Concurrently Secure Computation in Constant Rounds

Sanjam Garg
UCLA
sanjamg@cs.ucla.edu

Vipul Goyal
MSR India
vipul@microsoft.com

Abhishek Jain
UCLA
abhishek@cs.ucla.edu

Amit Sahai
UCLA
sahai@cs.ucla.edu

May 20, 2011

Abstract

We study the problem of constructing concurrently secure computation protocols in the plain model, where no trust is required in any party or setup. While the well established UC framework for concurrent security is impossible to achieve in this setting, a meaningful notion of concurrent security based on *super-polynomial simulation* (SPS) is achievable and has been extensively studied [Pas03, PS04, BS05, LPV09, CLP10]. The recent work of [CLP10] obtains a concurrently secure computation protocol in the plain model with SPS security under standard assumptions, but requires a number of rounds of interaction that is polynomial in the security parameter.

In this work, we obtain the first concurrently secure computation protocol in the plain model with SPS security that uses only a *constant* number of rounds and requires only *standard assumptions.* To accomplish our result, we introduce a new proof technique that significantly reduces the demands placed on "rewinding techniques" employed in previous work. We believe that our techniques are of independent interest and likely to be applicable in other settings related to secure concurrent composition.

Contents

1 Introduction

The notion of *secure computation* is central to cryptography. Introduced in the seminal works of [Yao86, GMW87], secure multi-party computation allows a group of (mutually) distrustful parties $P_1, \ldots, P_n$, with private inputs $x_1, \ldots, x_n$, to jointly compute any functionality f in such a manner that the honest parties obtain correct outputs and no group of malicious parties learn anything beyond their inputs and prescribed outputs. The original definition of secure computation, although very useful and fundamental to cryptography, is only relevant to the *stand-alone* setting where security holds only if a single protocol session is executed in isolation. As it has become increasingly evident over the last two decades, stand-alone security does not suffice in real-world scenarios where several protocol sessions may be executed *concurrently* – a typical example being protocols executed over modern networked environments such as the Internet.

Concurrent Security. Towards that end, the last decade has seen a push towards obtaining protocols that have strong concurrent *composability* properties, i.e., protocols that remain secure even when executed concurrently with other protocols in an arbitrary manner. The framework of *universal composability* (UC) was introduced by Canetti [Can01] to capture the security requirements in such a setting. Unfortunately, soon after the introduction of the UC framework, strong impossibility results were shown ruling out the existence of UC secure protocols for most functionalities of interest [CF01, CKL06]. These results were further generalized [Lin04, BPS06] to rule out the existence of secure protocols even in various less demanding settings. All these impossibility results refer to the "plain model," where parties do not trust any external entity or setup. (The original UC framework inherently assumes authenticated communication, but this can be avoided by considering the model of [BCL$^+$05] and using signature schemes, to consider UC security in the "truly plain model.")

To overcome these impossibility results, UC secure protocols were proposed based on various "trusted setup assumptions" such as a common random string that is published by a *trusted party* [CF01, CLOS02, BCNP04, CPS07, Kat07, CGS08]. In fact, this approach has yielded protocols with even stronger security guarantees, such as generalized UC security [CDPW07, DKSW09] and resource fairness [GMPY06]. Nevertheless, a driving goal in cryptographic research is to eliminate the need to trust other parties[1]. In the context of UC secure protocols based on setup assumptions, while there has been some recent effort [GO07, GK08, GGJS11] towards reducing the extent of trust in any single party (or entity), obviously this approach cannot completely eliminate trust in other parties (since that is the very premise of a trusted setup assumption). Ideally, we would like to obtain concurrently-secure protocols in the *plain model* (which is the main focus of this paper).

Super-Polynomial Simulation. To address the problem of concurrent security for secure computation in the plain model, a few candidate definitions have been proposed, including input-indistinguishable security [MPR06] and super-polynomial simulation. The notion of security with *super-polynomial simulators* (SPS) is one where the adversary in the ideal world is allowed to run in (fixed) super-polynomial time. Very informally, SPS security guarantees that any polynomial-time attack in the real execution can also be mounted in the ideal world execution, albeit in super-polynomial time. This is directly applicable and meaningful in settings where ideal world security is guaranteed statistically or information-theoretically (which would be the case in most "end-user" functionalities that have been considered, from privacy-preserving data mining to electronic voting). As a result, SPS has been extensively studied. SPS security for concurrently composable zero knowledge proofs was first studied by [Pas03], and SPS security for concurrently composable secure computation protocols was first studied by [PS04, BS05]. The SPS definition guarantees security with respect to concurrent self-composition of the secure computation protocol being studied, and guarantees security with respect to general

[1] As Bob Dylan puts it, "If you need somebody you can trust, trust yourself."

concurrent composition with arbitrary other protocols in the context of super-polynomial adversaries.

In recent years, the design of secure computation protocols in the plain model with SPS security has been the subject of several works [PS04, BS05, LPV09, CLP10]. Very recently, Canetti, Lin, and Pass [CLP10] obtained the first secure computation protocol that achieves SPS security based on *standard assumptions*[2]. Unfortunately, however, the improvement in terms of assumptions comes at the cost of the round complexity of the protocol. Specifically, the protocol of [CLP10] incurs *polynomial-round complexity*[3].

Round Complexity. The latency of sending messages back and forth has been shown to often be the dominating factor in the running time of cryptographic protocols [MNPS04, BDNP08]. Therefore, round complexity is an efficiency parameter of special importance, and often dictates the feasibility of actual usage of a protocol[4]. Indeed, round complexity has been the subject of a great deal of research in cryptography. For example, in the context of concurrent zero knowledge (ZK) proofs, round complexity was improved in a sequence of works [RK99, KP01, PRS02] from polynomial to slightly super-logarithmic (that nearly matches the lower bound w.r.t. black-box simulation [CKPR01]). Indeed, achieving constant-round concurrent ZK (with non black-box simulation) is a widely-respected open problem. The round complexity of non-malleable commitments in the stand-alone and concurrent settings has also been studied in several works [DDN00, Bar02, PR05b, PR05a, LP09, Wee10, Goy11, LP11], improving the round complexity from logarithmic rounds to constant rounds under minimal assumptions. These results are celebrated both for improving round complexity and for introducing new insights about simulation-based proofs of security. We observe that for the setting of concurrently secure computation protocols with SPS security, the situation is much worse since the only known protocol that achieves SPS security based on standard assumptions incurs polynomial-round complexity [CLP10]. This suggests the following natural question:

What is the asymptotic round complexity of concurrently composable secure computation protocols with SPS security, based on standard assumptions?

1.1 Our Results

In this work, we resolve this question by exhibiting a *constant-round* concurrently composable secure computation protocol that achieves SPS security based on only *standard assumptions.* Our construction only uses black-box simulation techniques. At the heart of our result is a new proof technique that significantly reduces the demands placed on "rewinding techniques" employed in previous work (more details below). In contrast to prior works where several powerful tools were employed to obtain positive results, e.g., CCA-secure commitments [CLP10], our new proof technique allows us to only use relatively less powerful primitives, such as standard non-malleable commitments. We believe that our techniques are of independent interest and likely to be applicable in other settings related to secure concurrent composition (see below).

[2] In fact, the work of [CLP10], together with [PS04, BS05], considers the stronger "angel-based security model" of [PS04]. In this work, we focus only on SPS security.

[3] We note in passing that under standard assumptions, if a protocol uses n^{10} rounds where n is a computational security parameter, then one can equivalently say that the protocol uses only k^{ϵ} rounds by setting the security parameter $k = n^{10/\epsilon}$. If the original protocol allowed the adversary to succeed with an advantage negligible in n, then the adversary will also only be able to attain advantage negligible in k.

[4] One can see the fundamental nature of round complexity as a measure of efficiency by considering the following: As time goes on, computing power (by means of improvements in speed or parallelism) becomes cheaper and network pipes become wider, greatly decreasing sensitivity to computational and communication complexities in cryptographic protocols. Indeed, as the computational power of honest parties and adversaries grows, under standard assumptions like one-way functions, this heavily favors the honest parties. However, the inefficiency caused by rounds of communication is tied to the speed of light, which is constant (see e.g. [Ein05]), and therefore beyond the assistance of technological advances in computing.

Our positive result relies on the nearly minimal assumptions that constant-round (semi-honest) oblivious transfer (OT) exists and collision-resistant hash functions (CRHFs) exist.[5] Furthermore, we note that our proof of security is *black-box*.

1.2 Our Techniques

A ubiquitous technique for simulation-based proofs in cryptography is that of *rewinding* the adversary. In the concurrent setting (which is the setting we consider in this paper), where an adversary can interleave messages from different protocols in any arbitrary manner, rewinding an adversary (to correctly simulate each session) is often problematic and typically requires a large number of rounds (in a single protocol). For example, in the context of concurrent zero knowledge, the best known result [PRS02] requires super-logarithmic round complexity, which nearly matches the lower bound w.r.t. black-box simulation [CKPR01].

A common technique employed by previous works [BS05, LPV09] to get around this issue (in order to achieve constant-rounds) is *complexity-leveraging* [CGGM00]. Roughly speaking, this means that we assume primitives A and B such that A cannot be broken in polynomial time, but can be broken in time $T(k)$ for some super-polynomial function T, while we assume that B *cannot* be broken even in time $T(k)$. Unfortunately, however, complexity leveraging inherently relies on non-standard assumptions (such as sub-exponentially hard one-way functions). Since our focus is on obtaining constant-round protocols based on standard assumptions, we do not make any use of complexity leveraging. We therefore instead wrestle with the problem of using rewinding in the concurrent setting.

To deal with the problem of concurrent rewinding, we develop a novel proof technique that, very crucially, significantly relaxes the properties that we need from our rewinding technique, which in turn allows us to obtain our result. In the following discussion, we give a more detailed intuition behind our techniques, where we assume somewhat greater familiarity with recent work in this area.

We first note that all prior works on obtaining secure computation protocols with SPS security crucially use the super-polynomial time simulator to "break" some cryptographic scheme and extract some "secret information". Then, to avoid any complexity-leveraging type technique, and yet argue security, the technique used in [CLP10] was to replace the super-polynomial time simulator with a polynomial-time rewinding "hybrid experiment" via a hybrid argument in the security proof. Indeed, this is why their protocol incurs large round complexity (so as to facilitate concurrent-rewinding). We also make use of rewinding, but crucially, in a weaker way. The main insights behind our rewinding technique are explained as follows:

- We first note that (like other works) we will restrict our usage of rewinding only to the creation of "look-ahead threads". Very roughly, this means that a rewinding simulator never changes its actions on the "main thread" of execution; and as such, the rewinding is employed only to extract some information from the adversary. Here, we again stress that our final simulator does not perform any rewinding, and that we only perform rewindings in hybrid experiments to bridge the gap between the real and ideal world executions.

- Now that we use rewindings only to extract some information from the adversary, and only in hybrid experiments, we make the critical observation that, in fact, we can make use of the secret inputs of the honest parties in the look-ahead threads. Indeed, in *all* our intermediate hybrid experiments, we perform rewindings to create look-ahead threads where we make "judicious" use of the honest party's inputs. In this manner, we eventually end up with a rewinding (hybrid) simulator that simulates the *main thread* without the honest party's inputs, but still uses them in the look-ahead threads (in a manner that guarantees extraction). This is our main conceptual

[5] We believe that our assumption of CRHFs can be removed by employing techniques from the recent work of [LPTV10], leaving only the minimal assumption that constant-round OT exists. We leave this for the full version of this paper.

deviation from prior work, where, to the best of our knowledge, honest party's inputs were only used in *some* intermediary hybrids, with the main goal being to eventually remove their usage even from the look-ahead threads. We show that this is in fact unnecessary, since our final simulator does not perform any rewindings, but instead runs in super-polynomial time to extract the same information that was being earlier extracted via rewinding in the hybrid experiments. We only need to argue that the main thread output by the rewinding (hybrid) experiment and the main thread output by the final simulator be indistinguishable. Indeed, we are able to argue that there is only a small statistical distance between our final simulator (that corresponds to the ideal execution) and the previous rewinding-based hybrid experiment. This statistical distance corresponds to the probability that the rewinding-based extraction is unsuccessful, since the SPS extraction is always successful.

- We further note that since we use the honest party's inputs in the look-ahead threads, we can bypass complex *recursive* rewinding schedules used in previous works and simply use "local rewindings" that only require constant rounds (in fact, only "one slot").

- Finally, we observe that since we perform rewindings only in hybrid experiments, we do not need the rewinding to succeed with probability negligibly close to 1, as is needed for concurrent ZK. Instead, we only require rewinding to succeed with probability $1 - \epsilon$, where ϵ is related to the success probability of the distinguisher that is assumed to exist for the sake of contradiction. This observation, yet again, allows us to use a simpler rewinding strategy.

- Our overall proof strategy only makes use of relatively well understood primitives like standard non-malleable commitments. This is a departure from [CLP10] which introduces a new primitive called CCA-secure commitment schemes.

At this point, an informed reader may question the feasibility of a "sound implementation" of the above approach. Indeed, a-priori it is not immediately clear whether it is even possible for the simulator to "cheat" on the main thread, yet behave honestly in look-ahead threads *at the same time*. In a bit more detail, recall that any given look-ahead thread shares a prefix with the main thread of execution. Now consider any session i on a look-ahead thread. Note that since some part of session i may already be executed on the *shared prefix*, it is not clear how the simulator can continue simulating session i on the look-ahead thread *without ever performing any recursive rewindings* if it was already cheating in session i on the shared prefix.

We address the above issues by a careful protocol design that guarantees that a rewinding simulator can always extract some "trapdoor" information *before* it "commits" to cheating in any session. As a result, during the simulation, whenever a look-ahead thread is forked at any point from the main thread, the simulator can either always continue cheating, or simply behave honestly (without any conflict with the main thread) in any session.

We further note that the use of honest party inputs in look-ahead threads leads to some additional challenges in constructing a "valid" sequence of hybrid arguments for the proof of security. We address this by creating a novel sequence of hybrid arguments that deviates from the most common approach of giving a "session-by-session" hybrid argument. Specifically, our sequence of hybrids "touches" each session i several times, *not necessarily in succession*, with the exact hybrid changes depending on the overall scheduling of messages by the adversary. (In contrast, typically such hybrid arguments involve modifications to any session i by a series of *successive* hybrids regardless of the scheduling.)

Finally, we believe that our new proof technique will be valuable in other contexts related to concurrent composition of protocols, including ones that do not consider super-polynomial simulation. Indeed, note that in our proof, SPS is used only at the very last step.

Proof Overview. For a more technical overview of the main ideas involved in our proof technique, we refer the reader to Section 5. This technical overview refers to the technical description of the protocol, given in Section 4.

Other Related Work. Here we discuss some additional prior work related to the work in this paper. We note that while the focus of this work is on SPS security as a means to obtain concurrently-secure protocols in the plain model, some recent works have investigated alternative security models for the same. Very recently, [GS09, GJO10] considered a model where the ideal world adversary is allowed to make additional queries (as compared to a single query, as per the standard definition) to the ideal functionality per session. The techniques we use in our work are related to, but quite different from, their work (which in turn builds on techniques from the work of [BPS06]). Another model for concurrent security in the plain model called input indistinguishable computation (IIC), was proposed by [MPR06], as mentioned above. Very roughly, IIC is a weakening of the standard notion of secure computation in that we only require that an adversary cannot distinguish between any two different executions – where the honest parties use different inputs in each execution – as long as the adversary obtains the same outputs in each execution.

2 UC Security

In this section we briefly review UC security. For full details see [Can00]. For the sake of completeness we include a short introduction that has been taken verbatim from [CLP10] in Appendix A. Following [GMR89, Gol01], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant.

Security of protocols. Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality.

Securely realizing an ideal functionality. We say that a protocol Π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π, or it is interacting with $\mathcal{S}$ and parties running ϕ. This means that, from the point of view of the environment, running protocol Π is 'just as good' as interacting with ϕ. We say that Π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0, 1\}$.

Definition 1 *Let Π and ϕ be protocols. We say that Π UC-emulates ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.*

Definition 2 *Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol. We say that Π UC-realizes $\mathcal{F}$ if Π UC-emulates the ideal process $\Pi(\mathcal{F})$.*

UC Security with Super-polynomial Simulation We next provide a relaxed notion of UC security by giving the simulator access to super-poly computational resources. The universal composition theorem generalizes naturally to the case of UC-SPS, the details of which we skip.

Definition 3 *Let Π and ϕ be protocols. We say that Π UC-SPS-emulates ϕ if for any adversary $\mathcal{A}$ there exists a* super-polynomial time *adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.*

Definition 4 *Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol. We say that Π UC-SPS-realizes $\mathcal{F}$ if Π UC-SPS-emulates the ideal process $\Pi(\mathcal{F})$.*

For simplicity of exposition, in the rest of this paper we assume authenticated communication; that is, the adversary may deliver only messages that were actually sent. (This is however not essential as shown previously [BCL$^+$05].)

3 Building Blocks

We now discuss the main cryptographic primitives that we use in our construction.

3.1 Statistically Binding String Commitments

In our protocol, we will use a (2-round) statistically binding string commitment scheme, e.g., a parallel version of Naor's bit commitment scheme [Nao91] based on one-way functions. For simplicity of exposition, in the presentation of our results in this manuscript, we will actually use a non-interactive perfectly binding string commitment.[6] Such a scheme can be easily constructed based on a 1-to-1 one way function. Let $\textsc{com}(\cdot)$ denote the commitment function of the string commitment scheme. For simplicity of exposition, in the sequel, we will assume that random coins are an implicit input to the commitment function.

3.2 Extractable Commitment Scheme

We will also use 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 $\textsc{com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding string commitment scheme (as described in Section 3). 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 $\textsc{com}$. Let $B \leftarrow \textsc{com}(\mathtt{str})$, and $A_i^0 \leftarrow \textsc{com}(\alpha_i^0)$, $A_i^1 \leftarrow \textsc{com}(\alpha_i^1)$ for every $i \in [k]$.

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

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

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.

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

Modified Commitment Scheme. Due to technical reasons, we will also use a minor variant, denoted $\langle C', R' \rangle$, of the above commitment scheme. Protocol $\langle C', R' \rangle$ is the same as $\langle C, R \rangle$, except that for a given receiver challenge string, the committer does not "open" the commitments, but instead simply reveals the appropriate committed values (without revealing the randomness used to create the corresponding commitments). More specifically, in protocol $\langle C', R' \rangle$, on receiving a challenge string $v_1, \ldots, v_n$ from the receiver, the committer uses the following strategy: for every $i \in [k]$, if $v_i = 0$, C' sends α_i^0, otherwise it sends α_i^1 to R'. Note that C' does not reveal the decommitment values associated with the revealed shares.

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

Proving Validity of the Commit Phase. We say that commit phase between C' and R' is *valid* with respect to a value $\hat{\mathtt{str}}$ if there exist values $\{\hat{\alpha}_i^0, \hat{\alpha}_i^1\}_{i=1}^k$ 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 can define *validity* condition for the commitment protocol $\langle C, R \rangle$ in a similar manner.

3.3 Constant-Round Non-Malleable Zero Knowledge Argument

In our main construction, we will use a constant-round non-malleable zero knowledge (NMZK) argument for every language in **NP** with perfect completeness and negligible soundness error. In particular, we will use a specific (stand-alone) NMZK protocol, denoted $\langle P, V \rangle$, based on the concurrent-NMZK protocol of Barak et al [BPS06]. Specifically, we make the following two changes to Barak et al's protocol: (a) Instead of using an $\omega(\log n)$-round PRS preamble [PRS02], we simply use the one-slot commitment scheme $\langle C, R \rangle$ (described above). (b) Further, we require that the non-malleable commitment scheme being used in the protocol be constant-round and public-coin w.r.t. receiver. We note that such commitment schemes are known due to Pass, Rosen [PR05b]. Further, in Section 3.4, we show how to adapt the scheme of Goyal [Goy11] to incorporate the public-coin property.[7] We now describe the protocol $\langle P, V \rangle$.

[7] We note that while the commitment scheme of [PR05b] admits a non black-box security proof, the security proof of Goyal's scheme is black-box. As such, the resultant NMZK protocol has a black-box security proof as well.

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

PREAMBLE PHASE.

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

POST-PREAMBLE PHASE.

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

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 constant-round public-coin extractable non-malleable commitment scheme.

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

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

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

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

Decoupling the Preamble Phase from the Protocol. Note that the preamble phase in $\langle P, V \rangle$ is independent of the proof statement and can therefore be executed by P and V *before* the proof statement is fixed. Indeed, this is the case when we use $\langle P, V \rangle$ in our main construction in Section 4. Specifically, in our main construction, the parties first engage in multiple executions of $\langle C, R \rangle$ at the beginning of the protocol. Later, when a party (say) P_i wishes to prove the validity of a statement π to (say) P_j, then P_i and P_j engage in an execution of the post-preamble phase of $\langle P, V \rangle$ for statement π. The protocol specification fixes a particular instance of $\langle C, R \rangle$ that was executed earlier as the preamble phase of this instance of $\langle P, V \rangle$. In the description of our main construction, we will abuse notation and sometimes refer to the post-preamble phase as $\langle P, V \rangle$.

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

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

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

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

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

3.4 Constant Round Public-Coin Non-Malleable Commitments

In this section, we sketch a modification to Goyal's commitment scheme to make it public coin. The main idea is to use a (public-coin) witness-indistinguishable argument (as opposed to a ZK argument) to prove consistency. More specifically, the idea is to commit to two strings in parallel and then prove that at least one of them was a valid commit phase. In the opening phase, if both of them were valid, the receiver simply takes the larger among the two as the committed value. This ensures that the commitment scheme is still hiding without affecting the proof of non-malleability.

In this section, we describe our basic protocol for "small" tags with one sided non-malleability. They can be extended to the general case by relying on techniques from [PR05b] (while maintaining the public-coin property). We assume that each execution has a tag $tag \in [2n]$. Denote by ℓ the value $k \cdot tag$. Let $\text{COM}(m)$ denote a commitment to the message m with the first message σ under the statistically binding commitment scheme of Naor. Whenever we need to be explicit about the randomness used to generate the commitment, we denote it as $\text{COM}(m; r)$ where r is the said randomness. The commitment scheme $\langle C, R\rangle$ between a committer $\mathcal{C}$ trying to commit to ν and a receiver $\mathcal{R}$ proceeds as follows.

Commitment Phase.

0. **Initialization Message.** The receiver $\mathcal{R}$ generates the first message σ of the Naor commitment scheme and sends it to $\mathcal{C}$.

   The committer now runs the primary slot phase and the verification message phase (together called the commit phase) for two strings $\nu[0]$ and $\nu[1]$ in parallel. It sets $\nu[0]$ to be the actual value ν it wants to commit to and sets $\nu[1] = 0$. For $b \in \{0, 1\}$, do the following (with fresh randomness each time):

   Primary Slot

1. The committer $\mathcal{C}$ generates ℓ pairs of random strings $\{\alpha_i^0, \alpha_i^1\}_{i\in[\ell]}$ (with length of each string determined by the security parameter). $\mathcal{C}$ further generates commitments of these strings $\{A_i^0 = \text{COM}(\alpha_i^0), A_i^1 = \text{COM}(\alpha_i^1)\}_{i\in[\ell]}$ and sends them to $\mathcal{R}$ ($\mathcal{C}$ uses fresh randomness to generate each commitment).

2. The receiver $\mathcal{R}$ generates and sends to $\mathcal{C}$ a random ℓ-bit challenge string $ch = (ch_1, \ldots, ch_\ell)$.

3. The committer $\mathcal{C}$ sends to $\mathcal{R}$ the values $\alpha_1^{ch_1}, \ldots, \alpha_\ell^{ch_\ell}$. Note that $\mathcal{C}$ *does not* send the openings associated with the corresponding commitments. $\mathcal{R}$ responds with an acknowledgement message on receiving these values [8].

4. **Verification Message.** Define ℓ strings $\{\alpha_i\}_{i\in[\ell]}$ such that $\alpha_i = \alpha_i^0 \oplus \alpha_i^1$ for all $i \in [\ell]$. $\mathcal{C}$ generates ℓ commitments $B_i = \text{COM}(\nu[b]; \alpha_i)$ for $i \in [\ell]$ and sends them to $\mathcal{R}$. (That is, randomness α_i is used to generate the i-th commitment to ν).

[8] This is done for technical reasons to ensure that this and the next message by $\mathcal{C}$ are in different rounds of the protocol

5. **Consistency Proof.** The committer $\mathcal{C}$ and the receiver $\mathcal{R}$ now engage in a witness indistinguishable argument protocol where $\mathcal{C}$ proves to $\mathcal{R}$ that at least one of the above commit phases is "valid". That is, for at least one commit phase (say indexed by b), there exist values $\hat{\nu}[b]$, $\{\hat{\alpha}_i, \hat{\alpha}_i^0, \hat{\alpha}_i^1\}_{i \in [\ell]}$ such that for all i:
   - $\hat{\alpha}_i^0 \oplus \hat{\alpha}_i^1 = \hat{\alpha}_i$, and,
   - commitments A_i^0 and A_i^1 are valid commitments to the strings $\hat{\alpha}_i^0$ and $\hat{\alpha}_i^1$ respectively under some random tape, and,
   - commitment B_i is a valid commitment to $\hat{\nu}[b]$ under the random tape $\hat{\alpha}_i$.

Decommitment Phase. The committer $\mathcal{C}$ simply reveals the committed value ν and the randomness used in running steps 1 to 4. The receiver $\mathcal{R}$ checks if the messages in the primary slot and the verification message were computed honestly using the revealed randomness *for at least one commit phase.* If so, $\mathcal{R}$ takes the value committed ν to be the larger among the (either one or two) valid committed values. If none of the commit phases is valid, $\mathcal{R}$ takes the value committed to as $\perp$.

Lemma 1 *The commitment scheme $\langle C, R \rangle$ is computationally hiding (in the stand alone setting).*

The proof is this lemma is relies on a standard hybrid argument. To go from one committed value ν_1 to the other ν_2, we use the follow strategy. In the first hybrid, use the witness corresponding to the second commit phase (in which we commit 0) to complete the WI argument. Next, we replace the value ν_1 by ν_2 in the first commit phase. Finally, use the witness corresponding to the first commit phase to complete the WI argument.

Lemma 2 *The commitment scheme $\langle C, R \rangle$ is statistically binding (in the stand alone setting).*

The proof of this lemma is similar to that in the original scheme of Goyal [Goy11].

Theorem 1 *The commitment scheme $\langle C, R \rangle$ is a one sided non-malleable commitment scheme against a synchronizing adversary.*

The proof of this theorem is similar to that in the original scheme of Goyal [Goy11]. Note that the proof of non-malleability in [Goy11] only relies on the standalone (computational) hiding property of the left session as well as the soundness of the consistency phase (and does not additionally rely on the proof of consistency being a zero-knowledge). Both of these properties are preserved as we replace the ZK protocol by a WI one. In addition, the second parallel commit phase on the left (which is a commitment to 0) is essentially independent of the rest of the left session and could have been generated by the man-in-the-middle on its own. Hence these modifications do not affect the non-malleability of the commitment scheme. More details will be provided in the final version.

3.5 Constant-Round Statistically Witness Indistinguishable Arguments

In our construction, we shall use a constant-round 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 protocol can be constructed by using $\omega(\log n)$ 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 constant-round statistically hiding commitment scheme [NY89, HM96, DPP97].

3.6 Semi-Honest Two Party Computation

We will also use a constant-round semi-honest two party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ for any functionality $\mathcal{F}$ in the stand-alone setting. The existence of such a protocol follows from the existence of constant-round semi-honest 1-out-of-2 oblivious transfer [Yao86, GMW87, Kil88].

4 Our Construction

Let $\mathcal{F}$ be any well-formed functionality[9] that admits a constant round two-party computation protocol in the semi-honest setting. In particular, $\mathcal{F}$ can be a universal functionality. In this section we will give a protocol Π that UC-SPS-realizes $\mathcal{F}$. Note that in the UC framework any two parties (say P_i and P_j) might interact as per the protocol Π on initiation by the environment for some session corresponding to a SID *sid*. For simplicity of notation, we will describe the protocol in terms of two parties P_1 and P_2, where these roles could be taken by any two parties in the system. Further we will skip mentioning the SID to keep the protocol specification simple.

In order to describe our construction, we first recall the notation associated with the primitives that we use in our protocol. Let $\textsc{com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding commitment scheme, and let $\langle C, R \rangle$ denote the one-slot extractable commitment scheme, and $\langle C', R' \rangle$ be its modified version (see Section 3.2). Further, we will use our constant-round NMZK protocol $\langle P, V \rangle$ (see Section 3.3), a constant-round SWI argument $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$, and a constant-round *semi-honest* two party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ that securely computes $\mathcal{F}$ as per the standard simulation-based definition of secure computation.

Let P_1 and P_2 be two parties with inputs x_1 and x_2 provided to them by the environment $\mathcal{Z}$. Let n be the security parameter. Protocol $\Pi = \langle P_1, P_2 \rangle$ proceeds as follows.

I. Trapdoor Creation Phase.

1. $P_1 \Rightarrow P_2$: P_1 samples a random string σ_1 (of appropriate length; see below) and engages in an execution of $\langle C, R \rangle$ with P_2, where P_1 commits to σ_1. We will denote this commitment protocol by $\langle C, R \rangle_{1 \to 2}$.

2. $P_2 \Rightarrow P_1$: P_2 now acts symmetrically. That is, P_2 samples a random string σ_2 and commits it via an execution of $\langle C, R \rangle$ (denoted as $\langle C, R \rangle_{2 \to 1}$) with P_1.

3. $P_1 \Rightarrow P_2$: P_1 creates a commitment $com_1 = \textsc{com}(0)$ to bit 0 and sends com_1 to P_2. P_1 and P_2 now engage in an execution of (the post-preamble phase of) $\langle P, V \rangle$, where P_1 proves that com_1 is a commitment to bit 0. The commitment protocol $\langle C, R \rangle_{2 \to 1}$ (executed earlier in step 2) is fixed as the preamble phase for this instance of $\langle P, V \rangle$ (see Section 3.3).

4. $P_2 \Rightarrow P_1$: P_2 now acts symmetrically.

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. As discussed earlier in Section 1.2, in order to bypass the need of recursive rewindings (even though we consider concurrent security), we want to ensure that a "hybrid" simulator (that performs rewindings) can always extract a "trapdoor" *before* it begins cheating in any protocol session. Here, we achieve this effect by de-coupling the preamble phase of $\langle P, V \rangle$ from the post-preamble phase (see Section 3.3) and executing the preamble phase at the very beginning of our protocol.

[9] See [CLOS02] for a definition of well-formed functionalities.

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 $\hat{x}_1 \| \hat{r}_1$ (see Section 3.2), *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 $\hat{x}_1 \| \hat{r}_1$ (see Section 3.2), 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.

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

Theorem 2 *Assume the existence of constant round semi-honest OT and collision resistant hash functions.Then for every well-formed functionality $\mathcal{F}$, there exists a constant-round protocol that UC-SPS-realizes $\mathcal{F}$.*

We prove the above claim by arguing that the protocol $\Pi = \langle P_1, P_2 \rangle$ described earlier UC-SPS-realizes $\mathcal{F}$. Note that our simulator will run in sub-exponential time, where the desired parameters can be obtained by using a "scaled-down" security parameter of the commitment scheme COM. We prove this in the next section.

5 Proof of Security

In order to prove Theorem 2, we will first construct a super-polynomial time simulator $\mathcal{S}$ that simulates the view of $\mathcal{A}$ in the UC setting. We will then argue that the output distributions of environment in the real and the ideal world executions are computationally indistinguishable, thus satisfying Definition 4. We describe the construction of $\mathcal{S}$ in Section 5.1 and give a proof outline in Section 5.2. Finally, we argue the correctness of simulation in Section 6. We first give some notation.

Notation. In the UC framework, any two parties (say P_i and P_j) might interact as per the protocol Π on initiation by the environment $\mathcal{Z}$. Simulator has to simulate[10] the view of the corrupted party if exactly one of the two parties is corrupted. In the other case the simulator does not have to do anything. In the sequel, we will use the notation H to denote the honest party and $\mathcal{A}$ to denote the corrupted party in any session. Let $\langle P, V\rangle_{H\to\mathcal{A}}$ denote the instance of $\langle P, V\rangle$ where H and $\mathcal{A}$ play the roles of prover P and verifier V respectively. Similarly, let $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{H\to\mathcal{A}}$ denote each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$ where H and $\mathcal{A}$ plays the roles of prover P and verifier V respectively. Now, recall that H plays the role of committer C in one instance of $\langle C, R\rangle$, where it commits to its preamble secret σ_H, and in one instance of $\langle C', R'\rangle$, where it commits to its input x_H and randomness r_H (to be used in the secure computation phase). We will reserve the notation $\langle C, R\rangle_{H\to\mathcal{A}}$ for the former case, and we will refer to the latter case by $\langle C', R'\rangle_{H\to\mathcal{A}}$. Further, we define $\langle P, V\rangle_{\mathcal{A}\to H}$, $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}$, $\langle C, R\rangle_{\mathcal{A}\to H}$, $\langle C', R'\rangle_{\mathcal{A}\to H}$ in the same manner as above, except that the roles of H and $\mathcal{A}$ are interchanged. Also, let $x_{\mathcal{A}}$ and $r_{\mathcal{A}}$ denote the input and random coins, respectively, of $\mathcal{A}$ (to be used in the secure computation phase). For the sake of simplicity, we will skip mentioning the session identifier unless necessary.

5.1 Description of Simulator $\mathcal{S}$

The simulator $\mathcal{S}$ consists of two parts, S_{main} and S_{ext}. Informally speaking, S_{main} is essentially the main simulator in that it interacts with the adversary $\mathcal{A}$. At various points during the simulation, S_{main} invokes S_{ext} in order to extract the following two values in each session: (a) the preamble secret $\sigma_{\mathcal{A}}$ committed by $\mathcal{A}$ in $\langle C, R\rangle_{\mathcal{A}\to H}$, and (b) the input $x_{\mathcal{A}}$ and randomness $r_{\mathcal{A}}$ committed by $\mathcal{A}$ in $\langle C', R'\rangle_{\mathcal{A}\to H}$. S_{ext} takes as input the transcript of an instance of the commitment protocol $\langle C, R\rangle_{\mathcal{A}\to H}$ (resp. $\langle C', R'\rangle_{\mathcal{A}\to H}$) and extracts the committed value $x_{\mathcal{A}}\|r_{\mathcal{A}}$ (resp., $\sigma_{\mathcal{A}}$) by running in super-polynomial time and breaking the hiding property of the commitment scheme COM (which is used in the construction of $\langle C, R\rangle$; see Section 3.2). We now give more details.

Description of S_{main}. We first describe the strategy of S_{main} in each phase of the protocol. For the sake of simplicity, below we describe the case in which the honest party sends the first message in the protocol. The other case, in which the adversary sends the first message, can be handled in an analogous manner and is omitted.

Trapdoor Creation Phase. In Steps 1 and 2 of the Trapdoor Creation Phase, the simulator follows the protocol specification and behaves exactly like an honest party. However, on the completion of the preamble $\langle C, R\rangle_{\mathcal{A}\to H}$ executed in Step 2, S_{main} extracts the preamble secret (committed by $\mathcal{A}$) by invoking S_{ext}. If S_{ext} returns a valid preamble secret $\sigma_{\mathcal{A}}$, then in Step 3, instead of committing to bit 0, S_{main} sends com_1 as a commitment to bit 1 and simulates the post-preamble phase of $\langle P, V\rangle_{H\to\mathcal{A}}$ in a straight-line manner (by using the preamble secret $\sigma_{\mathcal{A}}$; in the same manner as explained in Section 3.3). On the other hand, if S_{ext} returns $\perp$, then S_{main} executes Step 3 by following the honest party strategy. Finally, in Step 4, simulator again behaves just like an honest party.

[10] We only deal with static corruption.

As explained later in the description of S_{ext}, S_{ext} always succeeds in extracting the preamble secret $\sigma_{\mathcal{A}}$ in super-polynomial time as long as the commitment protocol $\langle C, R\rangle_{\mathcal{A}\to H}$ is *valid* (see Section 3.2) since $\langle C, R\rangle$ is a perfectly binding commitment scheme. In other words, S_{ext} only outputs $\perp$ if the commitment protocol $\langle C, R\rangle_{\mathcal{A}\to H}$ is not *valid*. Note that in this case, when S_{main} executes Step 3 in an honest fashion, $\mathcal{A}$ would fail with probability 1 in successfully decommitting to the preamble secret (since $\langle C, R\rangle$ is perfectly binding) during the post-preamble phase of $\langle P, V\rangle_{H\to\mathcal{A}}$. As a consequence, S_{main} (who is following the honest party strategy) will abort that session.

Input Commitment Phase.

1. In this phase, S_{main} first commits to a (sufficiently large) string of all zeros (unlike the honest party that commits to its input x_H and randomness r_H) in the execution of the commitment protocol $\langle C, R\rangle_{H\to\mathcal{A}}$. S_{main} then engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{H\to\mathcal{A}}$ with $\mathcal{A}$, where (unlike the honest party that uses the real witness) S_{main} uses the trapdoor witness. Note that the trapdoor witness is available to S_{main} since it committed to bit 1 in the trapdoor commitment phase.

2. Next, S_{main} behaves honestly in Step 2 of the Input Commitment Phase. However at the end of the phase, S_{main} extracts the input and randomness committed by $\mathcal{A}$ in $\langle C', R'\rangle_{\mathcal{A}\to H}$, by invoking S_{ext} with the transcript of $\langle C', R'\rangle_{\mathcal{A}\to H}$. If S_{ext} outputs $\perp$, then S_{main} stops its interaction with $\mathcal{A}$ and outputs a special abort message called $\mathtt{I\text{-}Abort}_1$. (Later, we show that S_{main} outputs $\mathtt{I\text{-}Abort}_1$ with only negligible probability.)

Secure Computation Phase. Let S_{sh} denote the simulator for the semi-honest two-party protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$ used in our construction. S_{main} internally runs the simulator S_{sh} on adversary's input $x_{\mathcal{A}}$. S_{sh} starts executing, and, at some point, it makes a call to ideal functionality $\mathcal{F}$ in the ideal world with an input string (say) $x_{\mathcal{A}}$. At this point, S_{main} makes a query $(sid, x_{\mathcal{A}})$[11] to $\mathcal{F}$. The output value received from $\mathcal{F}$ is forwarded to S_{sh}. S_{sh} runs further, and finally halts and outputs a transcript $\beta_{H,1}, \beta_{\mathcal{A},1}, \ldots, \beta_{H,t}, \beta_{\mathcal{A},t}$ of the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$, and an associated random string $\hat{r}_{\mathcal{A}}$. S_{main} now performs the following steps.

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

2. Now, in each round $j \in [t]$, S_{main} sends $\beta_{H,j}$. It then engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{H\to\mathcal{A}}$ with $\mathcal{A}$ where it uses the trapdoor witness (deviating from honest party strategy that used the real witness). Next, on receiving $\mathcal{A}$'s next message $\beta_{\mathcal{A},j}$ in the protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$, S_{main} engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}$ with $\mathcal{A}$ where it uses the honest verifier strategy. Finally at any stage, if the j^{th} message of the adversary is not $\beta_{\mathcal{A},j}$ and the proof $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}$ given immediately after this messages is accepted, then the simulator aborts all communication and outputs a special abort message called $\mathtt{I\text{-}Abort}_2$. (Later, we show that S_{main} outputs $\mathtt{I\text{-}Abort}_2$ with only negligible probability.)

Finally, simulator forwards all messages from the environment and the adversary sent to each other as such. And This completes the description of S_{main}. We now proceed to describe S_{ext}.

Description of S_{ext}. S_{ext} receives as input the transcript of an instance of either the commitment protocol $\langle C, R\rangle^{\sigma}_{\mathcal{A}\to H}$ or the commitment protocol $\langle C', R'\rangle_{\mathcal{A}\to H}$. On receiving such an input, S_{ext} runs in super-polynomial time and breaks the hiding property of the commitment scheme COM to extract

[11] Note that the session identifier *sid* corresponds to the specific session in which the parties are interacting. We have skipped mentioning it everywhere because all messages correspond the same session *sid*.

the value committed in the transcript of the commitment protocol. On successful extraction, S_{ext} returns the extracted value (which is either the preamble secret $\sigma_{\mathcal{A}}$, or the input and randomness $x_{\mathcal{A}}$, $r_{\mathcal{A}}$, depending upon the transcript received from S_{main}) to S_{main}.

In more detail, on receiving an input transcript from S_{main}, S_{ext} *breaks* (by running in super-polynomial time) each commitment in the transcript, including the commitment to the main value (which is either $\sigma_{\mathcal{A}}$ or $x_{\mathcal{A}} \| r_{\mathcal{A}}$) and the commitments to its secret shares. Note that each commitment represents a unique value since COM is perfectly binding. Then, if the secret shares thus extracted are not *consistent* with the main value (i.e., if the input transcript does not represent a *valid* commitment protocol; see Section 3.2), S_{ext} outputs $\perp$; otherwise, S_{ext} outputs the extracted value.

5.2 Proof Outline

We presented the description of the simulator in the previous section. For UC-SPS security we need to argue that for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ (running in super-poly time) such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π (referred to as real world), or it is interacting with $\mathcal{S}$ interacting with the trusted ideal functionality (referred to as ideal world). In order to argue this starting with the real world we will consider a sequence of hybrid experiments that lead to the ideal world. We will then argue indistinguishability of consecutive hybrids. Note that in the final hybrid the simulator runs in super-poly time. Since we are relying on computational assumptions that are secure only against polynomial time adversaries, the simulator cannot use its super-polynomial power across hybrids that are only computationally indistinguishable. We deal with this issue by having our simulator run in polynomial time in all the hybrids except the last one. Further the final switch to the last hybrid from the penultimate hybrid is based on a statistical argument and therefore the running time of the simulator in the last hybrid is irrelevant.

We will try to convey the main ideas of our proof by describing the penultimate hybrid (i.e., the last hybrid that runs in polynomial time) and explaining how it works. Consider the following sequence of hybrids:

- H_0: The real world interaction.
- $H_{1/2}$ [12]: This is the final poly-time hybrid. All hybrids before this hybrid will run in poly-time. The key point in this hybrid is that the simulator does not use the inputs of honest parties in the main thread. However, it uses the honest parties inputs in the look-ahead threads. The look-ahead threads are executed to help with extraction of preamble secrets and the inputs used by the adversary. We describe this hybrid next.
- H_1: The simulated interaction, as described in Section 5.1. This hybrid runs in super-poly time.

The penultimate hybrid – $H_{1/2}$. Before we describe our simulation strategy in this hybrid let us describe some notation: we will maintain two databases – $\texttt{Database}^{\sigma}$ and $\texttt{Database}^{x}$, and our hybrid will crucially refer to the notions of *special messages*, *honest execution, partial simulation* and *full simulation.*

Databases. In database $\texttt{Database}^{\sigma}$ we will store tuples of the form $(j, \sigma^j_{\mathcal{A}})$, where $\sigma^j_{\mathcal{A}}$ is the preamble secret committed by the adversary in session j. Similarly, in database $\texttt{Database}^{x}$ we will store tuples of the form $(j, x^j_{\mathcal{A}})$, where $x^j_{\mathcal{A}}$ is the input and the randomness committed by the adversary in session j.

Special message notation. In our protocol we will demarcate four messages as *special messages* –

[12] Same as hybrid $\mathcal{H}_{4m:6}$ in Section 6.

- The second message of the execution of $\langle C, R\rangle_{\mathcal{A}\to H}$.

- The last message of the execution of $\langle C, R\rangle_{\mathcal{A}\to H}$.

- The second message of the execution of $\langle C', R'\rangle_{\mathcal{A}\to H}$.

- The last message of the execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}$ given in the Input Commitment Phase.

Observe that for any session these messages happen in the order they are listed above. We will refer to the special messages of the j^{th} session by the names – *first special message of session* j, *second special message of session* j, *third special message of session* j and *fourth special message of session* j. A more elaborate description of the special messages and their properties in provided in Section 6.1.

Level of simulation. In order to abstract out some of the details relation to simulation, we start by describing the levels of simulation (or, the extent of simulation) we will use.

- *Honest Execution*: Honest execution in a particular session corresponds to the simulator following the honest party strategy in that session. Note that for this the simulator will need access to honest party input for that session.

- *Partial Simulation*: Partial simulation in a particular session corresponds to the simulator cheating only partially in the execution of that session. In this setting the simulator cheats everywhere except in the semi-honest 2-PC execution which is a part of the Secure Computation Phase. More specifically, in this setting our simulator provides a commitment to 1 instead of a commitment to 0 in the Trapdoor Creation Phase, cheats in the NMZK proof, cheats in all the SWI proofs and provides a commitment to "junk" (instead of honest party input and randomness) in the Input Commitment Phase. Note that for this kind of simulation the simulator will *still* need access to the honest party input, in addition to knowledge of the preamble secret committed by the adversary in Trapdoor Creation phase for the session it is partially simulating. We stress that the preamble secret is only needed after the second special message for the session has been received. Note that the simulator, however, does *not* need access to the input and the randomness committed to by the adversary in the Input Commitment Phase.

- *Full Simulation*: Full simulation in a particular session corresponds to the simulator cheating completely in the execution of that session. In this setting the simulator cheats everywhere, i.e., everywhere it was cheating in the partial simulation and also in the semi-honest 2-PC execution which is a part of the Secure Computation Phase. Note that for this kind of simulation the simulator will not need access to honest party input but will need the preamble secret committed by the adversary in Trapdoor Creation phase and the input and the randomness of the adversary committed in the Input Commitment Phase for the session being simulated. Also note that the preamble secret for a session is only needed after the second special message for the session has been received. Similarly the adversary's input and randomness are only needed after the fourth special message for the session has been received.

$\texttt{Lookup}(\texttt{Database}^{\sigma}, \texttt{Database}^{x}, \tau, i)$ Function: During the simulator's interaction with the adversary, which we will refer to as the *main thread*, it will make multiple calls to the $\texttt{Lookup}$ function. We will also refer to these function calls as a *look-ahead* thread. We start by giving a succinct description of this function. Our lookup function takes as input the two databases ($\texttt{Database}^{\sigma}$ and $\texttt{Database}^{x}$), the state of the adversary/environment τ so far, and a session name i as input. Note that this function *does not* itself update the databases in any way; it only uses their contents. The function simulates the execution (in the manner as explained below) until it receives a well-formed second special message or fourth special message for session i. If this takes place, it returns the relevant information necessary for the extraction of either the preamble trapdoor (in the case of a second special message) or the adversary's input and randomness (in the case of a fourth special message). The lookup function performs the simulation as follows: For each session $j \in [m]$, where m is the number of sessions, it behaves as follows:

- Case 1: $(j, \cdot) \notin \texttt{Database}^{\sigma} \bigwedge (j, \cdot) \notin \texttt{Database}^{x}$: For session j our simulator uses the honest execution strategy. (In other words, it executes the look ahead thread honestly using the honest party inputs).

- Case 2: $(j, \cdot) \in \texttt{Database}^{\sigma} \bigwedge (j, \cdot) \notin \texttt{Database}^{x}$: For session j our simulator uses the partial simulation strategy. Note that partial simulation in the j^{th} session requires access to the adversary's preamble secret for the j^{th} session and our simulator can obtain it from the database $\texttt{Database}^{\sigma}$.[13]

- Case 3: $(j, \cdot) \in \texttt{Database}^{\sigma} \bigwedge (j, \cdot) \in \texttt{Database}^{x}$: For session j our simulator uses the full simulation strategy. Note that full simulation in the j^{th} session requires access to the adversary's preamble secret and the adversary's input for the j^{th} session and our simulator can obtain it from the databases $\texttt{Database}^{\sigma}$ and $\texttt{Database}^{x}$, respectively. If $(j, \perp) \in \texttt{Database}^{x}$ then we abort with $\texttt{I-Abort}_1$.[14]

We stress that the working of the look-ahead thread depends on the contents of the databases. Furthermore, different look-ahead threads will differ from each other depending on what databases they are initiated with. Also, note that the Lookup function never calls itself recursively.

The main-thread simulation in hybrid $H_{1/2}$. Our simulator will perform full simulation in all sessions on the main thread of execution. However, in order to do this, it will perform some extra steps when it receives one of the special messages for any session. These steps are performed with the goal of populating the database $\texttt{Database}^{\sigma}$ (resp., $\texttt{Database}^{x}$) with the preamble secret (resp., adversary's input) for every session whose second (resp., fourth) special message has been received. As pointed out earlier, for the full simulation of the main thread, it suffices to obtain the preamble secret (resp., adversary's input) for those sessions whose second (resp., fourth) special message has already been received. For each session i encountered, the simulation will do the following in addition to the full simulation described above:

- *First or third special message of session i:* It starts $\texttt{Lookup}(\texttt{Database}^{\sigma}, \texttt{Database}^{x}, \tau, i)$ function k times.[15] We stress that all these look-ahead threads are executed before returning to the execution in the main thread. The state of the databases at the time of these function calls is used as input for the Lookup function. The simulator stores the output of these lookup functions calls for later use.

- *Second or fourth special message of session i:* When the simulator receives the second (resp., fourth) special message of session i, then our simulator scans the output of the look ahead threads it had started when it had received the first (resp., third) special message of session i. In these look-ahead threads it checks to see if it had received a second (resp., fourth) special message of the i^{th} session in any of the executions of the Lookup function calls. Our simulator aborts with a Rewind Abort[16] if this is not the case. On the other hand if it did receive the second (resp., fourth) special message of the i^{th} session in at least one of the look-ahead threads

[13] We argue in Lemma 7 that if $(j, \perp) \in \texttt{Database}^{\sigma}$ then we will not need the preamble secret.

[14] In Lemma 8 we will argue that this happens only with a negligible probability.

[15] k is a parameter that decides the number of look-ahead threads that we will execute. This number depends specifically on the proof details and is described in Section 6.

[16] Our simulator will output Rewind Abort with a small, yet noticeable probability. This probability will depend on the parameter k that we will adjust to ensure that the abort probability is low enough.

then it extracts the preamble secret (resp., adversary's input) of the i^{th} session and adds an entry for it in the database $\texttt{Database}^{\sigma}$ (resp., $\texttt{Database}^{x}$). If the extraction fails even though it did receive the second (resp., fourth) special message of the i^{th} session in at least one of the look-ahead threads (this could happen if, e.g., no consistent preamble secret was committed to by the adversary but still it managed to complete the SWI argument) then it just adds the entry $(i, \perp)$ to the database $\texttt{Database}^{\sigma}$ (resp., $\texttt{Database}^{x}$) and continues[17].

We stress that from the above description it is clear that for any session i for which the second (resp., fourth) special message is received in the main thread (and given that Rewind Abort has not occurred) an entry of the form $(i, \cdot)$ is always made in the database $\texttt{Database}^{\sigma}$ (resp., $\texttt{Database}^{x}$). Hence, whenever the simulator will need the preamble secret (resp., adversary's input) in the main thread then it will indeed be available in the database $\texttt{Database}^{\sigma}$ (resp., $\texttt{Database}^{x}$). Furthermore, for each call to the $\texttt{Lookup}(\texttt{Database}^{\sigma}, \texttt{Database}^{x}, \tau, i)$ function, for all sessions whose second (resp., fourth) special message is already received in the main thread before reaching the state τ, we will use partial (resp., full) simulation. On the other hand for the rest of the sessions, the second special message has not been obtained and we will use the honest execution strategy.

From H_0 to $H_{1/2}$: We move from the experiment H_0 to experiment $H_{1/2}$ through a carefully designed series of hybrid experiments. Very roughly, there are four high level changes that we do for each session in going from H_0 to $H_{1/2}$: first is the creation of various look ahead threads to help extraction of preamble secret in the concurrent setting, second is moving from honest execution to partially simulated execution, third is the creation of various look ahead threads to help extraction of adversary's input and randomness in the concurrent setting, and fourth is in moving from partially simulated execution to fully simulated execution – ultimately avoiding the need of honest party input for the session. For each session we will do these changes one by one, in this order itself. However, the changes across different sessions will be interleaved based on the interleaving of special messages among different sessions. We sequentially consider the special messages *across all sessions* that occur on the main thread. Depending on the special message being considered we make the following changes:

- *First (resp., third) special messages of some session.* We create k look-ahead threads at the point where this message has been received. Observe that at this point the execution of the main thread is identical to the execution of the look-ahead threads just started.
- *Second special messages of session j.* Switch from honest execution to partially simulated execution for session j in the main thread.
- *Fourth special messages of session j.* Switch from partially simulated execution to fully simulated execution for session j in the main thread.

Observe that if these changes are made for each special message then we will finally end up fully simulating each session in the main thread. We provide more details in Section 6.

Last Step from $H_{1/2}$ to H_1. Note that our simulation never makes use of the inputs of the honest parties in the main thread. However, these inputs are used in the look-ahead threads. As pointed earlier, we are using rewindings only to populate the databases $\texttt{Database}^{\sigma}$ and $\texttt{Database}^{x}$. Therefore, we can take the last step from the hybrid $H_{1/2}$ to hybrid H_1 by switching from poly-time extraction (using rewindings) to super-poly time extraction. This avoids the need for all look-ahead threads and the

[17] We will argue later in Lemma 7 that if an entry $(i, \perp)$ is made to the database $\texttt{Database}^{\sigma}$ then the adversary will not be able to continue the i^{th} session to a point where the preamble secret is actually needed. In Lemma 8 we will argue that an entry $(i, \perp)$ is made to the database $\texttt{Database}^{x}$ with only negligible probability.

inputs of the honest parties in "one shot." The distribution of the view of the adversary/environment in the main thread still remains *statistically close* enough to the one in the previous hybrid, since the only difference is that `Rewind Abort`s do not occur in the final hybrid.

A detailed exposition of the proof of security follows.

6 Indistinguishability of the Views

We consider two experiments H_0 and H_1, where H_0 corresponds to the real world execution of $\langle P_1, P_2 \rangle$ while H_1 corresponds to the ideal world execution, as described below.

Experiment H_0: The simulator $\mathcal{S}$ simulates the honest parties H and in doing so it obtains their inputs and the specification on what sessions to take part in from the environment $\mathcal{Z}$. $\mathcal{S}$ does so by following the honest party algorithm, it generates the outputs of the honest party along with $\mathcal{A}$'s view. This corresponds to the real execution of the protocol. The output of the hybrid corresponds to the output of the environment, the outputs of the honest parties and the view of the adversary $\mathcal{A}$.

Experiment H_1: $\mathcal{S}$ simulates the honest parties (in the same manner as explained in the description of $\mathcal{S}$ in Section 5.1). The honest parties, for each session receive input from the environment, query the ideal functionality on their input and output the response they receive from the ideal functionality as their output. Again the output of the hybrid corresponds to the output of the environment, the outputs of the honest parties and the view of the adversary $\mathcal{A}$.

Let ν^i be a random variable that represents the output of H_i. We now claim that the output distributions of H_0 and H_1 are indistinguishable, as stated below:

Lemma 3 $\nu^0 \overset{c}{\equiv} \nu^1$

The proof of this lemma requires a careful hybrid argument that departs from previous work such as [BPS06, GJO10] in several crucial respects, most notably in our handling of look-ahead threads and our sequence of hybrids. More details are given below.

6.1 Getting started

We will prove Lemma 3 by contradiction. Suppose that the hybrids H_0 and H_1 are distinguishable in polynomial time, i.e., there exists a PPT distinguisher D that can distinguish between the two hybrids with a non-negligible probability (say) ϵ. More formally, $\left|Pr[D(\nu^0) = 1] - Pr[D(\nu^1) = 1]\right| > \epsilon$. Let $k = \frac{6 \cdot m}{\epsilon}$, where m is an upper bound on the number of sessions (such an upper bound is the running time of the adversarial environment). k is a parameter that corresponds to the number of rewindings (explained later) that we will perform.

We will now consider a series of hybrid experiments $\mathcal{H}_{i:j}$, where $i \in [1, 4m]$, and $j \in [1, 6]$. We define two additional hybrids – first, a dummy hybrid $\mathcal{H}_{0:6}$ that represents the real world execution (i.e., H_0, as defined above), and second, an additional hybrid $\mathcal{H}_{4m+1:1}$ that corresponds to the simulated execution in the ideal world (i.e., H_1, as defined above). For each intermediate hybrid $\mathcal{H}_{i:j}$, we define a random variable $\nu_{i:j}$ that represents the output (output of the environment, the outputs of the honest parties and the view of the adversary $\mathcal{A}$) of $\mathcal{H}_{i:j}$.

Below, we will establish (via the intermediate hybrid arguments) that no polynomial time distinguisher can distinguish between $\nu_{0:6}$ and $\nu_{4m+1:1}$ with a probability greater than ϵ, which is a contradiction. Before we jump into description of our hybrids, we first establish some notation and terminology.

In the sequel, we will make use of the notation described in Section 5. We now describe some additional notation that will be used in the proof. Again recall that we are in the UC setting and there could be multiple parties in the system and the simulator only needs to simulate the interactions between parties such that exactly one of them is corrupted. Without loss of generality, in the proof, for simplicity of notation we will assume that there is only one honest party (referred to as H) and only one malicious party (referred to as $\mathcal{A}$). Now our simulator must simulate the view of the malicious party in all the sessions it is a part of. Rather than referring to these sessions by session identification *sid* numbers we will use just a session number ℓ. Let m be a total bound on the total number of sessions. Then, for any session, we will have that $\ell \in [m]$. It should be easy to see that a simulator constructed in this simple setting with proper "book-keeping" will be able to achieve correct simulation in the general setting.

Special Messages Notation. Among all the m sessions, consider the m executions of $\langle C, R\rangle_{\mathcal{A}\to H}$, the m executions of $\langle C', R'\rangle_{\mathcal{A}\to H}$ and the m executions of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}$ given in the Input Commitment Phase. In these executions consider the following four messages, which we will refer to as *special messages* (denoted by SM):

1. The second message of $\langle C, R\rangle_{\mathcal{A}\to H}$ in the Trapdoor Creation Phase: Recall that at this point, the adversary will have committed to a value σ (in the first message), and the second message is the challenge sent by the honest party.

2. The last message of $\langle C, R\rangle_{\mathcal{A}\to H}$ in the Trapdoor Creation Phase: Recall that this is the response sent by the adversary to the challenge above. If two valid responses to different challenges can be obtained by the simulator, this allows the simulator to learn the value σ that the adversary committed to. Note that this message is only considered a special message if it is well-formed (that is, the receiver considers the openings of the commitments to be valid openings).

3. The second message of $\langle C', R'\rangle_{\mathcal{A}\to H}$ in the Input Commitment Phase: Recall that at this point, the adversary will have committed to its input and randomness x (in the first message), and the second message is the challenge sent by the honest party.

4. The last message of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}$ in the Input Commitment Phase: Recall that this is where the adversary proves that the response it sent to the challenge above was correct. If two valid responses to different challenges can be obtained by the simulator, this allows the simulator to learn the input and randomness value x that the adversary committed to. Note that this message is only considered a special message if it is well-formed (that is, the verifier accepts the proof).

Note that there are exactly 4 SM's for each one of the m sessions. Consider a numbered ordering of all the $4m$ occurrences of special messages across the m sessions (excluding any look-up threads, created in some hybrids). Let SM_i denote the i^{th} special message that appears in the interaction between the simulator and the adversary. Also, let $s(i)$ be the index of the protocol session that contains SM_i; that is, the message SM_i occurs during session number $s(i)$. Note that $s(i)$ will correspond to the same session for 4 distinct values of $i \in [4m]$, unless a session aborts and not all messages are scheduled, in which case fewer than 4 repetitions may occur. We will refer to the SM_i's by one of these 4 names – *first special message of session* $s(i)$, *second special message of session* $s(i)$, *third special message of session* $s(i)$ and *fourth special message of session* $s(i)$. These messages correspond to second message of $\langle C, R\rangle_{\mathcal{A}\to H}$ of session $s(i)$, last message of $\langle C, R\rangle_{\mathcal{A}\to H}$ of session $s(i)$, the second messages of $\langle C', R'\rangle_{\mathcal{A}\to H}$ of session $s(i)$ and the last messages of the $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}$ given in the Input Commitment Phase of session $s(i)$, respectively. Note that these four messages appear in this order[18] (but not necessarily consecutively) in the ordering of the $4m$ special messages.

[18] I.e., first messages comes before the second and so on.

Below when we use the terms $A \overset{c}{\equiv} B$, what we mean is that no PPT distinguisher can distinguish (except with negligible probability) between A and B. Similarly, when we use the terms $A \overset{s}{\equiv} B$, what we mean is that no unbounded distinguisher can distinguish (except with negligible probability) between A and B.

Soundness Condition. Looking ahead, while proving the indistinguishability of the outputs of our hybrid experiments, we will need to argue that in each session $\ell \in [m]$, the soundness property holds for $\langle P, V \rangle_{\mathcal{A} \to H}$ and that the trapdoor condition is false for each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}$. In the sequel, we will refer to this as the *soundness condition.* Now consider the NMZK instance $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ in session ℓ. Let $\pi^{\ell}_{\mathcal{A}}$ denote the proof statement for $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$, where, informally speaking, $\pi^{\ell}_{\mathcal{A}}$ states that $\mathcal{A}$ committed to bit 0 (earlier in the trapdoor creation phase). Note that the soundness condition "holds" if we prove that in each session $\ell \in [m]$, $\mathcal{A}$ commits to a valid witness to the statement y^{ℓ} in the non-malleable commitment (NMCOM) inside $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$. To this end, we define m random variables, $\{\rho^{\ell}_{i:j}\}^{m}_{\ell=1}$, where $\rho^{\ell}_{i:j}$ is the value committed in the NMCOM inside $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ as per $\nu_{i:j}$. Now, before we proceed to the description of our hybrids, we first claim that the soundness condition holds in the real execution. We will later argue that the soundness condition still holds as we move from one hybrid to another.

Lemma 4 *Let $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ and $\pi^{\ell}_{\mathcal{A}}$ be as described above corresponding to the real execution. Then, for each session $\ell \in [m]$, if the honest party does not abort the session (before the first message of the Secure Computation Phase is sent) in the view $\nu_{0:6}$, then $\rho^{\ell}_{0:6}$ is a valid witness to the statement $\pi^{\ell}_{\mathcal{A}}$, except with negligible probability.*

Intuitively, the above lemma follows due the knowledge soundness of the statistical zero knowledge argument of knowledge used in NMZK. We refer the reader to [Claim 2.5, [BPS06]] for a detailed proof.

Lemma 5 *Let $\langle P, V \rangle^{\ell}_{\mathcal{A} \to H}$ and $\pi^{\ell}_{\mathcal{A}}$ be as described above corresponding to the execution in any hybrid $\mathcal{H}_{i:j}$. Then $\forall i \in [m], j \in [6]$, for each session $\ell \in [m]$, if the honest party does not abort the session (before the first message of the Secure Computation Phase is sent) in the view $\nu_{i:j}$, then $\rho^{\ell}_{i:j}$ is a valid witness to the statement $\pi^{\ell}_{\mathcal{A}}$, except with negligible probability.*

We defer the proof of this lemma until later (see Section 6.2.1), as it will make use of other claims that we will prove about our hybrids.

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

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

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

6.2 Description of the Hybrids

For $i \in [1, 4m]$, the hybrid experiments are described as follows. Very roughly, for each i we will consider the corresponding i^{th} special message SM_i that appears and depending on this message we will decide what hybrids to proceed with. We stress that unlike many previous works, our hybrids will not proceed "session by session", but rather we will revisit the same session several times over the course of the sequence of hybrid experiments. We also stress that in the following hybrid experiments, we will initiate several look-ahead threads, *however once a look-ahead thread is started in a particular hybrid, the operation of that look-ahead thread is never modified in future hybrids*[19]. Future hybrids only modify how the simulator deals with the main thread of the simulation.

Further we will maintain two databases: $\mathtt{Database}^{\sigma}$ and $\mathtt{Database}^{x}$. In database $\mathtt{Database}^{\sigma}$ we will store tuples of the form $(j, \sigma^j_{\mathcal{A}})$, where $\sigma^j_{\mathcal{A}}$ is the preamble secret committed by the adversary in session j. Similarly, in database $\mathtt{Database}^{x}$ we will store tuples of the form $(j, x^j_{\mathcal{A}})$, where $x^j_{\mathcal{A}}$ is the input and the randomness committed by the adversary in session j.

Experiment $\mathcal{H}_{i:1}$: Same as hybrid $\mathcal{H}_{i-1,6}$, unless if SM_i is the first (or resp., third) special message of session $s(i)$, in which case it differs in the following manner: Hybrid $\mathcal{H}_{i:1}$ is same as hybrid $\mathcal{H}_{i-1:6}$, except that the simulator $\mathcal{S}$ starts k *look-ahead* threads at the point SM_i with freshly chosen challenge messages[20] in each of the look-ahead threads.[21] Note that the simulator strategy in each of the look-ahead threads is the same as the simulator strategy in the main thread[22] from hybrid $\mathcal{H}_{i-1:6}$, using the current contents of databases $\mathtt{Database}^{\sigma}$ and $\mathtt{Database}^{x}$. As pointed earlier, in future hybrids we will make changes to the main thread but look-ahead thread will never be modified. Further observe that in hybrid $\mathcal{H}_{i-1:6}$, no look-ahead threads are initiated after SM_i has been sent in the main thread and hence the simulation of the look-ahead threads will not involve any extra rewindings. If $\mathcal{A}$ does not send the second (resp., fourth) special message of session $s(i)$ in any of the look-ahead threads but it does so in the main thread then our simulator aborts. We specifically refer to this abort as `Rewind Abort`.

Note that simulator can use the second (resp, fourth) special messages sent in the main-thread and in one of the look-ahead threads to extract the preamble secret (resp., input) committed in the execution $\langle C, R\rangle^{s(i)}_{\mathcal{A}\to H}$ (resp., $\langle C', R'\rangle^{s(i)}_{\mathcal{A}\to H}$). When this happens then our simulator extracts $\sigma^{s(i)}_{\mathcal{A}}$ (resp., $x^{s(i)}_{\mathcal{A}}$) and adds the entry $(s(i), \sigma^{s(i)}_{\mathcal{A}})$ (resp., $(s(i), x^{s(i)}_{\mathcal{A}})$) to the database $\mathtt{Database}^{\sigma}$ (resp., $\mathtt{Database}^{x}$). However, it might not be possible to extract the corresponding committed value (even though the simulator does not output `Rewind Abort`). In this case the simulator proceeds without the extraction adding a special entry $(s(i), \perp)$ to the $\mathtt{Database}^{\sigma}$ (resp., $\mathtt{Database}^{x}$).

[19] This is true except for the last hybrid, in which all look-ahead threads are eliminated with super-polynomial simulation.

[20] There is an negligibly small probability that the freshly chosen challenge message in a look-ahead thread is chosen to be the same as in the main thread. Because this probability is negligible, we will proceed in the proof under the assumption that this does not occur.

[21] The simulator completes all the look-ahead before it returns to continue the main thread.

[22] It is instructive to note that this strategy is same as the one described in the $\mathtt{Lookup}(\mathtt{Database}^{\sigma}, \mathtt{Database}^{x}, \tau, i)$ function described in Section 5.2. In this hybrid the main thread and all the look ahead threads follow this strategy.

We have the following claims:

$$\forall \text{ PPT } D \left|Pr[D(\nu_{i-1:6}) = 1] \quad - \quad Pr[D(\nu_{i:1}) = 1]\right| \leq \frac{1}{k} + \mathsf{negl}(n) \tag{1}$$

$$\forall \text{ PPT } D \left|Pr[D(\rho^1_{i-1:6}, \ldots, \rho^m_{i-1:6}) = 1] \quad - \quad Pr[D(\rho^1_{i:1}, \ldots, \rho^m_{i:1})) = 1]\right| \leq \frac{1}{k} + \mathsf{negl}(n) \tag{2}$$

Proving Equations 1 *and* 2. Note that the only difference between $\mathcal{H}_{i-1:6}$ and $\mathcal{H}_{i:1}$ is that $\mathcal{S}$ outputs a `Rewind Abort` in case the adversary sends second (resp., fourth) special message of session $s(i)$ in the main thread but does not do so in any one of the k look-ahead threads. Since the main-thread and the look-ahead threads are identical, by simple swapping [PRS02] argument, the probability that the simulator outputs a `Rewind Abort` is bounded by $1/k$. Further assuming that the simulator does not output `Rewind Abort` the hybrids $\mathcal{H}_{i-1:6}$ and $\mathcal{H}_{i:1}$ are close. Note that in this hybrid, the values extracted using the new look-ahead threads started in this hybrid are not utilized in any way. Hence our claim follows.

Looking ahead: Intuition for how look-ahead threads are handled without any recursive rewinding. As argued in Lemma 6 below, for every $j < i$, if SM_j is a second (resp., fourth) special message of session $s(j)$ then there exists an entry $(s(j), \cdot)$ in $\mathtt{Database}^\sigma$ (resp., $\mathtt{Database}^x$). Hence, for all these sessions for which an entry exists in the databases we can in fact use the extracted value and cheat (and we will do so in future hybrids). However, this entry could actually be $(s(j), \perp)$. In Lemma 7 below we argue that if the tuple $(s(j), \perp)$ is present in $\mathtt{Database}^\sigma$, then the adversary will not be able to continue the j^{th} session, and so there will be no need to simulate it. On the other hand we note that if $(s(j), \perp)$ is present in $\mathtt{Database}^x$ and if our simulator needs to use this value at some point then it aborts with the special abort message $\mathtt{I\text{-}Abort}_1$. This event happens with a negligible probability (argued in Lemma 8 below), and hence will not affect our analysis.

On the other hand, in sessions for which the second (resp., fourth) special message has not been received before SM_i, in the look-ahead threads we will not have an entry in $\mathtt{Database}^\sigma$ (resp., $\mathtt{Database}^x$) and we will behave "honestly," for such sessions.

Now we state and give the proof for Lemma 6.

Lemma 6 *For every $j < i$ if SM_j is the second (resp., fourth) special message of session $s(j)$ and `Rewind Abort` did not happen, then there exists an entry $(s(j), \cdot)$ in $\mathtt{Database}^\sigma$ (resp., $\mathtt{Database}^x$) at the time that SM_i occurs in the main thread.*

Proof. This follows immediately from our simulation strategy. Note that for every $j < i$, if SM_j is a first (resp., third) special message of session $s(j)$ then we have already started its look ahead threads, and that means that we will be able to extract the preamble secret (resp., adversary's input) as soon as the second (resp., fourth) special message of session $s(j)$ is received in the main thread, and then we always make an entry $(s(j), \cdot)$ in $\mathtt{Database}^\sigma$ (resp., $\mathtt{Database}^x$) unless `Rewind Abort` happens. ■

Invariant Lemmas. Now we establish two lemmas pertaining to the setting in which extraction fails even though `Rewind Abort` did not happen. We note these lemmas hold across all hybrids.

Lemma 7 *If $(j, \perp) \in \mathtt{Database}^\sigma$, then the adversary will not be able to continue the j^{th} session past the step where it opens all the commitments made in $\langle C, R\rangle^j_{\mathcal{A}\to H}$ (within Step 3 of Trapdoor Creation Phase).*

Proof. Since the entry $(j, \perp)$ was made to $\mathtt{Database}^\sigma$, this means that the extraction of $\sigma^j_{\mathcal{A}}$ had failed even though the second special message of session j was reached in the main thread as well as one of the look ahead threads (as `Rewind Abort` did not happen). Also note that the commitments used in

$\langle C, R\rangle^{j}_{\mathcal{A}\to H}$ are perfectly binding. From these two facts it follows that the commitment $\langle C, R\rangle^{j}_{\mathcal{A}\to H}$ is in fact *invalid*. Since the adversary has to open the commitment within Step 3 of Trapdoor Creation Phase, the adversary can not do so legitimately. This completes the proof. ■

Lemma 8 *Let E_j be the event that the simulator receives the fourth special message of session j in the main thread and in at least one of the look ahead threads, but the commitment $\langle C', R'\rangle^{j}_{\mathcal{A}\to H}$ completed in either the main thread or in considered the look-ahead thread is invalid. More formally, E_j is the event*[23] $\big((j,\cdot) \in \texttt{Database}^{x}\big) \bigwedge \big(\langle C', R'\rangle^{j}_{\mathcal{A}\to H}$ *is* invalid *in main thread* $\vee \langle C', R'\rangle^{j}_{\mathcal{A}\to H}$ *is* invalid *in look-ahead thread* $\big)$*, where we consider the specific look-ahead thread that lead to the creation of the entry $(j,\cdot)$. Then E_j happens with negligible probability.*

Proof. We prove this by contradiction. Lets say that actually for some session j, event E_j happens with non-negligible probability. We will then contradict the soundness of the SWI proof system.

We first note that by *soundness condition* (Lemma 5) we have that the adversary always sends a commitment to the bit 1 in the Trapdoor Creation phase for the j^{th} session. From this it follows that there is only one witness for the SWI protocol in the Input Commitment Phase. Therefore in order for an adversary to make an *invalid* commitment or for extraction to fail (in $\langle C', R'\rangle^{j}_{\mathcal{A}\to H}$) it has to cheat in at least one of the SWI proofs among the ones provided in the main thread and the look-ahead thread (in which fourth special message of $s(i)$ is reached). Further note that existence of $(j,\cdot) \in \texttt{Database}^{x}$ implies that the fourth special message of the j^{th} session was indeed reached in the main thread and at least one of the look-ahead threads. Finally, it follows from the soundness of the SWI protocol that the event E_j happens with a negligible probability. ■

Change from hybrid $\mathcal{H}_{i:1}$ to hybrid $\mathcal{H}_{i:6}$. Hybrid $\mathcal{H}_{i:6}$ is same as hybrid $\mathcal{H}_{i:1}$ unless SM_i is the second (resp., fourth) special message of session $s(i)$. If this is the case then we change how we simulate session $s(i)$ in the main thread. More specifically, we start partially simulating (resp., fully simulating)[24] session $s(i)$ in the main thread in hybrid $\mathcal{H}_{i:6}$. Further note that since second (resp., fourth) special message of session $s(i)$ has already been received in the main thread and `Rewind Abort` did not happen, therefore there exists an entry $(s(i),\cdot)$ in $\texttt{Database}^{\sigma}$ (resp., $\texttt{Database}^{x}$). However this entry could actually be $(s(i),\perp)$. We consider this specific case now. If SM_i is the second special message of session $s(i)$ then (as argued in Lemma 7) we will not need this value. On the other hand, if SM_i is the fourth special message of session $s(i)$ then we will abort with the abort message $\texttt{I-Abort}_1$. By Lemma 8 this happens only with a negligible probability.

Experiment $\mathcal{H}_{i:2}$: Same as $\mathcal{H}_{i:1}$, except if SM_i is the second special message of session $s(i)$, then $\mathcal{S}$ simulates the post-preamble phase of $\langle P, V\rangle^{s(i)}_{H\to\mathcal{A}}$ in a *straight-line* manner, as explained in Section 3.3, by making use of the entry in $\texttt{Database}^{\sigma}$ for the session $s(i)$. Recall that no look-ahead threads are started once the execution reaches SM_i on the main thread. All the changes in the main thread, as explained below, are performed *after* SM_i is reached.

We now claim that,

$$\nu_{i:1} \overset{c}{\equiv} \nu_{i:2} \tag{3}$$

$$\forall \ell \in [m] \quad \rho^{\ell}_{i:1} \overset{c}{\equiv} \rho^{\ell}_{i:2} \tag{4}$$

The proof of the claim is in Section 7.

[23] Note that our simulator outputs $\texttt{I-Abort}_1$ if $(j,\perp) \in \texttt{Database}^{x}$. Note that if entry $(j,\perp) \in \texttt{Database}^{x}$ is made, then E_j must have happened. And this implies that our simulator outputs $\texttt{I-Abort}_1$ with negligible probability.

[24] Recall that by partially simulating (resp., fully simulating) a session $s(i)$ we mean "cheating in the commitments, NMZK and SWI" (resp., "cheating in cheating in the commitments, NMZK, SWI as well as the semi-honest simulation") for session $s(i)$. Recall that this intuitive terminology was introduced in Section 5.2 so as to abstract out the details of the hybrids from $\mathcal{H}_{i:1}$ to $\mathcal{H}_{i:6}$.

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

We now claim that,

$$\begin{aligned} \nu_{i:2} &\overset{c}{\equiv} \nu_{i:3} && (5)\\ \forall \ell \quad \rho^{\ell}_{i:2} &\overset{c}{\equiv} \rho^{\ell}_{i:3} && (6) \end{aligned}$$

The proof of the claim is in Section 7.

Experiment $\mathcal{H}_{i:4}$: Same as $\mathcal{H}_{i:3}$, except if SM_i is the second special message of session $s(i)$, then $\mathcal{S}$ uses the trapdoor witness (instead of the real witness) in each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{H\to\mathcal{A}}$ in session $s(i)$. Note that the trapdoor witness for each of these SWI must be available to the simulator at this point since it earlier committed to bit 1 in phase I of session $s(i)$.

We now claim that,

$$\begin{aligned} \nu_{i:3} &\overset{s}{\equiv} \nu_{i:4} && (7)\\ \forall \ell \quad \rho^{\ell}_{i:3} &\overset{c}{\equiv} \rho^{\ell}_{i:4} && (8) \end{aligned}$$

The proof of the claim is in Section 7.

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

1. Instead of sending honest commitments to $x_H \| r_H$ and its secret shares, $\mathcal{S}$ sends commitments to random strings as the first message.
2. On receiving a challenge string from $\mathcal{A}$, instead of honestly revealing the committed shares (as per the challenge string), $\mathcal{S}$ sends random strings to $\mathcal{A}$.

We now claim that,

$$\begin{aligned} \nu_{i:4} &\overset{c}{\equiv} \nu_{i:5} && (9)\\ \forall \ell \in [m] \quad \rho^{\ell}_{i:4} &\overset{c}{\equiv} \rho^{\ell}_{i:5} && (10) \end{aligned}$$

The proof of the claim is in Section 7.

Experiment $\mathcal{H}_{i:6}$: Same as $\mathcal{H}_{i:5}$, except if SM_i is the fourth special message of session $s(i)$, then $\mathcal{S}$ "simulates" the execution of $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ in session $s(i)$, in the following manner. Let S_{sh} be the simulator for the semi-honest two party protocol $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ used in our construction. $\mathcal{S}$ internally runs the simulator S_{sh} for the semi-honest two party protocol $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$ on $\mathcal{A}$'s input in session $s(i)$ that was extracted earlier and is found in $\mathtt{Database}^x$. When S_{sh} makes a query to the trusted party with some input, $\mathcal{S}$ responds to this query by using its input for session $s(i)$. The response from the trusted party is passed on to S_{sh}. S_{sh} finally halts and outputs a transcript of the execution of $\langle P^{\mathsf{sh}}_1, P^{\mathsf{sh}}_2\rangle$, and an associated random string for the adversary.

Now, $\mathcal{S}$ forces this transcript and randomness on $\mathcal{A}$ and if at any point $\mathcal{A}$ responds differently (than the expected response) but succeeds in making the simulator accept in the SWI proof provided immediately after the message is sent, the simulator aborts all communication and outputs $\texttt{I-Abort}_2$. We claim that during the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, each reply of $\mathcal{A}$ must be consistent with this transcript, except with negligible probability. Note that we have already established from the previous hybrids that the *soundness condition* holds (except with negligible probability) at this point[25]. This means that the trapdoor condition is false for each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}^{s(i)}$.

We now claim that,

$$\nu_{i:5} \overset{c}{\equiv} \nu_{i:6} \tag{11}$$

$$\forall \ell \in [m] \quad \rho_{i:5}^{\ell} \overset{c}{\equiv} \rho_{i:6}^{\ell} \tag{12}$$

The proof of the claim is in Section 7.

6.2.1 Proof of Lemma 5

Proof of Lemma 5. This lemma can be argued by considering the increment in the probability with which the adversary can violate the soundness condition as we go across hybrids. Lemma 4 shows that the probability is negligible in the first hybrid. We will show this increment to be negligible for each pair of consecutive hybrids we consider. Note that the switch from hybrid $\mathcal{H}_{i-1:6}$ to hybrid $\mathcal{H}_{i:1}$ is a statistical change and the the entire statistical distance between the two is because in hybrid $\mathcal{H}_{i:1}$ the simulator outputs a `Rewind Abort` for either the first or third special message in session $s(i)$ with a probability equal (up to negligible additive terms) to the statistical difference between the two hybrids, whereas the probability of this specific type of `Rewind Abort` was zero in hybrid $\mathcal{H}_{i-1:6}$. Because there is clearly no soundness condition violation when a hybrid outputs `Rewind Abort`, and the entire statistical distance between hybrids (up to additive negligible terms) is due to this increased probability of a `Rewind Abort`, we immediately have that the probability of a soundness condition violation can only increase by a negligible amount from hybrids $\mathcal{H}_{i-1:6}$ to hybrid $\mathcal{H}_{i:1}$. The argument for the rest of the hybrids follows based on the indistinguishability of the random variables $\rho_{i:j}^1, \ldots, \rho_{i:j}^m$ and $\rho_{i:j+1}^1, \ldots, \rho_{i:j+1}^m$ for every $j \in \{1, \ldots, 5\}$ based on Equations 4, 6, 8, 10 and 12.

6.2.2 Finishing the proof of security

So, far: Based on the hybrids, combining the distinguishing advantage of a distinguisher in the hybrids so far we have that:

$$\forall \text{ PPT } D \ |Pr[D(\nu_{4m:6}) = 1] \quad - \quad Pr[D(\nu_{0:6}) = 1]| \leq \frac{2m}{k} + \mathsf{negl}(n) \tag{13}$$

$$\forall \text{ PPT } D \ |Pr[D(\rho_{4m:6}^1, \ldots, \rho_{4m:6}^m) = 1] \quad - \quad Pr[D(\rho_{0:6}^1, \ldots, \rho_{0:6}^m) = 1]| \leq \frac{2m}{k} + \mathsf{negl}(n) \tag{14}$$

Further note that in hybrid $\nu_{0:6}$ simulator never outputs an `Rewind Abort`. Therefore the probability of the simulator outputting `Rewind Abort` in hybrid $\nu_{4m:6}$ is bounded by $\frac{2m}{k} + \mathsf{negl}(n)$.

Experiment $\mathcal{H}_{4m+1:1}$: In this hybrid we get rid of all the re-windings and instead uses super polynomial simulation (as described in description of the simulator) to obtain all the adversarial preamble secrets and adversarial inputs. Further, instead of generating the output provided to S_{sh} on its own, our simulator obtains the output by querying the trusted party. Note that this hybrid is same as the simulated execution in the ideal world.

[25] This is argued in the proof of Lemma 5. Note that there is no circularity here, since the proof of Lemma 5 is given by showing that for each successive hybrid, the probability of a soundness condition violation can only increase by a negligible amount.

Note that in hybrid $\mathcal{H}_{4m:6}$ the simulator outputs `Rewind Abort` (bounded by $\frac{2m}{k} + \mathsf{negl}(n)$) but our simulator (in hybrid $\mathcal{H}_{4m+1:1}$) never outputs `Rewind Abort` in this hybrid. Further, we consider the event E as $E_1 \vee E_2 \vee \cdots E_m$. Recall that event E_j is the event that the commitment provided by adversary in $\langle C', R' \rangle^j_{\mathcal{A} \to H}$ for session $j \in [m]$ is not *valid* or that $(j, \perp)$ entry was made to $\texttt{Database}^\sigma$ given that an entry was made (i.e., `Rewind Abort` did not happen). Further, using Lemma 8 for every $j \in [m]$ $Pr[E_j]$ is negligible. By union bound we have that $Pr[E]$ is also negligible.

We start our argument by conditioning on the fact that the simulator does not outputs `Rewind Abort` and that E does not happen. Next we observe that the commitments that the simulator "breaks" by running in super-poly time are *valid* (and simulator was also able to extract it) and therefore the value extracted via "breaking" must be same as the value simulator obtained by "rewinding." Therefore conditioned on the fact that simulator does not output `Rewind Abort` and E does not happen the two hybrids – $\mathcal{H}_{4m:6}$ and $\mathcal{H}_{4m+1:1}$ are identical.

On the other hand note that the probability that the simulator outputs `Rewind Abort` or E happens in hybrid $\nu_{4m:6}$ is bounded by $\frac{2m}{k} + \mathsf{negl}(n)$. Therefore, we have that:

$$\forall \text{ PPT } D \ |Pr[D(\nu_{4m:6}) = 1] \quad - \quad Pr[D(\nu_{4m+1:1}) = 1]| \leq \frac{2m}{k} + \mathsf{negl}(n) \tag{15}$$

$$\forall \text{ PPT } D \ |Pr[D(\rho^1_{4m:6}, \ldots, \rho^m_{4m:6}) = 1] \quad - \quad Pr[D(\rho^1_{4m+1:1}, \ldots, \rho^m_{4m+1:1}) = 1]| \leq \frac{2m}{k} + \mathsf{negl}(n) \tag{16}$$

Finally, since $k = \frac{6m}{\epsilon}$, using Equations 13, 14, 15 and 16, we have that:

$$\forall \text{ PPT } D \ |Pr[D(\nu_{0:6}) = 1] \quad - \quad Pr[D(\nu_{4m+1:1}) = 1]| \leq \frac{2\epsilon}{3} + \mathsf{negl}(n) \tag{17}$$

$$\forall \text{ PPT } D \ |Pr[D(\rho^1_{0:6}, \ldots, \rho^m_{0:6}) = 1] \quad - \quad Pr[D(\rho^1_{4m+1:1}, \ldots, \rho^m_{4m+1:1}) = 1]| \leq \frac{2\epsilon}{3} + \mathsf{negl}(n) \tag{18}$$

Which contradicts our original assumption that there exists a distinguisher D that distinguishes between $\mathcal{H}_{0:6}$ and $\mathcal{H}_{4m+1:1}$ (same as H_0 and H_1) with a probability greater than, a non-negligible probability, ϵ.

7 Hybrid Indistinguishability Details

The following text is adapted from the proofs in [GJO10], which in turn was based in part on [BPS06].

7.1 Proof of Equation 3 and 4

Let $\pi^{s(i)}_H$ denote the proof statement in $\langle P, V \rangle^{s(i)}_{H \to \mathcal{A}}$. Let $\sigma^{s(i)}_{\mathcal{A}}$ denote the trapdoor value committed by the $\mathcal{A}$ in the preamble phase of $\langle P, V \rangle^{s(i)}_{H \to \mathcal{A}}$ that $\mathcal{S}$ has already extracted. Then, recall that $\mathcal{S}$ performs the following steps to simulate the post-preamble phase of $\langle P, V \rangle^{s(i)}_{H \to \mathcal{A}}$ in $\mathcal{H}_{i:2}$:

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

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

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

Now, to prove equations 3 and 4, we will create three intermediate hybrids $\mathcal{H}_{i:1:1}$, $\mathcal{H}_{i:1:2}$, and $\mathcal{H}_{i:1:3}$. Hybrid $\mathcal{H}_{i:1:1}$ is identical to $\mathcal{H}_{i:1}$, except that it changes its strategy to perform step 1 (as described above). Hybrid $\mathcal{H}_{i:1:2}$ is identical to $\mathcal{H}_{i:1:1}$, except that it changes its strategy to perform step 3. Finally, hybrid $\mathcal{H}_{i:1:3}$ is identical to $\mathcal{H}_{i:1:2}$, except that it changes its strategy to perform step 2. Note that $\mathcal{H}_{i:1:3}$ is identical to $\mathcal{H}_{i:2}$.

We now claim the following:

$$\nu_{i:1} \overset{s}{\equiv} \nu_{i:1:1} \tag{19}$$

$$\forall \ell \in [m] \quad \rho_{i:1}^{\ell} \overset{c}{\equiv} \rho_{i:1:1}^{\ell} \tag{20}$$

$$\nu_{i:1:1} \overset{s}{\equiv} \nu_{i:1:2} \tag{21}$$

$$\forall \ell \in [m] \quad \rho_{i:1:1}^{\ell} \overset{c}{\equiv} \rho_{i:1:2}^{\ell} \tag{22}$$

$$\nu_{i:1:2} \overset{c}{\equiv} \nu_{i:1:3} \tag{23}$$

$$\forall \ell \in [m] \quad \rho_{i:1:2}^{\ell} \overset{c}{\equiv} \rho_{i:1:3}^{\ell} \tag{24}$$

Note that equation 3 follows by combining the results of equations 19, 21, and 23. Similarly, equation 4 follows by combining the results of equations 20, 22, and 24. We now prove the above set of equations.

Proving Equations 19 and 20. We first note that SCOM and SZKAOK can together be viewed as a statistically hiding commitment scheme. Let $\overline{\text{SCOM}}$ denote this new commitment scheme. Then, equation 19 simply follows from the (statistical) hiding property of $\overline{\text{SCOM}}$.

In order to prove equation 20, let us first assume that the claim is false, i.e., $\exists \ell \in [m]$ such that $\rho_{i:1}^{\ell}$ and $\rho_{i:1:1}^{\ell}$ are distinguishable by a PPT distinguisher D. We will create a standalone machine M^* that is identical to $\mathcal{H}_{i:1}$, except that instead of simply committing to a string of all zeros using $\overline{\text{SCOM}}$, M^* takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$. Additionally, M^* "exposes" the NMCOM in $\langle P, V\rangle_{\mathcal{A}\to H}^{\ell}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Let us describe the interaction between M^* and C in more detail. M^* first sends the trapdoor secret $\sigma_{\mathcal{A}}^{s(i)}$ to C. Now, when C starts the execution of $\overline{\text{SCOM}}$ in $\langle P, V\rangle_{H\to\mathcal{A}}^{s(i)}$, M^* forwards the messages from C to $\mathcal{A}$; the responses from $\mathcal{A}$ are forwarded externally to C. Note that if C commits to a string of all zeros in the $\overline{\text{SCOM}}$ execution, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:1}$. On the other hand, if C commits to the preamble secret $\sigma_{\mathcal{A}}^{s(i)}$, then the (C, M^*, R) system is equivalent to $\mathcal{H}_{i:1:1}$. We will now construct a computationally unbounded distinguisher D' that distinguishes between these two executions, thus contradicting the statistically hiding property of $\overline{\text{SCOM}}$. D' simply extracts the value inside the NMCOM received by R and runs D on this input. D' outputs whatever D outputs. By our assumption, D's output must be different in these two experiments; this implies that D' output is different as well, which is a contradiction.

Proving Equations 21 and 22. Equation 21 simply follows due to the statistical witness indistinguishability property of SZKAOK. Equation 22 also follows from the same fact; the proof details are almost identical to the proof of equation 20 and therefore omitted.

Proving Equations 23 and 24. Equation 23 simply follows from the hiding property of NMCOM. To see this, we can construct a standalone machine M that internally runs $\mathcal{S}$ and $\mathcal{A}$ and outputs the view generated by $\mathcal{S}$. M is identical to $\mathcal{H}_{i:1:2}$ except that in the post-preamble phase of $\langle P,V\rangle^{s(i)}_{H\to\mathcal{A}}$, instead of simply committing (using NMCOM) to a valid witness (to the proof statement $\pi^{s(i)}_{H}$), it takes this commitment from an external sender C (who is given the valid witness) and "forwards" it internally to $\mathcal{A}$. If the external sender C honestly commits to the witness, then the (C,M) system is identical to $\mathcal{H}_{i:1:2}$; otherwise if C commits to an all zeros string, then the above system is identical to $\mathcal{H}_{i:1:3}$. Equation 23 therefore follows from the hiding property of NMCOM.

In order to prove equation 24, we will use the non-malleability property of NMCOM. Let us assume that equation 24 is false, i.e., $\exists \ell \in [m]$ such that $\rho^{\ell}_{i:1:2}$ and $\rho^{\ell}_{i:1:3}$ are distinguishable by a PPT machine. We will construct a standalone machine M^* that is identical to the machine M described above, except that it will "expose" the non-malleable commitment inside $\langle P,V\rangle^{\ell}_{\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Now, if C commits to the witness to $\pi^{s(i)}_{H}$, then the (C,M^*,R) system is identical to $\mathcal{H}_{i:1:2}$, whereas if C commits to a random string, then the (C,M^*,R) system is identical to $\mathcal{H}_{i:1:3}$. From the non-malleability property of NMCOM, we establish that the value committed by M^* to R must be computationally indistinguishable in both cases.

7.2 Proof of Equation 5 and 6

Equation 5 simply follows from the (computationally) hiding property of the commitment scheme COM.

In order to prove equation 6, we will leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in NMZK. Let us first assume that equation 6 is false, i.e., $\exists \ell \in [m]$ such that $\rho^{\ell}_{i:2}$ and $\rho^{\ell}_{i:3}$ are distinguishable by a PPT distinguisher. Note that it cannot be the case that the NMCOM inside $\langle P,V\rangle^{\ell}_{\mathcal{A}\to H}$ concludes *before* $\mathcal{S}$ sends the non-interactive commitment $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$ in session $s(i)$, since in this case, the execution of NMCOM is independent of $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$. Now consider the case when the NMCOM inside $\langle P,V\rangle^{\ell}_{\mathcal{A}\to H}$ concludes *after* $\mathcal{S}$ sends $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$.

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

7.3 Proof of Equation 7 and 8

Equation 7 simply follows from the statistical witness indistinguishability of SWI by a standard hybrid argument.

In order to prove equation 8, let us first consider the simpler case where $\mathcal{S}$ uses the trapdoor witness only in the *first* instance (in the order of execution) of SWI in session $s(i)$ where the honest party plays the role of the prover. In this case, we can leverage the "statistical" nature of the witness indistinguishability property of SWI in a similar manner as in the proof of equation 20. Then, by a standard hybrid argument, we can extend this proof for multiple SWI.

7.4 Proof of Equation 9 and 10

Proving Equations 9 and 10. In order to prove these equations, we will define two intermediate hybrids $\mathcal{H}_{i:4:1}$ and $\mathcal{H}_{i:4:2}$. Experiment $\mathcal{H}_{i:4:1}$ is the same as $\mathcal{H}_{i:4}$, except that $\mathcal{S}$ also performs steps 1 as described above. Experiment $\mathcal{H}_{i:4:2}$ is the same as $\mathcal{H}_{i:4:1}$, except that $\mathcal{S}$ also performs step 2 as described above. Therefore, by definition, $\mathcal{H}_{i:4:2}$ is identical to $\mathcal{H}_{i:5}$.

We now claim the following:

$$\nu_{i:4} \overset{c}{\equiv} \nu_{i:4:1} \tag{25}$$

$$\forall \ell \in [m] \quad \rho^{\ell}_{i:4} \overset{c}{\equiv} \rho^{\ell}_{i:4:1} \tag{26}$$

$$\nu_{i:4:1} \overset{c}{\equiv} \nu_{i:4:2} \tag{27}$$

$$\forall \ell \in [m] \quad \rho^{\ell}_{i:4:1} \overset{c}{\equiv} \rho^{\ell}_{i:4:2} \tag{28}$$

Note that equation 9 follows by combining the results of equations 25 and 27. Similarly, equation 10 follows by combining the results of equations 26 and 28. We now prove the above set of equations.

Proving Equations 25 and 26. Equation 25 simply follows from the (computational) hiding property of the commitment scheme COM.

In order to prove equation 26, let us first consider the simpler case where $\mathcal{S}$ only modifies the first commitment in in $\langle C', R' \rangle_{H \to \mathcal{A}}$. In this case, we can leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in NMZK. The proof details are the same as the proof of equation 6 (described below) and are therefore omitted. Then, by a standard hybrid argument, we can extend this proof to the case where $\mathcal{S}$ modifies all the commitments in $\langle C', R' \rangle_{H \to \mathcal{A}}$.

Proving Equations 27 and 28. Note that the main-thread is identical in hybrids $\mathcal{H}_{i:4:1}$ and $\mathcal{H}_{i:4:2}$ since we are only changing some random strings to other random strings; furthermore, the strings being changed are not used elsewhere in the protocol. Equations 27 and 28 follow as a consequence.

7.5 Proof of Equation 11 and 12

Informally speaking, equation 11 follows from the semi-honest security of the two-party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ used in our construction. We now give more details.

We will construct a standalone machine M that is identical to $\mathcal{H}_{i:5}$, except that instead of engaging in an honest execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ with $\mathcal{A}$ in session $s(i)$, it obtains a protocol transcript from an external sender C and forces it on $\mathcal{A}$ in the following manner. M first queries the ideal world trusted party on the extracted input of $\mathcal{A}$ for session $s(i)$ in the same manner as explained above for $\mathcal{S}$. Let $x_{\mathcal{A}}^{s(i)}$ denote the extracted input of $\mathcal{A}$. Let $x_H^{s(i)}$ denote the input of the honest party in session $s(i)$. Let O be the output that M receives from the trusted party. Now M sends $x_H^{s(i)}$ along with $x_{\mathcal{A}}^{s(i)}$ and O to C and receives from C a transcript for $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ and an associated random string. M forces this transcript and randomness on $\mathcal{A}$ in the same manner as $\mathcal{S}$ does. Now, the following two cases are possible:

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

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

In the first case, the (C, M) system is identical to $\mathcal{H}_{i:5}$, while in the second case, the (C, M) system is identical to $\mathcal{H}_{i:6}$. By the (semi-honest) security of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, we establish that the output of M must be indistinguishable in both the cases, except with negligible probability. This proves equation 11.

Proving Equation 12. We will leverage the semi-honest security of the two-party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ and the extractability property of the non-malleable commitment scheme in NMZK to prove equation 12.

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

References

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *FOCS*, pages 345–355, 2002.

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

[BCNP04] B. Barak, R. Canetti, J.B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security*, pages 257–266, 2008.

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

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

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552. IEEE Computer Society, 2005.

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

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

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, 2004.

[CDPW07] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *TCC*, 2007.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, Lecture Notes in Computer Science, pages 19–40. Springer, 2001.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for uc secure computation using tamper-proof hardware. EUROCRYPT, 2008.

[CKL06] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\widetilde{\Omega}\left(\log n\right)$ rounds. In *STOC*, pages 570–579, 2001.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, pages 541–550, 2010.

[CPS07] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259, 2007.

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

[DKSW09] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In *TCC*, pages 146–162, 2009.

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

[Ein05] Albert Einstein. Zur elektrodynamik bewegter krper. *Annalen der Physik*, 17:891–921, 1905.

[GGJS11] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Bringing people of different beliefs together to do uc. In *TCC*, pages 311–328, 2011.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In *CRYPTO*, pages 277–294, 2010.

[GK08] Vipul Goyal and Jonathan Katz. Universally composable multi-party computation with an unreliable common reference string. In *TCC*, pages 142–154, 2008.

[GMPY06] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. In *TCC*, pages 404–428, 2006.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

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

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In *CRYPTO*, pages 323–341, 2007.

[Gol01] Oded Goldreich. *Foundation of Cryptography - Basic Tools.* Cambridge University Press, 2001.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one-way functions. In *STOC*, 2011.

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

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

[Kat07] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Eurocrypt*, 2007.

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

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in polyloalgorithm rounds. In *STOC*, pages 560–569, 2001.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.

[LP09] Huijia Lin and Rafael Pass. Non-malleability amplification. In *STOC*, pages 189–198, 2009.

[LP11] Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In *STOC*, 2011.

[LPTV10] Huijia Lin, Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable zero knowledge proofs. In *CRYPTO*, pages 429–446, 2010.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC*, pages 179–188, 2009.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[MPR06] Silvio Micali, Rafael Pass, and Alon Rosen. Input-indistinguishable computation. In *FOCS*, pages 367–378. IEEE Computer Society, 2006.

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

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, pages 33–43, 1989.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.

[PR05a] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *FOCS*, pages 563–572, 2005.

[PR05b] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *STOC*, pages 533–542, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.

[RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT*, pages 415–431, 1999.

[Ros04] Alon Rosen. A note on constant-round zero-knowledge proofs for np. In *TCC*, pages 191–202, 2004.

[Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *FOCS*, pages 531–540, 2010.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE, 1986.

A UC Security

In this section we briefly review UC security. For full details see [Can00]. A large part of this introduction has been taken verbatim from [CLP10]. We first review the model of computation, ideal protocols, and the general definition of securely realizing an ideal functionality. Next we present hybrid protocols and the composition theorem.

The basic model of execution. Following [GMR89, Gol01], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Specifically, an ITM has three tapes that can be written to by other ITMs: the input and subroutine output tapes model the inputs from and the outputs to other programs running within the same "entity" (say, the same physical computer), and the incoming communication tapes and outgoing communication tapes model messages received from and to be sent to the network. It also has an identity tape that cannot be written to by the ITM itself. The identity tape contains the program of the ITM (in some standard encoding) plus additional identifying information specified below. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs*, or ITIs, that represent interacting processes in a running system. Specifically, an ITI is an ITM along with an identifer that distinguishes it from other ITIs in the same system. The identifier consists of two parts: A session-identifier (SID) which identifies which protocol instance the ITM belongs to, and a party identifier (PID) that distinguishes among the parties in a protocol instance. Typically the PID is also used to associate ITIs with "parties", or clusters, that represent some administrative domains or physical computers.

The model of computation consists of a number of ITIs that can write on each other's tapes in certain ways (specified in the model). The pair (SID,PID) is a unique identifier of the ITI in the system.

With one exception (discussed within) we assume that all ITMs are probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by M is at most n^c, where n is the overall number of bits written on the *input tape* of M in this run. (In fact, in order to guarantee that the overall protocol execution process is bounded by a polynomial, we define n as the total number of bits written to the input tape of M, *minus the overall number of bits written by M to input tapes of other ITMs.*; see [Can01].)

Security of protocols. Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. Below we overview the model of protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

The model for protocol execution. The model of computation consists of the parties running an instance of a protocol Π, an adversary $\mathcal{A}$ that controls the communication among the parties, and an *environment* $\mathcal{Z}$ that controls the inputs to the parties and sees their outputs. We assume that all parties have a security parameter $n \in \mathbb{N}$. (We remark that this is done merely for convenience and is not essential for the model to make sense). The execution consists of a sequence of *activations*, where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some other ITM) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is given an external input z and is the first to be activated. In its first activation, the environment invokes the adversary $\mathcal{A}$, providing it with some arbitrary input. In the context of UC security, the environment can from now on invoke (namely, provide input to) only ITMs that consist of a single instance of protocol Π. That is, all the ITMs invoked by the environment must have the same SID and the code of Π.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape or report information to $\mathcal{Z}$ by writing this information on the subroutine output tape of $\mathcal{Z}$. For simplicity of exposition, in the rest of this paper we assume authenticated communication; that is, the adversary may deliver only messages that were actually sent. (This is however not essential as shown in [Can04, BCL$^+$05].)

Once a protocol party (i.e., an ITI running Π) is activated, either due to an input given by the environment or due to a message delivered by the adversary, it follows its code and possibly writes a local output on the subroutine output tape of the environment, or an outgoing message on the adversary's incoming communication tape.

The protocol execution ends when the environment halts. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z, r)$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running protocol Π on security parameter n, input z and random input $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, r_2, \ldots$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$; $r_{\mathcal{A}}$ for $\mathcal{A}$, r_i for party P_i). Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z)$ random variable describing $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z, r)$ where r is uniformly chosen. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n, z)\}_{n\in\mathbb{N},z\in\{0,1\}^*}$.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal protocol* for carrying out the task at hand. A key ingredient in the ideal protocol is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a "trusted party") that interacts with the parties and the adversary. More specifically, in the ideal protocol for functionality $\mathcal{F}$ all parties simply hand their inputs to an ITI running $\mathcal{F}$. (We will simply call this ITI $\mathcal{F}$. The SID of $\mathcal{F}$ is the same as the SID of the ITIs running the ideal protocol. (the PID of $\mathcal{F}$ is null.)) In addition, $\mathcal{F}$ can interact with the adversary according to its code. Whenever $\mathcal{F}$ outputs a value to a party, the party immediately copies this value to its own output tape. We call the parties in the ideal protocol dummy parties. Let $\Pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$.

Securely realizing an ideal functionality. We say that a protocol Π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π, or it is interacting with S and parties running ϕ. This means that, from the point of view of the environment, running protocol Π is 'just as good' as interacting with ϕ. We say that Π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0, 1\}$.

Definition 5 *Let Π and ϕ be protocols. We say that Π UC-emulates ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have* $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Definition 6 *Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol. We say that Π UC-realizes $\mathcal{F}$ if Π UC-emulates the ideal process $\Pi(\mathcal{F})$.*

Hybrid protocols. Hybrid protocols are protocols where, in addition to communicating as usual as in the standard model of execution, the parties also have access to (multiple copies of) an ideal functionality. Hybrid protocols represent protocols that use idealizations of underlying primitives, or alternatively make *trust assumptions* on the underlying network. They are also instrumental in stating the universal composition theorem. Specifically, in an $\mathcal{F}$-hybrid protocol (i.e., in a hybrid protocol with access to an ideal functionality $\mathcal{F}$), the parties may give inputs to and receive outputs from an unbounded number of copies of $\mathcal{F}$.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, giving input to a copy of $\mathcal{F}$ is done by writing the input value on the input tape of that copy. Similarly, each copy of $\mathcal{F}$ writes the output values to the subroutine output tape of the corresponding party. It is stressed that the adversary does not see the interaction between the copies of $\mathcal{F}$ and the honest parties.

The copies of $\mathcal{F}$ are differentiated using their SIDs. All inputs to each copy and all outputs from each copy carry the corresponding SID. The model does not specify how the SIDs are generated, nor does it specify how parties "agree" on the SID of a certain protocol copy that is to be run by them. These tasks are left to the protocol. This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks.

The definition of a protocol securely realizing an ideal functionality is extended to hybrid protocols in the natural way.

The universal composition operation. We define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$-hybrid protocol, and let Π be a protocol that securely realizes $\mathcal{F}$. The composed protocol ρ^{Π} is constructed by modifying the code of each ITM in ρ so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of Π with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of Π, with the contents of that message given to Π as new input. Each output value generated by a copy of Π is treated as a message received from the corresponding copy of $\mathcal{F}$. The copy of Π will start sending and receiving messages as specified in its code. Notice that if Π is a $\mathcal{G}$-hybrid protocol (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is ρ^{Π}.

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the composition theorem basically says that if Π is a protocol that UC-realizes $\mathcal{F}$ then, for any $\mathcal{F}$-hybrid protocol ρ, we have that an execution of the composed protocol ρ^{Π} "emulates" an execution of protocol ρ. That is, for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and protocol ρ^{Π} or with $\mathcal{S}$ and protocol ρ, in a UC interaction. As a corollary, we get that if protocol ρ UC-realizes $\mathcal{F}$, then so does protocol ρ^{Π}. [26]

Theorem 3 (Universal Composition [Can01].) *Let $\mathcal{F}$ be an ideal functionality. Let ρ be a $\mathcal{F}$-hybrid protocol, and let Π be a protocol that UC-realizes $\mathcal{F}$. Then protocol ρ^{Π} UC-emulates ρ.*

An immediate corollary of this theorem is that if the protocol ρ UC-realizes some functionality $\mathcal{G}$, then so does ρ^{Π}.

UC Security with Super-polynomial Simulation We next provide a relaxed notion of UC security by giving the simulator access to super-poly computational resources. The universal composition theorem generalizes naturally to the case of UC-SPS, the details of which we skip.

Definition 7 *Let Π and ϕ be protocols. We say that Π UC-SPS-emulates ϕ if for any adversary $\mathcal{A}$ there exists a* super-polynomial time *adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have* $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Definition 8 *Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol. We say that Π UC-SPS-realizes $\mathcal{F}$ if Π UC-SPS-emulates the ideal process $\Pi(\mathcal{F})$.*

[26] The universal composition theorem in [Can01] applies only to "subroutine respecting protocols", namely protocols that do not share subroutines with any other protocol in the system.