuming* NMCom *is a secure non-malleable commitment scheme with respect to extraction, the invariant holds in* Hyb_3.

Proof. Proof is identical to that of Claim 20. □

Claim 40. *Assuming* NMCom *is a secure non-malleable commitment scheme,* Hyb_3 *is indistinguishable from* Hyb_2

Proof. The only difference between Hyb_3 and Hyb_2 is that the simulator commits to the trapdoor in the main look ahead thread.

Assume, for the sake of contradiction, that that the views are distinguishable. Then there exists an adversary $\mathcal{D}$ such that $\mathcal{D}$ can distinguish between Hyb_3 and Hyb_2 with non-negligible advantage. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{NMCom}}$ that breaks the hiding of the non-malleable commitment scheme NMCom with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{NMCom}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{NMCom}}$. $\mathcal{A}_{\mathsf{NMCom}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen NMCom messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The NMCom messages from P_i to P_j are exposed to the external challenger. Specifically, in round 1, set $\mathsf{ncom}_1^{i\to j} := \mathsf{ncom}_1$ where ncom_1 is received from $\mathcal{C}_{\mathsf{NMCom}}$.

$\mathcal{A}_{\mathsf{NMCom}}$ creates a set of 5 look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. In each look-ahead thread, $\mathcal{A}_{\mathsf{NMCom}}$ computes $\mathsf{ncom}_3^{i\to j}$ as a commitment to $\perp$. From the definition of the NMCom scheme, $\mathcal{A}_{\mathsf{NMCom}}$ can do this even without knowing the randomness used to generate $\mathsf{ncom}_1^{i\to j}$. These 5 threads are all GOOD with respect to some party H with noticeable probability. With the 5 threads, $\mathcal{A}_{\mathsf{NMCom}}$ can successfully run the input and trapdoor extraction phase.

On receiving $\mathsf{ncom}_1^{i\to j}$, $\mathcal{A}_{\mathsf{NMCom}}$ forwards it to $\mathcal{C}_{\mathsf{NMCom}}$ along with pair of values $(\widetilde{\mathsf{r}}, \mathsf{t}_j)$ where t_j was obtained during the extraction phase, and $\widetilde{\mathsf{r}}$ is a random value.

$\mathcal{A}_{\mathsf{NMCom}}$ receives a third round message ncom_3^L which is either a commitment to $\perp$ or t_j. This is sent to $\mathcal{A}$ as the value $\mathsf{ncom}_3^{i\to j}$ on the main thread. The rest of the messages are obtained in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. Depending on which value was committed we are either in Hyb_3 or Hyb_2. On completion of the execution, the view is input to $\mathcal{D}$ and the output returned is the output of $\mathcal{A}_{\mathsf{NMCom}}$

By our assumption, $\mathcal{D}$ can distinguish between the two hybrids with noticeable probability ε. Therefore, with non-negligible advantage $\frac{\varepsilon}{n^2}$, $\mathcal{A}_{\mathsf{NMCom}}$ wins the challenge game with $\mathcal{C}_{\mathsf{NMCom}}$ which breaks the hiding property of NMCom. Thus, ε must be negligible, and thus the views are indistinguishable. □

Claim 41. *Assuming the bounded rewinding witness indistinguishability* RWI*, the invariant holds in* Hyb_4.

Proof. Proof is identical to that of Claim 22. □

Claim 42. *Assuming the bounded rewinding witness indistinguishability* RWI*,* Hyb_4 *is indistinguishable from* Hyb_3

Proof. The only difference between Hyb_4 and Hyb_3 is that the simulator switches the witness in the RWI for $\mathcal{L}_b$.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{D}$ can distinguish between Hyb_4 and Hyb_3 with non-negligible advantage. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{RWI}}$ that breaks the bounded rewinding security of RWI with non-negligible probability.

The proof is similar to that of Claim 40 and Claim 41. □

Claim 43. *Assuming the bounded rewinding witness indistinguishability* RWI*, the invariant holds in* Hyb_5.

Proof. Proof is identical to that of Claim 24. □

Claim 44. *Assuming the bounded rewinding witness indistinguishability* RWI*,* Hyb_5 *is indistinguishable from* Hyb_4

Proof. The only difference between Hyb_5 and Hyb_4 is that the simulator switches the witness in the RWI for $\mathcal{L}_a$.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{D}$ can distinguish between Hyb_5 and Hyb_4 with non-negligible advantage. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{RWI}}$ that breaks the bounded rewinding security of RWI with non-negligible probability.

The proof is similar to that of Claim 40 and Claim 41. □

Claim 45. *The invariant holds in* Hyb_6.

Proof. The claim is trivially true since the change is made only in the fourth round. □

Claim 46. *Assuming the witness indistinguishability* WI*,* Hyb_6 *is indistinguishable from* Hyb_5

Proof. The only difference between Hyb_6 and Hyb_5 is that the simulator switches the witness in the WI for $\mathcal{L}_c$.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{D}$ can distinguish between Hyb_6 and Hyb_5 with non-negligible advantage. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{WI}}$ that breaks the witness indistinguishability of WI with non-negligible probability.

The proof is similar to that of Claim 40 and Claim 41. We point out that since only the second round of WI overlaps with the rewinding rounds, we don't need the external challenger to handle rewinds since the responses on the look-ahead threads, that are run only till the end of third round, are discarded. □

Claim 47. *Assuming the rewinding security of* REComm*,* Hyb_7 *is indistinguishable from* Hyb_6

Proof. This is proved via a sequence of hybrids given below.

This is done by a sequence of hybrids mentioned below. We note that we separate the look-ahead threads into two separate types: (i) to extract trapdoor, (ii) to extract input. In our hybrids, we shall only make changes to type (ii) threads.

Roughly, we first argue that if we switch to committing to "junk" in the RECom in any thread, the invariant continues to hold in that thread. For this, we will rely on the bounded rewind security of RECom. This ensures that on such threads, the adversary's input does not also switch to "junk" on these threads. We then use these threads as look-ahead threads to extract the adversary's inputs. These threads can be completed without having to forward messages to an external challenger since we can respond to committing to "junk", which can be done without knowledge of the randomness for that instance of RECom. This avoids a potential circularity with regards to extracting from the RECom while maintaining the hiding property of honestly evaluated RECom.

$\mathsf{Hyb}_{7,0}$: **Change main thread** RECom **to random:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the main thread to send "junk" responses. Specifically, for every honest party P_i and malicious party P_j do the following:

- for every $\ell \in [N]$, pick a new degree 4 polynomial q_ℓ.
- compute $\mathsf{recom}_{3,\ell}$ as $(0 \oplus \mathsf{q}_\ell(0), \mathsf{q}_\ell(\mathsf{z}_\ell))$.

Given that we changed our RECom to random, we want to claim that the adversary's input has not also become random.

Claim 48. *Assuming the security of* Com *and the existence of* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{7,0}$.

Proof. We prove that the invariant holds in the look-ahead threads that we make the changes in. We know that the invariant holds Hyb_6. The only difference between Hyb_6 and $\mathsf{Hyb}_{7,0}$ is that the simulator uses random polynomials to compute the third round messages of RECom on the main thread. An alternate way to think of this is that either the polynomials used inside Com and that used to compute the third round of RECom are the same, or they're independently sample random polynomials. Thus we think of the change as $\mathsf{Sim}_{\mathsf{Hyb}}$ switching the commitment in Com from polynomials p to q while using p to compute the third round of RECom. This is in fact done by a sequence of hybrids where only a single Com is changed at a time. For simplicity, we proceed with the assumption that in this hybrid, only a single commitment was changed. Assume, for the sake of contradiction, that the invariant doesn't hold in $\mathsf{Hyb}_{7,0}$. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ causes event E to occur with non-negligible probability. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{Com}}$ that breaks the hiding property of Com with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{Com}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{Com}}$. $\mathcal{A}_{\mathsf{Com}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen Com messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The Com messages from P_i to P_j are exposed to the external challenger. Specifically, $\mathcal{A}_{\mathsf{Com}}$ sends two challenges $(\mathsf{p}_\ell, \mathsf{q}_\ell)$ to $\mathcal{C}$. And sets $\mathsf{recom}_{1,\ell}^{i\to j} := \mathsf{Com}$ where Com is received from $\mathcal{C}_{\mathsf{Com}}$. Depending on the challenge used by $\mathcal{C}_{\mathsf{Com}}$, we are either in Hyb_6 or $\mathsf{Hyb}_{7,0}$.

$\mathcal{A}_{\mathsf{Com}}$ creates 2 look ahead threads where it runs rounds 2 and 3 of the protocol alone. Now $\mathcal{A}_{\mathsf{Com}}$ runs the extractor $\mathsf{Ext}_{\mathsf{NMCom}}$ of the non-malleable commitment scheme using the message in both the threads that correspond to the non-malleable commitment from malicious party P_j to honest party P_i. Let the output of $\mathsf{Ext}_{\mathsf{NMCom}}$ be t^*. $\mathcal{A}_{\mathsf{Com}}$ checks if $\mathsf{TDValid}(t^*, \mathsf{td}_{1,i}) = 1$. If so, it outputs 1 to indicate $\mathsf{Hyb}_{7,2}$ and 0 otherwise.

By our assumption, the invariant doesn't hold. Thus $\mathsf{Ext}_{\mathsf{NMCom}}$, on adversary P_{j^*}'s commitment to P_i, outputs a valid trapdoor t_{i^*} for the trapdoor generation messages of the honest party P_{i^*} with non-negligible probability ε. With probability at least $\frac{1}{n^2}$, where n is the total number of players, this corresponds to honest party P_i and malicious party P_j picked randomly by $\mathcal{A}_{\mathsf{Com}}$. Therefore, with non-negligible probability $\frac{\varepsilon}{n^2}$, $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^* as a valid trapdoor. Since the invariant holds in Hyb_6, if $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs t^*, it must be the case that we're in $\mathsf{Hyb}_{7,0}$ with non-negligible probability. That is, when $\mathsf{Ext}_{\mathsf{NMCom}}$ outputs a valid trapdoor, it must correspond to $\mathcal{A}_{\mathsf{Com}}$ receiving the challenge using input q. This breaks the security of Com, which is a contradiction. Thus the invariant must also hold for $\mathsf{Hyb}_{7,0}$.

This works because as long as the number of threads created to extract from NMCom is less than B_{recom}, which is in fact true, since otherwise, the "random" polynomial no longer appears random. It should be noted that we don't need to extract the adversary's input for the reduction, and thus no use of creating any Type (ii) threads. □

Claim 49. *Assuming the security of* Com, $\mathsf{Hyb}_{7,0}$ *is indistinguishable from* Hyb_6

Proof. Since we're only making changes in a look-ahead thread, all we need to do is argue that the adversary doesn't switch to "junk" commitments when we make the change. The only difference between Hyb_6 and $\mathsf{Hyb}_{7,0}$ is that the simulator uses random polynomials to compute the third round messages of REComlook-ahead threads.

Assume, for the sake of contradiction, that this isn't true. Then there exists an adversary $\mathcal{A}$ such that for some honest party P_{i^*} and malicious party P_{j^*}, $\mathcal{A}$ commits RWI proofs for $\mathcal{L}_a$ in Ecom such that the probability of accept in the two cases in non-negligible. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{Com}}$ that breaks the security of Com with non-negligible probability.

The proof is similar to that of Claim 21 and Claim 48. □

$\mathsf{Hyb}_{7,1}$: **Create Type (ii) look-ahead thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ creates Type (ii) threads that are identical to the main thread. These will be used to extract the adversary's input. We create as many needed for the extraction of the adversary's input.

Claim 50. *Assuming the security of* Com *and the existence of* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{7,1}$.

Proof. This trivially follows from the fact that invariant holds in $\mathsf{Hyb}_{7,0}$ are identical to the main thread. □

Claim 51. *Assuming the security of* Com*,* $\mathsf{Hyb}_{7,1}$ *is indistinguishable from* Hyb_6

Proof. This follows as in the proof of Claim 49. □

$\mathsf{Hyb}_{7,2}$: **Change main thread** RECom **to 0:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the main thread to commit to 0. Specifically, for every honest party P_i and malicious party P_j do the following:

– compute $\mathsf{recom}_{3,\ell}$ as $(0 \oplus \mathsf{p}_\ell(0), \mathsf{p}_\ell(\mathsf{z}_\ell))$.

where p_ℓ are the polynomials committed to in the first round.

Claim 52. *Assuming the security of* Com *and the existence of* $\mathsf{Ext}_{\mathsf{NMCom}}$*, the invariant holds in* $\mathsf{Hyb}_{7,2}$.

Proof. The proof follows as in 48. □

Claim 53. *Assuming the security of* Com*,* $\mathsf{Hyb}_{7,2}$ *is indistinguishable from* Hyb_6

Proof. The proof follows as in 49 □

Note that $\mathsf{Hyb}_{7,2} \equiv \mathsf{Hyb}_7$

Thus Hyb_7 is indistinguishable from Hyb_6. □

Claim 54. *Assuming that* Π *is a secure protocol instantiated with rewinding secure* ROT*, hiding of* OT *against malicious senders, hiding of* Ecom*, bounded rewind witness indistinguishability of* RWI*,* Hyb_8 *is indistinguishable from* Hyb_7

Proof. This is proved via a sequence of hybrids. The reasoning behind this sub-division is that if we directly make changes to the protocol on the main thread, a subtle issue shows up during reduction. Namely, the look ahead threads currently employ an honest strategy using the inputs 0 for the underlying MPC. Additionally, they use the same first round message as the main thread. Although, no proof is sent in the clear, honest behavior is proven via the commitment and the OT receiver message on these look ahead thread, which requires knowledge of the randomness used for the underlying MPC. This is problematic during a reduction to the security of the underlying MPC. We will utilize the fact that our proofs are not sent in the clear to get around this issue.

Recall that we separate out the look ahead threads based on their purpose. Type (i) look-ahead threads are used to extract the trapdoor, while the type (ii) look-ahead threads are used to extract the inputs.

$\mathsf{Hyb}_{8,0}$: **Change type (i) threads** Ecom **to 0:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (i) threads to commit to 0 instead of commitment to RWI third round messages corresponding to $\mathcal{L}_a$.

Claim 55. *Assuming the hiding of* Ecom*,* $\mathsf{Hyb}_{8,0}$ *is indistinguishable from* Hyb_7

Proof. Since we're making changes only to type (i) threads used to extract trapdoor, we need to ensure we're still able to extract the trapdoor with this change. This can be done without having to rewind to the extract the trapdoor. On receiving the third round of the adversary's message on these threads, we can use TDOut to check if the trapdoor messages sent by the adversary satisfy validity. If there is a noticeable change in the validity condition being satisfied, we can break the hiding property of Ecom.

Assume there exists an adversary $\mathcal{D}$ that results in the trapdoor validity check being passed in $\mathsf{Hyb}_{8,0}$ and Hyb_7 with non-negligible difference. We will use this to create an adversary $\mathcal{A}_{\mathsf{Ecom}}$ to break the hiding property of Ecom with non-negligible probability. Note that we make changes to these threads one at a time.

$\mathcal{A}_{\mathsf{Ecom}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages in the main and look ahead threads other than the Ecom messages from P_i to P_j on the changed look ahead thread are identical. The Ecom messages from P_i to P_j are exposed to the external challenger. The first two rounds are forwarded to and from the adversary. Then, $\mathcal{A}_{\mathsf{Ecom}}$ sends to the challenger the pair of values $(0, \mathsf{rwi}_{a,3}^{i\to j})$

$\mathcal{A}_{\mathsf{Ecom}}$ receives a third round message ecom_3 which is either a commitment to 0 or $\mathsf{rwi}_{a,3}^{i\to j}$. This is sent to $\mathcal{A}$ as the third round message. The rest of the messages are obtained in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. Depending on which value was committed we are either in $\mathsf{Hyb}_{8,0}$ or Hyb_7. On completion of the execution, check the validity condition of the trapdoor messages sent using TDOut. If the validity check passes, output 0 (to indicate we're in Hyb_7), else output 1.

By our assumption, $\mathcal{D}$ results in non-negligible difference in the validity condition being verified it the two hybrids with noticeable probability difference ε. Therefore, with non-negligible advantage $\frac{\varepsilon}{n^2}$, $\mathcal{A}_{\mathsf{Ecom}}$ wins the challenge game with $\mathcal{C}_{\mathsf{Ecom}}$ which breaks the hiding property of Ecom. Thus, ε must be negligible, and thus we continue to extract the trapdoor.

Since we continue to extract trapdoor, $\mathsf{Hyb}_{8,0}$ is indistinguishable from Hyb_7.

□

Note that we don't need to argue invariant here to argue indistinguishability since the look ahead threads we extract inputs from are unchanged.

$\mathsf{Hyb}_{8,1}$: **Change Type (i) threads receiver input to 0 in** OT**:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (i) threads to use receiver input 0 in OT instead of the third round messages of RWI corresponding to $\mathcal{L}_b$.

Claim 56. *Assuming the hiding of* OT *against malicious senders,* $\mathsf{Hyb}_{8,1}$ *is indistinguishable from* $\mathsf{Hyb}_{8,0}$

Proof. The proofs follows identically as in Claim 55. The only difference being the now the OT messages are exposed to the external challenger. Since we're still able to extract the trapdoor, $\mathsf{Hyb}_{8,1}$ is indistinguishable from $\mathsf{Hyb}_{8,0}$.

□

We now make changes to look ahead threads of Type (ii)

$\mathsf{Hyb}_{8,2}$: **Switch** RWI **proofs for** $\mathcal{L}_a$ **on Type (ii) threads:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (ii) threads to switch to the "trapdoor witness" in the RWI proofs for $\mathcal{L}_a$. This is done a single thread at a time.

We need to ensure that we still continue extracting the inputs of the adversary. This is done by proving the invariant holds in each of the threads when we make this change.

Claim 57. *Assuming bounded rewind witness indistinguishability of* RWI*, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$ *the invariant holds in* $\mathsf{Hyb}_{8,2}$.

Proof. The proof follows identically as in Claim 22. □

Claim 58. *Assuming bounded rewind witness indistinguishability of* RWI*,* $\mathsf{Hyb}_{8,2}$ *is indistinguishable from* $\mathsf{Hyb}_{8,1}$.

Proof. The proof follows identically as in Claim 23. □

$\mathsf{Hyb}_{8,3}$: **Switch** RWI **proofs for** $\mathcal{L}_b$ **on Type (ii) threads:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (ii) threads to switch to the "trapdoor witness" in the RWI proofs for $\mathcal{L}_b$. This is done a single thread at a time.

Claim 59. *Assuming bounded rewind witness indistinguishability of* RWI*, and the existence of an extractor* $\mathsf{Ext}_{\mathsf{NMCom}}$ *the invariant holds in* $\mathsf{Hyb}_{8,3}$.

Proof. The proof follows identically as in Claim 22. □

Claim 60. *Assuming bounded rewind witness indistinguishability of* RWI*,* $\mathsf{Hyb}_{8,3}$ *is indistinguishable from* $\mathsf{Hyb}_{8,2}$.

Proof. The proof follows identically as in Claim 23. □

$\mathsf{Hyb}_{8,4}$: **Switch** RWI **proofs for** $\mathcal{L}_b$ **on Type (ii) threads:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (ii) threads to switch to the "trapdoor witness" in the RWI proofs for $\mathcal{L}_b$. This is done a single thread at a time.
$\mathsf{Hyb}_{8,5}$: **Simulate** Π **on main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the transcript of the underlying protocol Π. For a complete description of the changes, refer to the description of Hyb_8.

Claim 61. *Assuming that* Π *is a secure protocol instantiated with rewinding secure* ROT*, the invariant holds in* $\mathsf{Hyb}_{8,5}$.

Proof. Here, since the invariant only depends on the first three rounds, we need to prove that the invariant holds conditioned on the view of the first three rounds. The proof is similar to Claim 31. □

Claim 62. *Assuming that* Π *is a secure protocol instantiated with rewinding secure* ROT*,* $\mathsf{Hyb}_{8,5}$ *is indistinguishable from* $\mathsf{Hyb}_{8,4}$.

Proof. The only difference between $\mathsf{Hyb}_{8,5}$ and $\mathsf{Hyb}_{8,4}$ is how the transcript of the underlying protocol Π is computed.

Assume, for the sake of contradiction, that that the views are distinguishable. Then there exists an adversary $\mathcal{D}$ such that $\mathcal{D}$ can distinguish between $\mathsf{Hyb}_{8,5}$ and $\mathsf{Hyb}_{8,4}$ with non-negligible advantage. We will use this adversary to create an adversary $\mathcal{A}_\Pi$ that breaks the indistinguishability of Π when instantiated with bounded rewinding secure ROT with non-negligible probability. Essentially, we rely on the fact that if the ROT is rewinding secure, then the transcript of Π for an honest and simulated transcript are indistinguishable.

We now describe the working of $\mathcal{A}_\Pi$ which interacts with the challenger $\mathcal{C}_\Pi$. All messages other than the Π messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The Π messages from are exposed to the external challenger. Specifically, in round 1, set $\{\mathsf{msg}_{1,i}\}_{P_i \in \mathcal{H}} := \overrightarrow{\mathsf{msg}}_1$ where $\overrightarrow{\mathsf{msg}}_1$ is received from $\mathcal{C}_\Pi$. Send to $\mathcal{C}_\Pi$ $\{\mathsf{msg}_{1,i}\}_{P_i \notin \mathcal{H}}$ that is sent by $\mathcal{A}$. The response from $\mathcal{C}_\Pi$, $\overrightarrow{\mathsf{msg}}_2$ is parsed as $\{\mathsf{msg}_{2,i}\}_{P_i \in \mathcal{H}} := \overrightarrow{\mathsf{msg}}_2$.

$\mathcal{A}_\Pi$ then creates a set of 3 type (i) look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. This is done to extract the trapdoor. This doesn't require generating proofs on these look-ahead threads. Now, with the extracted trapdoor, $\mathcal{A}_\Pi$ then creates a set of 5 type (ii) look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. In each look-ahead thread, $\mathcal{A}_\Pi$ forwards the $\{\mathsf{msg}_{2,i}\}_{P_i \notin \mathcal{H}}$ sent by $\mathcal{A}$ in each look-ahead thread to $\mathcal{C}_\Pi$. These are simply ROT messages and will be responded to by $\mathcal{C}_\Pi$. The response is likewise forwarded to $\mathcal{A}$. These 5 threads are all GOOD with respect to some party H with noticeable probability. With the 5 threads, $\mathcal{A}_\Pi$ can successfully run the extraction phase. Note that in these threads, $\mathcal{A}_\Pi$ can use the "trapdoor witness" extracted using the type (i) threads.

On completion of the extraction phase, prior to the third round on the main thread, $\mathcal{A}_\Pi$ sends to $\mathcal{C}_\Pi$ all parties inputs $(\{\mathsf{x}_i, \mathsf{r}_i\}_{i \in [n]}, y)$ to $\mathcal{C}_\Pi$. $\mathcal{C}_\Pi$ then either responds with the simulated last message or the honest execution for the rest of the transcript. The rest of the messages are obtained in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. Depending on the choice of $\mathcal{C}_\Pi$ we are either in Hyb_8 or Hyb_7. On completion of the execution, the view is input to $\mathcal{D}$ and the output returned is the output of $\mathcal{A}_\Pi$

By our assumption, $\mathcal{D}$ can distinguish between the two hybrids with non-negligible probability ε. Therefore, with non-negligible advantage ε, $\mathcal{A}_\Pi$ wins the challenge game with $\mathcal{C}_\Pi$ which breaks the security of Π when rewinding security of ROT is maintained. Thus, ε must be negligible, and thus the views are indistinguishable.

□

Remark 12. *For the case of the implicit abort, in the above sub-hybrid we replace the* Π *messages of honest parties to be computed honestly using input 0. The arguments then follow identically as above and Claims 31 and 32.*

Now we undo the changes made to the look ahead threads. The proofs follow identically as argued above, and are skipped.

$\mathsf{Hyb}_{8,6}$: **Switch RWI proofs for $\mathcal{L}_b$ on Type (ii) threads:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (ii) threads to switch back to the "real witness" in the RWI proofs for $\mathcal{L}_b$. This is done a single thread at a time.

$\mathsf{Hyb}_{8,7}$: **Change Type (i) threads receiver input to RWI message in OT:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (i) threads to use receiver input to be the third round message of RWI, corresponding to $\mathcal{L}_b$, in OT, instead of 0.

$\mathsf{Hyb}_{8,8}$: **Change Type (i) threads Ecom to RWI message:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ modifies the third round of the type (i) threads to commit to RWI third round messages corresponding to $\mathcal{L}_a$ instead of the commitment to 0.

Note that $\mathsf{Hyb}_{8,8} \equiv \mathsf{Hyb}_8$

□

Claim 63. *The invariant holds in* Hyb_9.

Proof. The claim is trivially true since there are no changes to the main thread. □

Claim 64. Hyb_9 *is indistinguishable from* Hyb_8 *except with negligible probability.*

Proof. Except with negligible probability, the extraction from OT succeeds, and therefore the simulator does not abort. $\square$

Claim 65. *The invariant holds in* Hyb_{10}.

Proof. The claim is trivially true since the change is made only in the fourth round. $\square$

Claim 66. *Assuming the security of* GC *and sender's* OT *messages,* Hyb_{10} *is indistinguishable from* Hyb_9

Proof. This is established by the creating the following sub-hybrids.

$\mathsf{Hyb}_{10,0}$: **Change** OT **sender's message on main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ changes how the sender OT is computed. We extract from ot to obtain the adversary's receiver message. Use the receiver value extracted from the ot to change the sender OT to include only a single label of the garbled circuit. Specifically, $\forall j \neq i$, compute

$$\mathsf{ot}_4^{i\to j} \leftarrow \mathsf{OT}_4((\mathsf{lab}_{i,v|_j}, \mathsf{lab}_{i,v|_j})).$$

where v is the extracted receiver string from $\mathsf{ot}_3^{i\to j}$.

Claim 67. *Assuming the security of sender's* OT *messages,* $\mathsf{Hyb}_{10,0}$ *is indistinguishable from* Hyb_9

Proof. The only difference between $\mathsf{Hyb}_{10,0}$ and Hyb_9 is that the simulator $\mathsf{Sim}_{\mathsf{Hyb}}$ switches the sender OT input to using the same label twice P_i if it receives a non-accepting RWI proof for $\mathcal{L}_a$.

Assume, for the sake of contradiction, that that the views are distinguishable. Then there exists an adversary $\mathcal{D}$ such that $\mathcal{D}$ can distinguish between $\mathsf{Hyb}_{10,0}$ and Hyb_9 with non-negligible advantage. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{OT}}$ that breaks the sender's security in OT with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{OT}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{OT}}$. $\mathcal{A}_{\mathsf{OT}}$ picks randomly an honest party P_i and a random malicious party P_j. All messages other than the chosen OT messages are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The OT messages from P_i to P_j are exposed to the external challenger. Specifically, in round 1, send to $\mathcal{C}_{\mathsf{OT}}$ the first round OT message $\mathsf{ot}_1^{i\to j}$ sent by $\mathcal{A}$. Receive ot_2 and set $\mathsf{ot}_2^{i\to j} := \mathsf{ot}_2$.

$\mathcal{A}_{\mathsf{OT}}$ creates sufficiently many look-ahead threads, in each of which, it runs rounds 2 and 3 of the protocol alone. In each look-ahead thread, $\mathcal{A}_{\mathsf{OT}}$ re-sends the same $\mathsf{ot}_2^{i\to j}$ message in the second round. Only $\mathsf{ot}_3^{i\to j}$ on the main thread is forwarded to $\mathcal{C}_{\mathsf{OT}}$. With the the look ahead threads threads, $\mathcal{A}_{\mathsf{OT}}$ can successfully run the extraction phase to extract the OT receiver bit from P_j to be v. Send $\left(\mathsf{lab}_{i,0|_j}, \mathsf{lab}_{i,1|_j}\right)$ and $\left(\mathsf{lab}_{i,v|_j}, \mathsf{lab}_{i,v|_j}\right)$ to $\mathcal{C}_{\mathsf{OT}}$ as challenges. Note that we don't need rewind security here since the look ahead threads are only executed up to the third round. And for the alternating message OT, a new adversarial receiver message doesn't have to be answered on the look ahead threads.

The rest of the messages are obtained in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. Depending on pair was used as sender input we are either in $\mathsf{Hyb}_{10,0}$ or Hyb_9. On completion of the execution, the view is input to $\mathcal{D}$ and the output returned is the output of $\mathcal{A}_{\mathsf{OT}}$

By our assumption, $\mathcal{D}$ can distinguish between the two hybrids with noticeable probability ε. Therefore, with non-negligible advantage $\frac{\varepsilon}{n^2}$, $\mathcal{A}_{\mathsf{OT}}$ wins the challenge game with $\mathcal{C}_{\mathsf{OT}}$

which breaks the sender security of OT. Thus, ε must be negligible, and thus the views are indistinguishable. $\square$

$\mathsf{Hyb}_{10,1}$: **Simulate garbled circuit:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ computes a garbled circuit to output $\perp$ if either in **implicit abort** or **opaque** case. Specifically, in this case,

$$\left(\mathsf{C}_i, \widetilde{\mathsf{lab}}_i\right) \leftarrow \mathsf{Garble}\left(\mathsf{C}_\perp\right)$$

Claim 68. *Assuming the security of* GC*,* $\mathsf{Hyb}_{10,1}$ *is indistinguishable from* $\mathsf{Hyb}_{10,0}$

Proof. The only difference between $\mathsf{Hyb}_{10,1}$ and $\mathsf{Hyb}_{10,0}$ is that the simulator $\mathsf{Sim}_{\mathsf{Hyb}}$ switches the garbled circuit to a circuit for each relevant P_i. Note that this is a functionally equivalent circuit given the condition we choose to switch. Namely, either there is an implicit abort, or that P_i receives a wrong RWI proof via OT. The changes are made through a sequence of sub-hybrids, where in each sub-hybrid only a single circuit is switched.

Assume, for the sake of contradiction, that that the views are distinguishable. Then there exists an adversary $\mathcal{D}$ such that $\mathcal{D}$ can distinguish between $\mathsf{Hyb}_{10,1}$ and $\mathsf{Hyb}_{10,0}$ with non-negligible advantage. We will use this adversary to create an adversary $\mathcal{A}_{\mathsf{GC}}$ that breaks GC security with non-negligible probability.

We now describe the working of $\mathcal{A}_{\mathsf{GC}}$ which interacts with the challenger $\mathcal{C}_{\mathsf{GC}}$. All messages other than the garbled circuit are computed in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. The GC messages from P_i are exposed to the external challenger. Specifically, in round four, it sends as challenges to $\mathcal{C}_{\mathsf{GC}}$, $(\mathsf{C}_\perp, v)$ and $(\mathsf{C}[\mathsf{msg}_{4,i}, \mathbf{T}^{\bullet\to i}_{\mathsf{rwi}_b}[2], \mathsf{st}^{\bullet\to i}_b, \mathsf{r}^{\bullet\to i}_{\mathsf{rwi}_b}], v)$ where v is the concatenation of all extracted/generated receiver values for all parties other than P_i. $\mathcal{C}_{\mathsf{GC}}$ then returns a garbled circuit $\overline{\mathsf{C}}$ and labels corresponding to the input v, $\widehat{\mathsf{lab}}$. These are set as $\overline{\mathsf{C}}_i := \overline{\mathsf{C}}$ and $\widehat{\mathsf{lab}}_i := \widehat{\mathsf{lab}}$. The rest of the messages are obtained in the same manner as $\mathsf{Sim}_{\mathsf{Hyb}}$. Depending on challenge bit used by $\mathcal{C}_{\mathsf{GC}}$ we are either in $\mathsf{Hyb}_{10,1}$ or $\mathsf{Hyb}_{10,0}$. On completion of the execution, the view is input to $\mathcal{D}$ and the output returned is the output of $\mathcal{A}_{\mathsf{GC}}$.

By our assumption, $\mathcal{D}$ can distinguish between the two hybrids with noticeable probability ε. Therefore, with non-negligible advantage, $\mathcal{A}_{\mathsf{GC}}$ wins the challenge game with $\mathcal{C}_{\mathsf{GC}}$ which breaks the security of GC. Thus, ε must be negligible, and thus the views are indistinguishable.[19] $\square$

$\mathsf{Hyb}_{10,2}$: **Change** OT **sender's message on main thread:** In this hybrid, $\mathsf{Sim}_{\mathsf{Hyb}}$ changes how the sender OT is computed.

Change the sender OT to include back to include both labels of the garbled circuit. Specifically, $\forall j \neq i$, compute

$$\mathsf{ot}_4^{i\to j} \leftarrow \mathsf{OT}_4((\mathsf{lab}_{i,0|_j}, \mathsf{lab}_{i,1|_j})).$$

Claim 69. *Assuming the security of sender's* OT *messages,* $\mathsf{Hyb}_{10,2}$ *is indistinguishable from* Hyb_9

Proof. The proof follows identically as in Claim 67. $\square$

Note that $\mathsf{Hyb}_{10,2} \equiv \mathsf{Hyb}_{10}$.

$\square$

[19]For ease of proof, we use the indistinguishability definition instead of the simulation definition as given in the prelims. This is easily rectified by having another intermediate hybrid where the simulated garbled circuit it used.

Claim 70. *The invariant holds in* $\mathsf{Hyb}_{\mathsf{IDEAL}}$.

Proof. The claim is trivially true since the main thread remains unchanged. □

Claim 71. Hyb_{10} *is indistinguishable from* $\mathsf{Hyb}_{\mathsf{IDEAL}}$ *except with probability at most* $\frac{\mu}{4} + \mathsf{negl}(\lambda)$.

Proof. This is argued in two cases depending on the probability with which the adversary abort.

Case 1: $\Pr[\mathsf{not\ abort}] \geq \frac{\mu}{4}$**:**

Suppose the adversary doesn't cause an abort with probability greater that $\frac{\mu}{4}$. Let us analyze the probability with which $\perp_{\mathsf{extract}}$ is output by $\mathsf{Sim}_{\mathsf{Hyb}}$. By the Chernoff bound, in Hyb_{11}, except with negligible probability, in the set of $\frac{5 \cdot n \cdot \lambda}{\mu}$ threads, there will be at least 5 GOOD threads with respect to some honest party P_{i^*}. Also in $\mathsf{Hyb}_{\mathsf{IDEAL}}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ will run an expected polynomial number of threads to get 12λ (which is greater than $5 \cdot n$) GOOD threads. Thus the extractions will be successful in except with negligible probability.

Therefore the only difference between $\mathsf{Hyb}_{\mathsf{REAL}}$ and Hyb_{11} is that in Hyb_{11}, after extraction, $\mathsf{Sim}_{\mathsf{Hyb}}$ samples the main thread $\frac{\lambda}{\mu}$ times while in $\mathsf{Hyb}_{\mathsf{REAL}}$, $\mathsf{Sim}_{\mathsf{Hyb}}$ first estimates the probability of not aborting to be ε' and then re-samples the main thread $\min\left(2^{\lambda}, \frac{\lambda^2}{\varepsilon'}\right)$ times. The rest of the proof follows in a very similar manner to the proof of claim 5.8 in [Lin16]. That is, we show that if "Check Abort" step succeeds, the simulator in $\mathsf{Hyb}_{\mathsf{IDEAL}}$ fails only with negligible probability using the claim in [Lin16]. Also, by a Markov argument, we know that Hyb_{11}, if the "Check Abort" step succeeds, the simulation successfully forces the output and hence, this completes the proof.

Case 2: $\Pr[\mathsf{not\ abort}] < \frac{\mu}{4}$**:**

Suppose the adversary doesn't cause an abort with probability smaller than $\frac{\mu}{4}$. Then, in both hybrids, $\mathsf{Sim}_{\mathsf{Hyb}}$ aborts at the end of the "Check Abort" step except with probability $\frac{\mu}{4}$. Thus, in this case, the adversary's view in $\mathsf{Hyb}_{\mathsf{IDEAL}}$ and Hyb_{11} is indistinguishable except with probability at most $\frac{\mu}{4} + \mathsf{negl}(\lambda)$.

□

We now calculate the probability that the adversary can distinguish between $\mathsf{Hyb}_{\mathsf{REAL}}$ and $\mathsf{Hyb}_{\mathsf{IDEAL}}$.

Except in two cases, every pair of hybrids are indistinguishable except with negligible probability. In the two special cases, the hybrids are indistinguishable except with probability $\frac{\mu}{4} + \mathsf{negl}(\lambda)$. Thus, $\mathsf{Hyb}_{\mathsf{REAL}}$ and $\mathsf{Hyb}_{\mathsf{IDEAL}}$ are indistinguishable except with probability $\frac{\mu}{2} + \mathsf{negl}(\lambda)$. This contradicts our assumption that there must be an adversary $\mathcal{A}$ that can distinguish the REAL and IDEAL executions with probability at least μ.

4.6 Bidirectional to Alternating message model

In [GMPP16] the authors prove that there does not exist a 3-round protocol in the bidirectional message model for tossing $\omega(\log \lambda)$ coins which can be proven secure via blackbox simulation. To prove this theorem, the authors show how to reschedule a 3-round protocol in the bidirectional message into a 4-round non-simultaneous protocol thus contradicting the impossibility of [KO04]. In this section we extend the proof approach used in [GMPP16] to show the following. Let $\Pi^{\leftrightarrow} =$

$(A^{\leftrightarrow}, B^{\leftrightarrow})$ be a k-round two-party protocol (2PC) that securely computes the function f in the bidirectional message model. f takes the inputs of the parties $A^{\leftrightarrow}$ and $B^{\leftrightarrow}$, that we denote with $x_{A^{\leftrightarrow}}$ and $x_{B^{\leftrightarrow}}$ respectively, and outputs $y_{A^{\leftrightarrow}}$ and $y_{B^{\leftrightarrow}}$, where $y_{A^{\leftrightarrow}}$ corresponds to the output of $A^{\leftrightarrow}$ and $y_{B^{\leftrightarrow}}$ corresponds to the output of $B^{\leftrightarrow}$. We show how to obtain a k-round 2PC protocol $\Pi^{\leftrightarrows} = (A, B)$ in the alternating message model, in which *at least* one party gets the output.

Theorem 16. *Any k-round two party protocol (2PC) $\Pi^{\leftrightarrow}$ that securely computes f in the bidirectional message model, proven secure via blackbox simulation, can be turned into a k-round two party protocol in the alternating message model, in which* at least *one party gets the output of f.*

Proof. We show how to obtain a k-round 2PC protocol $\Pi^{\leftrightarrows} = (A, B)$, in the alternating message model, that securely computes f in which only one party gets the output. Without loss of generality, we assume that only the party B gets the output. We denote with m_i^A the message that the party $A^{\leftrightarrow}$ sends in the i-th round of $\Pi^{\leftrightarrow}$, and with m_i^B the message that the party $B^{\leftrightarrow}$ in the i-th round of $\Pi^{\leftrightarrow}$ with $1 \leq i \leq k$.

In $\Pi^{\leftrightarrows}$ the party A computes its messages by internally running $A^{\leftrightarrow}$, and the same does B with $B^{\leftrightarrow}$. We provide an high level description of $\Pi^{\leftrightarrows}$ in Fig 4.6. The main observation that makes possible to reschedule the messages of $\Pi^{\leftrightarrow}$ in the alternating message model is that the message that $B^{\leftrightarrow}$ sends to $A^{\leftrightarrow}$ in the last round of $\Pi^{\leftrightarrow}$ can be removed given that A does not need to compute the output. Moreover, the security of $\Pi^{\leftrightarrow}$ is proved by considering a rushing adversary. This means that a message that an honest party sends in the round i-th of $\Pi^{\leftrightarrow}$ has to be independent from the message that the other party sends in the i-th round. We propose a more formal description of $\Pi^{\leftrightarrows}$ in Fig. 4.7.

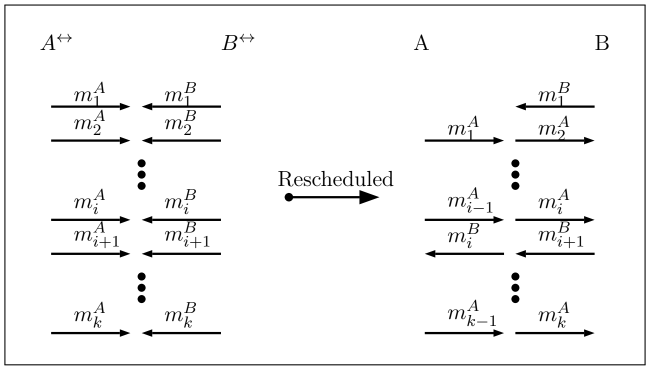


Figure 4.6: High level description of the rescheduled messages.

We start by considering the case in which B is corrupted (we denote a corrupted party P with $P^\star$). Then we need to build an expected PPT simulator $\mathcal{S}^{\leftrightarrows}$ that satisfies the Definition 15. Since $\Pi^{\leftrightarrow}$ is secure, then there exists a simulator $\mathcal{S}^{\leftrightarrow}$ in the ideal world for any corrupt $B^{\leftrightarrow\star}$ executing the simultaneous message exchange protocol $\Pi^{\leftrightarrow}$. Our simulator $\mathcal{S}^{\leftrightarrows}$ is constructed using $\mathcal{S}^{\leftrightarrow}$ and works as follows.

1. $\mathcal{S}^{\leftrightarrows}$, upon receiving m_1^B from $B^\star$, forwards m_1^B to $\mathcal{S}^{\leftrightarrow}$. $\mathcal{S}^{\leftrightarrow}$ outputs (m_1^A, m_2^A) (note that the inner simulator must be able to produce m_2^A even before seeing the second message m_2^B of party $B^{\leftrightarrow\star}$ given that the $B^{\leftrightarrow\star}$ is rushing). Moreover, $\mathcal{S}^{\leftrightarrows}$ acts as a proxy between $\mathcal{S}^{\leftrightarrow}$ and the ideal functionality and whenever $\mathcal{S}^{\leftrightarrow}$ asks to rewind the adversary, $\mathcal{S}^{\leftrightarrows}$ rewinds $B^\star$

Round-1: B runs $\Pi^{\leftrightarrow}$ on behalf of $B^{\leftrightarrow}$ thus obtaining the m_1^B and sends it to A

Round-i with $1 < i < k$, $i \mod 2 = 0$: A runs $\Pi^{\leftrightarrow}$ on behalf of $A^{\leftrightarrow}$ to compute the messages (m_{i-1}^A, m_i^A) and sends them to B .

Round-i with $1 < i < k, i \mod 2 \neq 0$: B runs $\Pi^{\leftrightarrow}$ on behalf of $B^{\leftrightarrow}$ to compute the messages (m_{i-1}^B, m_i^B) and sends them to A.

Round-k: A runs $\Pi^{\leftrightarrow}$ on behalf of $A^{\leftrightarrow}$ to compute the messages (m_{k-1}^A, m_k^A) and sends them to B.

Figure 4.7: $\Pi^{\leftrightarrows}$ description.

2. Upon receiving the message $m_i = (m, m')$ from $B^\star$ in the i-th round, with $1 < i < k-1$, $\mathcal{S}^{\leftrightarrows}$, sends m to $\mathcal{S}^{\leftrightarrow}$, receives m_{i-1}^A and sends also m' to $\mathcal{S}^{\leftrightarrow}$. $\mathcal{S}^{\leftrightarrow}$ now outputs m_i^A which $\mathcal{S}^{\leftrightarrows}$ sends to $B^\star$.
3. Upon receiving the message $m_{k-1} = (m, m')$ from $B^\star$ in the k-th round, $\mathcal{S}^{\leftrightarrows}$ sends m to $\mathcal{S}^{\leftrightarrow}$, receives m_{k-1}^A and sends also m' to $\mathcal{S}^{\leftrightarrow}$. $\mathcal{S}^{\leftrightarrow}$ now outputs m_k^A and $\mathcal{S}^{\leftrightarrows}$ sends (m_{k-1}, m_k) to $B^\star$. In the end $\mathcal{S}^{\leftrightarrows}$ sends an abort message to $\mathcal{S}^{\leftrightarrow}$ (to indicate that the adversary has not sent the last message) and outputs what $\mathcal{S}^{\leftrightarrow}$ outputs.

It should be easy to see that $\mathcal{S}^{\leftrightarrows}$ emulates correctly $B^{\leftrightarrow\star}$ and hence $\mathcal{S}^{\leftrightarrows}$ represents a good adversary for the ideal world. The proof for the case in which A is corrupted is similar, with the difference that the last message output by the inner simulator $\mathcal{S}^{\leftrightarrow}$ is not forwarded to $A^\star$. $\square$

4.7 Open Problems

While our work concludes a long line of research on constructing round-optimal MPC from minimal assumptions, studying round-optimal protocols for stronger security notions such as that of *identifiable abort*, where participants can prove that a participant misbehaved, still remain open.

Chapter 5

Founding Secure Computation on Blockchains

5.1 Overview

We start with an overview of the techniques before moving on to the technical sections. Recall that an adversary is said to be blockchain-active if can read from and post to the blockchain (Section 2.14). We observe that if an adversary is blockchain-active, it can "detect" that it is being rewound by posting the transcript of the interaction so far on the blockchain. In more detail, upon getting an incoming message, the adversary concatenates the entire transcript with a session ID and submits it to the blockchain Oracle. Before giving a response, the adversary waits for the next block to be mined and checks the following: the transcript it posted on the blockchain has indeed appeared, and, no such transcript (for the same session and the same round) appeared on any of the prior blocks. If the check passes (which is guaranteed in the real execution), the adversary proceeds honestly with computing and sending the next protocol message. We show that it would be impossible for any polynomial-time simulator to rewind this adversary which forms the basis of our black-box impossibility result for zero-knowledge.

Constructing Black-Box Zero-Knowledge Protocols. To overcome the above problems posed by blockchains, we look towards blockchains for a solution as well. Our idea is to make the *protocol* blockchain active as well. Specifically, we let the honest prover keep track of the blockchain state, and, if the number of new blocks mined since the beginning of the protocol exceed a fixed number k, abort. Thus, the honest parties use the blockchain to implement a time-out mechanism. We emphasize, however, that we do not require the honest parties to have synchronized clocks. The only requirement placed is that the protocol must be finished in an a priori bounded amount of time, *as measured by the progress of the blockchain*. For example, while using Bitcoin, if k is set to 20, this gives the parties nearly 3.5 hours to finish the zero-knowledge protocol before a time-out occurs (since a block is mined roughly every 10 minutes in Bitcoin). For simplicity, we will treat the parameter k as a constant (even though our constructions can handle an arbitrary value of k by scaling the round complexity of the protocol appropriately).

We devise a construction for black-box zero-knowledge proofs where the number of "slots" (or rewinding opportunities) in the protocol is higher than k. While the adversary can send any information to the blockchain Oracle at any point of time, there can be at most k points in the protocol execution where the adversary actually *receives* from the Oracle a new (unforgeable) mined block. However by our construction, this would still leave several slots in the protocol where the simulator is free to rewind (without having to forge the blockchain state).

A potentially complication in the design of the simulator arises from the fact that, apart from the newly mined blocks, the adversary can also "listen in" on the network communication in real time. This could consist of various (honest party) transactions currently outstanding on the network and waiting to be included in the next block. This is formalized by *buffer reads* in the model of Badertscher et. al [BMTZ17]. We handle this problem by having the simulator simply replay the honest-party outstanding transactions since they could not have changed from the main thread to the look-ahead thread. The adversarial outstanding transactions (which might change from thread to thread) in the current thread are already known to the simulator since the simulator can read all outgoing messages from the adversary. The above ideas form the basis of our first positive result modulo the issue of simulation time which is discussed next.

The Issue of Simulation Time. Interestingly, the fact that blockchains can be used to implement a global unforgeable clock presents a novel challenge in proving security against blockchain-active adversaries, that to the best of our knowledge, does not arise elsewhere in cryptography. Typically in cryptography, the running time of the simulator is larger than the running time of the adversary. This means that the number of blocks mined during a simulated execution may be higher than the number of blocks mined during a real execution. Then, the number of mined blocks can be used as "side-channel" information to distinguish real and simulated executions, if the adversary and the distinguisher are blockchain-active! Such a difficulty does not arise in the plain model since the simulator is assumed to have complete control over the clock of the adversary (including the ability to freeze it).

To address this issue, we seek to construct a simulator whose running time is the same as the real protocol execution. Towards that end, we build upon techniques from the notion of precise zero-knowledge [MP06]. To start with, it would seem that we need to construct a simulator with precision exactly 1, something that is currently not known to be possible. To resolve this problem, our key observation is that there is a crucial difference between the *time that the simulator takes to finish* and *the number of computation steps it executes*. In particular, if the simulator can execute a number of computations *in parallel*, it could potentially perform more computations than the prover in the real execution, and yet, finish in the same amount of time. Our rewinding strategy would run several threads of execution in parallel (e.g., by making several copies of the adversary code) and ensure that by the time the *main*[1] thread finishes, all the rewound execution threads have finished as well. To ensure that the simulation succeeds, our simulator is necessarily required to have a super-constant number of rewinding opportunities (which can be pursued in parallel). Such a simulator would give a guarantee of the following form: any information learnt by an adversarial verifier in the protocol could also be produced from scratch by an algorithm which is capable of running sufficient (polynomial) number of computations in parallel. For example, a quad core processor is capable of running 4 parallel computations.

We believe that the issue of simulation time is one of independent interest. In particular, developing an understanding of the time required by the simulator (as opposed to the number of computation steps) could shed additional light on the knowledge complexity of cryptographic constructions as well as motivate the study of strong notions of security.

[1]The thread output by the simulator is referred to as the main thread.

Lower Bound on Round Complexity of Black-Box Zero-Knowledge. We prove that constant round ZK arguments are impossible w.r.t black-box simulation in the blockchain-hybrid model. Our impossibility result holds even for expected polynomial-time simulators.

Consider an adversarial verifier that waits for a fixed constant time c before responding to any message from the prover. Our proof works in two steps:

1. Recall that black-box simulators can only query the adversarial verifier as an Oracle. However, the simulator may choose to make these queries *in parallel* rather than sequentially by making several copies of the adversary state (and hence, increasing the number of available Oracles).

   In the first step, we assume that the simulator is *memory bounded*. This means that at any given time, the simulator may only have a bounded (strict polynomial) number of copies (say) $q(\cdot)$ of the adversary. Furthermore, since the verifier takes time c to answer each query, the total number of queries the simulator may make to the adversary in a given time t can be bounded by $\frac{q \cdot t}{c}$ (an a priori bounded strict polynomial). Now we observe the following:

   – The simulator must terminate within roughly t steps where t is the time an honest prover takes to complete the proof. To see this, let r be an upper bound on the number of blocks that can created in the time taken by the honest prover to complete the proof. We consider a blockchain active adversary that observes the state of the blockchain when the protocol starts, and posts a transcript on the completion of the proof. If it notices that more than r blocks have been created since the protocol started, it concludes that it is interacting with the simulator.

   – Thus, the overall number of queries (and hence) the running time of the simulator is a strict polynomial. Now, we can directly invoke the result of Barak and Lindell [BL02] that rules out constant-round ZK arguments with strict polynomial-time black-box simulation.

2. The above only rules out a simulator with "a priori bounded parallelism." However what if, e.g., the number of parallel queries the simulator may make to the verifier cannot be a priori bounded (and instead we only require that the simulator finish in a priori bounded number of computational steps)? In particular, the simulator may see the responses to the queries made so far, and, *adaptively* decide to increase the number of parallel queries (i.e., the number of copies of the adversary)? This case is more tricky and as such, the ideas from the work of [BL02] don't apply.

   To resolve this issue, we crucially rely upon the fact that by carefully choosing the delay parameter c and an aborting probability for the adversary, the number of such "adaptive steps" can be bounded by a constant. Thereafter, we argue that in each adaptive step, if the simulator increases the number of parallel copies by more than an a priori bounded polynomial factor, it runs the risks of blowing the number of computation steps to beyond expected polynomial. On the other hand if the number of parallel copies blow up by at most a fixed polynomial factor, since the number of adaptive steps is a constant, the simulator is still using "bounded parallelism" (a case already covered by our previous step). The full proof is delicate and can be found in Section 5.5.

Concurrent Secure computation. We now proceed to describe the main ideas behind our positive result for concurrent self-composable secure computation. We start by recalling the intuition behind the impossibility of concurrent secure computation w.r.t. black-box simulation in the plain model.

A primary task of a simulator for a secure computation protocol is to extract the adversary's input. A black-box simulator extracts the input of the adversary by rewinding. However, in the concurrent setting, extracting the input of the adversary in each session is a non-trivial task. In particular, given an adversarial scheduling of the messages of concurrent sessions, it may happen that in order to extract the input of the adversary in a given session s, the simulator rewinds past the beginning of another session s' that is interleaved inside the protocol messages of session s. When this happens, the adversary may change its input in session s'. Thus, the simulator would be forced to query the ideal functionality more than once for the session s'.

Indeed, as shown in [Lin04], this intuition can be formalized to obtain a black-box impossibility result for concurrent self-composition w.r.t. the standard definition of secure computation, where only one query per session is allowed. While Lindell's impossibility result is only w.r.t. black-box simulation, subsequent works have shown impossibility of concurrent secure computation even w.r.t. non-black-box simulation [BPS06, Goy12, AGJ+12, GKOV12].

In order to overcome the impossibility results, our starting idea is the following: prior to the start of a protocol, each party must commit to its input and randomness on the blockchain. It must then wait for its commitment string to be posted on the blockchain before sending any further message in the protocol. Similar to our ZK protocols (with stand-alone security), we use a time-out mechanism to place an upper bound on the number of blocks that can be mined during a session. Then, by using sufficiently many rewinding "slots," we can ensure that there exist some slots in each session where the adversary is guaranteed to not see new block (and hence no new interleaved sessions), making them "safe" for rewinding. Note, however, that this mechanism does not bound the overall number of concurrent sessions since an adversary can start any polynomial number of sessions *in parallel*.

Once we have the above protocol template, the key technical challenge is to perform concurrent extraction of the adversary's inputs in all of the sessions. Since there are multiple "unsafe" rewinding slots in every session (wherever a new block is mined), we need to extract adversary's inputs in all of the sessions under the constraint that only the safe slots are rewound. Unfortunately, commonly known rewinding strategies in the concurrent setting [RK99, KP01, PRS02] rewind all parts of the protocol transcripts (potentially multiple times). Therefore, they immediately fail in our setting.

In order to solve this problem, we develop a new concurrent rewinding strategy. The starting idea towards developing this rewinding strategy is the observation that our particular setting has some similarities to the work of Goyal et al. [GLP+15] who were interested in a seemingly unrelated problem: designing commitment schemes that are secure w.r.t. chosen commitment attacks [CLP10]. Goyal et al. introduced what they call the "robust extraction lemma" that guarantees concurrent extraction even if a *constant* number of "breakpoints" – that cannot be rewound – are interspersed throughout the overall transcript of the concurrent sessions. These breakpoints are analogous to the unsafe points in our setting.

While this serves as a useful starting point, robust extraction is not directly applicable to our setting since overall, the number of external blocks seen by the adversary (the equivalent of breakpoints in [GLP+15]) cannot be bounded. Indeed, if the number of sessions is T, the number of blocks can only be upper bounded by $T \cdot k$ (if e.g., all the sessions are sequential).

Our main observation is that the concurrent adversary can only choose one of the following: either too much concurrency, or too many newly mined blocks, but not both. This allows us to come up with a new variant and analysis of the robust extraction lemma which we believe could be of independent interest. In particular, our new variant uses twice as many slots as the one used by the robust extraction lemma. We refer the reader to the technical sections for more details.

5.2 Definitions and Preliminaries

Unless otherwise specified, we consider the adversaries that have access to the global functionality $\mathcal{G}_{\mathsf{ledger}}$, and thus the view includes messages received from and sent to $\mathcal{G}_{\mathsf{ledger}}$. Thus, when we denote that two distributions representing the views of parties with access to $\mathcal{G}_{\mathsf{ledger}}$ are computationally indistinguishable in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model, we give distinguisher access to the global $\mathcal{G}_{\mathsf{ledger}}$ functionality. An immediate consequence of this is that, any view generated by a simulator using a privately initialized $\mathcal{G}_{\mathsf{ledger}}$ functionality will be trivially distinguished from the real execution by the distinguisher that views the state of the global $\mathcal{G}_{\mathsf{ledger}}$.

5.2.1 Zero Knowledge in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model

Definition 35. *An interactive protocol* (P, V) *for a language* L *is zero knowledge in the* $\mathcal{G}_{\mathsf{ledger}}$*-hybrid model if the following properties hold:*

- **Completeness.** *For every* $x \in L$,

$$\Pr\Big[\ \mathsf{out}_{\mathsf{V}}\ \langle \mathsf{P}(x, w), \mathsf{V}(x)\rangle = 1\Big] = 1$$

- **Soundness.** *There exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.* $\forall x \notin L$ *and for all adversarial prover* P^*.

$$\Pr\Big[\ \mathsf{out}_{\mathsf{V}}\ \langle \mathsf{P}^*(x), \mathsf{V}(x)\rangle = 1\Big] \leq \mathsf{negl}(\lambda)$$

- **Zero Knowledge.** *For every* PPT *adversary* V^**, there exists a* PPT *simulator* Sim *such that the probability ensembles*
  - $\Big\{\mathsf{View}_{\mathsf{V}}\ \langle \mathsf{P}(x, w), \mathsf{V}(x, z)\rangle\ \Big\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*}$
  - $\Big\{\mathsf{Sim}(x, z)\Big\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*}$

  are computationally indistinguishable *in the* $\mathcal{G}_{\mathsf{ledger}}$*-model.*

5.2.2 Concurrently Secure Computation in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model

In this work, we consider a malicious, static adversary that chooses whom to corrupt before the execution of the protocol. The adversary controls the scheduling of the concurrent executions. We only consider *computational* security and therefore restrict our attention to adversaries running in probabilistic polynomial time. We denote computational indistinguishability by $\approx_c$, and the security parameter by λ. We do not require fairness and hence in the ideal model, we allow a corrupt party to receive its output in a session and then optionally block the output from being delivered to the honest party, in that session. Further, we only consider "security with abort". To formalize the above requirement and define security, we follow the standard paradigm for defining secure computation (see also [Lin08]). 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* by an adversary in the ideal model. More details follow.

IDEAL MODEL. We first define the ideal world experiment, where there is a trusted party for computing the desired two-party functionality $\mathcal{F} : \{0,1\}^{r_1} \times \{0,1\}^{r_2} \rightarrow \{0,1\}^{s_1} \times \{0,1\}^{s_2}$. Let P_1

and P_2 denote the two parties in a single execution. In total. let there be k parties $Q_1, Q_2, \cdots, Q_k$, where each party may be involved in multiple sessions with possibly interchangeable roles, i.e. Q_i may play the role of P_1 in one session and P_2 in some other session. Let the total number of executions be $m = m(n)$. For each $\ell \in [m]$, we will denote by P_1^ℓ, the party playing the role of P_1 in session ℓ. P_2^ℓ is defined analogously. The adversary may corrupt any subset of the parties in $Q_1, \ldots, Q_k$. The ideal world execution proceeds as follows:

I **Inputs:** There is a PPT *usage scenario* which gives inputs to all the parties. For each session $\ell \in [m]$, it gives inputs $x_\ell \in X \subseteq \{0,1\}^{r_1}$ to P_1^ℓ and $y_\ell \in Y \subseteq \{0,1\}^{r_2}$ to P_2^ℓ. The adversary is given auxiliary input $z \in \{0,1\}^*$, and chooses the subset of the parties to corrupt, say M. The adversary receives the inputs of the corrupted parties.

II **Session initiation:** When the adversary wishes to initiate the session number ℓ, it sends a $(\mathsf{start\text{-}session}, \ell)$ message to the trusted party. On receiving a message of the form $(\mathsf{start\text{-}session}, \ell)$, the trusted party sends $(\mathsf{start\text{-}session}, \ell)$ to both P_1^ℓ and P_2^ℓ.

III **Honest parties send inputs to the trusted party:** Upon receiving $(\mathsf{start\text{-}session}, \ell)$ from the trusted party, an honest party P_i^ℓ sends its real input along with the session identifier. More specifically, if P_1^ℓ is honest, it sends (ℓ, x_ℓ) to the trusted party. Similarly, an honest P_2^ℓ sends (ℓ, y_ℓ) to the trusted party.

IV **Corrupted parties send inputs to the trusted party:** At any point during execution, a corrupted part P_1^ℓ may send a message (ℓ, x'_ℓ) to the trusted party, for any string x'_ℓ (of appropriate length) of its choice. Similarly, a corrupted party P_2^ℓ sends (ℓ, y'_ℓ) to the trusted party, for any string y'_ℓ (of appropriate length) of its choice.

V **Trusted party sends results to the adversary:** For a session ℓ, when the trusted party has received messages from both P_1^ℓ and P_2^ℓ, it computes the output for that session. Let x'_ℓ and y'_ℓ be the inputs received from P_1^ℓ and P_2^ℓ, respectively. It computes the output $\mathcal{F}(x'_\ell, y'_\ell)$. If either P_1^ℓ or P_2^ℓ is corrupted, it sends $(\ell, \mathcal{F}(x'_\ell, y'_\ell))$ to the adversary. If neither of the parties is corrupted, then the trusted party sends the output message $(\ell, \mathcal{F}(x'_\ell, y'_\ell))$ to both P_1^ℓ and P_2^ℓ.

VI **Adversary instructs the trusted party to answer honest players:** For a session ℓ, where exactly one of the party is corrupted, the adversary, depending on its view up to this point, may send the message (output, ℓ) to the trusted party. Then, the trusted party sends the output $(\ell, \mathcal{F}(x'_\ell, y'_\ell))$, computed in the previous step, to the honest party in session ℓ.

VII **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 including messages exchanged with the $\mathcal{G}_{\mathsf{ledger}}$ functionality.

The ideal execution of a function $\mathcal{F}$ with security parameter λ, input vectors $\overrightarrow{x}, \overrightarrow{y}$, auxiliary input z to Sim and the set of corrupted parties M, denoted by $\mathsf{IDEAL}^{\mathcal{F}}_{M,\mathsf{Sim}}(\lambda, \overrightarrow{x}, \overrightarrow{y}, z)$, is defined as the output pair of the honest parties and the ideal world adversary Sim from the above ideal execution.

REAL MODEL. We now consider the real model in which a real two-party protocol is executed (and there exists no trusted third party). Let $\mathcal{F}, \overrightarrow{x}, \overrightarrow{y}, z$ be as above and let Π be a two-party protocol for computing $\mathcal{F}$. Let $\mathcal{A}$ denote a non-uniform probabilistic polynomial time adversary that controls any subset M of parties $Q_1, \ldots, Q_k$. The parties run concurrent executions of the protocol Π, where the honest parties follow the instructions of Π in all executions. The honest party initiates a new session

ℓ, using the input provided whenever it receives a start-session message from $\mathcal{A}$. The scheduling of all messages throughout the execution is controlled by the adversary. That is, the execution proceeds as follows: the adversary sends a message of the form (ℓ, msg) to the honest party. The honest party then adds msg to its view of session ℓ and replies according to the instructions of Π and this view in that session. 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 in the execution of the protocol, which includes messages exchanged with the $\mathcal{G}_{\mathsf{ledger}}$ functionality.

The real concurrent execution of Π with security parameter λ, input vectors $\overrightarrow{x}, \overrightarrow{y}$, auxiliary input z to $\mathcal{A}$ and the set of corrupted parties M, denoted by $\mathsf{REAL}^{\mathcal{F}}_{M,\mathcal{A}}(\lambda, \overrightarrow{x}, \overrightarrow{y}, z)$, is defined as the output pair of the honest parties and the real world adversary $\mathcal{A}$ from the above real world process.

Definition 36. *Let $\mathcal{F}$ and Π be as above. Then protocol Π for computing $\mathcal{F}$ is a concurrently secure computation protocol in the $\mathcal{G}_{ledger}$-hybrid model if for every probabilistic polynomial time adversary $\mathcal{A}$ in the real model, there exists a probabilistic polynomial time adversary* Sim *in the ideal model such that for every polynomial $m = m(n)$, every input vectors $\overrightarrow{x} \in X^m$, $\overrightarrow{y} \in Y^m$, every $z \in \{0,1\}^*$, and every subset of corrupt parties M, the following*

$$\left\{\mathsf{IDEAL}^{\mathcal{F}}_{M,\mathsf{Sim}}(\lambda, \overrightarrow{x}, \overrightarrow{y}, z)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathsf{REAL}^{\mathcal{F}}_{M,\mathcal{A}}(\lambda, \overrightarrow{x}, \overrightarrow{y}, z)\right\}_{\lambda\in\mathbb{N}}$$

holds in the $\mathcal{G}_{ledger}$-hybrid model.

5.2.3 (Multi-slot) Extractable Commitment Protocol $\langle C, R\rangle$

Let $\mathsf{Com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding string commitment scheme. Let λ denote the security parameter. The commitment scheme $\langle C, R\rangle$ between the committer C and the receiver R is described as follows. Note that we have already described Extractable Commitments schemes before (Section 4.2.2), but in this chapter we shall consider a *multi-slot* notion. The notion in Section 4.2.2 can be considered to be a single-slot variant of this notion.

Commit Phase: This consists of two stages, namely, the Init stage and the Challenge-Response stage, described below:

INIT: To commit to a n-bit string σ, C chooses $(\ell \cdot N)$ independent random pairs of n-bit strings $\{\alpha^0_{i,j}, \alpha^1_{i,j}\}^{\ell,N}_{i,j=1}$ such that $\alpha^0_{i,j} \oplus \alpha^1_{i,j} = \sigma$ for all $i \in [\ell], j \in [N]$. C commits to all these strings using Com, with fresh randomness each time. Let $B \leftarrow \mathsf{Com}(\sigma)$, and $A^0_{i,j} \leftarrow \mathsf{Com}(\alpha^0_{i,j})$, $A^1_{i,j} \leftarrow \mathsf{Com}(\alpha^1_{i,j})$ for every $i \in [\ell], j \in [N]$.

CHALLENGE-RESPONSE: For every $j \in [N]$, do the following:

- Challenge : R sends a random ℓ-bit challenge string $v_j = v_{1,j}, \ldots, v_{\ell,j}$.
- Response : $\forall i \in [\ell]$, if $v_{i,j} = 0$, C opens $A^0_{i,j}$, else 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 $\langle C, R\rangle$.

Notation. We introduce some terminology that will be used in the remainder of this work. We refer to the committed value σ as the *preamble secret*. A slot_i of the commitment scheme consists of the i'th Challenge message from R and the corresponding Response message from C. Thus, in the above protocol, there are N slots.

5.3 Black-box Zero Knowledge

In this section we will describe a $\omega(1)$ round zero-knowledge protocol that can be proven secure using a black-box simulator. We start with a description of the graph hamiltonicity proof protocol, and then build our protocol atop this protocol.

5.3.1 Graph Hamiltonicity Zero-knowledge Proof

As a starting point, we describe the the Hamiltonicity proof system [Blu87]. In the simplest setting, the prover proves the existence of a Hamiltonian cycle in graph. The description of the protocol can be found in figure 5.1.

Properties. The protocol is zero-knowledge when a single instance is run, and thus witness indistinguishable. Witness indistinguishability holds even when the protocol is run in parallel. In addition, the above protocol satisfies the notion of *special simulation*, where the simulator can trivially simulate the proof if it is aware of the verifier's challenge. This, similar to witness indistinguishability, holds even when the protocol is run in parallel.

Roughly, the idea to simulate the protocol when we know the challenge prior to the first message sent by the prover, is the following:

– If the challenge bit is 0, then commit to adjacency matrix of the permuted graph $\pi(G)$.

– On the other hand, if the challenge bit is 1 commit to the complete graph K_λ.

It is easy to see that the conditions for the corresponding challenge is met by the verifier. It also follows that the simulation holds when the protocol runs in parallel.

5.3.2 Our Protocol

The high level idea for our protocol is that the verifier commits to its challenge via the multi-round extractable commitment described in section 5.2.3, and reveals the challenge in place of the second round of the Hamiltonicity proof system. Since we are constructing a proof system where the prover has unbounded computational power, we require the commitment by the verifier to be statistically hiding so that an unbounded adversarial prover is not able to guess the challenge. We refer to the multi-round extractable commitment as the preamble.

In the preamble, the challenge committed to by the verifier is retrieved by rewinding the verifier in each of the slots. As long as the rewinding is successful in one of the slots, the committed challenge can be extracted. But in the presence of the blockchain (abstracted by the $\mathcal{G}_{\mathsf{ledger}}$ functionality) this becomes difficult. Consider a verifier that sends the challenge received by the prover in a given slot to $\mathcal{G}_{\mathsf{ledger}}$, and waits for the state to expand to include the challenge before responding to the challenge. It then checks in the state if there is another challenge from the prover for the same slot. If this is the case, it knows that it has been rewound, and will abort the protocol. Thus, in the

Hamiltonicity proof system

Common Input: A directed graph $G = (V, E)$ with $\lambda \overset{\text{def}}{=} |V|$.

Auxiliary Input for Prover: a directed Hamiltonian, $C \subset E$, in G.

1. Select a random permutation π, of the vertices V and using a statistically binding commitment scheme commit to the entries of the adjacency matrix of the permuted graph.
2. The verifier uniformly selects a bit σ and sends it to the prover.
3. The prover sends a message based on the value of σ it receives:
   - if $\sigma = 0$, the prover sends π along with the decommitment to all the values it had committed to earlier.
   - if $\sigma = 1$, the prover decommits only to entries in the permuted adjacency matrix $(\pi(u), \pi(v))$ with $(u, v) \in C$.
4. The verifier first checks if all the values decommitted to by the prover are valid (with respect to their corresponding commitment).
   - if $\sigma = 0$, the verifier checks if the revealed graph is the original graph permuted by π.
   - if $\sigma = 1$, the verifier checks if all the revealed values on the matrix is 1 and form a cycle.

   The verifier accepts if and only if both the initial checks, and the checks corresponding to the challenge verify .

Figure 5.1: Hamiltonicity proof system

simulated setting, the verifier will abort with a disproportionate probability in comparison to the real execution.

The trivial solution of not relaying messages from the verifier to $\mathcal{G}_{\mathsf{ledger}}$ on the look ahead threads does not work because the verifier can refuse to respond unless the state expands.

Thus, to overcome this issue, we design a protocol in the blockchain-hybrid model, where the protocol requires all parties to access $\mathcal{G}_{\mathsf{ledger}}$ in order to participate in the protocol. In our protocol, we just require that during the preamble, the local state of each party increases by at most k. But since parties may have different views of thus state, we must be careful when we claim the state size increase for other parties. But since $\mathcal{G}_{\mathsf{ledger}}$ guarantees that $|\mathsf{state}_{\mathsf{P}} - \mathsf{state}_{\mathsf{V}}| \leq \mathsf{windowSize}$, we are guaranteed that if the size of the state of one party increases by k, the size of the state of any other party can increase by at most $\mathsf{windowSize} + k$ (with maximum when both parties point to the head of the state initially).

If we set the number of rounds of the preamble to be $m > k + \mathsf{windowSize}$, we are guaranteed to have at least $m - (k + \mathsf{windowSize})$ slots where the state does not expand during the slot. For simplicity we assume k to be a constant, but our protocol can handle arbitrary k by scaling the number of rounds accordingly. The high level idea then is to just rewind in the slots where the state has not expanded, and thus the verifier does not expect the state to expand before it responds, and thus messages to or from $\mathcal{G}_{\mathsf{ledger}}$ can be kept from the verifier on the look ahead threads. Of course the exact number of rounds would depend on the exact simulator strategy. In our protocol, the number of rounds in the preamble is set to be $m = \omega(1)$. We should point out that $k > \mathsf{windowSize}$ to avoid trivial aborts in an honest execution of the protocol since otherwise the parties may start off with states that may then be k behind the head of the state, and in one computation step catches up to the head, thereby increasing local state size by k, and thus causing an abort. The complete protocol is presented in Figure 5.2.

Theorem 17. *The protocol* BCA-ZK *is a Zero-Knowledge Proof with black-box simulation in the* $\mathcal{G}_{\mathsf{ledger}}$-hybrid model.

Completeness. The parameter k for protocol needs to be selected appropriately such that honest provers do not "time-out" prior to completion of the first phase (PRS preamble). Once this is ensured, the completeness of the protocol follows immediately from the completeness of the underlying Hamiltonicity proof system. We set k to be a constant satisfying this property.

A honest prover sends random challenges in the first phase and performs appropriate checks. In the second phase, it uses its witness to the statement to answer the verifier's challenge.

Soundness. On a high level, soundness follows from the statistical hiding property of the commitment scheme and the soundness of the underlying Hamiltonicity proof system. Because of statistical hiding, other than with negligible probability, even an unbounded prover cannot break the hiding of the commitment scheme. The values revealed by the verifier are randomly chosen strings, independent of the the challenge string. Thus, other than with negligible probability, no information about the challenge string is revealed by the end of Phase I. The rest of the protocol then relies on the soundness of the Hamiltonicity proof system, which simply requires that the challenge from the verifier is random and unknown to the prover.

For the reduction, we use prover $\mathcal{P}$, breaking the soundness of our protocol, to construct a prover $\mathcal{P}_{\mathsf{HC}}$ that breaks the soundness of the underlying Hamiltonicity proof system.

$\mathcal{P}_{\mathsf{HC}}$ behaves as follows:

– Commit to random challenge in the initial commitment for σ and likewise commits to $2m\lambda$ random strings and send to $\mathcal{P}$. From the statistical hiding property, $\mathcal{P}$'s view is indistinguishable when $\mathcal{P}_{\mathsf{HC}}$ commits to random strings.

– Respond to $\mathcal{P}$'s challenges honestly.

– Forward $\mathcal{P}$'s first message in Phase II to the external challenge verifier.

– Let the challenge verifier respond with be $\widetilde{\sigma}$. We break the binding property of the the commitment scheme to find decommitments that are consistent with $\widetilde{\sigma}$. This applies to the initial commitment and the unopened commitments in Phase I.

– Send these decommitments to the prover.

– When the prover sends the third message in Phase II, relay it to the external challenge verifier. If $\mathcal{P}$ breaks soundness, so does $\mathcal{P}_{\mathsf{HC}}$.

Protocol BCA-ZK

Common Input: An instance x of a language L with witness relation R_L, the security parameter λ, the time out parameter k and the round parameter $m := m(\lambda)$.

Auxiliary Input for Prover: a witness w, such that $(x, w) \in R_L$, size of local state from the ledger $i_{\mathsf{P}} := |\mathsf{state}_{\mathsf{P}}|$.

Auxiliary Input for Verifier: size of local state from the ledger $i_{\mathsf{V}} := |\mathsf{state}_{\mathsf{V}}|$.

Phase I: Prior to each message sent in this phase, the respective party checks if the size of the state is such that $|\mathsf{state}_{\mathsf{P}}| < i_{\mathsf{P}} + k$ (correspondingly $|\mathsf{state}_{\mathsf{V}}| < i_{\mathsf{V}} + k$ for the verifier). If not, the party aborts.

1. Prover uniformly select a first message for a two round *statistically hiding commitment scheme* and send it to the verifier.
2. Verifier uniformly selects $\sigma \in \{0,1\}^\lambda$, and $m\lambda$ pairs of λ-bit strings $(\sigma^0_{\ell,p}, \sigma^1_{\ell,p})$ for $\ell \in [\lambda], p \in [m]$ such that $\forall \ell, p: \ \sigma^0_{\ell,p} \oplus \sigma^1_{\ell,p} = \sigma$. It commits to all $2m\lambda + 1$ selected strings using the statistically hiding commitment scheme. The commitments are denoted by $\alpha, \{\alpha^b_{\ell,p}\}_{b\in\{0,1\},\ell\in[\lambda],p\in[m]}$.
3. For $p = 1$ to m:
   (a) Prover sends an λ-bit challenge string $r_p = r_{1,p}, \dots, r_{\lambda,p}$ to the verifier.
   (b) Verifier decommits $\alpha^{r_{1,p}}_{1,p}, \dots, \alpha^{r_{\lambda,p}}_{\lambda,p}$ to $\sigma^{r_{1,p}}_{1,p}, \dots, \sigma^{r_{\lambda,p}}_{\lambda,p}$.
4. The prover proceeds with the execution if and only if all the decommitments send by the verifier are valid.

Phase II: The prover and verifier engage in λ parallel executions of the Hamiltonicity protocol as described below:

1. The prover sends the first message of the Hamiltonicity proof system.
2. The verifier decommits α to σ. And also reveals all $m\lambda$ commitments not decommitted to in the earlier phase.
3. The prover checks if decommitted values $\sigma, \{\sigma^b_{\ell,p}\}_{b\in\{0,1\},\ell\in[\lambda],p\in[m]}$ are valid decommitments. Additionally, check if $\forall \ell, p: \ \sigma^0_{\ell,p} \oplus \sigma^1_{\ell,p} = \sigma$. If any of the checks fail, abort. Else, send the third message of the Hamiltonicity proof system.
4. Verifier checks if all conditions of the Hamiltonicity proof system are met. It accepts if and only if this is the case.

Figure 5.2: Protocol for zero-knowledge proof in the blockchain aware setting.

Thus with only negligible probability difference, $\mathcal{P}_{\mathsf{HC}}$ breaks the soundness of the Hamiltonincity proof system.

We note that we need statistical hiding in Phase I because an all powerful prover should not be able to guess the challenge bits. And we require statistical binding in the Hamiltonicity proof in Phase II because we don't want the prover to be able to break binding property to change the adjacency matrix that it committed to.

Zero-knowledge. We need to construct a simulator Sim such that the following ensembles are computationally indistinguishable in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model

$$\left\{\mathsf{View}_{\mathsf{V}}\left\langle \mathsf{P}(x,w),\mathsf{V}(x,z)\right\rangle\right\}_{x\in L,w\in R_L(x),z\in\{0,1\}^*},\left\{\mathsf{Sim}(x,z)\right\}_{x\in L,w\in R_L(x),z\in\{0,1\}^*}.$$

We now describe the simulator in Figure 5.3.

Simulator Sim

1. Send the first message of the two round statistically hiding commitment scheme.
2. At any point in the simulation, if on the main thread the size of the state of *either* the prover or verifier increases by k, then quit and output the view of the verifier. Unless otherwise specified, the simulator relays messages between the $\mathcal{G}_{\mathsf{ledger}}$ and the verifier.
3. For each $p \in [m]$,
   - Select a random challenge string for the main thread, and $\lambda - 1$ challenge strings for the look-ahead threads. All the threads are run in parallel.
   - On the look ahead thread, no messages to or from the $\mathcal{G}_{\mathsf{ledger}}$ are relayed.
   - A slot on the look ahead thread terminates if the slot completes successfully, or if the size of the local state increased after the threads were created (or if the adversary aborts).

   Note that there might be look-ahead threads from earlier slots running in parallel while the main thread may have progressed further.
4. If in each of the look-ahead threads, the slot is terminated prior to completion (all $m \cdot (\lambda - 1)$ of them), output $\perp_{\mathsf{rewind}}$ and exit.
5. Since the abort condition does not hold, other than with negligible probability, the challenge string has been obtained. Use the challenge string to simulate Phase II of the protocol.

Figure 5.3: Simulator for zero-knowledge proof in $\mathcal{G}_{\mathsf{ledger}}$-hybrid model.

In order to prove zero-knowledge, we consider an intermediate simulator Sim_1 that receives a witness w to the statement x. Sim_1 on input x, w, z proceeds identically to the honest execution except that in Phase I, before the challenge message from the prover P in each slot, we sample $\lambda - 1$ other challenge messages and fork look ahead threads to be run in parallel. Thus, there are a total of $m \cdot (\lambda - 1)$ look ahead threads.

The main and look ahead threads are run in parallel. They are executed almost identically but for the following difference: In the look ahead threads, none of the queries made to the $\mathcal{G}_{\mathsf{ledger}}$ are forwarded to the $\mathcal{G}_{\mathsf{ledger}}$. As described in section 2.14, the READ queries to $\mathcal{G}_{\mathsf{ledger}}$ are simulated within each thread without having to query $\mathcal{G}_{\mathsf{ledger}}$. In addition, the look ahead threads are run until either a block is created, or the slot completes, whichever happens first (the adversary may also abort).

At the completion of Phase I, if none of the slots on the look ahead thread completed before being terminated, output $\perp_{\mathsf{rewind}}$. (i.e. for none of the slots look ahead thread completed successfully with a response from the verifier.)

We note that to optimize the simulator, we can stop creation of look ahead threads if at any point a look ahead thread successfully completes its slot. But for simplicity of exposition, we do not do so here.

Now, we claim the following:

Claim 72. *The following are statistically indistinguishable in the $\mathcal{G}_{ledger}$-hybrid model:*

$$\left\{\mathsf{View}_{\mathsf{V}}\left\langle\mathsf{P}(x,w),\mathsf{V}(x,z)\right\rangle\right\}_{x\in L,w\in R_L(x),z\in\{0,1\}^*},\left\{\mathsf{Sim}_1(x,w,z)\right\}_{x\in L,w\in R_L(x),z\in\{0,1\}^*}$$

Proof. Since the main thread remains unchanged and the only difference is the creation of look ahead threads, all we need to argue is that $\perp_{\mathsf{rewind}}$ is output with negligible probability. To this end, we make the following additional claim.

Claim 73. *The probability with which Sim_1 outputs $\perp_{\mathsf{rewind}}$ is negligible.*

Proof. We prove this by contradiction. Assume that the probability is noticeable. We note that Sim_1 outputs $\perp_{\mathsf{rewind}}$ only if Phase I completes, and in all the look ahead threads, the simulator is forced to terminate it before it completes (i.e. if the size of the state increased before the look ahead threads were completed). It should be noted that since the adversary (with restrictions) controls when the state expands, it could potentially attempt to cause the state to increase at different rates in the look ahead thread. This makes no difference to the analysis since this can just be thought of as the adversary waiting for the state to expand before completion of the slot. For ease of notation, we shall call a slot good if the state did not expand after the slot started, and before its completion. This can refer to either a slot in the main thread, or in the look ahead thread. Thus, by this notation, Sim_1 outputs $\perp_{\mathsf{rewind}}$ only if the preamble completes and none of the slots on the look ahead threads are good.

To prove this, we shall consider yet another intermediate simulator Sim_1'. It is identical to Sim_1, but instead of picking the main thread challenge, and then the look-ahead challenges, pick λ random strings and assign one of them to be the main thread challenge, and the remaining strings will be challenges on the look ahead thread. (The only difference is that in this case is that the main challenge is decided randomly from a set of challenges, as opposed to fixing a main thread and then choosing the look ahead threads.) Hence the probability that Sim_1' outputs $\perp_{\mathsf{rewind}}$ is identical to the probability that Sim_1 outputs $\perp_{\mathsf{rewind}}$.

To recall, Sim_1' outputs $\perp_{\mathsf{rewind}}$ if both the following conditions hold:

– $k' = (k + \mathsf{windowSize})$ blocks are not mined prior to completion of Phase I on the main thread. This ensures that at least $m - k'$ slots on the main thread are good.

– For each $p \in [m]$, every look ahead slot is not good.

Now let us look at the probability with which this happens. If the Phase I does not abort, then we know that there are at least $m - k$ slots on the main thread T that are good. And for the failure condition to be met, all the look-ahead slots at the same position terminate before completion (as the state expanded before it completed). Thus given the view prior to the slot, the corresponding good slot was chosen with probability $1/\lambda$. We can repeat this analysis for each such good slot on the main thread. Given that we choose the random string to be the main thread challenge independently for each $p \in [m]$, for any choice of m strings of length λ each we get:

$$\Pr\left[\mathsf{Sim}_1' \text{ outputs } \bot_{\mathsf{rewind}}\right] \leq \frac{1}{\lambda^{m-k}} \leq \lambda^k \cdot \mathsf{negl}(\lambda) \tag{5.1}$$

when $m = \omega(1)$.

Thus, from our assumption that k is a constant, we get the probability that Sim_1' outputs $\bot_{\mathsf{rewind}}$ to be negligible. This is also gives us the requirement that the number of rounds in the protocol must be $\omega(1)$. □

This gives us a contradiction to our assumption that the probability with which Sim_1 outputs $\bot_{\mathsf{rewind}}$ is non-negligible. □

Now in the last step, the only change from Sim_1 is that we use the special simulation property of Hamiltonicity proof system. Now this becomes identical to our described simulator. View indistinguishability follows trivially as a consequence. Thus we get the following claim.

Claim 74. *The following are computationally indistinguishable in the $\mathcal{G}_{ledger}$-hybrid model:*

$$\left\{\mathsf{Sim}_1(x, w, z)\right\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*} \textit{ and } \left\{\mathsf{Sim}(x, w, z)\right\}_{x \in L, w \in R_L(x), z \in \{0,1\}^*}$$

5.4 Concurrent Self Composable Secure Computation

In this section, we will construct a two-party protocol that is secure under concurrent self composition. We follow the line of works [CGJ15, GGJ13, GJO10] that rely on realizing an extractable commitment scheme that remains extractable even when there are multiple concurrent copies of this scheme in execution. Thus we construct our protocol in a two-step process. First, we describe a modified version of the multi-round extractable commitment preamble in the blockchain-hybrid model and show that we can extract from each session when multiple sessions are executed concurrently. Next, we plug our constructed concurrently extractable commitments into the compilers constructed in [CGJ15, GGJ13, GJO10] to achieve a concurrently secure two-party computation protocol.

5.4.1 Concurrently Extractable Commitment

In this section we present our construction of the concurrently extractable commitment scheme in the blockchain-hybrid model. We will refer to this as the modified PRS preamble. The idea for the modified PRS preamble is quite simple. Prior to starting the preamble, the party needs to post the first message to $\mathcal{G}_{\mathsf{ledger}}$. It is guaranteed that it will appear in the view of every party within the next $\Delta := 4 \cdot \mathsf{windowSize}$ blocks. Once the local state increase by Δ blocks, it sends the same message to the receiver. Posting to $\mathcal{G}_{\mathsf{ledger}}$ gives the party an “expiry period” of k-blocks after the Δ wait i.e., all slots of the preamble must be completed before the size of the state increases by a total of $\Delta + k$. As in the case of zero-knowledge, if the size of the state of a party increases by $\Delta + k$, for any other party the size of the state can have increased by at most $\Delta + k + \mathsf{windowSize}$, which is a constant when k is a constant. This needs to be taken into account when choosing the parameters ℓ and k. The formal description of the protocol is given below.

Protocol $\langle C, R\rangle_{\mathsf{BCA}}$

Common Input: The security parameter λ, the time-out parameter k, and the round parameter $2 \cdot \ell := \ell(\lambda)$.

Input to the Committer: the value σ to be committed, size of local state from the ledger $i_C := |\mathsf{state}_C|$.

Input to the Receiver: size of local state from the ledger $i_R := |\mathsf{state}_R|$.

Commitment:

1. Committer uniformly selects $\sigma \in \{0,1\}^\lambda$, and $2 \cdot \ell \cdot \lambda$ pairs of λ-bit strings $(\sigma^0_{\ell,p}, \sigma^1_{\ell,p})$ for $\ell \in [\lambda], p \in [2 \cdot \ell]$ such that $\forall \ell, p :\ \sigma^0_{\ell,p} \oplus \sigma^1_{\ell,p} = \sigma$. It generate commitments to all $2(2 \cdot \ell) \cdot \lambda + 1$ selected strings using the statistically binding commitment scheme. The commitments are denoted by $\alpha, \{\alpha^b_{\ell,p}\}_{b \in \{0,1\}, \ell \in [\lambda], p \in [2 \cdot \ell]}$. Send a SUBMIT query of these commitments to $\mathcal{G}_{\mathsf{ledger}}$. By our assumption, these will be guaranteed to appear in every party's state (at the same position) when $|\mathsf{state}_C| = i_C + \Delta$. Let it appear in index i of the state.
2. The committer sends to receiver the commitments along with the index i of the state that it appears in. The receiver verifies if the commitments were indeed in the designated index of the state.
3. Prior to each message subsequently sent, the respective party checks if the size of the state is such that $|\mathsf{state}_R| < i_R + k + \Delta$ (correspondingly $|\mathsf{state}_C| < i_C + k + \Delta$ for the committer). If not, the party aborts.

   For $p = 1$ to m:

   (a) Receiver sends an λ-bit challenge string $r_p = r_{1,p}, \ldots, r_{1^\lambda,p}$ to the committer.

   (b) Committer decommits $\alpha^{r_{1,p}}_{1,p}, \ldots, \alpha^{r_{\lambda,p}}_{\lambda,p}$ to $\sigma^{r_{1,p}}_{1,p}, \ldots, \sigma^{r_{\lambda,p}}_{\lambda,p}$.

5.4.2 Simulation-Extraction Strategy

In this section, we will describe a simulation-extraction strategy. The goal of this strategy is to extract the value committed by an adversary in every session of multiple concurrent executions of the modified preamble described above. We present a new concurrent strategy where our starting point is the simulator described in [GLP+15]. The relevant description has been reproduced here.

The scheduling of messages is controlled by the adversary $\mathcal{A}^*$, and when $\mathcal{A}^*$ sends the p-th message of a session s, it immediately receives the next message of s. The only exception to this are the special messages relevant to the $\mathcal{G}_{\mathsf{ledger}}$. In this case, $\mathcal{A}^*$ sends a message to $\mathcal{G}_{\mathsf{ledger}}$, and can proceed with the other parts of the execution. Once $\mathcal{G}_{\mathsf{ledger}}$ sends a message to $\mathcal{A}^*$, $\mathcal{A}^*$ immediately receives it. This is not universal across all queries to $\mathcal{G}_{\mathsf{ledger}}$ since a READquery is immediately responded to by $\mathcal{G}_{\mathsf{ledger}}$, and needs to be handled accordingly.

The state of $\mathcal{A}^*$ at any given point consists of its view up to that point. The starting state of $\mathcal{A}^*$ is denoted by sta_0, which is its state before it receives its first message. In addition, $\mathsf{LIVE}(\mathsf{sta})$ denotes the set of live sessions when the execution is at the state sta. When the preamble starts, $\mathcal{A}^*$ sends a START along with its commitment, and on successful completion of the preamble expects a message of the form (END, α) from the simulator. As described in [GLP+15], we will require this to be the value $\mathcal{A}^*$ committed in preamble.

The simulator Sim receives as auxiliary input a string $z \in \{0,1\}^*$, and the security parameter λ. It incorporates $\mathcal{A}^*$ as a black-box, and let $T = T(\lambda)$ be the maximum number of sessions (of the modified preambles) that are started by $\mathcal{A}^*$. Unlike the setting described in [GLP+15], the external messages refer to those from $\mathcal{G}_{\mathsf{ledger}}$. We let the execution on look ahead threads proceed when the adversary has sent a message to $\mathcal{G}_{\mathsf{ledger}}$ without forwarding the query, and only abort the thread when the size of the state increases. Recall, from the description of $\mathcal{G}_{\mathsf{ledger}}$, the adversary also receives SUBMIT messages every time an honest party submits something to $\mathcal{G}_{\mathsf{ledger}}$. These are passed on to $\mathcal{A}^*$ on both the main and look ahead threads. This stems from the fact that these can be replayed in each thread of execution. On the look ahead thread, when $\mathcal{A}^*$ makes read queries, we follow the strategy outlined in the model in section 2.14. These READ queries are answered local to a thread without having to pass on queries to $\mathcal{G}_{\mathsf{ledger}}$ as follows: The adversary, and thus the simulator, is aware of the transactions/data sent by honest parties since $\mathcal{G}_{\mathsf{ledger}}$ sends them to the adversary as and when they arrive. These messages can be replayed within threads. For the transactions/data sent by the adversary, the simulator maintains a local buffer for each thread, collecting but not forwarding these queries. When the look ahead threads make a READ query, combine the state from the start of the thread (since we have not aborted the thread, we're guaranteed that the state hasn't expanded), the honest SUBMIT queries along with the SUBMIT queries sent by the adversary **only local to that thread**. It is crucial to note that we do not make any changes to state.

Sim starts by setting $(1^\lambda, z)$ on $\mathcal{A}^*$'s input tape, and a sufficiently long uniform string on its random tape. Sim then starts the recurse procedure:

$$(\mathsf{sta}, \mathcal{T}) \leftarrow \mathsf{extract}(2 \cdot \ell \cdot T, \mathsf{sta}_0, \emptyset, 1, \emptyset, 0)$$

The terminology used in the recurse procedure are:

- t is the block length (of a block of recursion), and the base case will occur when $t = 1$.
- sta refers to the starting state.
- $\mathcal{T}$ refers to the table containing solutions.
- f is used to denote if the execution lies on the main thread. $f = 1$ if and only if the block lies on the main thread of execution.
- aux refers to auxiliary tables that are used in special cases (see [GLP+15] for details).
- id refers to the identity of a block of execution used to uniquely identify it.

Throughout its execution, messages of recurse are forwarded back and forth between $\mathcal{G}_{\mathsf{ledger}}$ and $\mathcal{A}^*$. The final output of Sim is the first output of recurse, namely sta which is also known as the main thread of execution.

```
Procedure $\mathsf{recurse}(t, \mathsf{sta}, \mathcal{T}, \mathsf{f}, \mathsf{aux}, \mathsf{id})$

1. If $t = 1$, repeat the following keeping in mind that if at any point, a session "expires"
   (state size has increased by $k + \Delta$ since session information posted to $\mathcal{G}_{\mathsf{ledger}}$), abort the
   session:
   (a) If the next message is START, check if relevant information is in the state and start
       a new session $s$.
         – send $r \leftarrow\$ \{0,1\}^n$ as the challenge of the first slot of $s$.
         – add entry $(s : 1, r, \_\_)$ to $\mathcal{T}$.
   (b) If the next message is the slot-$i$ challenge of an existing session $s$.
         – send $r \leftarrow\$ \{0,1\}^n$ as the slot-$i$ challenge of $s$.
         – add entry $(s : i, r, \_\_)$ to $\mathcal{T}$.
   (c) If the next message is the slot-$i$ response, say $\gamma$, of an existing session $s$.
         – If $\gamma$ is a valid message.
             – update entry $(s : i, r, \_\_)$ to $(s : i, r, \gamma)$.
             – if $i = 2 \cdot \ell$, i.e., it is the last slot, send $(\mathsf{END}, \mathsf{extract}(s, \mathsf{id}, \mathcal{T}, \mathsf{aux}))$.
         – Otherwise, if $\gamma = \perp$, abort session $s$ and add $(s : \perp, \perp, \perp)$ to $\mathcal{T}$.
         – Update sta to be the current state of $\mathcal{A}^*$
         – return $(\mathsf{sta}, \mathcal{T})$.
   (d) If the next message is a message from $\mathcal{A}^*$ to the $\mathcal{G}_{\mathsf{ledger}}$
         – If READ message, drop the message, simulate the READ response (as de-
           scribed) and continue.
         – If $\mathsf{f} = 0$, i.e., it is a look ahead block, then drop the message (stop it from
           reaching $\mathcal{G}_{\mathsf{ledger}}$) and continue.
         – If $\mathsf{f} = 1$, i.e., it is the main thread, then forward the message to $\mathcal{G}_{\mathsf{ledger}}$, and
           continue.
   (e) If the next message is an expanded state from the $\mathcal{G}_{\mathsf{ledger}}$ to $\mathcal{A}^*$
         – If $\mathsf{f} = 0$, i.e., it is a look ahead block, then return $(\mathsf{sta}, \mathcal{T})$.
         – If $\mathsf{f} = 1$, i.e., it is the main thread, do the following:
             – Update sta to be the current state of $\mathcal{A}^*$
             – For every live session $s \in \mathsf{LIVE}(\mathsf{sta})$, do the following:
                 – $\times_{s,\mathsf{id}} = \mathsf{true}$
                 – for every block $\mathsf{id}'$ that contain the block id, set $\times_{s,\mathsf{id}'} = \mathsf{true}$.
   (f) If other messages from $\mathcal{G}_{\mathsf{ledger}}$, pass to $\mathcal{A}^*$ and continue.
2. If $t > 1$,
     # Rewind the first half twice
   (a) $(\mathsf{sta}_1, \mathcal{T}_1) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}, \mathcal{T}, 0, \mathsf{aux}, \mathsf{id} \circ 1)$          [look-ahead block $C'$]
   (b) Let $\mathsf{aux}_2 := (\mathsf{aux}, \mathcal{T}_1 \setminus \mathcal{T})$,
       $(\mathsf{sta}_2, \mathcal{T}_2) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}, \mathcal{T}, \mathsf{f}, \mathsf{aux}_2, \mathsf{id} \circ 2)$          [main block $C$]
```

```
    # Rewind the second half twice
    (c) Let $\mathcal{T}^* := \mathcal{T}_1 \cup \mathcal{T}_2$,
        $(\mathsf{sta}_3, \mathcal{T}_3) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}_2, \mathcal{T}^*, 0, \mathsf{aux}, \mathsf{id} \circ 3)$        [look-ahead block $D'$]
    (d) Let $\mathsf{aux}_2 := (\mathsf{aux}, \mathcal{T}_1 \setminus \mathcal{T})$,
        $(\mathsf{sta}_4, \mathcal{T}_4) \leftarrow \mathsf{recurse}(t/2, \mathsf{sta}_2, \mathcal{T}^*, \mathsf{f}, \mathsf{aux}_4, \mathsf{id} \circ 4)$        [main block $D$]

    (e) return $(\mathsf{sta}_4, \mathcal{T}_3 \cup \mathcal{T}_4)$.
```

```
Procedure $\mathsf{extract}(s, \mathsf{id}, \mathcal{T}, \mathsf{aux})$

  1. Attempt to extract a value for $s$ from $\mathcal{T}$.
  2. If extraction fails, consider every block $\mathsf{id}_1$ for which $\times_{s,\mathsf{id}_1} = \mathsf{true}$.
       – Let $\mathsf{id}_1'$ be the sibling of $\mathsf{id}_1$, with input/output tables $\mathcal{T}_{\mathsf{in}}, \mathcal{T}_{\mathsf{out}}$ respectively.
       – Attempt to extract from $\mathsf{aux}_{\mathsf{id}_1'} := \mathcal{T}_{\mathsf{out}} \setminus \mathcal{T}_{\mathsf{in}}$;     (included in aux).
  3. If all attempts fail, abort the simulation and return ExtractFail.
     Otherwise return the extracted value.
```

Although we describe the simulator in terms of a recursive rewinding strategy, to ensure that the adversary does not gain any side channel leakage in the form of increase of state size for the state in $\mathcal{G}_{\mathsf{ledger}}$, all threads are run in parallel. The total number of threads is given by the recursion $h(2 \cdot \ell \cdot T(\lambda)) \leq 4 \cdot h(\ell \cdot T(\lambda))$ which gives us $h(2 \cdot \ell \cdot T(\lambda)) \leq (\ell \cdot T(\lambda))^2 \mathsf{poly}(n)$ threads. Thus if the total number of sessions and slots are polynomial, we have only polynomial many threads. And by our assumption of the simulator having access to arbitrary polynomial parallelism, we can run this threads in parallel.

Claim 75. Sim *succeeds other than with negligible probability.*

To prove this claim, we shall rely on the following robust extraction lemma. Informally, the lemma states that there exists a simulator that can extract the commitment made by the adversary without having to rewind an external protocol Π. The lemma states this by describing an online extractor $\mathcal{E}$ which can run in super-polynomial time to extract the committed value. We refer the reader to [GLP+15] for further details.

Lemma 5 (Robust Extraction Lemma [GLP+15]). *There exists an interactive Turing Machine* Sim *("robust simulator") such that for every $\mathcal{A}^*$, for every $\Pi = \langle B, A \rangle$, there exists a party $\mathcal{E}$ ("online extractor"), such that for every $\lambda \in \mathbb{N}$, for every $x \in \mathsf{dom}_B(\lambda)$, and every $z \in \{0,1\}^*$, the following conditions hold:*

1. **Validity constraint.** *For every output ν of $\mathsf{REAL}^{\mathcal{A}^*}_{\mathcal{E},\Pi}(\lambda, x, z)$, for every* PRS *preamble s (appearing in ν) with transcript τ_s, if there exists a unique value $v \in \{0,1\}^\lambda$ and randomness ρ such that $\mathsf{open}_{\mathsf{PRS}}(\tau_s, v, \rho) = 1$, then*
$$\alpha_s = v,$$
where α_s is the value $\mathcal{E}$ sends at the completion of preamble s.

2. **Statistical simulation.** *If $k = k(\lambda)$ and $\ell = \ell(\lambda)$ denote the round complexities of Π and the* PRS *preamble respectively, then the statistical distance between distributions* $\mathsf{REAL}^{\mathcal{A}^*}_{\mathcal{E},\Pi}(\lambda, x, z)$ *and* $\mathsf{out}_s \left\langle B(1^\lambda, x), \mathsf{Sim}^{\mathcal{A}^*}(1^\lambda, z) \right\rangle$ *is given by:*

$$\Delta(n) \leq \frac{1}{2^{\Omega(\ell - k \cdot \log T(n))}},$$

where $T(n)$ is the maximum number of total PRS *preambles between $\mathcal{A}^*$ and $\mathcal{E}$. Further the running time of* Sim *is* $\mathsf{poly}(\lambda) \cdot T(n)^2$.

The reason the lemma doesn't directly apply to our setting is that there needs to be a "gap" between the number of slots of the preamble and the number of external messages. For instance, if the number of external messages are constant and the number of slots super-constant, we get a simulator that fails with negligible probability. Unfortunately in our setting, we can loosely upper bound the number of external state expansion messages by $T \cdot k$, which is no longer a constant.

To see why this our previous point about simulating the buffer is important important, each READ query and its corresponding response from $\mathcal{G}_{\mathsf{ledger}}$ will count as an external message by our definition. Since there is no prior bound on the number of such queries the adversary can make, we cannot hope to use the approaches listed above directly.

The proof of the above claim follows an argument of contradiction. Assume there is an adversary $\mathcal{A}$ for which the above simulator Sim fails, we shall construct using $\mathcal{A}$ and Sim, a new adversary $\widetilde{\mathcal{A}}$ such that there are only a constant number of external messages, but the simulator $\widetilde{\mathsf{Sim}}$ described in [GLP+15] fails with noticeable probability, thus violating the robust extraction lemma.

As a matter of technicality, and for ease of proof, the adversaries $\mathcal{A}$ and $\widetilde{\mathcal{A}}$ participate in slightly different preambles. $\mathcal{A}$ participates in preambles that have $2 \cdot \ell$ slots, while $\widetilde{\mathcal{A}}$ participates in preambles that have only ℓ slots. Thus, care must be taken when we $\widetilde{\mathcal{A}}$ forwards messages from $\mathcal{A}$.

Before we describe the constructed adversary, $\widetilde{\mathcal{A}}$, we introduce some notation. Let sta_i be the state of the adversary $\mathcal{A}$ on the main thread, when the START_i message is sent on this thread. We partitions the set $\mathsf{LIVE}(\mathsf{sta}_i)$ into two sets $\mathsf{HALF}(\mathsf{sta}_i)$ and $\mathsf{HALF}^c(\mathsf{sta}_i) = \mathsf{LIVE}(\mathsf{sta}_i) \setminus \mathsf{HALF}(\mathsf{sta}_i)$, where $\mathsf{HALF}(\mathsf{sta}_i)$ is the set of all preambles that have completed at least half (ℓ) of their slots (but not all of them since they're in $\mathsf{LIVE}(\mathsf{sta}_i)$). Intuitively, these preambles for these sessions already contain enough information on the tables generated by Sim, and thus there is no need to forward them to $\widetilde{\mathsf{Sim}}$.

Recall, in the preamble, the first message that is sent along with START, is the commitment to v and $2 \cdot \ell \cdot \lambda$ pairs $\left(v^0_{i_1,i_2}, v^1_{i_1,i_2}\right)$ for , $i_1 \in [2 \cdot \ell], i_2 \in [\lambda]$ such that $\forall i_1 \in [2 \cdot \ell], i_2 \in [\lambda]\ \ v^0_{i_1,i_2} \oplus v^1_{i_1,i_2} = v$. We denote this message for a session u to be ExtCom_u. Additionally, we denote by $\mathsf{ExtCom}_u[p : p{+}i_3]$ for $p + i_3 \leq 2 \cdot \ell$, the (truncated) commitment consisting of the commitment to v as before and of $(i_3 + 1) \cdot \lambda$ pairs $\left(v^0_{i_1,i_2}, v^1_{i_1,i_2}\right)$ for $i_1 \in [p, p + i_3]\,, i_2 \in [\lambda]$. Thus $\mathsf{ExtCom}_u = \mathsf{ExtCom}_u[1 : 2 \cdot \ell]$.

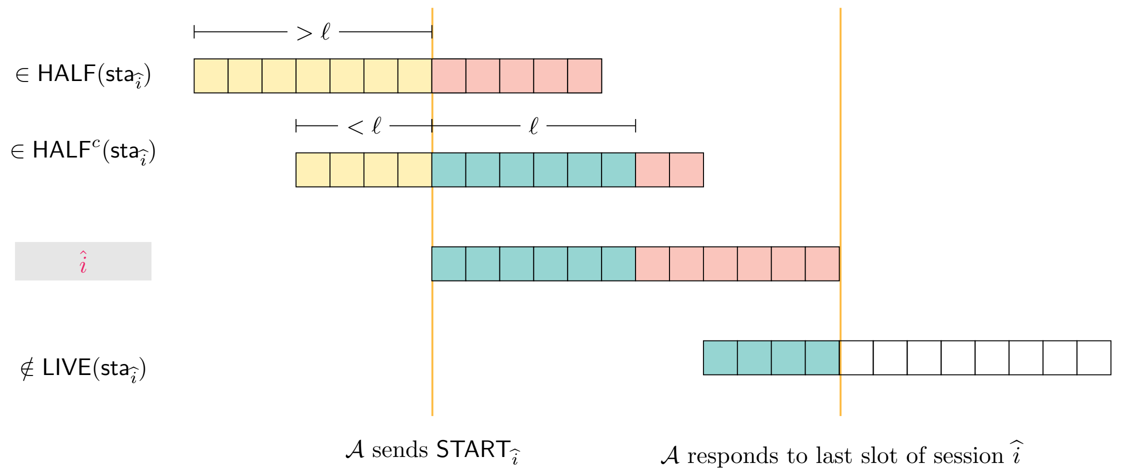
Consider session $j \in \mathsf{HALF}^c(\mathsf{sta}_i)$. Let p^i_j be the slot of session j for which $\mathcal{A}$ received a challenge, but did not send a response. i.e. $p^i_j - 1$ slots were completed in session j. Given that $j \in \mathsf{HALF}^c(\mathsf{sta}_i)$, we have $1 \leq p^i_j \leq \ell$. We will use $\mathsf{ExtCom}_u[p^i_j + 1 : p^i_j + \ell]$ as the preamble commitment for $\widetilde{\mathcal{A}}$. This leaves slots $p^i_j + \ell + 1$ to $2 \cdot \ell$ when $p^i_j < \ell$ that need to be dealt with appropriately.

Given the notation and the idea described, we construct the adversary $\widetilde{\mathcal{A}}$ as follows:

```
adversary $\widetilde{\mathcal{A}}(1^\lambda, z)$

1. Guess the slot $\widehat{i}$ for which the simulator Sim fails to extract from $\mathcal{A}$.
2. Initialize Sim with random coins and auxiliary input $z$. Sim will in turn initialize $\mathcal{A}$ to
   use in a black-box manner.
3. Prior to $\widetilde{\mathcal{A}}$ sending out any messages, it executes the interaction between $\mathcal{A}$ and Sim up
   to the point that $\mathcal{A}$ sends $\mathsf{START}_{\widehat{i}}$ on the main thread. It does so by relaying messages
   between Sim and $\mathcal{A}$ in each parallel execution till $\mathcal{A}$ outputs $\mathsf{START}_{\widehat{i}}$. During the execu-
   tion of the interaction between Sim and $\mathcal{A}$, if Sim send messages to the $\mathcal{G}_{\mathsf{ledger}}$, $\widetilde{\mathcal{A}}$ sends
   this message too. When $\widetilde{\mathcal{A}}$ receives a state expansion message from the $\mathcal{G}_{\mathsf{ledger}}$, this is
   forwarded to Sim.
4. On receiving $\mathsf{START}_{\widehat{i}}$ and the corresponding $\mathsf{ExtCom}_{\widehat{i}}$,
     output $\mathsf{START}_{\widehat{i}}$ and $\mathsf{ExtCom}_{\widehat{i}}[1:\ell]$
     forward response received to $\mathcal{A}$.
5. For all other messages, the behavior is defined as follows:
   – If the next message from $\mathcal{A}$ is a $\mathsf{START}_j$ message for session $j$,
       output $\mathsf{START}_j$ and $\mathsf{ExtCom}_j[1:\ell]$
       forward response received to $\mathcal{A}$.
   – If the next message from $\mathcal{A}$ is the slot-$p$ response, say $\gamma$, of session $j$.
     – If $j \in \mathsf{HALF}(\mathsf{sta}_{\widehat{i}})$      //sessions with at least half the slots completed
       – if $p < 2 \cdot \ell$
           respond internally with the challenge for slot $p+1$.
       – if $p = 2 \cdot \ell$      //last slot of the main preamble
           use Sim's extract procedure to extract the value $v_j$ committed in session
         $j$.
           respond with $(\mathsf{END}_j, v_j)$
     – If $j \in \mathsf{HALF}^c(\mathsf{sta}_{\widehat{i}})$      //sessions with at least half the slots remaining
       – if $p = p_j^{\widehat{i}}$,
           output $\mathsf{START}_j$ and $\mathsf{ExtCom}_j[p+1 : p+\ell]$.
           forward response received to $\mathcal{A}$.
       – if $p_j^{\widehat{i}} < p < p_j^{\widehat{i}} + \ell$,
           output $\gamma$.
           forward response received to $\mathcal{A}$.
       – if $p = p_j^{\widehat{i}} + \ell$,      //last slot of modified preamble
         output $\gamma$.
         on receiving $(\mathsf{END}_j, \widetilde{v_j})$, store $\widetilde{v_j}$ and respond internally with the chal-
         lenge for slot $p+1$.
       – if $p_j^{\widehat{i}} + \ell < p < 2 \cdot \ell$,
           respond internally with the challenge for slot $p+1$.
```

```
        – if $p = 2 \cdot \ell$,      //last slot on main preamble
              use the stored value $\widetilde{v_j}$ and respond with $(\mathsf{END}_j, \widetilde{v_j})$.
    – If $j \notin \mathsf{LIVE}(\mathsf{sta}_{\widehat{i}})$      #sessions started after (and including) session $\widehat{i}$
        – if $p < \ell$,
              output $\gamma$.
              forward response received to $\mathcal{A}$.
        – if $p = \ell$,      //last slot of modified preamble
              output $\gamma$.
              on receiving $(\mathsf{END}_j, \widetilde{v_j})$, store $\widetilde{v_j}$ and respond internally with the challenge for slot $p + 1$.
        – if $\ell < p < 2 \cdot \ell$,
              respond internally with the challenge for slot $p + 1$.
        – if $p = 2 \cdot \ell$,      //last slot on main preamble
              use the stored value $\widetilde{v_j}$ and respond with $(\mathsf{END}_j, \widetilde{v_j})$.
  – If the next message from $\mathcal{A}$ is a message to $\mathcal{G}_{\mathsf{ledger}}$, output this query.
  – On receiving state expansion message from the $\mathcal{G}_{\mathsf{ledger}}$, forward to $\mathcal{A}$.
6. Quit on receiving response to the last slot in session $\widehat{i}$.
```



$> \ell$
$\in \mathsf{HALF}(\mathsf{sta}_{\widehat{i}})$
$< \ell$
ℓ
$\in \mathsf{HALF}^c(\mathsf{sta}_{\widehat{i}})$
$\widehat{i}$
$\notin \mathsf{LIVE}(\mathsf{sta}_{\widehat{i}})$
$\mathcal{A}$ sends $\mathsf{START}_{\widehat{i}}$
$\mathcal{A}$ responds to last slot of session $\widehat{i}$

Figure 5.4: We illustrate above the different types of sessions as described earlier, where the slots are colored to indicate how $\widetilde{\mathcal{A}}$ deals with them: (i) 🟨 - the responses to these slots are obtained by passing messages between $\mathcal{A}$ and Sim; (ii) 🟥 - the responses to these slots to $\mathcal{A}$ are answered internally by $\widetilde{\mathcal{A}}$ generating a random challenge and discarding the response without passing message to Sim; (iii) 🟩 - $\widetilde{\mathcal{A}}$ exposes these slots to the external $\widetilde{\mathsf{Sim}}$, by conveying challenges and responses between $\widetilde{\mathsf{Sim}}$ and $\mathcal{A}$ (note that only ℓ such slots exist in each session); and (iv) ☐ - these slots are never run since $\widetilde{\mathcal{A}}$ stops once the last slot response of session $\widehat{i}$ is received.

Figure 5.4 provides an illustration of the strategy employed by $\widetilde{\mathcal{A}}$. At a high level, $\widetilde{\mathcal{A}}$ does not make any calls to Sim once $\mathcal{A}$ sends $\mathsf{START}_{\widehat{i}}$. For the sessions in $\mathsf{HALF}(\mathsf{sta}_{\widehat{i}})$, we rely on the fact that Sim has responses for at least half the slots in these sessions - enabling it to extract since any constant fraction of sessions should allow Sim to extract. Due to the time-out mechanism, there can only be a constant number of external messages once $\mathcal{A}$ sends $\mathsf{START}_{\widehat{i}}$. We would like to make a minor technical note at this point. While we stated that we wanted an adversary $\widetilde{\mathcal{A}}$ that receives only a constant number of external messages (state expansion messages from $\mathcal{G}_{\mathsf{ledger}}$), prior to $\widetilde{\mathcal{A}}$ sending any messages for the session, it waits for some additional external messages. Unfortunately, these may not be a constant. But, this makes little difference as the "core" of the transcript still contain only a constant number of external messages, and rewinding at these points do not affect the external messages sent early on.

5.4.3 The Protocol

In this section, we describe our concurrent secure computation protocol Π in the $\mathcal{G}_{\mathsf{ledger}}$-hybrid model for a general functionality $\mathcal{F}$. Our protocol is, in fact, the same as the one presented in [GJO10, GGJ13, CGJ15], except that we use the concurrently extractable commitment from Section 5.4.1. Indeed, the core ingredient of the compiler in [GJO10] (which is also used in [GGJ13, CGJ15]) is a concurrently extractable commitment, and in particular, it follows from these works that if there exists a concurrent simulator for the extractable commitment, then the resultant compiled protocol securely evaluates the function $\mathcal{F}$.

For completeness, we recall the protocol here. The proof of security for our case follows in essentially an identical fashion to [GJO10] with the main difference being that our simulator only performs a single ideal world query per session (while the simulator performs multiple ideal world queries per session in their work). We discuss other minor differences in Section 5.4.3.2.

5.4.3.1 Building Blocks

Statistical Binding String Commitments. 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, however, in the presentation of our results, we will use a non-interactive perfectly binding string commitment. Let $\mathsf{Com}(\cdot)$ denote the commitment function of the string commitment scheme.

Statistical Witness Indistinguishable Arguments. 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].

Specifically, in our discussion of the Blum Hamiltonicity protocol (Section 5.3.1), the prover used a statistically binding commitment to commit to the (permuted) graph. This ensured that the resultant protocol was a *proof*. If one were to replace the commitment with a commitment scheme that was *statistically* hiding but *computationally* binding, then we have a *statistical* zero-knowledge argument (as opposed to a computational zero knowledge proof) where the soundness is 1/2. By [FS90] since (i) every zero-knowledge protocol is also witness indistinguishable while maintaining the corresponding security property (i.e. computational/statistical/perfect); and (ii) composes under parallel repetition; we have that $\omega(\log k)$ parallel repetitions brings the soundness error to be negligible while maintaining statistical witness indistinguishability.

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 ideal functionality $\mathcal{F}$ in the stand-alone setting. The existence of such a protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ follows from [Yao86, GMW87a, Kil88].

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 requiring 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.

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 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 $\omega(\log n)$-round extractable commitment scheme [PRS02], we will use the N-round extractable commitment scheme $\langle C, R \rangle$ (described in Section 5.2.3). (b) Further, we require that the non-malleable commitment scheme being used in the protocol be public-coin w.r.t. receiver[2]. We now describe the 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 $x \in L$. P additionally has a private input w (witness for x). 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 execution of $\langle C, R \rangle$ (Section 5.2.3) 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).

[2]The original NMZK 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 [DDN91] 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 x (i.e., $R(x, w) = 1$, 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.

Remark 13. *We discussed above how one can repeat the Blum Hamiltonicity protocol with statistically hiding commitments to achieve SWI. If one were to repeat the Hamiltonicity protocol with statistically hiding commitments* sequentially *then we have a statistical zero-knowledge argument since zero-knowledge protocols compose sequentially. The Blum Hamiltonicity protocol is also an argument of knowledge since an extractor can obtain a witness from any two accepting transcripts.*

Modified Extractable Commitment Scheme $\langle C', R'\rangle$ Due to technical reasons, in our secure computation protocol, we will also use a minor variant, denoted $\langle C', R'\rangle_{\mathsf{BCA}}$, of the extractable commitment scheme presented in 5.4.1. Protocol $\langle C', R'\rangle_{\mathsf{BCA}}$ is the same as $\langle C, R\rangle_{\mathsf{BCA}}$, 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 $\langle C', R'\rangle_{\mathsf{BCA}}$, on receiving a challenge string $v_j = v_{1,j}, \ldots, v_{\ell,j}$ from the receiver, the committer uses the following strategy: for every $i \in [\ell]$, 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 $\langle C', R'\rangle_{\mathsf{BCA}}$ 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".

We remark that the extraction proof for the simulation-extraction procedure also holds for the $\langle C', R'\rangle_{\mathsf{BCA}}$ commitment scheme.

5.4.3.2 Protocol Description

Notation. Let $\mathsf{Com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding commitment scheme. Let $\langle C, R\rangle_{\mathsf{BCA}}$ denote the N-round extractable commitment scheme and $\langle C', R'\rangle_{\mathsf{BCA}}$ be its modified version as described above. For the description, we drop the subscript and refer to them as $\langle C, R\rangle$ and $\langle C', R'\rangle$ respectively. Let $\langle P, V\rangle$ denote the modified version of the CNMZK argument of Barak et al. [BPS06]. Further, let $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$ denote a SWI argument and let $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ denote a semi-honest two party computation protocol $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ 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 n be the security parameter. The protocol proceeds as follows.

Protocol BCA-CONC

I. Trapdoor Creation Phase.

1. $P_1 \Rightarrow P_2$: P_1 creates a commitment $\mathsf{com}_1 = \mathsf{Com}(0)$ 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 $\mathsf{com}_2 = \mathsf{Com}(0)$ 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 $\beta_{1,j} = P_1^{\mathsf{sh}}(T_{1,j}, x_1, r_1'')$ 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) $\beta_{1,j} = P_1^{\mathsf{sh}}(T_{1,j}, \hat{x}_1, \hat{r}_1 \oplus r'_1)$

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

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

Proof of Security. Our proof of security follows in almost an identical fashion to [GJO10, GGJ13, CGJ15]. The main difference is that due to the property of our concurrent extractor (Section 5.4.2), our simulator only needs to make one ideal world query per session (as opposed to multiple ideal world queries). Indeed, this is why we achieve standard concurrent security, while [GJO10, GGJ13, CGJ15] achieve security in the so-called multiple-ideal-query model.

Our indistinguishability hybrids also follow in the same manner as in [GJO10, GGJ13, CGJ15]. There is one minor difference that we highlight. The hybrids of [GJO10, GGJ13, CGJ15] maintain a "soundness invariant", where roughly speaking, it is guaranteed that whenever an honest party changes its input in any sub-protocol used within the secure computation protocol, the value committed by the adversary in the non-malleable commitment (inside the CNMZK) does not change, except with negligible probability. In some hybrids, this property is argued via extraction from the non-malleable commitment.

In our setting, we have to be careful with such an extraction since a blockchain-active adversary may try to keep state using $\mathcal{G}_{\mathsf{ledger}}$. However, the key point is that for such a soundness argument, the reduction can use a locally initialized $\mathcal{G}_{\mathsf{ledger}}$ that it controls (and can therefore modify arbitrarily). This follows from the fact that we do not care about the view of an adversary in such a reduction to be indistinguishable to a distinguisher that has access to $\mathcal{G}_{\mathsf{ledger}}$. In fact, it will trivially be distinguishable. But since a locally initialized $\mathcal{G}_{\mathsf{ledger}}$ is indistinguishable to the adversary that is simply allowed to interact using the given interface (i.e. efficiently simulatable), the adversary's behavior does not change. Using this idea, we can perform extraction as in the plain model.

5.5 Impossibility of Constant Round Black-Box Zero Knowledge

In this section, we prove the impossibility of constant round zero-knowledge protocols w.r.t. black-box simulation in the blockchain-hybrid model. The starting point for our result is the beautiful work by Barak and Lindell[BL02] who showed that it is impossible to construct (non-trivial) constant round zero knowledge arguments or proofs with respect to black-box simulation if the simulator runs in strict polynomial time. At a high level, their impossibility result constructs a verifier with appropriate probability of abort in a given step such that with noticeable probability an honest execution will complete, but the simulator "runs out of time" when it attempts to gain any advantage over an honest prover. While they prove their impossibility result w.r.t. strict polynomial time black-box simulators, we extend their impossibility in our blockchain-hybrid model to stronger classes of simulators. In addition to simulators which run in strict polynomial time, we consider two additional classes of simulators: (1) simulators that can run in expected polynomial time, but have an a priori bounded memory, and hence at any time, can make a fixed polynomial number of queries to the adversary by running them in parallel; and (2) simulators that can run in expected polynomial time, but can make an unbounded number of queries to the adversary by running them in parallel. It is easy to see that the second class is stronger than the first. These results complement our positive results and demonstrate that our constructed protocols are tight.

For our setting, a *universally constant* round protocol will be such that (i) the number of rounds are constant; (ii) and there is a constant upper bound on the size that the state can increase by during an execution of the protocol.

For a constant round protocol, let us first consider the simplest setting where the simulator can only run for some strict polynomial time. In addition, it can make a fixed number of polynomial queries in parallel to the adversary by making copies. While the polynomial can be arbitrary, it is fixed in advance. Thus, the effective number of computation steps for the simulator is a strict polynomial. We can thus apply directly the result from [BL02] to construct a verifier that prevents the simulator from gaining any advantage over an honest prover.

Bounded memory simulator. Now we let us consider an expected polynomial time simulator with bounded memory. This means that at any given time, the simulator may only have a strict polynomially bounded number of parallel executions. To invoke the result from Barak-Lindell [BL02], we need to describe a verifier strategy that forces the simulator to always run in strict polynomial time. Intuitively this means that for the given verifier strategy, any simulator running in super-polynomial time would leak side-channel information, i.e. the verifier would realize it was being run in super-polynomial time. We assume that a k-round protocol gives some upper bound r on how much the size of the state can expand during the execution of the protocol. This is enforced by the ExtendPolicy function. See appendix A.1.

We describe below an adversarial verifier strategy that forces the simulator to run in strict polynomial time:

- At the start of the protocol, the verifier obtains the state $\mathsf{state}_{\mathsf{V}}$ from $\mathcal{G}_{\mathsf{ledger}}$. Let the size of the state be $i_{\mathsf{V}} := |\mathsf{state}_{\mathsf{V}}|$.
- Behave according to underlying honest strategy.
- When the prover sends its last message p_ℓ, the verifier sends the signed transcript to $\mathcal{G}_{\mathsf{ledger}}$, and waits for $\mathsf{state}_{\mathsf{V}}$ to include the transcript. Let i^* be the index of the state that the transcript appear in. It then checks if $i^* - i_{\mathsf{V}}$ is larger than r, if so it outputs a special abort symbol $\perp$.

Note that unlike plain model, due to the presence of $\mathcal{G}_{\mathsf{ledger}}$, a verifier's view is not completely determined by the messages it receives from the prover.

Given that the interval between state being expanded is some polynomial (since the adversary controls this in a restricted manner), a super-polynomial running time would ensure that the increase in state size is not a constant. Thus, to avoid a trivial distinguisher that looks for the special abort, the simulator must run in strict polynomial time.

Unbounded memory. We now proceed to extend the impossibility to simulators that have no a priori bound on the number of parallel queries they make to the verifier, but still run in expected polynomial time. Here, we need to be wary of the simulator adaptively choosing to increase number of parallel queries.

While the result [BL02] does not apply to this setting, our results will build on their verifier strategy. Our adversarial verifier waits for constant time c_1 (here time is in terms of computation steps) on getting the query, answers correctly with prob ϵ and aborts with prob $1-\epsilon$. We will choose the probability such that $\epsilon > 1/q(\lambda)$ for some polynomial $q(\cdot)$. The probability (over P's random coins) that an honest P causes the honest verifier to accept is $p = \epsilon^c$, where c is the number of rounds in the protocol. Since c is a constant, an honest prover will convince the verifier with noticeable probability as desired.

Let $\epsilon = \epsilon(\lambda)$ be some value to be determined later. Let $\mathcal{H} = \{H_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of $f(\lambda)$-wise independent hash function, such that for every $h \in H_\lambda, h : \{0,1\}^{\leq c \cdot m} \to \{0,1\}^\lambda$, where $\{0,1\}^{\leq c \cdot m}$ denotes all strings of length at most $c \cdot m$. $f(\lambda)$ will be determined later. c denotes the number of prover messages, and $m = m(\lambda)$ denotes the longest prover message of the protocol. We present the detailed strategy below:

Verifier V

Random tape: (h, r) - h defines a function in H_λ, and r is of the length of the random tape required by the honest verifier strategy **Input:** Series of prover messages $q = (\alpha_1, \ldots, \alpha_i)$

1. Step 1 - decide whether or not to abort:
   (a) Compute $h(q')$ for every prefix q' of q. That is, for every $j(1 \leq j \leq i)$, compute $h(\alpha_1, \ldots, \alpha_j)$.
   (b) Wait c_1 time and abort (by outputting the special symbol $\perp$), unless for every j, the first log $\left(\frac{1}{\epsilon}\right)$ bits of $h(\alpha_1, \ldots, \alpha_j)$ are equal to 0.

   (Since the definition of V is by its next message function, we have to ensure that it replies to q only if it would not have aborted on messages sent prior to q in an interactive setting. This is carried out by checking that it would not have aborted on all prefixes of q.)
2. Step 2 - if not aborting,
   (a) Run the honest verifier on input $\alpha_1, \ldots, \alpha_i$ and with random tape r, and obtain its response β.
   (b) Wait c_1 time and output β.

For simplicity of exposition, we assume that the verifier is always convinced. Our analysis naturally extends to the more general setting, where this may not hold.

Now consider the behavior of the same verifier in a simulation by Sim. Sim can issue any number of queries (in parallel) to the adversary at any point of time (by making copies). Consider epochs each of length c_1. In each epoch, the simulator may make a query at any point (and will get the answer only in the next epoch after time c_1 has elapsed). Represent the total number of queries the simulator makes in epoch i by q_i. Note that the Sim can only run for fixed constant c_2 epochs (else our earlier proof would apply). Let $\widetilde{p}$ be the probability that Sim output a transcript $(\alpha_1, \ldots, \alpha_c)$ such that the honest verifier accepts, and Sim received a non-aborting response for every prefix query. By the zero-knowledge property, $\widetilde{p}$ is at least $p - \mathsf{negl}(\lambda)$. Specifically, because ϵ is chosen such that it is an inverse of a polynomial, we have $\widetilde{p} \geq p/2$. Since the simulator Sim is black-box, it does not know ϵ and can only observe the output of the queries.

We show that in each adaptive step, if Sim increases the number of parallel copies by more than an a priori bounded polynomial, it no longer runs in expected polynomial time.

Epoch 1 . Consider epoch 1. During this epoch, by construction of the verifier, the simulator has not received any response from the verifier. Irrespective, it makes q_1 queries to the verifier.

We claim that other than with negligible probability (over the coins of Sim), there exists a polynomial $f_1(\lambda)$ such that $q_1 \leq f_1(\lambda)$.

If it is not the case, with noticeable probability Sim runs in super-polynomial time. Thus, Sim's expected running time would become super-polynomial as well, violating our restriction.

For the subsequent analysis, we define the event E to be,

$$\text{Event } \mathsf{E} \coloneqq \text{ Sim has seen fewer than } c+1 \text{ queries answered up until that point.}$$

We shall now bound the number of queries for each epoch conditioned on event E occurring. Specifically, we will show that if Sim has seen fewer than $c+1$ queries, then the number of queries it makes is bounded by some polynomial in each epoch. We will then argue that any simulator that makes fewer than $c+1$ queries to generate an accepting transcript (i.e. essentially generating a transcript without rewinding), can be used by a cheating prover to break soundness.

Epoch 2. Now consider epoch 2. During this epoch, it receives responses to the queries made in epoch 1. Let us represent this by a "configuration" vector v_1 of dimension q_1 where each entry is either $\bot$ if it aborted, or $\top$ otherwise. We split our analysis of the number of queries q_2, in epoch 2, into two cases (both conditioned on event E):

Case I. If the number of non-aborts in v_1 are at most c, i.e. event E holds, then there exists a polynomial $f_2(\lambda)$ such that for every configuration of v_1, $q_2 \leq f_2(\lambda)$.

Case II. If the number of non-aborts in v_1 are at most c, i.e. event E holds, there is no such f_2, i.e. there is a configuration v_1^* such that the number of queries q_2 is super-polynomial. Given that we've conditioned on event E, there at most c "$\top$" symbols in every characteristic vector v_1. Thus, the total number of such characteristic vectors are

$$\binom{q_1}{0} + \binom{q_1}{1} + \cdots + \binom{q_1}{c}.$$

Since c is a constant, the above number is a polynomial (other than with negligible probability) since we've already established that q_1 is a polynomial. Thus conditioned on event E, every possible configuration v_1^* occurs with noticeable probability. If Sim runs in super-polynomial for a configuration v_1^*, then by the above argument, Sim now runs in super-polynomial time with noticeable probability. Thus, Sim's expected running time would become super-polynomial as well, violating our restriction that expected running time must be polynomial.

Thus, if the event E holds, other than with negligible probability the simulator makes q_2 number of queries, that are bounded by some polynomial.

Epochs $3, \cdots, c_2$**.** For $i \in \{2, \ldots, c_2\}$, we follow identically the analysis for q_2. But here, the characteristic vector v_i would include a symbol ($\bot$ or $\top$) for all queries made before epoch i started. Thus, its dimension would be $\sum_{i=1}^{i-1} q_i$.

Thus, conditioned on E, the queries of Sim are bounded by $f_1(\lambda), f_2(\lambda), \ldots, f_{c_2}(\lambda)$ in their respective epochs. We denote by $f \coloneqq f(\lambda) = \sum_{i=1}^{c_2} f_i(\lambda)$ the bound on the number of queries made by the simulator given event E holds.

From the formula that for n Bernoulli trials where each trial succeeds with probability p, the probability of having k or more successes is at most $\binom{n}{k}p^k$. In our setting, a trial is a query message from the simulator, and the trial succeeds with probability ϵ. Thus the probability of having $c+1$ or more successes in f trials is at most $\binom{f}{c+1}\epsilon^{c+1}$.

Consider the event good where Sim outputs a verifying transcript, but receives exactly c non-aborting responses from the verifier. From the earlier discussion, Sim must output an accepting transcript with probability at least $p/2 = \epsilon^c/2$. And since the probability of having $c+1$ or more successes in f trials is at most $\binom{f}{c+1}\epsilon^{c+1}$, the event good occurs is computed below where E is the event defined earlier, i.e. Sim receives fewer than $c+1$ queries

$$\Pr[\mathsf{Sim\ succeeds}] = \Pr\left[\mathsf{Sim\ succeeds} \mid \overline{\mathsf{E}}\right] \cdot \Pr\left[\overline{\mathsf{E}}\right] + \Pr[\mathsf{Sim\ succeeds} \mid \mathsf{E}] \cdot \Pr[\mathsf{E}]$$

Since Sim outputs a transcript with c responses, we have that $\Pr[\mathsf{good}] = \Pr[\mathsf{Sim\ succeeds} \wedge \mathsf{E}] = \Pr[\mathsf{Sim\ succeeds} \mid \mathsf{E}] \cdot \Pr[\mathsf{E}]$, i.e. making fewer than $c+1$ queries corresponds to making exactly c queries (other than with negligible probability). Therefore, we have,

$$\begin{aligned}\Pr[\mathsf{good}] &= \Pr[\mathsf{Sim\ succeeds}] - \Pr\left[\mathsf{Sim\ succeeds} \mid \overline{\mathsf{E}}\right] \cdot \Pr\left[\overline{\mathsf{E}}\right] \\ &\geq \Pr[\mathsf{Sim\ succeeds}] - 1 \cdot \Pr\left[\overline{\mathsf{E}}\right] \\ &\geq \epsilon^c/2 - \binom{f}{c+1}\epsilon^{c+1}\end{aligned}$$

If we can set the value of ϵ to be such that $\Pr[\mathsf{good}]$ is noticeable, we can rely on the proof in [BL02]. Specifically, this is because we can then use Sim to construct a cheating prover that is able convince an honest verifier with the same probability.

To this end, we set $\epsilon = \frac{1}{4\cdot\binom{f}{c+1}}$. Note that since f is a polynomial, and c is a constant, there exists a polynomial such that $\epsilon(n) = 1/p(\lambda)$. Thus the difference between $\epsilon^c/2$ and $\binom{f}{c+1}\epsilon^{c+1}$ is $\epsilon^c/4$, which is an inverse polynomial as desired. We refer the reader to [BL02] for the full details.

5.6 Black-Box Impossibility of Zero Knowledge in the Plain Model

In this section we shall show that in the plain model, it is impossible to construct a zero knowledge proof system against blockchain active adversaries (BCA).

Recall that the main advantage of a black-box simulator over an adversary is its ability to rewind the adversary. We now sketch a simple verifier strategy that makes it impossible for the simulator to successfully rewind the verifier.

For any polynomial ℓ, let $(p_1, v_1, p_2, v_2, \cdots, p_\ell, v_\ell, p_{\ell+1})$ denote the sequence of messages in a purported ZK protocol. Let $\mathsf{state}_\mathsf{V}$ be the $\mathcal{G}_\mathsf{ledger}$ state of the verifier. The adversarial verifier strategy is as follows:

- On receiving a prover message p_i, send the transcript $(p_1, v_1, p_2, v_2, \cdots, p_i)$ (along with the corresponding session id) to $\mathcal{G}_\mathsf{ledger}$, and wait for the size of the state to increase by $4 \cdot \mathsf{windowSize}$.
- Now, check for the if the following two conditions hold:
  - the sent message is on the state; and
  - there is no other transcript of the form $(\widetilde{p}_1, \widetilde{v}_1, \widetilde{p}_2, \widetilde{v}_2, \cdots, \widetilde{p}_i)$ for the same session id anywhere in the state.

Only if both the checks pass, proceed by sending the honest response v_i. It is not explicitly stated, but the verifier maintains the state of the blockchain, and updates its state as it receives valid blocks.

It is clear that in the interaction with the honest prover, the verifier behaves honestly and completes the protocol.

Now, consider any simulator. For the simulator to gain any advantage over a prover, the simulator must receive responses from the verifier for at least two different i-th messages p_i and $\widetilde{p}_i$ for some i. From the verifier's strategy, we know that in both cases the verifier sends the transcript to $\mathcal{G}_{\mathsf{ledger}}$ and waits for the state to expand sufficiently before responding. If the posted transcript does not appear on the state after the specified wait period, the verifier aborts. This ensures that the simulator cannot rewind the verifier without detection, and thus gains no advantage over an honest prover.

5.7 UC Impossibility

In this section, we will show that it is impossible to achieve universal composition security [Can01] in the blockchain model. This might seem surprising given the positive results of [CJS14] in the non-programmable global random oracle (RO) model, which seems to have a similar flavor.

A crucial difference in these settings is that in the case of the non-programmable global RO model, the posts by the honest parties to the oracle, and their corresponding responses, are private. This is not true for the blockchain model as the queries are public to everyone. Importantly, this in turn implies that the environment sees the actual blockchain, and not a view that the simulator presents to the environment. This prevents the simulator from changing any query made by the adversary to the blockchain. If the simulator does change the query, the environment will always be able to tell the difference between the real and ideal execution. Intuitively, seeing the queries to blockchain is not unique to the adversary, and thus does not constitute a new capability.

For this impossibility, we work in a slightly different model where we allow parties to make an *outstanding query request* to the blockchain, asking for outstanding queries that have not been included in the block. This closely models what happens in practice since parties broadcast the information they want to post to the blockchain to anyone connected to them. This explicitly models the fact that parties can see queries made to the blockchain. We show the impossibility of UC secure commitments [CF01] in the blockchain model. We define the ideal functionality of the commitment here, and refer to the reader to [Can01, CF01] for details and definitions.

Let us assume to the contrary, that there exists a simulator Sim for an adversarial committer. Then, we shall describe an adversarial receiver that uses Sim to extract the value committed by the committer. The adversarial committer works as follows:

1. Initialize Sim.
2. Behave as a "fake committer" by passing any message sent by the committer to Sim, and vice-versa.
3. As and when blocks are received from the blockchain, pass them on to Sim.
4. When the committer makes a query q to the blockchain, this query is sent to Sim on behalf of the "fake committer".
5. Since Sim cannot change the query made by the adversary, when it makes the same query q to the blockchain, we block this query. But as before, when a subsequent block is mined, we

Functionality $\mathbf{F}_{\mathsf{COM}}$

$\mathbf{F}_{\mathsf{COM}}$ proceeds as follows with parties $P_1, \ldots, P_n$ and an adversary Sim.

1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0, 1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and Sim.
2. Upon receiving a value $(\texttt{Open}, sid, P_i, P_j)$ from P_i, proceed as follows: If some value b was previously recorded, then send the message $(\texttt{Open}, sid, P_i, P_j, b)$ to P_j and Sim and halt. Otherwise halt.

Figure 5.5: The ideal commitment functionality

pass it on to Sim.

6. When Sim makes a query to the UC commitment ideal functionality, stop and output this as the committed value.

5.8 Open Problems

Our work demonstrates that under a decentralized trust assumption, namely that of the existence of a secure blockchain, one can circumvent certain cryptographic impossibilities. Both (i) studying additional lower bounds that can be circumvented with the help of blockchains; and (ii) more broadly considering further *reasonable* trust assumptions; are interesting open problems. One should note that what constitutes a “reasonable” trust assumption could vary depending on the setting considered.

Bibliography

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 468–499, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 4, 5, 6, 52, 53, 59

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press. 7, 10

[AGJ$^+$12] 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 Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 443–460, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. 9, 131

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany. 6, 53

[AJ17] Prabhanjan Ananth and Abhishek Jain. On secure two-party computation in three rounds. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 612–644, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. 6

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd Annual Symposium on Foundations of Computer Science*, pages 106–115, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press. 29

[BCG$^+$18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018. 16, 30

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science*, pages 186–195, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press. 9

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 505–514, New York, NY, USA, May 31 – June 3, 2014. ACM Press. 3, 4

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 51–70, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. 9

[BGJ$^+$18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 459–487, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. 4, 5, 6, 53, 55, 56, 57, 59, 61, 62, 65, 67, 68, 81, 86, 87, 97

[BGK$^+$18] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. Cryptology ePrint Archive, Report 2018/378, 2018. `https://eprint.iacr.org/2018/378`. 7, 24, 27

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 645–677, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. 4, 5, 6, 53

[BIN97] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In *38th Annual Symposium on Foundations of Computer Science*, pages 374–383, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press. 44, 47

[BJY97] Mihir Bellare, Markus Jakobsson, and Moti Yung. Round-optimal zero-knowledge arguments based on any one-way function. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 280–305, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany. 8

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. 7, 10

[BKOV17] Saikrishna Badrinarayanan, Dakshita Khurana, Rafail Ostrovsky, and Ivan Visconti. Unconditional UC-secure computation with (stronger-malicious) PUFs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 382–411, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. 9

[BKP19] Nir Bitansky, Dakshita Khurana, and Omer Paneth. Weak zero-knowledge beyond the black-box barrier. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1091–1102, Phoenix, AZ, USA, June 23–26, 2019. ACM Press. 6, 29, 30, 33, 36

[BL02] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In *34th Annual ACM Symposium on Theory of Computing*, pages 484–493, Montréal, Québec, Canada, May 19–21, 2002. ACM Press. 130, 153, 154, 157

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 500–532, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. 5, 6, 22, 52, 56, 57, 78, 79

[Blu81] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *Advances in Cryptology – CRYPTO'81*, volume ECE Report 82-04, pages 11–15, Santa Barbara, CA, USA, 1981. U.C. Santa Barbara, Dept. of Elec. and Computer Eng. 15, 179

[Blu87] Manual Blum. How to prove a theorem so no one else can claim it. In *International Congress of Mathematicians*, pages 1444–1451, 1987. 135, 149, 179

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press. 4, 6

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 7, 10, 24, 26, 27, 129, 174

[BOV03] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 299–315, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. 18

[BP04] Boaz Barak and Rafael Pass. On the possibility of one-message weak zero-knowledge. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 121–132, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. 19, 30, 36

[BP12] Nir Bitansky and Omer Paneth. Point obfuscation and 3-round zero-knowledge. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 190–208, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany. 6

[BP15] Nir Bitansky and Omer Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 401–427, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. 3, 19

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *47th Annual Symposium on Foundations of Computer Science*, pages 345–354, Berkeley, CA, USA, October 21–24, 2006. IEEE Computer Society Press. 9, 131, 150, 151

[BV17] Nir Bitansky and Vinod Vaikuntanathan. A note on perfect correctness by derandomization. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 592–606, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. 3, 19

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press. 8, 9, 10, 28, 158

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany. 7, 10, 28

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany. 8, 9, 158

[CGJ15] Ran Canetti, Vipul Goyal, and Abhishek Jain. Concurrent secure computation with optimal query complexity. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 43–62, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. 141, 149, 153

[CGJ$^+$17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 719–728, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. 7, 10, 27

[CGJ19] Arka Rai Choudhuri, Vipul Goyal, and Abhishek Jain. On round optimal secure multiparty computation from minimal assumptions. Cryptology ePrint Archive, Report 2019/216, 2019. `https://ia.cr/2019/216`. 7

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for UC secure computation using tamper-proof hardware. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 545–562, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany. 9

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 597–608, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. 7, 9, 28, 158

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany. 8, 9

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press. 9

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st Annual Symposium on Foundations of Computer Science*, pages 541–550, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press. 131

[CO19] Michele Ciampi and Rafail Ostrovsky. Four-round secure multiparty computation from general assumptions. Cryptology ePrint Archive, Report 2019/214, 2019. https://eprint.iacr.org/2019/214. 7

[COSV16] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Concurrent non-malleable commitments (and more) in 3 rounds. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 270–299, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. 71

[COSV17a] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 711–742, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. 4, 6

[COSV17b] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Round-optimal secure two-party computation from trapdoor permutations. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 678–710, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. 4, 6

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances* *in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. 2

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, New Orleans, LA, USA, May 6–8, 1991. ACM Press. 53, 70, 150

[DFK$^+$14] Dana Dachman-Soled, Nils Fleischhacker, Jonathan Katz, Anna Lysyanskaya, and Dominique Schröder. Feasibility and infeasibility of secure computation with malicious PUFs. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 405–420, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. 9

[DL20] Hila Dahari and Yehuda Lindell. Deterministic-prover zero-knowledge proofs. Cryptology ePrint Archive, Report 2020/141, 2020. `https://eprint.iacr.org/2020/141`. 2

[DN00] Cynthia Dwork and Moni Naor. Zaps and their applications. In *41st Annual Symposium on Foundations of Computer Science*, pages 283–293, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press. 53

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *30th Annual ACM Symposium on Theory of Computing*, pages 409–418, Dallas, TX, USA, May 23–26, 1998. ACM Press. 10

[Fei90] Uriel Feige. Alternative models for zero knowledge interactive proofs. 1990. 179

[FNV17] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Predictable arguments of knowledge. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 121–150, Amsterdam, The Netherlands, March 28–31, 2017. Springer, Heidelberg, Germany. 2, 3, 4, 29, 31, 41, 42, 44, 46, 47

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. 8

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd Annual ACM Symposium on Theory of Computing*, pages 416–426, Baltimore, MD, USA, May 14–16, 1990. ACM Press. 8, 150, 180

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 529–561, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. 7, 9, 10, 27

[GGJ13] Vipul Goyal, Divya Gupta, and Abhishek Jain. What information is leaked under concurrent composition? In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 220–238, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. 141, 149, 153

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. 2, 16, 17, 54

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *30th Annual ACM Symposium on Theory of Computing*, pages 151–160, Dallas, TX, USA, May 23–26, 1998. ACM Press. 6

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany. 9

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 277–294, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. 141, 149, 153

[GK96a] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, June 1996. 86, 92

[GK96b] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996. 2, 4, 8, 52

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. 10

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 291–323, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 10

[GKM$^+$00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. The relationship between public key encryption and oblivious transfer. In *41st Annual Symposium on Foundations of Computer Science*, pages 325–335, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press. 86

[GKOV12] Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 424–442, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. 9, 131

[GKW15] Romain Gay, Iordanis Kerenidis, and Hoeteck Wee. Communication complexity of conditional disclosure of secrets and attribute-based encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 485–502, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. 6

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, WA, USA, May 15–17, 1989. ACM Press. 15, 179

[GLP$^+$15] Vipul Goyal, Huijia Lin, Omkant Pandey, Rafael Pass, and Amit Sahai. Round-efficient concurrently composable secure computation via a robust extraction lemma. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 260–289, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. 131, 142, 143, 145, 146

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 448–476, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 2, 4, 5, 6, 125

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th Annual ACM Symposium on Theory of Computing*, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press. 8, 175, 178

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. 19, 20, 67

[GMR89a] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. 2, 3, 52

[GMR89b] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. 175

[GMW87a] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press. 4, 8, 150

[GMW87b] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. 52

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994. vii, 1, 2, 29, 33, 51, 175, 176, 177

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. 20

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 97–111, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany. 3, 19

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 695–704, San Jose, CA, USA, June 6–8, 2011. ACM Press. 4, 6, 70

[Goy12] Vipul Goyal. Positive results for concurrently secure computation in the plain model. In *53rd Annual Symposium on Foundations of Computer Science*, pages 41–50, New Brunswick, NJ, USA, October 20–23, 2012. IEEE Computer Society Press. 9, 131

[GPR16] Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 1128–1141, Cambridge, MA, USA, June 18–21, 2016. ACM Press. 68, 69, 72, 73, 74

[GR19] Vipul Goyal and Silas Richelson. Non-malleable commitments using Goldreich-Levin list decoding. In David Zuckerman, editor, *60th Annual Symposium on Foundations of Computer Science*, pages 686–699, Baltimore, MD, USA, November 9–12, 2019. IEEE Computer Society Press. 55, 68

[GS18a] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 425–454, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany. 16, 30

[GS18b] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. 56, 78, 79

[GVW01] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP 2001: 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 334–345, Heraklion, Crete, Greece, July 8–12, 2001. Springer, Heidelberg, Germany. 4

[HHK$^+$05] Iftach Haitner, Omer Horvitz, Jonathan Katz, Chiu-Yuen Koo, Ruggero Morselli, and Ronen Shaltiel. Reducing complexity assumptions for statistically-hiding commitment. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 58–77, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany. 15, 149

[HHPV18] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 488–520, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. 4, 5, 6, 53, 61

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. 15, 19

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, Heidelberg, Germany, 2010. vii, 182

[HM96] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany. 16

[HPV16] Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Composable security in the tamper-proof hardware model under minimal complexity. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 367–399, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. 7, 28

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 577–594, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. 6

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany. 4

[JKKR17] Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Ron Rothblum. Distinguisher-dependent simulation in two rounds and its applications. In Jonathan

Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 158–189, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 53, 55

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany. 9

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th Annual ACM Symposium on Theory of Computing*, pages 20–31, Chicago, IL, USA, May 2–4, 1988. ACM Press. 4, 150

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 644–653, Baltimore, MA, USA, May 22–24, 2005. ACM Press. 10

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press. 16, 30

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 335–354, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany. 4, 125

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 578–595, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany. 4, 6

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in polyloalgorithm rounds. In *33rd Annual ACM Symposium on Theory of Computing*, pages 560–569, Crete, Greece, July 6–8, 2001. ACM Press. 131

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. 10

[KSS11] Jeff Kahn, Michael Saks, and Clifford Smyth. The dual bkr inequality and rudich's conjecture. *Combinatorics, Probability and Computing*, 20(2):257–266, 2011. 14

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 705–734, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. 7, 24, 27

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science*, pages 394–403, Cambridge, MA, USA, October 11–14, 2003. IEEE Computer Society Press. 9

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 203–222, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. 9, 131

[Lin08] Yehuda Lindell. Lower bounds and impossibility results for concurrent self composition. *Journal of Cryptology*, 21(2):200–249, April 2008. 9, 132

[Lin16] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. `https://eprint.iacr.org/2016/046`. 125, 182

[LPV08] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable commitments from any one-way function. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 571–588, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany. 70

[LS91] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 353–365, Santa Barbara, CA, USA, August 11–15, 1991. Springer, Heidelberg, Germany. vii, 18, 179, 180

[LS19] Alex Lombardi and Luke Schaeffer. A note on key agreement and non-interactive commitments. Cryptology ePrint Archive, Report 2019/279, 2019. `https://eprint.iacr.org/2019/279`. 56, 86

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing*, pages 306–315, Seattle, WA, USA, May 21–23, 2006. ACM Press. 129

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, January 1991. 15, 149

[NOVY98] Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect zero-knowledge arguments for NP using any one-way permutation. *Journal of Cryptology*, 11(2):87–108, March 1998. 15, 149

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. 3, 19

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, pages 33–43, Seattle, WA, USA, May 15–17, 1989. ACM Press. 16

[Ore87] Yair Oren. On the cunning power of cheating verifiers: Some observations about zero knowledge proofs (extended abstract). In *28th Annual Symposium on Foundations of Computer Science*, pages 462–471, Los Angeles, CA, USA, October 12–14, 1987. IEEE Computer Society Press. 175

[ORS15] Rafail Ostrovsky, Silas Richelson, and Alessandra Scafuro. Round-optimal black-box two-party computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 339–358, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. 23

[ORSV13] Rafail Ostrovsky, Vanishree Rao, Alessandra Scafuro, and Ivan Visconti. Revisiting lower and upper bounds for selective decommitments. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 559–578, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, *36th Annual ACM Symposium on Theory of Computing*, pages 232–241, Chicago, IL, USA, June 13–16, 2004. ACM Press. 4, 6

[PR05] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *46th Annual Symposium on Foundations of Computer Science*, pages 563–572, Pittsburgh, PA, USA, October 23–25, 2005. IEEE Computer Society Press. 70

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *43rd Annual Symposium on Foundations of Computer Science*, pages 366–375, Vancouver, BC, Canada, November 16–19, 2002. IEEE Computer Society Press. 58, 61, 62, 131, 150

[PSs17] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. 10

[PW10] Rafael Pass and Hoeteck Wee. Constant-round non-malleable commitments from subexponential one-way functions. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 638–655, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. 4, 6

[RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 415–431, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany. 131

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, MD, USA, May 14–16, 1990. ACM Press. 20

[Ros04] Alon Rosen. A note on constant-round zero-knowledge proofs for NP. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 191–202, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. 61, 62

[Rud88] Steven Rudich. Limits on the provable consequences of one-way functions. 1988. 14

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science*, pages 543–553, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press. 69

[TW87] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *28th Annual Symposium on Foundations of Computer Science*, pages 472–482, Los Angeles, CA, USA, October 12–14, 1987. IEEE Computer Society Press. 175

[Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st Annual Symposium on Foundations of Computer Science*, pages 531–540, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press. 4, 6

[Yao82a] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press. 8, 23

[Yao82b] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press. 15, 179

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press. 4, 54, 150

Appendix A

A.1 Extend Policy for Bitcoin

We copy verbatim from [BMTZ17] the properties ensured by ExtendPolicy:

1. The speed of the ledger is not too slow. This is implemented by defining an upper bound $\mathsf{maxTime}_{\mathsf{window}}$ on the time interval within which at least windowSize state blocks have to be added. This is known as minimal chain-growth.
2. The speed of the ledger is not too fast. This is implemented by defining a lower bound $\mathsf{minTime}_{\mathsf{window}}$ on the time interval such that the adversary is not allowed to propose new blocks if windowSize or more blocks have already been added during that interval.
3. The adversary cannot create too many blocks with arbitrary (but valid) contents. This is formally enforced by defining an upper bound η on the number of these so-called adversarial blocks within a sequence of state blocks. This is knows as chain quality. Formally, this is enforced by requiring that a certain fraction of blocks need to satisfy higher quality standards (to model blocks that were honestly generated).
4. Last, but not the least, ExtendPolicyguarantess that if a transaction is "old enough", and still valid with respect to the actual state, then it is included into the state. This is a week form of guaranteeing that a transaction will make it into the state unless it is in conflict.

The formal description can be found in [BMTZ17].

A.2 The Many Flavors of Zero-Knowledge

In our chapter on *deterministic prover zero-knowledge*, we briefly discussed the existence of various different notions of zero-knowledge, choosing to focus on (bounded) auxiliary-input zero knowledge. In this section, we present the various notions, and the relationship between these notions. This will help provide some context for both the impossibility in [GO94] (discussed further in Section A.3), and the results presented in this thesis.

We do not repeat here the definitions of completeness and soundness (see Section 3.2.1 for details), and focus solely on the zero-knowledge definitions.

We start with the original formulation of zero-knowledge by Goldwasser, Micali and Rackoff [GMR85], hereon referred to as the *GMR* zero-knowledge definition.

Definition 37 (*GMR* Zero-knowledge). *For every* PPT *verifier* V^**, there exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{V}^*}$*, such that*

$$\left\{\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x,w), \mathsf{V}^{*(x)}\rangle\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)}} \approx_c \left\{\mathsf{Sim}_{\mathsf{V}^*}(x)\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)}} .$$

The subsequent stronger notions were described in various works [Ore87, GO94, TW87, GMR89b]

Definition 38 (*Auxiliary-input* Zero-knowledge). *For every* PPT *verifier* V^**, there exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{V}^*}$*, such that*

$$\left\{\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x,w), \mathsf{V}^*(x,y)\rangle\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)\\ y\in\{0,1\}^*}} \approx_c \left\{\mathsf{Sim}_{\mathsf{V}^*}(x,y,1^t)\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)\\ y\in\{0,1\}^*}} .$$

Definition 39 (*Universal-simulation* Zero-knowledge). *There exists a* PPT *simulator* Sim*, such that for every* PPT *verifier* V^* *of running time at most* $t(\lambda)$*,*

$$\left\{\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x,w), \mathsf{V}^*(x)\rangle\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)}} \approx_c \left\{\mathsf{Sim}(\mathsf{V}^*,1^t,x)\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)}} .$$

Definition 40 (*Black-box-simulation* Zero-knowledge). *There exists a* PPT *simulator* Sim*, such that for every* PPT *verifier* V^**,*

$$\left\{\mathsf{View}_{\mathsf{V}^*}\langle \mathsf{P}(x,w), \mathsf{V}^*\rangle\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)}} \approx_c \left\{\mathsf{Sim}^{\mathsf{V}^*}(x)\right\}_{\substack{\lambda\in\mathbb{N},\\ x\in\mathcal{L}\cap\{0,1\}^\lambda,\\ w\in R_{\mathcal{L}}(x)}} .$$

A.2.1 Relationship Between Notions of Zero-knowledge

We state here (without proof), the relationships between the various notions discussed above. $\mathsf{Cl}(\mathit{def})$ denotes the class of all interactive proof (or argument) systems satisfying the requirements of definition *def*.

Theorem 18 ([Ore87]).

$$\mathsf{Cl}(\textit{Auxiliary-input}) = \mathsf{Cl}(\textit{Universal-simulation})$$

Intuitively, an *auxiliary-input* simulator can be constructed from a *universal* simulator by hard-coding t bits of the auxiliary input y into the description of the verifier, where t is the running time. A *universal* simulator can be constructed from an *auxiliary-input* simulator by consider the *auxiliary-input* simulator for the universal Turing machine, and setting the auxiliary-input to be the description of the verifier.

We note that the existence of a universal simulator is *not* guaranteed from the existence of a *bounded* auxiliary-input zero-knowledge since the running time can be larger than the auxiliary-input.

Theorem 19 ([GO94]).

$$\mathsf{Cl}(\textit{Black-box-simulation}) \subseteq \mathsf{Cl}(\textit{Auxiliary-input}) \subseteq \mathsf{Cl}(\textit{GMR})$$

The proof follows in a fairly straightforward manner from the corresponding definitions.

A.3 Goldreich-Oren [GO94] impossibility for Deterministic Provers

Here we sketch out the impossibility of deterministic prover auxiliary-input zero-knowledge for non-trivial languages as defined in [GO94]. Before we can state the theorem, we must establish what it means for a language to be trivial. Below, we define the class BPP which we state to be the class of trivial languages.

Definition 41 (Complexity class BPP). *A language $\mathcal{L}$ is in BPP if there exists a PPT machine M such that*

Completeness. *if $x \in \mathcal{L}$,* $\Pr[\mathsf{M}(x) = \mathsf{accept}] \geq 1 - \mathsf{negl}(|x|)$.

Soundness. *if $x \notin \mathcal{L}$,* $\Pr[\mathsf{M}(x) = \mathsf{accept}] < \mathsf{negl}(|x|)$.

Any interactive proof system for $\mathcal{L} \in$ BPP is *trivial* since a PPT verifier can always run the corresponding M for $\mathcal{L}$ on input any x, and decide whether to accept without interacting with a prover.

We now state the formal theorem below, where auxiliary-input zero-knowledge is defined in Definition 38.

Theorem 20 ([GO94]). *Let $\mathcal{L}$ be any language. If $\mathcal{L}$ has an auxiliary-input zero-knowledge protocol with deterministic provers, then $\mathcal{L} \in$ BPP.*

Let (P, V) be the *deterministic prover* zero-knowledge protocol for $\mathcal{L}$ with the corresponding auxiliary-input zero knowledge simulator Sim. To show that $\mathcal{L} \in$ BPP, we construct using Sim a machine M as defined in Definition 41.

Assume that the verifier V sends the first message. This can be assumed without loss of generality since the protocol can always be modified to send an empty string to be the empty string if it is not the case (or a random string if public coin). Before we proceed, let us establish some notation. Specifically, let $\overrightarrow{\tilde{\mathsf{v}}}_i := (\tilde{\mathsf{v}}_1, \cdots, \tilde{\mathsf{v}}_i)$. We define $\mathsf{V}^*_{\overrightarrow{\tilde{\mathsf{v}}}_i}$ to be the verifier that has hardcoded the strings $\tilde{\mathsf{v}}_1, \cdots, \tilde{\mathsf{v}}_i$, and for its first i messages uses the string $\tilde{\mathsf{v}}_j$ as its j-th message, and can behave arbitrarily after it has sent the i-th message.

Note that $\mathsf{V}^*_{\overrightarrow{\tilde{\mathsf{v}}}_i}$ is *not* the prescribed verifier for the underlying protocol. We are ready to describe M, utilizing the underlying prescribed verifier V and simulator Sim.

```
M(x):
  1. Choose a random $r$ //the randomness to be used with the prescribed verifier V
  2. For each $i \in [\rho]$,
      (a) Compute $\mathsf{v_i} := \mathsf{V}(x, \mathsf{p}_1, \cdots, \mathsf{p}_{i-1}; r)$
      (b) Set $\tilde{\mathsf{v}}_i := \mathsf{v}_i$.
      (c) $(\{\tilde{\mathsf{p}}_j\}_{j=1}^{\rho}, \widetilde{r}) \leftarrow \mathsf{Sim}\left(x, \mathsf{V}^*_{\overrightarrow{\tilde{\mathsf{v}}}_i}, 1^t\right)$. //where $\mathsf{V}^*_{\overrightarrow{\tilde{\mathsf{v}}}_i}$ is as defined above
      (d) Set $\mathsf{p}_i := \tilde{\mathsf{p}}_i$. //ignore the remaining outputs of Sim
  3. Output $\mathsf{V}(x, \mathsf{p}_1, \cdots, \tilde{p}_\rho; r)$ //accept or reject with the prescribed verifier
```

We provide a high level sketch of the soundness and completeness properties for the machine M described above. For simplicity, we consider here the case of efficient provers, and we refer the reader to [GO94] for the more general setting.

Soundness. When $x \notin \mathcal{L}$, if M outputs accept with non-negligible probability, we can break the soundness of the underlying protocol (P, V) using the same strategy as in M. This follows from the following observations regarding M: (i) M uses the prescribed verifier V to compute the verifier messages using the randomness r sampled in Step 1; (ii) additionally, the aforementioned randomness r is not used other than to execute the prescribed verifier V.

To build a cheating prover P^* interacting with a verifier V, run a modified machine $\tilde{\mathsf{M}}$ which (i) skips Steps 1, 2(a) and 3; and (ii) sets $\tilde{\mathsf{v}}_i$ in Step 2(b) to be the message P^* receives from V. From the description of P^* if follows that if M accepts with non-negligible probability, P^* succeeds with the (same) non-negligible probability.

Completeness. Completeness follows from the completeness and zero-knowledge properties of the underlying protocol. From the completeness of the protocol, we know that when $x \in \mathcal{L}$, the honest prover produces an accepting transcript. In fact, since the prover is deterministic, for *any* fixed witness w and verifier randomness r, there is a unique protocol transcript, i.e. prover messages are fixed by r and w. From the zero-knowledge property, the simulator must produce (other than with negligible probability) the same prover messages in the simulated transcript as in the real transcript. We do not provide the details here, but at a high level this follows from the fact that (i) verifier $\mathsf{V}^*_{\vec{\mathsf{v}}_i}$ behaves *like* a prescribed verifier when the auxiliary input includes the prescribed verifier messages with the exception that the Sim must produce simulated verifier randomness consistent with the passed verifier messages; and (ii) a non-uniform distinguisher with access to the witness w can generate the protocol transcript by itself and compare with simulated transcript.

In conjunction with Theorem 19, the above impossibility also rules out the existence of *deterministic prover* zero-knowledge with *black-box-simulation* but does not have any bearing on the [GMR85] notion of zero-knowledge.

A.4 Lapidot-Shamir [LS91] Three Round Witness Indistinguishable Proof

We describe here the 3 message (round) witness indistinguishable proof implicit in the work of [LS91], as presented in [Fei90]. The protocol, as in Blum's Hamiltonicity protocol [Blu87][1], is presented for the NP-complete language Graph Hamiltonicity, which we denote below.

$$\mathcal{L}_{\mathsf{HC}} = \left\{ G = (V, E) \text{ where } n \overset{\text{def}}{=} |V| \;\middle|\; \exists C \subset E \;\text{ s.t. } C \text{ is a directed Hamiltonian} \right\}$$

The protocol uses a non-interactive *statistically binding* commitment schemes (Section 2.3), which can be constructed assuming injective one-way functions [Blu81, Yao82b, GL89].

The Graph $G = (V, E)$ is represented by a $n \times n$ matrix M_G,

$$M_G[u][v] = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

The protocol $(\mathsf{P}_{\mathsf{HC}}, \mathsf{V}_{\mathsf{HC}})$ is presented in Figure A.1. $(\mathsf{P}_{\mathsf{HC}}, \mathsf{V}_{\mathsf{HC}})$ is repeated k times in parallel (with independent randomness for each execution) to obtain the protocol $(\mathsf{P}_{\mathsf{HC}}^{\otimes k}, \mathsf{V}_{\mathsf{HC}}^{\otimes k})$, where $\mathsf{V}_{\mathsf{HC}}^{\otimes k}$ accepts *if and only if* all k sessions of V_{HC} accept.

We then have the following imported theorem.

Imported Theorem 4 ([LS91]). *Assuming non-interactive commitments,* $(\mathsf{P}_{\mathsf{HC}}^{\otimes k}, \mathsf{V}_{\mathsf{HC}}^{\otimes k})$ *is a 3 round delayed-input witness indistinguishable proof systems.*

We provide a informal argument below for each of the desired properties.

Input-delayed. The prover only needs to know the number of vertices in a graph G in the first round of the protocol, the actual graph G and its corresponding cycle (witness) is not required until round 3.[2]

Public coin. The verifier only sends k random bits which it uses to evaluate the outcome of the protocol, and thus satisfies the public coin property.

Completeness. Completeness follows in a straightforward manner from the description of the protocol.

Soundness. We argue that the soundness of $(\mathsf{P}_{\mathsf{HC}}, \mathsf{V}_{\mathsf{HC}})$ is $1/2 + \mathsf{negl}(\lambda)$ which is *amplified* by parallel repetition to be $1/2^k + \mathsf{negl}(\lambda)$. For $(\mathsf{P}_{\mathsf{HC}}, \mathsf{V}_{\mathsf{HC}})$ we in fact argue something stronger. Specifically, if a cheating prover P^* is able to convince a verifier with probability (noticeably) larger than $1/2$, then we can *extract* a witness for $G \in \mathcal{L}_{\mathsf{HC}}$. This clearly implies soundness since any $G \notin \mathcal{L}_{\mathsf{HC}}$ does not possess a valid witness.

If a cheating prover P^* is able to convince a verifier with probability (noticeably) larger than $1/2$, then it must be the case that P^* is able to answer *both* challenges ($b = 0$ and $n = 1$) from the

[1]We have previously described the Blum Hamiltoncity protocol in Section 5.3.1, and it bears resemblance to the [LS91] protocol described here, with the notable exception that Blum's Hamiltonicity protocol is not input delayed.

[2]This property is used in the NIZK construction of [LS91] where this protocol is implicit.

3 message [LS91] Witness Indistinguishable Proof $(\mathsf{P}_{\mathsf{HC}}, \mathsf{V}_{\mathsf{HC}})$ for L_{HC}

Common Input: A directed graph $G = (V, E)$ with $\lambda \overset{\text{def}}{=} |V|$.

Auxiliary Input for Prover: a directed Hamiltonian, $C \subset E$, in G.

1. Prover P_{HC} computes the first message as
   (a) Pick a random cycle on n nodes to generate a graph H, and construct the corresponding adjacency matrix A_H.
   (b) Commit to each entry of A_H using Com.

   send the $n \times n$ matrix of committed values to the verifier VHC.
2. Verifier VHC computes the second message to be a random bit $b \leftarrow\!\!\$ \{0, 1\}$.
3. Prover P_{HC} computes the third message as
   – if $b = 0$, set d to be the decommitment to *all* positions of A_H.
   – if $b = 1$, let π the permutation that maps from H onto the cycle $C \subset E$. Set d to include π and the decommitment to all entries in $A_{\pi(H)}$ that *do not* correspond to edges in G.

   send d to the verifier V_{HC}.
4. Verifier V_{HC} first checks if all the values decommitted to by the prover are valid (with respect to their corresponding commitment).
   – if $b = 0$, the verifier checks if the revealed graph is a cycle on n vertices.
   – if $b = 1$, the verifier checks if all the revealed values correspond to non-edges of $\pi(H)$ with value 0.

   The verifier accepts if and only if both the initial checks, and the checks corresponding to the challenge verify .

Figure A.1: Hamiltonicity proof system

verifier. Specifically, P^* is able to *both* decommit to a cycle graph H, *and* a permutation π from H to G. Combining the two, one can obtain the cycle C in G. Since the protocol utilizes a statistically binding commitment scheme, the argument holds (other than with negligible probability) for computationally unbounded cheating provers as well. Thus, the protocol is a *proof*.

Witness Indistinguishability. We will argue the zero-knowledge property of the above protocol for a *single* iteration $(\mathsf{P}_{\mathsf{HC}}, \mathsf{V}_{\mathsf{HC}})$, and use the fact that a zero-knowledge protocol is also witness indistinguishable [FS90] - a property that composes with parallel execution [FS90], i.e. k parallel repetitions with fresh randomness $(\mathsf{P}_{\mathsf{HC}}^{\otimes k}, \mathsf{V}_{\mathsf{HC}}^{\otimes k})$ remains witness indistinguishable.

Zero-knowledge of $(\mathsf{P}_{\mathsf{HC}}, \mathsf{V}_{\mathsf{HC}})$ follows from the fact that one can generate a simulated transcript by guessing the verifier challenge bit b and committing to either (i) a cycle graph if $b = 0$; or (ii) “empty” graph if $b = 1$. The third message of the simulation remains identical for $b = 0$ while for $b = 1$, the simulator picks a random permutation π and opens to the appropriate non-edges (in fact, there are *no* edges in the committed graph).

A.5 Hazay-Lindell [HL10] Analysis

In the description of the simulator Sim in Section 4.5, if the adversary $\mathcal{A}$ did not abort (or only implicitly aborted), we sampled transcripts until there were $12 \cdot \lambda$ non-aborting transcripts before proceeding with the simulation. The purpose was to estimate the aborting probability ε' to be within a constant factor of the true (unknown) aborting probability ε. This ensured that the expected running time of the simulator was $\mathsf{poly}(\lambda)$ for some polynomial poly.

Here we elaborate on how we arrive at the specific number of non-aborting transcripts $12 \cdot \lambda$. This discussion is taken largely from [HL10] (Section 6.5.3). We refer the reader to [HL10, Lin16] for a detailed discussion on why such estimations are necessary.

Let m be the total number of non-aborting transcripts needed to estimate ε to within a constant factor, i.e. from our discussion above we shall show that $m = 12 \cdot \lambda$ suffices. The estimate ε' is set to be m/T where T is the number of attempts to obtain m non-aborting transcripts.

The required value of m follows from the following Lemma used in [HL10].

Lemma 6 (Tail inequality for Geometric Variables). *Let $X_1, \cdots, X_m$ be m independent random variables with geometric distribution with probability ε, i.e. for every i, $\Pr[X_i = j] = (1-\varepsilon)^{j-1} \cdot \varepsilon$. Let $X = \sum_{i=1}^{m}$ and let $\mu = \mathbb{E}[X] = m/\varepsilon$. Then, for every δ,*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{m\delta^2}{2(1+\delta)}} \quad . \tag{A.1}$$

For $0 \leq \delta \leq 1/2$,

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{m\delta^2}{3(1-\delta)}} \quad . \tag{A.2}$$

While the proof of Equation A.1 in the Lemma above is presented in [HL10], we provide below the proof of Equation A.2 which follows in a similar manner but provided here for completeness.

Proof. For any α, let Y_α denote the number of 1s appearing in a prefix of length α for Bernoulli trials where each bit is 1 with probability ε. Then,

$$\widetilde{\mu}_\alpha := \mathbb{E}[Y_\alpha] = \alpha \cdot \varepsilon \quad .$$

By our description of the random variable X in the lemma, $X \leq (1-\delta)\mu$ if and only if $Y_{(1-\delta)\mu} \geq m$. This follows from the fact that m successful trials in Y_α occurred *before* α attempts, which is what $X \leq \alpha$ conveys. Therefore,

$$\Pr[X \leq (1-\delta)\mu] = \Pr\left[Y_{(1-\delta)\mu} \geq m\right] \quad .$$

We can focus on upper bounding the term $\Pr\left[Y_{(1-\delta)\mu} \geq m\right]$. Since $\mu = m/\varepsilon$, we have that $\widetilde{\mu}_\alpha = (1-\delta)m$ for $\alpha = (1-\delta)\mu$. Thus, for $\alpha = (1-\delta)\mu$ we have,

$$\begin{aligned} \left(1 + \frac{\delta}{1-\delta}\right)\widetilde{\mu}_\alpha &= \left(1 + \frac{\delta}{1-\delta}\right)(1-\delta)m \\ &= m \quad . \end{aligned}$$

Writing out $Y_{(1-\delta)\mu}$ in a more convenient form to apply the Chernoff bound, we have,

$$\Pr\left[Y_{(1-\delta)\mu} \geq m\right] = \Pr\left[Y_{(1-\delta)\mu} \geq \left(1 + \frac{\delta}{1-\delta}\right)\widetilde{\mu}_{(1-\delta)\mu}\right] \quad .$$

Using the Chernoff bound $\Pr[Y \geq (1+\beta)\widetilde{\mu}] \leq e^{-\frac{\beta^2 \widetilde{\mu}}{3}}$ for $0 \geq \beta \geq 1$, we have

$$\begin{aligned}
\Pr\big[Y_{(1-\delta)\mu} \geq m\big] &= \Pr\left[Y_{(1-\delta)\mu} \geq \left(1+\frac{\delta}{1-\delta}\right)\widetilde{\mu}_{(1-\delta)\mu}\right] \\
&\leq e^{-\left(\frac{\delta}{1-\delta}\right)^2 \frac{\widetilde{\mu}_{(1-\delta)\mu}}{3}} \\
&= e^{-\left(\frac{\delta}{(1-\delta)}\right)^2 \frac{(1-\delta)m}{3}} \\
&= e^{-\frac{\delta^2}{1-\delta}\frac{m}{3}}
\end{aligned}$$

for $0 \leq \delta \leq 1/2$ since those are the range of values for δ that ensure $\beta = \delta/(1-\delta) \leq 1$. □

Now, to apply the above lemma, let X_i denote the random variable that equals to the number of attempts required for the i-th non-aborting transcript, and and let $\delta = \pm 1/2$. Thus, each X_i is a geometric distribution with probability ε. Clearly, from our description above, $X = \sum_{i=1}^{m}$ corresponds to the total number of sample T. From the lemma above,

$$\begin{aligned}
\Pr\left[X \leq \frac{m}{2\varepsilon} \vee X \geq \frac{3m}{2\varepsilon}\right] &\leq \Pr\left[X \leq \frac{m}{2\varepsilon}\right] + \Pr\left[X \geq \frac{3m}{2\varepsilon}\right] \\
&= \Pr\left[X \leq \left(1-\frac{1}{2}\right)\frac{m}{\varepsilon}\right] + \Pr\left[X \geq \left(1+\frac{1}{2}\right)\frac{m}{\varepsilon}\right] \\
&\leq e^{-\frac{m(1/2)^2}{3(1-1/2)}} + e^{-\frac{m(1/2)^2}{2(1+1/2)}} \\
&= e^{-m/6} + e^{-m/12} \leq 2 \cdot e^{-m/12}
\end{aligned}$$

Thus the above bounds the probability that the estimate is *not* between $2\varepsilon/3$ and 2ε to be $2e^{-m/12}$. By setting, $m := 12 \cdot \lambda$, we have that the above probability is bounded by $1/2^{\lambda}$.

Sim also has an additional time out of 2^{λ} steps. This is to cover the case that ε' is not estimated within a constant factor of ε with $12 \cdot \lambda$ non-aborting transcripts. But from above, since this happens only with probability $1/2^{\lambda}$, the expected running time of the simulator remains polynomial.