Round-Efficient Concurrently Composable Secure Computation via a Robust Extraction Lemma

Vipul Goyal
MSR India
vipul@microsoft.com

Omkant Pandey
UT Austin
omkant@cs.utexas.edu

Amit Sahai
UCLA
sahai@cs.ucla.edu

Abstract

We consider the problem of constructing protocols for secure computation that achieve strong concurrent and composable notions of security in the plain model. Unfortunately UC-secure secure computation protocols are impossible in this setting, but the Angel-Based Composable Security notion offers a promising alternative. Until now, however, under standard (polynomial-time) assumptions, only protocols with polynomially many rounds were known to exist.

In this work, we give the first $\tilde{O}(\log n)$-round secure computation protocol in the plain model that achieves angel-based composable security in the concurrent setting, under standard assumptions. We do so by constructing the first $\tilde{O}(\log n)$-round CCA-secure commitment protocol. Our CCA-secure commitment protocol is secure based on the minimal assumption that one-way functions exist.

A central tool in obtaining our result is a new *robust concurrent extraction lemma* that we introduce and prove, based on the minimal assumptions that one-way functions exist. This robust concurrent extraction lemma shows how to build concurrent extraction procedures that work even in the context of an "external" protocol that cannot be rewound by the extractor. We believe this lemma can be used to simplify many existing works on concurrent security, and is of independent interest. In fact, our lemma when used in conjunction with the concurrent-simulation schedule of Pass and Venkitasubramaniam (TCC'08), also yields a constant round construction based additionally on the existence of quasi-polynomial time ($\mathcal{PQT}$) secure one-way functions.

1 Introduction

The notion of *secure multi-party computation* protocols 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 yet no group of malicious parties learn anything beyond their inputs and the prescribed outputs. These early results on secure computation [Yao86, GMW87], along with a rich body of followup works that further refined and developed the concept [GL90, GMW91, Bea91, MR91, Can00, PW01, Can01, Gol04], demonstrated that the delicate task of designing secure protocols can be captured by general secure computation.

Much of the early literature on secure computation only considered the *stand-alone* setting where security holds only if a single execution of the protocol takes place, in isolation with no other cryptographic activity in the system. We call this security *stand-alone security*. While stand-alone security may be sufficient for basic purposes, it does not suffice in today's more complex networked environments where other cryptographic protocols might be running in the system simultaneously.

Concurrent Security. To deal with more complex systems, the last decade has seen a push towards obtaining protocols that have strong concurrent *composability* properties. For example, we could require concurrent self-composability: the protocol should remain secure even when there are multiple copies executing concurrently. The framework of *universal composability* (UC) was introduced by Canetti [Can01] to capture a more general security requirement for a protocol that may be executed concurrently with not only several copies of itself but also with other protocols in an arbitrary manner.

Unfortunately, strong impossibility results have been shown ruling out the existence of secure protocols in the concurrent setting. UC secure protocols for most functionalities of interest have been ruled out in [CF01, CKL03]. Protocols in even less demanding settings of concurrent security were ruled on in [Lin04, BPS06, AGJ$^+$12, Goy12]. We stress that, in fact, the latest sequence of these impossibility results provide an *explicit attack* in the concurrent setting using which the adversary may even fully recover the input of an honest party (see, e.g., the chosen protocol attack in [BPS06]). Hence, designing secure protocols in the concurrent setting is a question of great theoretical interest as well as practical motivation.

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]. Nevertheless, a driving goal in cryptographic research is to eliminate the need to trust other parties. The main focus of this paper is to obtain concurrently-secure protocols in the *plain model*.

Relaxing the Security Notion. To address the problem of concurrent security for secure computation in the plain model, a few candidate definitions have been proposed, the most well studied one being that of *super-polynomial simulation* [Pas03, PS04, BS05]. The notion of security with *super-polynomial simulators* (SPS) [Pas03, PS04, BS05] 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).

Angel-based UC security. To formalize the notion of SPS security in a way that allows modular analysis and provides composability, Prabhakaran and Sahai [PS04] put forward the notion of *angel-based composable security*. Very roughly, in the angel based security notion, the parties (including the simulator and the adversary) are all polynomial time but have access to an angel which will perform certain *specific* super-polynomial time computations. This angel-based definition is in contrast to the case where the simulator is given direct access to super-polynomial computation power: in this case, the resulting security notion is *not* closed under composition and thus does not permit a modular protocol design in the concurrent setting[1]. A construction for concurrently secure computation in the angel based composable security model were given in [PS04, BS05], but only based on non-standard super-polynomial hardness assumptions.

Very recently, Canetti, Lin, and Pass [CLP10] obtained the first secure computation protocol that achieves angel-based composable security based on *standard polynomial-time assumptions*. 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*.

[1]However we note that according to this weaker SPS security notion, concurrently secure protocols in constant rounds are now known [GGJS12].

A follow up work of Lin and Pass [LP12] considers the problem of designing *black-box* constructions for secure computation in the concurrent setting. They propose a protocol making only a black-box use of oblivious transfer satisfying the angel-based composable security notion. However the round complexity of their protocol continues to remain polynomial.

We note that 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, BNP08]. Indeed, round complexity has been the subject of a good 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]). The round complexity of non-malleable commitments in the stand-alone and concurrent settings has also been studied in several works[DDN91, Bar02, PR05b, PR05a, LP09, Wee10, Goy11, LP11], improving the round complexity from logarithmic rounds to constant rounds under minimal assumptions. We observe that for the setting of concurrently secure computation protocols with angel-based composable security, the situation is worse since the only known protocols that achieves angel-based composable security based on standard assumptions incurs polynomial-round complexity [CLP10, LP12]. This raises the following natural question:

"Do there exists round-efficient protocols in the concurrent setting satisfying the angel-based composable notion of security based on standard assumptions?"

Our Results. We answer the above question in the affirmative and provide a $\widetilde{O}(\log n)$ round construction of concurrently secure computation in the plain model. Our construction satisfies the angel-based composable notion of security [PS04, CLP10]. To obtain our result, we construct a "CCA-secure commitment" protocol in $\widetilde{O}(\log n)$ rounds, based only the assumption that one-way functions exist. CCA secure commitments were introduced in [CLP10]; roughly speaking, a commitment protocol is CCA-secure if it remains hiding even when the adversary is given an oracle that can open all commitment values (except the commitment given as a challenge to the adversary). In [CLP10], Canetti et al. show how to construct a protocol that securely realizes any functionality—under the angel-based composable notion of security—given an (appropriate) protocol for CCA secure commitments (see full version of [CLP10]). Prior to our work, the best known construction for CCA secure commitments based on standard (polynomial time) assumptions, required n^{ϵ} rounds [CLP10, LP12]. In contrast, the round complexity of our protocol matches that of the best known constructions for concurrent extractable commitment schemes [PRS02, MOSV06].

A robust concurrent extraction lemma. A key technical tool that we introduce is a lemma that allows *robust* extraction of secrets from an adversarial committer A^* in the *concurrent* setting. We call this lemma, the *robust concurrent extraction lemma*, which is of independent interest. Roughly speaking, the lemma is a strengthening of the concurrent extraction mechanism for the PRS preamble [PRS02] (we shall call this the PRS commitment), and states that concurrent extraction can be performed even in the presence of an external protocol which cannot be rewound by the "simulator."

More precisely, consider an adversarial committer A^* who commits to multiple values in *concurrent* sessions of the PRS commitment to honest receivers; let us label these sessions as the **right** sessions. Simultaneously, A^* participates in a **left** execution of an arbitrary k-round protocol, denoted $\Pi := \langle B, A \rangle$. Then, the robust concurrent-extraction lemma states that for every A^* there exists a simulator S which, *without rewinding the external party B in its execution of* Π, extracts the values committed to by A^* in every session of the PRS commitment. Furthermore, if ℓ is the

round complexity of the PRS preamble, and T is the running time of A^*, then S only fails with probability that is exponentially small in $\ell - O(k \cdot \log T)$.

In order to capture correctness of extraction, we formulate our lemma by considering a "real-world" experiment in which A^* receives the values committed to in every valid PRS commitment from an exponentially powerful party $\mathcal{E}$, called the "online extractor." The extraction of values are provided by $\mathcal{E}$ as soon as a PRS session ends. Our lemma states, intuitively speaking, that both the adversary A^* and the exponentially powerful party $\mathcal{E}$ can be replaced with a polynomial-time simulator, that still interacts with the external party B in the external protocol Π. We remark that formulating the lemma in a generic way, so that it can handle as general usage of the PRS commitment as we have seen in the literature, is a delicate task. Nevertheless, we show that it is possible to precisely capture the concurrent-extraction property of the PRS commitment in a generic way without referring to any specific protocol that uses it.

An immediate benefit of our formulation is that when the PRS commitment is used inside a larger protocol, the task of "concurrent extraction" can be *formally* isolated from other parts of the protocol. This allows one to design hybrid experiments without having to worry about the extraction. We provide two procedures for this purpose—a simulator S and an online extraction $\mathcal{E}$—and demonstrate their use in the security proof of our protocol.

We also wish to remark that the ability to extract without rewinding B turns out to be a very useful tool during concurrent security proofs, and we expect this will have significant applications elsewhere. This flexibility simplifies security proofs (of even previous works) to a great extent. For example, a situation similar to our lemma arises in previous works on concurrent non-malleable zero-knowledge [BPS06, LPTV10], In these works, the problem is solved in an arguably ad-hoc fashion, which stops rewindings after a certain point in the simulation during certain hybrids. This, overall, leads to a rather delicate analysis, and the order of hybrid experiments becomes important.

By using our robust-extraction lemma, this problem can be avoided almost directly. We note however, that in our case, it is crucial that the rewindings not be stopped. This is because, in our situation, one needs to implement the super-polynomial angel from the beginning in each hybrid experiment. Hence, every session in every hybrid requires online extraction—which is possible either by using (all) rewindings or by using super-polynomial simulation.

Technical Overview. As mentioned before, the starting point of our construction is the robust extraction lemma. The basic problem encountered in proving the lemma is that given the entire transcript of interaction between A^* and the honest right parties, there are various "breakpoints" in the transcript (representing messages of the left protocol) which cannot be rewound. In addition, during rewindings (or look-ahead threads), the breakpoints can change their location and can even come earlier than expected (at which point the current thread must be discontinued and another one started). At a high level, we start by considering the necessary modification of the KP/PRS simulator so that when a breakpoint is encountered during the execution of a look-ahead thread, the look-ahead thread is stopped and abandoned.

Our proof shows that even with this modification, the simulation still succeeds. The key observation is that each breakpoint can "spoil" at most a $d = \log T$ number of "recursive blocks" used in the swapping argument of [PRS02]. Since there are k breakpoints, this incurs an additional loss of $k \cdot \log T$ blocks. However, to execute this proof strategy, it becomes crucial to use the information learned during a "look-ahead block" in certain special sibling blocks. This new feature of our analysis must be done carefully to maintain the correctness of our swapping argument.

We note that we strive to obtain the best possible bounds for the round complexity ℓ of PRS that is necessary to extract for a given value of k (rounds of the left protocol). For this reason, we

choose to re-analyze the proof presented in [PRS02] (see also [PTV08])[2].

For the goal of constructing CCA secure commitments, a direct application of the robust extraction lemma will not be sufficient. This is because the number of rounds in the left and the right interaction will be the same, and the robust extraction lemma cannot work with respect to protocols with the same round complexity as the PRScommitment! To construct CCA secure commitments (which is our main technical goal), we instead build upon techniques from prior work on concurrent non-malleable zero-knowledge (CNMZK) [BPS06, GJO10, LPTV10]. At a high level, our protocol is simple: commit to the value using a regular commitment scheme, and then, prove the validity of the commitment using a concurrent zero-knowledge protocol that is also *simulation sound* [Sah99]. We note that we design our own protocol for this task since we strive to achieve a construction based on *one-way functions* only (for CCA secure commitments). Using techniques from [BPS06, LPTV10] is either not possible or results in stronger assumptions such as the existence of collision-resistant hash functions. Our protocol is presented in section 3.

A Constant Round Protocol from $\mathcal{PQT}$ One-way Functions. Our techniques can be seen as a general method which reduce the task of concurrent-extraction to that of concurrent-simulation. The method requires only a constant-factor blow up in the round-complexity (of a given concurrent-simulation method) and a polynomial-factor blow up in the running time of the (given) simulator. By applying our method to the concurrent-simulator of [PV08], we obtain a constant round protocol for CCA-secure commitments based only on the existence of one-way functions secure against adversaries running in time super quasi-polynomial time ($\mathcal{PQT}$) (see [PV08]).

Assuming more complex, and somewhat non-standard assumptions—namely the existence of adaptive one-way functions [PPV08]—a constant round protocol for CCA secure commitments is already known [PPV08]. In addition, our techniques can also be combined with the recent constant round concurrent zero-knowledge protocols of [CLP12, GS12][3], to obtain constant round CCA secure commitments from similar (non-standard) assumptions.

Related Works. The work of Garg, Goyal, Jain, and Sahai [GGJS12] is closely related to our work, who provide a constant round protocol under the non-composable SPS notion (instead of angel-based composable security). Their work requires the existence of statistically hiding commitment. Independently of [GGJS12], a recent work of Lin, Pass, and Venkitasubramaniam [LPV11] also provides a constant round protocol achieving non-composable SPS security, using very different techniques.

Other security notions that deal with concurrent security were presented in [MPR06, GGJS12] who propose the notion of *input indistinguishable computation*, and in [GS09, GJO10] who considered a modified ideal world that allows the adversary to make more output queries to the ideal functionality (than just one) per session.

[2]For example, one can consider the approach of applying a pigeonhole principle argument to argue that $\omega(\log n)$ slots must occur between some two breakpoints, and then trying to apply the PRS analysis simply to these slots. However, note that even if no breakpoints occur during these slots, *look-ahead threads* that are started during these slots can still encounter breakpoints, since the adversary can choose the scheduling adaptively. Dealing with this analytically would require further loss, and result in a worse asymptotic bound than ours for super-constant values of k. Our more direct approach shows how to amortize the gains made over all slots, even if only a few slots occur between some breakpoints.

[3]Chung, Lin, and Pass [CLP12] construct a constant-round concurrent zero-knowledge protocol under a new (falsifiable) assumption, namely the *existence of strong $\boldsymbol{P}$-certificate system*. Gupta and Sahai [GS12] also obtain the same result, albeit under a (new) knowledge-type assumption.

2 Robust Concurrent Extraction

In this section, we will prove the robust extraction lemma. We use standard notation. In particular, $A(x;r)$ denotes the process of evaluating (randomized) algorithm A on input x with random coins r, and $A(x)$ the process of sampling a uniform r and then evaluating $A(x;r)$. We define $A(x,y;r)$ and $A(x,y)$ analogously. The set of natural numbers is represented by $\mathbb{N}$. Unless specified otherwise, $n \in \mathbb{N}$ represents the security parameter available as an implicit input. when necessary. All inputs are assumed to be of length at most polynomial in n.

For two probability distributions D_1 and D_2, we write $D_1 \stackrel{c}{\equiv} D_2$ to mean that D_1 and D_2 are computationally indistinguishable. For two interactive Turing machines (ITM) A and B, we write $\mathsf{OUT}_B\left[A(1^n,x) \leftrightarrow B(1^n,y)\right]$, the output of B after an interaction with A where their inputs in the interaction are y and x respectively, and their random tapes are independent and uniform.

We assume familiarity with commitment schemes. Without loss of generality, we will be using commitment schemes with non-interactive reveal phase—i.e., the committer sends a single message (v,d) to decommit. For a commitment scheme $\langle C,R\rangle$, we denote by $\mathsf{open}_{\langle C,R\rangle}(c,v,d)$ the decommitment function. That is, the receiver accepts v as the value committed to in the commitment-transcript c if $\mathsf{open}_{\langle C,R\rangle}(c,v,d)$ outputs 1, and rejects otherwise. For statistically binding commitments, v is uniquely determined given c with high probability.

2.1 The PRS Preamble

The robust extraction lemma deals with the commitment preamble of Prabhakaran, Rosen, and Sahai [PRS02]. This preamble has been used in many prior works, and is often referred to as the PRS preamble. The preamble uses an underlying commitment scheme Com. Roughly speaking, the committer first commits to many shares of the value v to be committed using Com. This is followed by a several rounds where in each round, the receiver sends a random challenge, and the committer responds with appropriate decommitments. Each round is called a *slot*.

Essentially, the PRS preamble is an interactive commitment scheme, which is statistically binding (resp. hiding) if the underlying scheme Com is statistically binding (resp., hiding). The formal description of the PRS preamble is given in figure 1. As before, we write $\mathsf{open}_{\mathsf{PRS}}(c,v,\rho)=1$, to formally mean that there exists randomness ρ such that c is the transcript of the PRS preamble, executed between the honest committer with input v and randomness ρ and the honest receiver with randomness (equal to its challenges) appearing in c.

2.2 The Extraction Lemma

In this section, we present the robust extraction lemma. We will consider an adversary A^* who interacts in many sessions of the PRS preamble; simultaneously, A^* also participates in a single execution of a two party computation protocol Π. The running time of A^* is not necessarily polynomial in n. However, the lemma becomes trivial if the hiding/binding of the underlying commitment Com can be broken in $\mathrm{poly}(n)\cdot T^2$ time, where $T=T(n)$ is the maximum number of PRS preambles A^* initiates.

Simplifying Assumption. We assume, for the clarity of presentation, that the commitment scheme Com underlying the PRS preamble is *statistically binding*. Later, we will present the general form which deals with both kinds of Com, as well as varying round complexity of the preamble.

The security parameter is n, the value to be committed is $v \in \{0,1\}^n$, and the round-parameter is $\ell := \ell(n)$.

Commitment. The committer and the receiver execute the following steps.

1. The committer chooses $n\ell$ pairs of n-bit random strings $(v^0_{i,j}, v^1_{i,j})$ for $i \in [n], j \in [\ell]$ such that (for every i, j): $v^0_{i,j} \oplus v^1_{i,j} = v$. It commits to strings v, $v^b_{i,j}$ using the commitment scheme Com, for every $b \in \{0,1\}, i \in [n], j \in [\ell]$.
2. For $j = 1$ to ℓ:
   (a) the receiver sends a n-bit challenge string $r_j = r_{1,j}, \ldots, r_{n,j}$
   (b) the committer responds by sending a decommitment to strings $v^{r_{1,j}}_{1,j}, \ldots, v^{r_{n,j}}_{n,j}$

Decommitment. The committer decommits to all remaining strings which were not opened in the commit phase.

Figure 1: The PRS Preamble based on Com.

Protocol Π. Let $\Pi := \langle B, A \rangle$ be an arbitrary two-party computation protocol. We assume w.l.o.g. that both B and A receive a parameter $n \in \mathbb{N}$ as their first input. In addition, for a fixed $n \in \mathbb{N}$, let $\mathrm{dom}_B(n)$ denote the domain of valid (second) input for algorithm B, and $k := k(n)$ denote the round complexity of Π.

The robust-concurrent attack. Let A^* be an interactive Turing machine, called the adversary, $n \in \mathbb{N}$ the security parameter, and $x \in \mathrm{dom}_B(n)$ an input. In the robust-concurrent attack, A^* interacts with a special, not necessarily polynomial time, party $\mathcal{E}$ called the "online extractor." Party $\mathcal{E}$ simultaneously participates in one execution of the protocol Π, and several executions of the PRS preamble with A^*. Party $\mathcal{E}$ follows the (honest) algorithm $B(1^n, x)$ in the execution of Π with A^*. Further, it follows the (honest) receiver algorithm in each execution of the PRS preamble. If A^* successfully completes a PRS preamble s, $\mathcal{E}$ sends a string α_s to A^*, together with a special message $\textsc{end}_s$, to mark the completion of the preamble.

The scheduling of all messages in all sessions—Π as well as PRS preambles—is controlled by A^* including starting new sessions and finishing or aborting existing sessions. We adopt the following conventions [Ros04, PRS02]:

1. When A^* sends a round i message of session s, it immediately receives the next—i.e., $(i+1)$-st message of s; this is without loss of generality,[4] and holds for messages of Π as well.
2. If a session s has been aborted, A^* does not schedule any further messages of s.
3. If A^* starts a PRS preamble s, it also sends a special message, denoted $\textsc{start}_s$, immediately *after the last message* of step 1 of this preamble is completed (see figure 1). Message $\textsc{start}_s$ indicates that the challenge-response phase is about to start.

At some point, A^* halts. We say that A^* *launches* the robust-concurrent attack.

For $n \in \mathbb{N}, x \in \mathrm{dom}_B(n), z \in \{0,1\}^*$, let $\mathsf{REAL}^{A^*}_{\mathcal{E},\Pi}(n, x, z)$ denote the output of the following probabilistic experiment: on input 1^n and auxiliary input z, the experiment starts an execution of A^*. Adversary A^* launches the robust-concurrent attack by interacting with the special party $\mathcal{E}$ throughout the experiment, as described above. When A^* halts, the experiment outputs the view

[4] This is because the next message can be stored and delivered whenever needed during the attack.

of A^* which includes: all messages sent/received by A^* to/from $\mathcal{E}$, the auxiliary input z, and the randomness of A^*. $\square$

We are now ready to present the robust extraction lemma. Informally speaking, the lemma states that there exists an interactive Turing machine—a.k.a the robust simulator—whose output is statistically close to $\mathsf{REAL}^{A^*}_{\mathcal{E},\Pi}(n,x,z)$ *even if* the final response of $\mathcal{E}$ at the end of a successful PRS session is actually the value A^* commits to in that session. Further, the robust simulator does not "rewind" B, and runs in time polynomial in total sessions opened by A^*.

Lemma 1 (Robust Concurrent Extraction)**.** *There exists an interactive Turing machine S ("robust simulator"), such that for every A^*, for every $\Pi := \langle B, A\rangle$, there exists a party $\mathcal{E}$ ("online extractor"), such that for every $n \in \mathbb{N}$, for every $x \in \mathrm{dom}_B(n)$, and every $z \in \{0,1\}^*$, the following conditions hold:*

1. **Validity constraint.** *For every output ν of $\mathsf{REAL}^{A^*}_{\mathcal{E},\Pi}(n,x,z)$, for every PRS preamble s (appearing in ν) with transcript τ_s, if there exists a* unique *value $v \in \{0,1\}^n$ and randomness ρ such that $\mathsf{open}_{\mathsf{PRS}}(\tau_s, v, \rho) = 1$, then:*

$$\alpha_s = v,$$

   where α_s is the value $\mathcal{E}$ sends at the completion of preamble s.

2. **Statistical simulation.** *If $k = k(n)$ and $\ell = \ell(n)$ denote the round complexities of Π and the PRS preamble respectively, then the statistical distance between distributions $\mathsf{REAL}^{A^*}_{\mathcal{E},\Pi}(n,x,z)$ and $\mathsf{OUT}_{\mathrm{S}}\left[B(1^n,x) \leftrightarrow S^{A^*}(1^n,z)\right]$ is given by:*

$$\Delta(n) \leq 2^{-\Omega(\ell - k\cdot\log T(n))},$$

   where $T(n)$ is the maximum number of total PRS preambles between A^* and $\mathcal{E}$.[5] *Further, the running time of S is $\mathrm{poly}(n)\cdot T(n)^2$.*

We prove this lemma by presenting an explicit simulator S and a corresponding party $\mathcal{E}$. The explicit constructions appear in subsection 2.3, and the full proof of the lemma appears towards the end of the paper, in section 5. We now make some important remarks about the lemma.

Remarks.

1. The special party $\mathcal{E}$ is not completely defined by the lemma. In particular, when a PRS preamble s does not commit to a *unique and valid* value v, the value α_s sent by $\mathcal{E}$ to A^* is not defined. This can happen, e.g., when not all shares committed to in step 1 XOR to v. In such situations, $\mathcal{E}$ can choose whatever value α_s it wants. The only requirements on $\mathcal{E}$ are that it uses honest algorithms during the robust-concurrent attack *and* that for each successfully completed PRS preamble it satisfies the validity constraint. Every $\mathcal{E}$ satisfying these requirements is called a *valid* $\mathcal{E}$.

2. The PRS preamble is used in a variety of complex ways. For example, some protocols require opening the committed PRS-value (e.g., [PRS02, BPS06, OPV10], whereas some others may

[5] The lemma allows for exponential $T(n)$ as well. However, if it is too large—e.g., $T(n) = 2^{2n}$, the PRS preamble should be modified suitably. For example, the value v as well as the challenges in each slot, must be of length at least $n + 2\log T(n)$.

never open this value (e.g., [LPTV10, GJO10]). To be able to capture such uses generically, we do not enforce any consistency requirements on the PRS preambles. The choice of not fully defining $\mathcal{E}$ when PRS preamble is not valid provides sufficient flexibility to capture such generic uses of the preamble.

3. When the preamble is used in a larger protocol, a "main" simulator is used to prove the security of the larger protocol. Typically, the main simulator employs the rewinding strategy of [KP01, PRS02] to extract the PRS-values and simultaneously deals with other details of the protocol. Our lemma separates the task of extracting PRS-values from other necessary actions of the main simulator. This makes the overall proofs simpler. Party $\mathcal{E}$ then only acts as mechanism to transfer the extracted PRS-values back to the main simulator. The main simulator takes upon the role of A^* to receive extracted values from $\mathcal{E}$, while only dealing with other details of the larger protocol.

4. A consequence of the above two remarks is that when a PRS preamble is not consistent, we do not know what value α_s actually gets extracted. Our choice of the order of quantifiers allows $\mathcal{E}$ to depend on A^* as well as S. This essentially allows $\mathcal{E}$ to extract and supply the same value α_s (by running S internally) that a typical "main" simulator would extract for inconsistent PRS preambles.

5. Requirements 1 and 2 of the lemma imply that if we sample an output of the simulator and consider a PRS preamble s with transcript τ_s which contains a *unique* and *valid* value v, and receives α_s as $\mathcal{E}$'s response in the end, then except with probability $\Delta(n)$, it holds that $\alpha_s = v$.

2.3 A Robust Simulator and an Online Extractor

In this section, we present an explicit construction of a robust simulator S, and the (online extractor) party $\mathcal{E}$ for which (the robust extraction) lemma 1 holds. The simulator is a slight modification of [KP01, PRS02], to also deal with messages of Π, without rewinding them. We start by defining a few terms first.

The states of A^*. Recall that the scheduling of messages in the robust-concurrent attack is controlled by A^*, and when A^* sends the i-th message of a session s (either PRS or Π), it immediately receives the next message of s, namely the $(i+1)$-st message. The *state* of A^* at any given point during the attack consists of its *view up to that point*: it includes all messages sent/received by A^*, its auxiliary input z and its randomness. The *starting* (or *original*) state of A^*—denoted throughout by st_0—is its state before it receives the first message. If st denotes the state of A^* at some point during the robust-concurrent attack, the set of all PRS preambles which have not completed yet is denote by $\mathsf{LIVE}(\mathsf{st})$.

The robust simulator S. The simulator receives as input an auxiliary string $z \in \{0,1\}^*$, and the security parameter n. The simulator participates with an external party B of Π. Let $x \in \mathrm{dom}_B(n)$ and γ denote the input and uniformly chosen randomness of B. The simulator incorporates the adversary A^* as a black-box; let $T = T(n)$ define the maximum number of PRS preambles that A^* can open during the robust-concurrent attack.

Simulator S starts by setting $(1^n, z)$ on A^*'s input tape, and a sufficiently long uniform string on its random tape. The simulator then initiates a helper procedure recurse as follows:

$$(\mathsf{st}, \mathcal{T}) \leftarrow \mathsf{recurse}(T, \mathsf{st}_0, \emptyset, 1, \emptyset, 0).$$

Throughout its execution, messages of recurse are forwarded back and forth to $B(1^n, x; \gamma)$ and (the black-box) A^* as appropriate. Finally, the output of S is the first output of recurse, namely st; the output st is also known as the *main thread*. Procedure recurse is given in figure 2, each execution of recurse is called a block, and has a unique name denoted by id. □

The online extractor $\mathcal{E}$. Formally, an execution of $\mathcal{E}$ begins during a robust-concurrent attack. Let (γ, ρ) denote the random tape of $\mathcal{E}$, and (n, x, z) denote its inputs. $\mathcal{E}$ incorporates the program of A^*, and performs the following *internal steps* before it sends out its first message in the robust-concurrent attack,

1. $\mathcal{E}$ proceeds identically to the robust simulator algorithm S, using ρ as its random tape and $(1^n, z)$ as its inputs. To successfully proceed in this step, $\mathcal{E}$ uses (x, γ) to simulate the honest algorithm $B(1^n, x; \gamma)$, as well as black-box access to A^*.[6] However, $\mathcal{E}$ differs from S in its actions only when a PRS preamble s completes, in the manner described below.

2. Let s be a successfully completed PRS preamble; at this point S either extracts a value μ_s or reaches an ExtractFail. When this happens, $\mathcal{E}$ neither sends μ_s nor aborts the simulation; instead it proceeds as follows. First, $\mathcal{E}$ attempts to extract the *actual value* committed to in the preamble by inverting all instances of the underlying commitment Com. Then, it decides the value α_s, to be sent, as follows. If a valid and *unique* value v_s exists, set $\alpha_s = v_s$. Otherwise, it has following cases:

   (a) If there are more than one valid v_s, proceed as follows: if μ_s equals to any of them, set $\alpha_s = \mu_s$, otherwise, set α_s to be one of them chosen at random.

   (b) If no valid v_s exists, then proceed as follows: if $\mu_s = \mathsf{ExtractFail}$, set α_s to be a random value; otherwise, set $\alpha_s = \mu_s$.

3. After reaching the end of the simulation, $\mathcal{E}$ internally stores the randomness ρ_s and the values α_s for every PRS preamble s appearing on the *main thread*.

Having completed the steps above,[7] $\mathcal{E}$ is now ready to interact with the (outside) A^* launching the robust-concurrent attack, and proceeds as follows.

- If A^* sends a message intended for the (only) session of Π, $\mathcal{E}$ interacts with A^* by following actions of $B(1^n, x; \gamma)$. Likewise, if A^* sends a messages for a PRS preamble s, A^* follows the honest receiver algorithm of PRS preamble with *randomness* ρ_s already computed internally.

- If A^* successfully completes a PRS preamble s, $\mathcal{E}$ sends the already stored value α_s to A^*. □

3 CCA Secure Commitments in $\widetilde{O}(\log n)$ Rounds

In this section we apply our robust extraction lemma to construct a $\widetilde{O}(\log n)$-round protocol for CCA secure commitments. We will need the generalized version of the lemma which allows for *statistically hiding* PRS preamble as well; the general version appears in A. We will also use a non-malleable commitment scheme, denoted NMCom, that is *robust* w.r.t. constant round protocols [LP09]. Constant round constructions for such protocols are now known, see section C for details and definitions.

[6] Observe that although $\mathcal{E}$ depends on A^* here, it still uses A^* only as a black-box, as remarked earlier.

[7] We insist that all the steps above are *internal* to $\mathcal{E}$ and that as of now it has not sent any external message in the robust-concurrent attack. Further, it can successfully complete these steps since it has all the required inputs.

procedure $\mathsf{recurse}(t, \mathsf{st}, \mathcal{T}, \mathsf{f}, \mathsf{aux}, \mathsf{id})$:

1. If $t = 1$, **repeat**:
   (a) If the next message is START, start a new session s.
   – send $r \leftarrow \{0,1\}^n$ as the challenge of the first slot of s.
   – add entry $(s : 1, r, \_\_)$ to $\mathcal{T}$.
   (b) If the next message is the slot-i challenge of an existing session s.
   – send $r \leftarrow \{0,1\}^n$ as the slot-i challenge of s.
   – add entry $(s : i, r, \_\_)$ to $\mathcal{T}$.
   (c) If the next message is the slot-i *response*, say β, of an existing session s.
   – If β is a valid message:
   – update entry $(s : i, r_i, \_\_)$ to $(s : i, r_i, \beta)$.
   – if $i = \ell$, i.e., it is the last slot, send $(\text{END}, \mathsf{extract}(s, \mathsf{id}, \mathcal{T}, \mathsf{aux}))$.
   – Otherwise, if $\beta = \perp$, abort session j, and add $(s : \perp, \perp, \perp)$ to $\mathcal{T}$.
   – Update st to be the current state of A^*
   – **return** $(\mathsf{st}, \mathcal{T})$.
   (d) If the next message is a *response* from A^* for the external protocol Π.
   – If $\mathsf{f} = 0$, i.e., it is a look-ahead block, then **return** $(\mathsf{st}, \mathcal{T})$;
   – If $\mathsf{f} = 1$, i.e., it is the *main thread*), do the following:
   – send A^*'s message to the external party of Π, return the response to A^*.
   – Update st to be the current state of A^*
   – For every live session $s \in \mathsf{LIVE}(\mathsf{st})$, do the following:
   – $\times_{s,\mathsf{id}} = \mathsf{true}$,
   – for every block id' that contains the block id, set: $\times_{s,\mathsf{id}'} = \mathsf{true}$.
2. If $t > 1$,
   # Rewind the first half twice:
   (a) $(\mathsf{st}_1, \mathcal{T}_1) \leftarrow \mathsf{recurse}(t/2, \mathsf{st}, \mathcal{T}, 0, \mathsf{aux}, \mathsf{id} \circ 1)$ [look-ahead block C']
   (b) Let $\mathsf{aux}_2 = (\mathsf{aux}, \mathcal{T}_1 \setminus \mathcal{T})$,
   $(\mathsf{st}_2, \mathcal{T}_2) \leftarrow \mathsf{recurse}(t/2, \mathsf{st}, \mathcal{T}, \mathsf{f}, \mathsf{aux}_2, \mathsf{id} \circ 2)$ [main block C]
   # Rewind the second half twice:
   (c) $(\mathsf{st}_3, \mathcal{T}_3) \leftarrow \mathsf{recurse}(t/2, \mathsf{st}, \mathcal{T}^*, 0, \mathsf{aux}, \mathsf{id} \circ 3)$ [look-ahead block D']
   (d) Let $\mathcal{T}^* = \mathcal{T}_1 \cup \mathcal{T}_2$ and $\mathsf{aux}_4 = (\mathsf{aux}, \mathcal{T}_3 \setminus \mathcal{T}^*)$,
   $(\mathsf{st}_4, \mathcal{T}_4) \leftarrow \mathsf{recurse}(t/2, \mathsf{st}, \mathcal{T}^*, \mathsf{f}, \mathsf{aux}_4, \mathsf{id} \circ 4)$ [main block D]
   (e) **return** $(\mathsf{st}_4, \mathcal{T}_3 \cup \mathcal{T}_4)$.

procedure $\mathsf{extract}(s, \mathsf{id}, \mathcal{T}, \mathsf{aux})$:

1. Attempt to extract a value for s from $\mathcal{T}$.
2. If extraction fails, consider every block id_1 for which $\times_{s,\mathsf{id}_1} = \mathsf{true}$.
   – Let id'_1 be the sibling of id_1, with input/output tables $\mathcal{T}_{\text{in}}, \mathcal{T}_{\text{out}}$ respectively.
   – Attempt to extract from $\mathsf{aux}_{\mathsf{id}'_1} := \mathcal{T}_{\text{out}} \setminus \mathcal{T}_{\text{in}}$; (included in aux).
3. If all attempts fail, abort the simulation and **return** ExtractFail.
   Otherwise **return** the extracted value.

Figure 2: Procedures recurse and extract used by the robust simulator S.

The protocol has a very intuitive "commit-and-prove" structure where the committer commits to the value v using a PRS preamble, and then proves its consistency using what resembles a "concurrent simulation-sound" protocol. We start by recalling the notion of CCA secure commitments from [CLP10].

3.1 CCA Secure Commitments

Let $\langle C, R\rangle$ denote a statistically binding and computationally hiding commitment scheme. We assume w.l.o.g. that $\langle C, R\rangle$ has a non-interactive reveal phase—i.e., the committer simply sends (v, d). The decommitment is verified using a function $\mathsf{open}(c, v, d)$; that is, the receiver accepts v as the value committed in the commitment-transcript c if $\mathsf{open}(c, v, d)$ outputs 1, and rejects otherwise. A tag-based commitment scheme with $l(n)$-bit identities [PR05b, DDN91] is a commitment-scheme where in addition to 1^n, C and R also receive a "tag" (or *identity*) of length $l(n)$ as common input. We will consider schemes which are *efficiently checkable*: meaning that if R *accepts* in the interaction (with transcript c) then there exists a decommitment pair (v, d) such that $\mathsf{open}(c, v, d) = 1$.

In CCA-secure commitments, we consider an adversarial receiver A, who has access to an oracle $\mathcal{O}$, called the "decommitment oracle." The oracle participates with A in many *concurrent* sessions of (the commit phase of) $\langle C, R\rangle$, using tags of length $l(n)$, chosen adaptively by A. At the end of each session, if the session is accepting, the oracle returns the (unique) value committed by A in that session; otherwise it returns $\perp$. (In case there is more than one possible decommitment, $\mathcal{O}$ returns any one of them.)[8]

Roughly speaking, we say that a tag-based scheme $\langle C, R\rangle$ is CCA-secure if there exists a decomitment oracle $\mathcal{O}$ for $\langle C, R\rangle$, such that the hiding property of the scheme holds even for adversaries A with access to $\mathcal{O}$. Formally, let $\mathsf{IND}_b(\langle C, R\rangle, \mathcal{O}, A, n, z)$ denote the output of the following probabilistic experiment: on common input 1^n and auxiliary input z, the PPT adversary $A^{\mathcal{O}}$ (adaptively) chooses a pair of challenge values $(v_0, v_1) \in \{0,1\}^n$ and a tag $\mathsf{id} \in \{0,1\}^{l(n)}$, and receives a commitment to v_b using the tag id (note that A interacts with $\mathcal{O}$ throughout the experiment as described before); finally, when $A^{\mathcal{O}}$ halts, the experiment returns the output y of $A^{\mathcal{O}}$; y is replaced by $\perp$ if during the experiment, A sends $\mathcal{O}$ any commitment using the tag id.

Definition 1 (CCA-secure Commitments, [CLP10])**.** *Let $\langle C, R\rangle$ be a tag-based commitment scheme with tag-length $l(n)$, and $\mathcal{O}$ be a decommitment oracle for it. We say that $\langle C, R\rangle$ is* CCA-secure w.r.t. *$\mathcal{O}$, if for every* PPT ITM *A, every $z \in \{0,1\}^*$, and every sufficiently large n, it holds that:*

$$\mathsf{IND}_0(\langle C, R\rangle, \mathcal{O}, A, n, z) \stackrel{c}{\equiv} \mathsf{IND}_1(\langle C, R\rangle, \mathcal{O}, A, n, z).$$

We say that $\langle C, R\rangle$ is CCA-secure if there exists a decommitment oracle $\mathcal{O}'$ such that $\langle C, R\rangle$ is CCA-secure w.r.t. $\mathcal{O}'$.

An analogous version of the definition which considers many concurrent executions on left (instead of just one), is known to be equivalent to the current definition (via a simple hybrid argument). Also note that, for this reason, this definition implies concurrent non-malleable security for commitments [PR05a].

[8] Note that since $\langle C, R\rangle$ is efficiently checkable, and the session is accepting, such a valid decommitment always exists. In addition, note that since we only have statistical binding, this value is unique except with negligible probability.

3.2 Our Protocol for CCA Secure Commitments

We are now ready to present our CCA secure commitment protocol, denoted CCA-Com. The protocol employes the PRS preamble in *both* directions (similar to [GJO10, LPTV10]). However, our protocol is much simpler, and admits an easier security proof.

We now provide a quick overview of our protocol. Our protocol will also use a constant-round non-malleable commitment scheme—NMCom—that is robust w.r.t. 4 rounds. The formal description of our protocol appears in figure 3.

Let $v \in \{0,1\}^n$ be the value to be committed. Our commitment protocol, CCA-Com, consists of five phases. In the first phase, the committer C commits to v using a statistically-binding PRS preamble. We denote this instance of the preamble by PRS_1. In phase 2, the receiver R uses a statistically-*hiding* PRS preamble to commit to a random value $\sigma \in \{0,1\}^n$; we denote this instance of the PRS preamble by PRS_2.

In phase 3, C commits to 0^n using the *robust* NMCom scheme. In phase 4, R decommits to the value σ (of phase PRS_2). Finally, in phase 5, C uses a witness-indistinguishable (WI) *proof* system to prove that: "either there exists a valid value v in phase 1 or the value committed in NMCom is σ." It can use, for example, Blum's 3-round protocol repeated in parallel n times [Blu87]. We remark that it is necessary to use a proof system which ensures soundness against unbounded provers.

To decommit, C sends decommitments corresponding to the first phase, namely PRS_1. The round-complexity of both PRS preambles is $\ell \in \omega(\log n)$. We have the following theorem, whose proof appears in section 4.

Theorem 1. *Assuming the existence of collision-resistant hash functions, protocol* CCA-Com *presented in figure 3 is a $\widetilde{O}(\log n)$-round CCA secure commitment scheme for identities of length n.*

3.3 Construction based on One-way Functions

Protocol CCA-Com of the previous section requires the use of a statistically-hiding phase PRS_2. If we change the protocol so that PRS_2 is statistically-binding, we will not be able to prove that the right sessions of CCA-Com remain statistically binding. This is because of the presence of the super-polynomial time oracle $\mathcal{O}$. Indeed, without the oracle we can make PRS_2 statistically-binding and have a protocol based on one-way functions; but in the presence of oracle, it may happen that A^* is able to ensure that for some right session s, $\widetilde{u}_s = \widetilde{\sigma}_s$.

To avoid this problem, we must somehow remove PRS_2 yet still be able to later modify the left PRS slot-by-slot. The key observation is that in case of commitments, there is only one left session. Therefore, we can use a different simulation strategy which guarantees "soundness" w.r.t. unbounded committers as well.

Based on this observation, our one-way functions based protocol does not use PRS_2. Instead it uses a CoinFlip protocol in which no unbounded prover can succeed in "setting up a trapdoor" but a rewinding party can. Our new protocol, denoted CCA-Com*, appears in figure 4.

Theorem 2. *Assuming the existence of one way functions, protocol* CCA-Com* *presented in figure 4 is a $\widetilde{O}(\log n)$-round CCA secure commitment scheme for identities of length n.*

For the proof of this theorem see appendix B. Although the proof is very similar to that of theorem 1, one crucial point is that we need to re-use the idea of robust-simulation, and be careful about how we apply it.

Protocol CCA-Com. The committing algorithm is denoted by C. The receiver algorithm is denoted by R. The common input is the security parameter n and an identity id of length n. The private input of C is the value $v \in \{0,1\}^n$ to be committed. The protocol proceeds in following five phases:

- *Phase 1:* C commits to v using a statistically-binding PRS preamble. This instance of the preamble is denoted by PRS_1, and let τ_1 be the commitment-transcript.
- *Phase 2:* R commits to random $\sigma \leftarrow \{0,1\}^n$ using a statistically-hiding PRS preamble. This instance of the preamble is denoted by PRS_2.
- *Phase 3:* C commits to the all-zero string 0^n using NMCom, with common identity id. Let τ_3 be the commitment-transcript.
- *Phase 4:* R decommits to σ, by sending the appropriate decommitment strings.
- *Phase 5:* C proves to R, using a public coin, constant round WI *proof* system (e.g., n parallel repetitions of Blum's protocol), that either:

  (a) $\exists\ v \in \{0,1\}^n$ s.t. τ_1 is a valid PRS-commitment to v; OR
  (b) τ_3 is a valid commitment to σ as per NMCom.

Decommitment oracle $\mathcal{O}$. The oracle extracts the value committed to in transcript τ_1 of the first phase, and returns it.

Figure 3: Protocol CCA-Com.

4 Proof of Security for CCA-Com

We start by noting that the scheme is statistically-binding even against an unbounded cheating committer algorithm C^*. This is because of the following. PRS_2 is statistically-hiding for σ, and therefore except with negligible probability, the value committed to in NMCom (which is statistically-binding) does not equal σ. Then, from the soundness of the WI *proof*, it follows that C^* can succeed only when the PRS_1 is consistent. It will be important to have statistical-binding w.r.t. unbounded $C*$ since we will be dealing with the super-polynomial time oracle $\mathcal{O}$ in various hybrid experiments.

We now proceed to demonstrate the CCA security of our scheme. To do so, we will directly construct a PPT simulator SIM, which simulates the view of any CCA adversary $A^{*\mathcal{O}}$. SIM will not have access to the decommitment oracle $\mathcal{O}$, and uses A^* only as a black-box. Recall that $\mathcal{O}$ extracts the value from the transcript of PRS_1. SIM only receives 1^n and auxiliary-input z for A^*, as its own inputs.

Lemma 2. *There exists a strict polynomial time machine* SIM *such that for every* PPT ITM *machine* A^**, every* $z \in \{0,1\}^*$*, every sufficiently large* n*, and every* $b \in \{0,1\}$*, it holds that:*

$$\mathsf{SIM}^A(1^n, z) \stackrel{c}{\equiv} \mathsf{IND}_b(\langle C, R\rangle, \mathcal{O}, A, n, z).$$

Proof.

Protocol $\mathsf{CCA\text{-}Com}^*$. The committing algorithm is denoted by C. The receiver algorithm is denoted by R. The common input is the security parameter n and an identity id of length n. The private input of C is the value $v \in \{0,1\}^n$ to be committed. Com is a statistically-binding scheme.

Round parameters. Let $q := q(n)$ be a fixed function in $\omega(1)$, and let $\ell := \ell(n) = q(n) \cdot \log n \cdot \omega(1)$. Let $k \in O(1)$ be the round complexity of NMCom.

- *Phase 1:* C commits to v using the PRS preamble with parameter $\ell \in \omega(\log n)$. This instance of the preamble is denoted by PRS_1, and let τ_1 be the commitment-transcript.
- *Phase 2:* C and R execute the following $\mathsf{CoinFlip}$ protocol. In round $i \in [q]$:
  (a) C commits a "short" string $u_i \in \{0,1\}^{\log n}$ using Com,
  Let c_i be the commitment-transcript, and d_i a decommitment-string.
  (b) R sends a "short" random string $\sigma_i \in \{0,1\}^{\log n}$
  Define $u = (u_1, \ldots, u_q)$, $\sigma = (\sigma_1, \ldots, \sigma_q)$, and $d = (d_1, \ldots, d_q)$.
- *Phase 3:* C commits to string (u, d) using NMCom andidentity id. Let τ_3 be the commitment-transcript.
- *Phase 4:* C proves to R, using a public coin, constant round WI *proof*—e.g., n parallel repetitions of Blum's protocol—that either:
  (a) $\exists\ v \in \{0,1\}^n$ s.t. τ_1 is a valid PRS-commitment to v; OR
  (b) τ_3 is a valid commitment to (u, d) as per NMCom, such that $\forall i \in [q]$:
  – (i) $\mathsf{open}_{\mathsf{Com}}(c_i, u_i, d_i) = 1$, *and*
  – (ii) $u_i = \sigma_i$.

Decommitment oracle $\mathcal{O}$. The oracle extracts the value committed to in transcript τ_1 of the first phase, and returns it.

Figure 4: Protocol $\mathsf{CCA\text{-}Com}^*$ based on one-way functions.

Recall that during its execution A^* opens one session of $\mathsf{CCA\text{-}Com} := \langle C, R \rangle$ on left, while simultaneously interacting in multiple concurrent sessions of $\mathsf{CCA\text{-}Com}$ with $\mathcal{O}$, called the right sessions. Let $T = T(n)$ be a polynomial upper-bounding the number of right sessions of A^*.

Algorithm SIM will use the robust simulator S guaranteed by the (general version of) the robust-extraction lemma (see sections 2 and A), allowing it to extract the values committed to by A^* in all PRS preambles. It then uses these values to simulate the answers of $\mathcal{O}$, as well as to succeed in WI proof (by committing the extracted value in NMCom of the left session).

SIM uses two helper procedures: the robust simulator S and an "interface" algorithm I to be described shortly. Procedure I essentially "decouples" the PRS preambles from the rest of the $\mathsf{CCA\text{-}Com}$ protocol. It incorporates A^* as a black-box, and handles all messages of all sessions of $\mathsf{CCA\text{-}Com}$ internally and honestly, *except* for all the PRS preambles in which A^* acts as the *committer*. All these preambles are forwarded to outside PRS-receivers. That is, I participates in a robust-concurrent attack, interacting with the party $\mathcal{E}$. It executes PRS preambles with $\mathcal{E}$, and at the end of each PRS preamble s, I expects to receive a value α_s from $\mathcal{E}$. This value will be used internally.

SIM is a polynomial time machine without access to any super-polynomial time helper. Therefore, to run I, it runs the robust-simulator S providing it black-box access to the "adversary" procedure I. SIM outputs whatever S outputs. Formal descriptions follow.

Algorithm $\mathsf{SIM}(1^n, z)$**.** Return the output of $S^I(1^n, z)$, where procedure I—which has black-box access to adversary A^*—is described below.

Procedure $I(1^n, z)$**.** Procedure I launches the robust-concurrent attack, by committing in several PRS sessions to external receivers denoted $R_1, \ldots, R_T$. At the end of each preamble, it expects to receive a string α_s. I incorporates the CCA adversary A^* internally, as a black-box. I initiates an execution of A^*, simulating various sessions of $\mathsf{CCA\text{-}Com}$ that A^* opens as follows.

1. If A^* starts a new session s of $\mathsf{CCA\text{-}Com}$ on *right*, I starts by initiating a new session of the statistically-binding PRS_1 with an external receiver R_s. Then, messages of phase-1 of s are relayed between A^* and R_s. I simulates all other phases of s internally by following the honest algorithm for each phase.

2. If A^* starts a new session s of $\mathsf{CCA\text{-}Com}$ on *left*, I initiates a new session of the statistically-*hiding* preamble to be used as PRS_2 of s with an external receiver R'_s. I completes various phases of s as enumerated below.

   (a) *Phase 1:* I commits to an all zero string to A^*.

   (b) *Phase 2:* I simply relays messages between A^* and R'_s.

   (c) *Phase 3:* I commits to value α_s using NMCom (instead of 0^n). Value α_s was received from outside at the end of PRS_2 of sessions s.

   (d) *Phase 4:* If A^* correctly opens the value in PRS_2 of session s, I checks that the opened value is equal to the "fake witness" α_s. If not, it outputs a special symbol $\mathsf{ExtractFail}$, and halts. Otherwise it continues the execution.

   (e) *Phase 5:* I uses α_s and the randomness used in phase 3 (NMCom) to complete the WI proof in phase 5.[9]

[9] Note that, technically, I can use the valid witness corresponding to the PRS_1 phase s. This is since it committed to a valid value, namely 0^n. However, we choose to use α_s so that SIM will in fact be a valid simulator even for our concurrent non-malleable zero-knowledge protocols.

3. *Oracle answers:* If A^* successfully finishes a session s of $\mathsf{CCA\text{-}Com}$ on right, I sends the (already extracted) value α_s to A^*.

When A^* halts, I outputs the view of A^*, and halts. $\square$

Proving Indistinguishability. We are now ready to prove our lemma. We will prove this by using a series of hybrid experiments. Our hybrid experiments will be designed my making step-by-step changes to how I communicates with A^* internally in various phases of the protocol. For $i \in [7]$, we denote by ν^i the output of hybrid H_i.

Hybrid H_0: This hybrid is identical the experiment $\mathsf{IND}_b(\langle C, R\rangle, \mathcal{O}, A, n, z)$. Recall that in this experiment, $A^{*\mathcal{O}}$ receives a commitment to value $v_b \in \{0,1\}^n$ from the honest committer C, while interacting with the oracle $\mathcal{O}$. The output of the experiment consists of the view of A^*, which is also the output of H_0.

Hybrid H_1: This hybrid is identical to H_0, except for the following differences:

(a) H_1 does not forward the right sessions to $\mathcal{O}$. Instead, it executes all right sessions on its own, by playing the honest receiver strategy R. Denote by R_s the instance of R executed in session s.

(b) When A^* successfully completes the session R_s, H_1 queries a different oracle $\mathcal{O}'$, which acts as follows. The query to $\mathcal{O}'$ consists of only part of the commitment-transcript PRS_1 that belongs for the value to be committed. All other messages (e.g., commitments of shares and slot-messages) are not part of the query. In particular, if Com is non-interactive, the query will be commitment of the value $\widetilde{v}_s$. If there is a unique value defined by the query, $\mathcal{O}'$ extracts that value, and returns it as the answer. In all other cases, it behaves exactly as $\mathcal{O}$.

Hybrid H_2: This hybrid is identical to H_1; we use it to set up some notation. Define a procedure I_2, which is identical to procedure I (defined above), except that it it executes (all internal phases of) the left session honestly and that it uses the oracle $\mathcal{O}'$ as in H_1. That is, it only forwards the PRS preambles received from A^* outside, but executes all other phases internally and honestly. Formally, I_2 is identical is identical to I except:

(a) I_2 receives the value v_b to be committed in left sessions of $\mathsf{CCA\text{-}Com}$, as an input. Furthermore, I_2 commits to value v_b in PRS_1 of the left session (instead of 0^n, step 2(a)).

(b) When a session s of PRS_1 ends, I_2 expects to receive a value α_s from outside. If s is a statistically-binding sessions, then I_2 gives α_s to A^*. If s is statistically-hiding, it *ignores* the value s—there is only one such session corresponding to the left session.

(c) I_2 uses the valid witness, v_b and the randomness of PRS_1 to complete its WI proof in the left session (step 2(e)). I_2 does not check the validity of the "fake witness" (step 2(d)).

Observe that I_2 is essentially launching a robust-concurrent attack. It does not perform any rewindings. Hybrid H_2 simply runs the procedure $I_2^{A^*}$, and simulates PRS-receivers for it exactly as H_1 does; furthermore, when a statistically-binding session s finishes, H_2 forwards its commitment to $\mathcal{O}'$ and returns the oracles answer, denoted α_s, to I. On the other hand, if the statistically-hiding session of PRS ends, H_2 sends a random string α_s (which, by construction, is ignored by I_2, see point (b)). The output of H_2 is the view of I_2 (which in turn is the view of A^*).

Hybrid H_3: For procedure I_2, let $\mathcal{E}$ be our online extractor as defined in section 2.3 (with addendum from section A to deal with statistically-hiding PRS as well). Recall that $\mathcal{E}$ is a super-polynomial time machine, which facilitates the robust-concurrent attack. In particular, it acts honest receivers in all PRS preambles, as well as provides extractions for each one of them when they finish.

On input $(1^n, v_b, z)$, hybrid H_3 starts an execution of $I_2(1^n, v_b, z)$, making it interact with party $\mathcal{E}$. The output of H_3 is the output of the robust-concurrent attack, which in turn consists of the view of I_2 (and hence A^*). Therefore, H_3 differs significantly from H_2: it does not run PRS receivers and does not have access to $\mathcal{O}'$, since these are automatically done by $\mathcal{E}$ for H_3.

From hereon, we will only be making changes to the interface procedure I_2. All future hybrids, except for the last one, differ from H_2 in only that they use a modified version of I_2. Furthermore, changes are made to the phases of the *left session* only, of which there is only one.

Hybrid H_4: This hybrid is identical to H_3 except that instead of using I_2, it uses procedure I_4. Let α_s be the value that I_2 receives from $\mathcal{E}$ at the end of statistically-hiding PRS_2 of the *left session*. Furthermore, let σ_s be the value opened by A^* which appears internally, in the *left* session of $\mathsf{CCA\text{-}Com}$ (simulated for A^* by I_2).

Procedure I_4 is identical to I_2 except that when A^* sends σ_s along with valid openings, I_4 tests that $\alpha_s = \sigma_s$; it aborts the entire simulation if this test fails, and outputs $\mathsf{BindingFail}$. Recall that I_2 simply ignores α_s.

Hybrid H_5: This hybrid is identical to H_4 except that instead of using I_4, it uses a modified procedure I_5. Procedure I_5 is identical to I_4 except that instead of committing to 0^n, it commits to α_s in protocol NMCom of the *left* session s.

Hybrid H_6: Identical to H_5 except that instead of using I_5, it uses a modified procedure I_6. Procedure I_6 is identical to I_5 except that it uses the"fake witness" in the WI proof of the left session s. Recall that the "fake witness" consists of the value α_s and the randomness used in NMCom.

Hybrid H_7: Identical to H_6 except that instead of using I_6, it uses a modified procedure I_7. Procedure I_7 is identical to I_6 except it does not receive the input v_b, and commits to 0^n in the first phase PRS_1 of the left session. Observe that I_7 is in fact identical to I.

Hybrid H_8: This hybrid differs from H_7 in two crucial places. First, it does not receive the value v_b as input. Therefore, its only inputs are $(1^n, z)$. In addition, it does not use the party $\mathcal{E}$. Instead, H_8 simply runs the robust simulator S with inputs $(1^n, z)$ and black-box I_7. It outputs whatever S outputs. Observe that H_8 is in fact our original simulator SIM, presented earlier. □

Notation. Let ν^0 denote the output of H_0. Let u^0 be the variable denoting the value committed by C in NMCom in the left session, and let σ^0 be the variable denoting PRS-value opened by A^* in (phase-4 of) the left session. Since there is only one left session, we will not use any subscripts. Analogously, define variables $\widetilde{u}^0_s$ and $\widetilde{\sigma}^0_s$ for the s-th right session in hybrid H_0. Finally, we denote by $\widetilde{v}^0_s$ the value A^* commits in PRS_1 of the s-th right session in H_0. For $i \in \{0, \ldots, 8\}$ define values $\nu^i, \widetilde{u}^i_s, \widetilde{\sigma}^i_s$ and $\widetilde{v}^i_s$ w.r.t. the hybrid H_i analogously. Identity of the left session is referred to by id, and that of the s-th right sessions by $\widetilde{\mathsf{id}}_s$ without mention of the hybrid. Further, unless specified

otherwise, index s is in $[T]$, and $\widetilde{\mathsf{id}}_s$ is not equal to id, and the probability is taken over the randomness of the hybrid in consideration.

We start by noting that ν^0 is identical to $\mathsf{IND}_b(\langle C, R\rangle, \mathcal{O}, A, n, z)$. Next, we claim the following.

Claim 1. *For every hybrid $i \in \{0, \ldots, 4\}$ and for every right session $s \in [T]$,*

$$\nu^0 \stackrel{\text{s}}{\equiv} \nu^1 \equiv \nu^2 \stackrel{\text{s}}{\equiv} \nu^3 \stackrel{\text{s}}{\equiv} \nu^4 \tag{1}$$

$$\Pr\left[\widetilde{u}_s^i = \widetilde{\sigma}_s^i\right] \leq \text{negl}(n) \tag{2}$$

Proof. It is seen by way of construction of hybrids H_1 and H_2, that $\nu^0 \stackrel{\text{s}}{\equiv} \nu^1 \equiv \nu^2$.[10] In case of hybrid H_3 party $\mathcal{E}$, which also simulates PRS sessions exactly as H_2 does (since $\mathcal{E}$ is valid). Furthermore, $\mathcal{E}$ extracts the values α_s in the (first message) of the PRS preamble exactly as $\mathcal{O}'$ does in H_2. If a unique and valid α_s exists, the value α_s returned to I_2 is the same in both hybrids. However, when the value is not unique, $\mathcal{E}$ uses a different decision procedure to decide the value of α_s. Nevertheless, statistical-binding of PRS ensures that this happens with only negligible probability. It follows that the distribution of answers α_s in both hybrids is statistically close, and hence $\nu^2 \stackrel{\text{s}}{\equiv} \nu^3$.

Finally, in H_4 the only difference is that I_4 verifies that $\alpha_s = \sigma_s$ (i.e., the fake witness is correct). From the validity constraint 1(b) on $\mathcal{E}$ (see lemma 6, section A) this condition fails with only negligible probability. Therefore, $\nu^3 \stackrel{\text{s}}{\equiv} \nu^4$.

We first prove the second equation for hybrid H_0. Fix any right-session s of H_0. Observe that PRS_2 of s is statistically-hiding. Therefore, except with negligible probability, we have that value $\widetilde{\sigma}_s^0$ is not defined until after the completion of NMCom of session s. Since NMCom is statistically-binding, and there are exponentially many possible values for $\widetilde{\sigma}_s^0$, the claim follows (for hybrid H_0). Now, observe that the same argument applies for hybrids H_1 and H_2 as well. In case of H_3 and H_4, since $\mathcal{E}$ simulates PRS receivers honestly, the same argument applies to these hybrids as well. □

A corollary of the second equation is that in each of these hybrids, for every right session s that is accepting, there exists a *unique* and *valid*[11] value $\widetilde{v}_s^i$ to which A^* is committed to (except with negligible probability). This is because if the second equation holds, from the soundness of WI proof against unbounded adversaries,[12] phase PRS_1 must be consistent defining a unique and valid value. Therefore, in all future hybrids, we will continue to maintain the second equation as an invariant.

Claim 2. *We have that, $\nu^4 \stackrel{\text{c}}{\equiv} \nu^5$; and $\forall s$: $\Pr\left[\widetilde{u}_s^5 = \widetilde{\sigma}_s^5\right] \leq \text{negl}(n)$.*

Proof. Both hybrids H_4 and H_5 use party $\mathcal{E}$ which is super-polynomial time. Therefore, to prove the claim, we need to first need to eliminate the use of $\mathcal{E}$. Observe that this can be done by employing the robust simulator S. However, since we aim to reduce the claim to the non-malleability of NMCom, we would like one left execution and one right execution of NMCom that the robust-simulator S does not rewind.

Recall that the only difference between H_4 and H_5 is that they use different procedures: I_4 and I_5 respectively. The only difference between I_4 and I_5 is that the first one commits to $x = 0^n$

[10] First two hybrids are not identical since the oracle changes: $\mathcal{O}$ extracts from the full PRS transcript (and therefore always checks for consistency), whereas $\mathcal{O}'$ simply extracts from the (first) committing message of PRS. Statistical-binding ensures that this difference happens only in negligible cases.

[11] Recall that v is valid if all shares XOR to v; formally, there exists randomness ρ such that $\mathsf{open}_{\mathsf{PRS}}(\tau_1, v_s, \rho)$=1 where τ_1 is the commitment-transcript of PRS_1 of s.

[12] We need this since A^* does have access to super-polynomial computations via $\mathcal{O}$, and we do not know what he might be learning.

whereas the second one commits to $x = \alpha_s$ in phase NMCom of the left session. Recall that α_s is the value sent by $\mathcal{E}$ at the conclusion of PRS_2 of the left session.

We design an intermediate procedure I_*, which is identical to I_4 except for the following differences:

1. When the NMCom phase of the left session is about to begin, I_* sends $(0^n, \alpha_s)$ to an external committer.

2. I_* does not execute the NMCom of the left session internally. Instead, it expects to receive it from an external honest committer, denoted C_{NM}, who either commits to 0^n or α_s.

3. For a randomly chosen right session j, I_* does not execute the the NMCom of session s internally. Instead, it forwards the messages of this NMCom to an external honest receiver, denoted R^j_{NM}.

Therefore, I_* is executed several PRS preambles, and at the same time acting as a man-in-the-middle for protocol NMCom by receiving a commitment and making a commitment at the same time. Let B denote the party who runs algorithm C_{NM} as well as R^j_{NM} for I_*, both *honestly*. Observe that the number of rounds of interaction between B and I_* are $2k$ where k is the round-complexity of NMCom. The input x to B consists of the value to be committed by C_{NM}.

Viewed this way, I_* is an adversary who launches the robust-concurrent attack with respect to party B, and the party $\mathcal{E}$. Furthermore, if B commits to $x = 0^n$ then the execution is identical to that of H_4 with I_4; on the other hand, if $x = \alpha_s$, the execution is identical to that of H_5 with procedure I_5. That is,

$$\begin{aligned} \nu^4 &\equiv \mathsf{REAL}^{I_*}_{\mathcal{E},\Pi}(n, 0^n, z), \\ \nu^5 &\equiv \mathsf{REAL}^{I_*}_{\mathcal{E},\Pi}(n, \alpha_s, z). \end{aligned}$$

where protocol $\Pi = (B, P_2)$, and P_2 is "converse of B—that is P_2 acts as a receiver in one NMCom in which B is acting as a committer and vice-versa in the other.

Now, we can remove $\mathcal{E}$ and instead use the robust-simulator S to sample statistically close views. That is, suppose that we run S with I_* and consider the output $\mathsf{OUT}_{\text{S}}(x) := \mathsf{OUT}_{\text{S}}\left[B(1^n, 0^n) \leftrightarrow S^{I_*}(1^n, z)\right]$. By applying the robust concurrent extraction lemma, we have that statistical distance between $\mathsf{OUT}_{\text{S}}(x)$ and $\mathsf{REAL}^{I_*}_{\mathcal{E},\Pi}(n, x, z)$ is at most:

$$\Delta(n) \leq 2^{-\Omega(\ell - 2k \cdot \log T)} \leq \text{negl}(n),$$

for every $x \in \{0^n, \alpha_s\}$ since $\ell \in \omega(\log n)$, k is a constant, and T is at most a polynomial. Using this with the equations above, we get:

$$\nu^4 \overset{\text{s}}{\equiv} \mathsf{OUT}_{\text{S}}(0^n) \tag{3}$$

$$\nu^5 \overset{\text{s}}{\equiv} \mathsf{OUT}_{\text{S}}(\alpha_s) \tag{4}$$

By construction, $\mathsf{OUT}_{\text{S}}(x)$ is the output of S^{I_*} in an interaction with B on input x. Algorithm S^{I_*} is a PPT man-in-the-middle adversary for NMCom who receives a commitment to x from C_{NM}, and commits a value, say $\widetilde{u}_j(x)$, to R^j_{NM}. From non-malleability of NMCom w.r.t. to itself, we have that:

$$(\widetilde{u}_j(0^n), \mathsf{OUT}_{\text{S}}(0^n)) \overset{\text{c}}{\equiv} (\widetilde{u}_j(\alpha_s), \mathsf{OUT}_{\text{S}}(\alpha_s)) \tag{5}$$

It immediately follows from (3), (4), and (5), that $\nu^4 \stackrel{c}{\equiv} \nu^5$. Furthermore, suppose that the other part of the claim is false, so that for some session $s \in [T]$ value $\widetilde{u}_s^5 = \widetilde{\sigma}_s^5$ with noticeable probability p. Define this to be event bad_5 for hybrid H_5, and analogously define bad_4 for H_4.

Now observe that since j was chosen uniformly from T sessions, $j = s$ with probability $1/T$. Therefore, value $\widetilde{u}_s^5$ appears as the variable $\widetilde{u}_j(\alpha_s)$ with probability $1/T$. In addition, value $\widetilde{\sigma}_j(\alpha_s)$ is a part of $\mathsf{OUT}_{\mathrm{S}}(\alpha_s)$, and therefore, event bad_5 is efficiently observable given both $(\widetilde{u}_j(\alpha_s), \mathsf{OUT}_{\mathrm{S}}(\alpha_s))$; and it occurs with probability p/T which is noticeable. By an analogous argument, event bad_4 is also efficiently observable; furthermore bad_4 must occur with noticeable probability as well due to equation (5). This contradicts equation (2). Hence the claim. □

Claim 3. *We have that,* $\nu^5 \stackrel{c}{\equiv} \nu^6$*; and* $\forall s: \ \Pr\left[\widetilde{u}_s^6 = \widetilde{\sigma}_s^6\right] \leq \mathrm{negl}(n)$.

Proof. The proof of this claim is almost identical to the proof of claim 2. The only difference is that instead of using non-malleability w.r.t. itself property of NMCom, we use the fact that NMCom is robust w.r.t. every *interesting* 3-round protocol (i.e., one that "hides" its input, see section C). Since the only difference between H_5 and H_6 is in the WI part, the proof follows; we omit the details. □

Claim 4. *We have that,* $\nu^6 \stackrel{c}{\equiv} \nu^7$*; and* $\forall s: \ \Pr\left[\widetilde{u}_s^7 = \widetilde{\sigma}_s^7\right] \leq \mathrm{negl}(n)$.

Proof. Observe that in H_6, since the "fake witness" is being used in the WI part, the PRS_1 phase of the left session (which commits to input value v_b) does not have to be consistent. Therefore, we proceed as follows:

1. Design $\ell + 1$ intermediate hybrids $H_{6:i}$ for $i = \{0, \ldots, \ell - 1\}$ where $H_{6:0} = H_6$, $H_{6:\ell} = H_7$.
2. Hybrid $H_{6:i}$ is the same as $H_{6:i+1}$ except that in slot-i of PRS_1 phase of the left session, $H_{6:i+1}$ commits to shares of an all-zero string.
3. Now, following the proof of claim 4 and by using the robustness of NMCom w.r.t. the 3-round protocols, we conclude that $\nu^{6:i} \stackrel{c}{\equiv} \nu^{6:i+1}$ and that $\forall s: \ \Pr\left[\widetilde{u}_s^{6:i} = \widetilde{\sigma}_s^{6:i+1}\right] \leq \mathrm{negl}(n)$, where variables are analogously defined. The details of this part are repetitive, and omitted.

The claim now follows. □

Completing the proof. Note that the output of H_8, is statistically-close to ν^7 due to the robust-concurrent-extraction lemma. Therefore, combining all the equations we have that $\nu^0 \stackrel{c}{\equiv} \nu^8$. Since H_8 is identical tot he simulator, the lemma follows. ■

5 Proof of the Robust Extraction Lemma

We organize this section in two parts. The first part introduces important terminology used throughout the proof. The terminology is mostly consistent with previous literature [KP01, PRS02, Ros04, PTV08], and can be skipped by experts at first reading—except, for the notion of *breakpoints* (paragraph 2) and *unmarked view* of a block. In the second part, we present the actual proof of the lemma.

5.1 Terminology

We now introduce some terminology about the recursive structure of recurse. An execution of recurse is called a *block*. The six inputs of a block B, namely $(t, \mathsf{st}, \mathcal{T}, \mathsf{f}, \mathsf{aux}, \mathsf{id})$ are referred to as follows. Input t is called the *length*, st the *starting state*, and $\mathcal{T}$ the *starting solutions*. Input f is used as an identifier whose only purpose is to formally identify if B "lies on the main thread" (defined later); it does if $\mathsf{f} = 1$. Input aux is called the *auxiliary table* (or auxiliary solutions), used only in special cases. Finally, id is the *identity* of the block B, used to uniquely specify B in the full recursive structure of recurse.

Blocks of length $t = 1$ are called the *atomic* blocks. The output $(\mathsf{st}', \mathcal{T}')$ of a block consists of the *final* (or output) state and the *final* (or output) solutions. Each message of protocol Π is called a *breakpoint*, often dented by a cross mark $\times$. If B is an atomic block, and a message of Π is scheduled during its execution, we say that B contains a breakpoint; B may contain one or several breakpoints.

Let B be a block that is not atomic. B executes four blocks during its own execution, in the following order: C', C, D' and D. Blocks C', D' are called the *look-ahead blocks*; and C, D the *main blocks* (see steps 2(a) to 2(d) of procedure recurse). Furthermore, blocks C and C' (resp., D and D') are called *siblings*; they are the *children* of the same *parent* block B. We say that B *contains* its children blocks.

We say that blocks B_1 and B_2 are *consecutive*, denoted $B_1 \to B_2$, if the final state of B_1 is the starting state of B_2. We say that B_1 and B_2 are *connected* if $B_1 \to B_2$ or $B_2 \to B_1$. A thread h is represented by a sequence of consecutive blocks $(B_1, \ldots, B_m)$, i.e., $B_i \to B_{i+1}$ for every $i \in \{1, \ldots, m-1\}$. Further, if it holds that for every $i \in [m]$, block B_i in the sequence is an *atomic* block, then we say that the sequence is the *primitive sequence* of h. Note that a primitive sequence *uniquely* identifies a thread. That is, a thread h can be uniquely referred to by specifying the *identity* of each atomic block B_i in the sequence. Note that identities, by construction, are indeed unique for every block, including blocks that are not atomic. We say that threads h_1 and h_2 are the *same* if their primitive sequences are equal. We say that a thread h is an internal thread of a block B if every block B_i in the primitive sequence of h is contained in B. If B is not an atomic block, it contains many internal threads. The *base thread* h of a block B is an internal thread of B whose primitive sequence consists of only *main blocks*. That is every atomic block appearing in the primitive sequence of the base-thread is actually a *main* block of its parent block. By construction, the base thread of a block B is unique. Finally, we say that a block B *lies* on a thread h if there exists a sequence h' of blocks containing B such that the primitive sequence of both h and h is identical.

We note that the terminology so far is exclusively about the recursive structure of recurse. In particular, it allows us to refer to various blocks and threads without considering any specific execution with any given random tape or inputs. Fixing the length t of a block, completely fixes the flow chart of all recursive calls inside that block and hence its internal structure.

Main Thread: A thread h is called the *main thread*, if every sequence of blocks $\{B_i\}_i$ specifying h, is a *main block*. Equivalently, h is a main thread if every (atomic) block in the primitive sequence of h has receives the flag $\mathsf{f} = 1$. Observe that, by construction, there exists a *unique* main thread. All other threads are called *look-ahead* threads.

Recall that the *original* state of A^* is denoted by st_0, and consists of its auxiliary input z and its randomness. Also recall that the execution of A^* proceeds in rounds, where in round i of the interaction, advertsary A^*—who is participating in many PRS preambles as well as one session of

Π—sends its next message and immediately receives a response. The state of A^* at the beginning of the round i, consists of its original state st_0, and all messages that it sends or receives. This terminology allows us to refer to the state of A^* even within an atomic block B.

We say that a state st of A^* lies on a thread h if there exists an atomic block B_i in the primitive sequence of h such that the state of A^* is st at some point during the execution of B_i. Let st_1 and st_2 be two states of A^* that lie on h such that st_2 appears before st_1; then we denote by $\mathsf{st}_1 \to \mathsf{st}_2$, all messages that lie on h between st_1 and st_2, and call it a *segment* of a *part* of the thread h. We say that a block B *contains* a message v if v appears on an internal thread of B. A session s is said to lie on a thread h if all messages of s appear on h. Finally, we say that a block B contains the j-th slot of session s if B contains both (r_j, β_j), where r_j is the receiver-challenge in round j of s, and β_j the corresponding openings sent by A^*. Recall that (r_j, β_j) is a convincing slot if β_j consists of valid openings of commitments chosen by r_j.

Unmarked view/execution of a block. Consider all internal threads of a block B, which contain a breakpoint $\times$ (i.e., a message of Π). By construction, if a $\times$ appears on a look-ahead thread, the execution of the look-ahead thread stops as soon as $\times$ occurs—that is, there are no further messages scheduled after $\times$. However, if $\times$ occurs on the base-thread of B, *and* the base-thread is a part of the main-thread, the execution does not stop after $\times$, and continues. The unmarked view (or execution) of B consists of executing the part of each internal-thread h of B executed *only* up to the point where a $\times$ occurs on h. Therefore, if base-thread of B contains a $\times$ then the unmarked-execution will continue the base thread after the $\times$. We note that all threads and breakpoints $\times$ considered are internal to B, and that any breakpoint not contained in B does not affect unmarked view of B.

Now, we say that a message v is contained in the unmarked view of B if v appears on an internal thread of B in the unmarked execution of B. This means that v appears on internal thread of B before an breakpoint $\times$ appears on that thread. We define the term "slot-j of a session appears in the unmarked view of B" analogously.

5.2 Proof of Successful Simulation

Let us fix a *target* thread h and a *target* session s that lies on h. Formally, define a random tape ρ of the simulator to be *bad* if all slots of session s that lie on h are convincing, whereas all slots of s that lie on look-ahead threads are not convincing. We now define notion of *robust blocks*, and an ordering relation (adapted from [PRS02, Ros04, PTV08]).

Definition 2 (Robust Block)**.** *Let ρ be a fixed random tape. Let h be an arbitrary thread, and B an arbitrary block, all defined by the execution of* recurse *with random tape ρ. Further, let s be an arbitrary session of the* PRS *preamble that lies on h. We say that B is a* robust *block with respect to h and s if it satisfies the following conditions:*

(a) (Main block:) B lies on h and does not contain the START *message of session s.*

(b) (Full slot:) B contains a convincing slot of s (not necessarily on h) in the unmarked execution*. That is, a convincing slot appears on an internal thread of B, prior to any breakpoint $\times$ on that thread.*

(c) (Good sibling:) B' does not contain an END *of session s.*

(d) (Blocked END*:) B does not contain any* END *of s in the* unmarked view*.*

(e) (Locatable:) Every slot of s that lies on h and appears before B is convincing, and all other threads (that complete) before B, do not contain any convincing slot of s.

Let us first establish the following claim:

Claim 5. *If B is a robust block, then it is a main block (of its parent).*

PROOF. Suppose that B is not a main block. Then it is a look-ahead block, and therefore, there does not exist any block C for which $B \to C$. Since s completes on h, h contains the END of s, denoted END_s. Since END_s occurs after all slots of s, and B contains a full slot of s, we have that END_s cannot occur before B. Further, since there is no C such that $B \to C$, END_s cannot occur after B either. Therefore, END_s must occur in B. Consider the internal thread of B on which END_s occurs, say h'. By requirement (d), END_s cannot occur in the unmarked view of B. However, since B is a look-ahead block, by construction, the unmarked view is the entire view of B, and hence B cannot contain END_s. This is a contradiction. □

Recall that a block B contains all its children blocks, as well as all blocks that are contained by its children. If a block B_1 contains another block B_2, then B_2 does not contain B_1. We say that B_1, B_2 are *disjoint*, if B_1 does not contain B_2 and vice versa.

The following order relation will provide a correct order to swap blocks (unchanged from [PTV08]).

Definition 3. *Let C and B be two blocks on a common thread. We write $C > B$ iff*

- *C and B are disjoint, and C occurs before B, or*
- *C and B are* not *disjoint, and C is a larger block that contains B.*

Since these are the only two possible cases, relation $>$ defines a total order on any set of blocks that lie on the *same* thread. Now, let us formally define the *swap* operation. The swap operation takes as input a random tape ρ, and the identity of a block B. Let B' be the sibling of B. The, the swap operation exchanges part of ρ used to executed B with that used to execute B', and outputs the resulting tape.

Finally we define the undo function on any given random tape ρ'. The undo function is defined with respect to the target thread h and the target session s. Since we are dealing with an external party of protocol Π as well, we define undo also with respect to a fixed input x and randomness γ of the external party. The output of $\mathsf{undo}(\rho') := \mathsf{undo}_{x,\gamma,h,s}(\rho')$ is defined as follows:

1. Execute the simulation using recurse, with random tape ρ' and (x, γ) as (input, randomness) of the external party of Π. Call a block special if it *does not* lie on the target thread and contains a convincing slot of the target session. That is the block lies on a look ahead thread with a convincing slot of s.

2. Locate the first special block, D, after the START; that is, any other special block E after START satisfies $D > E$.

3. Swap the parts of ρ' used by D and its sibling, say D'; output the new random tape.

Lemma 3. *Let ρ be a random tape (not necessarily bad). Let B be a* robust *block with sibling B', with respect to h and s, when* recurse *is executed with random tape ρ. Further, let ρ' be the random tape obtained after swapping the parts of ρ used by blocks B and B'. Then,*

1. (Goodness): ρ' is a good *random tape with respect to h and s.*

2. *(Robustness): If C is a robust block on ρ and $C > B$, then C is also robust on ρ'.*

3. *(Reversibility):* $\mathsf{undo}(\rho') = \rho$.

Proof.

Note that B and B' are not necessarily symmetric: inputs f, aux differ for them. Therefore, the swap may change the internal execution of these blocks. The key point is that the unmarked view of B after the swap "does not shrink."

Goodness. Let us first establish the goodness property. Let st be the common starting-state and $\mathcal{T}$ be the common solutions table of blocks B and B'. Let aux and $\mathcal{T}_1$ be the auxiliary- and the output-tables of B', respectively. By construction, the auxiliary table of B is $\mathsf{aux}_2 = (\mathsf{aux}, \mathsf{aux}_{B'})$ where $\mathsf{aux}_{B'} = \mathcal{T}_1 \setminus \mathcal{T}$. The following claim says that $\mathsf{aux}_{B'}$ is never used to solve a session that lies in the unmarked view of B':

Claim 6. *Let u be a session, and id an atomic block. If $\mathsf{extract}(u, \mathsf{id})$ reads table $\mathsf{aux}_{B'}$, then B contains END_u; but it contains no END_u in the unmarked view of B. That is , a breakpoint $\times$ occurs between START_u and END_u.*

Proof. Consider an arbitrary main block id_1 with sibling id'_1. By construction, $\mathsf{extract}(u, \mathsf{id})$ reads $\mathsf{aux}_{\mathsf{id}'_1}$ if and only if all of the following conditions hold: (a) id reaches END_u, (b) id_1 contains id (and therefore END_u), and (c) $\times_{u,\mathsf{id}_1} = \mathsf{true}$ (meaning that a $\times$ occurs in block id_1 on the thread containing END_u). Conditions (a) and (b) prove the first part of the claim.

Consider the requirement (c). By construction, $\times_{u,\mathsf{id}_1} = \mathsf{true}$ if and only if execution in block id_1 reaches a $\times$ (i.e., a message of protocol Π is scheduled) while the session u is still "live"–i.e., START_u has occurred, but END_u has not yet occurred on the current thread. Therefore, if $\mathsf{extract}(u, \mathsf{id})$ reads $\mathsf{aux}_{\mathsf{id}'_1}$ then block id_1 contains a $\times$ between START_u and END_u.

By this argument, if $\mathsf{extract}(u, \mathsf{id})$ reads $\mathsf{aux}_{B'}$, we have that block B contains a $\times$ between START_u and END_u. This proves the second part of the claim. □

Claim 7. *The* unmarked *views of B and B' do not change after the swap.*

Proof. Let us first consider the block B. Before the swap, B receives $\mathsf{aux}_{B'}$ as (part of) its auxiliary input table, whereas after the swap, it does not. The key observation is that the execution of the unmarked view of B does not read the table $\mathsf{aux}_{B'}$. Indeed, suppose on the contrary, that it does. Then, there must exist a session u and an atomic block id such that $\mathsf{extract}(u, \mathsf{id})$ reads the table $\mathsf{aux}_{B'}$. From claim 6, there must occur a $\times$ before END_u occurs. Therefore, the call to $\mathsf{extract}(u, \mathsf{id})$ occurs only *after* the $\times$ mark. By definition, this call will not occur during the unmarked view/execution of B. Therefore, we have that the unmarked view of B never reads the table $\mathsf{aux}_{B'}$. Since $\mathsf{aux}_{B'}$ is the only missing part in B's input, it follows that the claim holds for B.[13]

Now consider the block B'. After the swap, B' becomes a main block. If B' does not lie on the *main thread*, all threads will still stop as soon as the first $\times$ mark occurs; therefore, by claim 6, the table aux_B (defined analogously to $\mathsf{aux}_{B'}$ in the swapped tape ρ') will not be accessed. On the other hand, if B lies on the main thread, some of its internal threads may continue the execution past their first (or even later) $\times$ marks. However, even in this case, by claim 6, executions of these threads prior to the first $\times$ do not use table aux_B, and therefore remain unchanged. The claim follows for B' as well. □

[13] Although input $\mathsf{f} = 0$ after the swap, this in fact only enforces that only the unmarked view of B executes.

Claim 8. *ρ' is a good tape.*

PROOF. Since B is a robust block, by definition, it contains a convincing slot of s and does not contain END_s in the *unmarked view* with tape ρ'. From claim 7, it follows that even after the swap, B will continue to have a convincing slot of s and no END_s.

Further, before the swap, B' does not contain END_s. Observe that since B' is a look-ahead block (see claim 5), this view is also it's unmarked view. Therefore, the unmarked view of B' does not contain END_s. After the swap, B' lies on the target thread h. We have two cases:

1. If B' does not lie on the *main* thread, the execution of B' does not change (since $\mathsf{f} = 0$). Since B' does not contain END_s, if END_s ever occurs on the target thread h, it will occur after both B and B', and therefore contain a slot of s that is opened twice. Hence ρ' is a good tape.

2. If B' lies on the main thread, some of the internal threads of B' may continue the execution past a $\times$ mark. First note that by claim 7, END_s still does not occur in the *unmarked* view of B'. Therefore, if a END_s occurs in the full view of B' on a thread h' which contains a $\times$ before END_s. When this happens, $\mathsf{extract}$ will access the table aux_B and find a convincing slot of s (since B contains a convincing slot and stores it in aux_B). Further, since END_s was reached in the full view B', this slot must have been opened again. Hence ρ' is a good tape. □

Robustness. Let C be a robust block on tape ρ such that $C > B$. Let C' be the sibling of C.

CASE 1: (C occurs before B). In this case, swapping of B and B' does not change the execution of either C or C'. It also does not change the execution between START and C, leaving all requirements of a robust block intact on ρ'.

CASE 2: (C contains B). In this case, swapping of B and B' does not affect the execution of C', and the execution between START of s and before C. Therefore the main block, good sibling, and locatable requirements still hold for C in ρ'. We have two cases.

The first case is when the the unmarked view of B is *not* a part of the unmarked view of C. Note that this happens when C contains a $\times$ on h before B. In this case, swapping of B and B' does not change the unmarked view of C, and therefore keeps the remaining requirements of a robust block–namely, full slot, good marking, and blocked END–intact on ρ'.

The other case is when the unmarked view of B is a part of the unmarked view of C. This happens when C does not contain any $\times$ mark on h before B. After the swap, B does not line on h, but instead becomes the look-ahead thread. However, even after the swap, B continues to be contained in C; further, the unmarked view of B also continues to be the part of unmarked view of C.[14] Since the unmarked view of B satisfies the remaining requirements of a robust block–namely, full slot, good marking, and blocked END, it follows that they hold for C as well on the tape ρ'.

Reversibility. Consider the execution of $\mathsf{undo}(\rho')$. undo starts by locating the first special block D. That is, D contains a convincing slot of s, does not lie on h, and for every other special block E, D either occurs before E or D contains E. After locating D, undo swaps the randomness used by D and its sibling D'.

[14]Note that the overall unmarked view of C in this case may have changed; in particular, this happens when before the swap: (a) B connects to some block D, (b) B contains a $\times$ on its base thread, *but* (c) B' does not contain a $\times$ on its base thread. Thus, D is not a part of the unmarked view of C on tape ρ. However, after the swap, D will be a part of the unmarked view of C.

We start by noting that on tape ρ', B does not lie on h; further as argued before, B contains a convincing slot of s. Therefore B is special on ρ'. Further observe that every block that occurs before cannot have a convincing slot of s due to the locatable requirement. Also, any block E that contains B must also contains B'; since B' which lies on h in tape ρ', so does E. Therefore E cannot be special. It follows that B is always the first special block on ρ', and undo swaps B with B'. This proves that $\mathsf{undo}(\rho') = \rho$. ■

The following claim shows that robust blocks can be swapped arbitrarily to generate many good tapes for each bad tape.

Lemma 4. *Let ρ be a bad random tape, and $\mathcal{R} = \{B_1, \ldots, B_m\}$ a set of robust blocks for ρ. Then, there exists a set of good tapes, denoted $\mathcal{G}(\rho, \mathcal{R})$ such that:*

1. $|\mathcal{G}(\rho, \mathcal{R})| \geq 2^m - 1$.
2. *For every bad random tape $\rho' \neq \rho$, and every set of robust blocks $\mathcal{R}'$ for ρ':* $\mathcal{G}(\rho, \mathcal{R}) \cap \mathcal{G}(\rho', \mathcal{R}') = \emptyset$.

Proof. By definition, all blocks lie on the same thread, and therefore have a total ordering. Without loss of generality, let $B_1, \ldots, B_m$. Choose any non-empty subset R of blocks in $\mathcal{R}$ and apply the swap operation on each one of them in the reverse order of their actual ordering (defined by relation $>$). Denote by ρ_R the resulting tape. From lemma 3, each swap leaves the first block in R still robust, and swapping the last block gives a good tape. Therefore, ρ_R is a good random tape. Since there are $2^m - 1$ non-empty subsets of $\mathcal{R}$, the first part of the claim follows.

To prove the second claim, assume that $\gamma \in \mathcal{G}(\rho, \mathcal{R}) \cup \mathcal{G}(\rho', \mathcal{R}')$. Since $\gamma \in \mathcal{G}(\rho, \mathcal{R})$ by reversibility of swap, applying undo repeatedly on γ until a bad tape is obtained returns ρ. Likewise, since $\gamma \in \mathcal{G}(\rho', \mathcal{R}')$, the same process returns ρ'. But since undo is a deterministic function, this is a contradiction. □

We say that the simulator S gets *stuck* if it outputs $\mathsf{ExtractFail}$. Further, we say that S gets *stuck on h in session s* if it outputs $\mathsf{ExtractFail}$ when sessions s ends on h—i.e., $\textsc{end}_s$ is scheduled by A^*. We now bound the probability of getting stuck.

Lemma 5. *Simulator S gets stuck with probability at most $2^{-\Omega(\ell - k \cdot \log T)}$.*

Proof.

We first prove the following claim.

Claim 9. *For every bad random tape ρ, the number of robust blocks is at least $\ell - (k+2) \cdot \log T$.*

Proof. Let us define the notion of a *minimal* block. We say that a block B is minimal for slot-j of session s, if and only if: B contains (the convincing) slot-j, but none of its children blocks contain slot-j. It is easy to see that for every session s there are ℓ minimal blocks. This is because each slot has at least one minimal block; furthermore, a block can be minimal for exactly one slot, since otherwise, it would have to contain two slots of s and at least one of them will be contained in one of its children blocks (by construction).

Next, each of these minimal blocks lies on h. Further, they all have a good sibling (requirement (c) in definition 2). This is because ρ is a bad tape and therefore it does not contain a convincing slot on any other look-ahead threads. Now, for every $\textsc{start}$, $\textsc{end}$, and $\times$ there are at most $\log T$ blocks that contain it. Such blocks cannot be robust, by definition, and there is a total of $(k+2) \cdot \log T$

of them. All the other remaining minimal blocks, satisfy the remaining requirements of definition 2, which is a total of $\ell - (k+2) \cdot \log T$. $\square$

By construction, the simulator can output ExtractFail if there exists a session s and thread h such that its random tape ρ is bad w.r.t. (h, s). It cal also output ExtractFail if in fact the tape is not bad, but the convincing slots were opened for the same PRS-challenge. However, the later only happens with probability $2^{-n} \cdot T^2 \cdot \text{poly}(n)$ throughout the simulation. Further, the former happens with probability $2^{-\Omega(\ell - k \cdot \log T)}$, and since there are only $T^2 \cdot \text{poly}(n)$ threads and sessions total, the claim follows form the union bound. ■

Completing the proof of the robust concurrent-extraction. It is easily seen that if a particular PRS preamble is contains a valid value, it must be consistent. However, whenever it is consistent, the simulator indeed extracts the correct value unless either it gets stuck or the commitment is not binding. Since both of these events occur with negligible probability, the validity constraint follows. The second part of the lemma follows from the fact that the simulator behaves exactly like $\mathcal{E}$, and that when PRS-sessions are consistent, the provided values are the same except with negligible probability; further, where they are not consistent, both S and $\mathcal{E}$ provides identically distributed values, to A^*. This completes the proof.

References

[AGJ+12] Shweta Agrawal, Vipul Goyal, Abhishek Jain, Manoj Prabhakaran, and Amit Sahai. New impossibility results for concurrent composition and a non-interactive completeness theorem for secure computation. In *CRYPTO*, pages 443–460, 2012.

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

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *Proc. 45th FOCS*, pages 186–195, 2004.

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

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

[BNP08] 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.

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

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

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

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Bob Werner, editor, *Proc.* 42*nd FOCS*, pages 136–147, 2001. Preliminary full version available as Cryptology ePrint Archive Report 2000/067.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. Report 2001/055, Cryptology ePrint Archive, July 2001. Extended abstract appeared in CRYPTO 2001.

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

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Euroctypt*, 2003.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. In *STOC*, 2001.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party computation. In *Proc.* 34*th 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. Full version: `http://www.cs.cornell.edu/~rafael/papers/ccacommit.pdf`.

[CLP12] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero knowledge from falsifiable assumptions. *IACR Cryptology ePrint Archive*, 2012:563, 2012.

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

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *STOC*, pages 542–552, 1991.

[GGJS12] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Concurrently secure computation in constant rounds. In *EUROCRYPT*, pages 99–116, 2012.

[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. Full version: `http://research.microsoft.com/en-us/um/people/vipul/pke.pdf`.

[GL90] Shafi Goldwasser and Leonid Levin. Fair computation of general functions in presence of immoral majority. In *Crypto '90*, pages 77–93, 1990.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In ACM, editor, *Proc.* 19*th STOC*, pages 218–229, 1987. See [Gol04, Chap. 7] for more details.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991. Preliminary version in FOCS' 86.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[GOLV12] Vipul Goyal, Rafail Ostrovsky, Chen-Keui Lee, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, 2012.

[Goy11] Vipul Goyal. Constant round non-malleable protocols using one way functions. In *STOC*, pages 695–704, 2011. See full version available on `http://research.microsoft.com/en-us/people/vipul/nmcom.pdf`.

[Goy12] Vipul Goyal. Positive results for concurrently secure computation in the plain model. In *FOCS*, pages 695–704, 2012.

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

[GS12] Divya Gupta and Amit Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. *CoRR*, abs/1210.3719, 2012.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logarithm rounds. In *Proc. 33th STOC*, pages 560–569, 2001. Preliminary full version published as cryptology ePrint report 2000/013.

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

[LP12] Huijia Lin and Rafael Pass. Black-box constructions of composable protocols without set-up. In *CRYPTO*, pages 461–478, 2012.

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

[LPV08] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable commitments from any one-way function. In *TCC*, pages 571–588, 2008.

[LPV11] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam, 2011. Personal communication.

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

[MOSV06] Daniele Micciancio, Shien Jin Ong, Amit Sahai, and Salil P. Vadhan. Concurrent zero knowledge without complexity assumptions. In *TCC*, pages 1–20, 2006.

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

[MR91] Silvio Micali and Phillip Rogaway. Secure computation. In *CRYPTO*, pages 392–404, 1991.

[OPV10] Rafail Ostrovsky, Omkant Pandey, and Ivan Visconti. Efficiency preserving transformations for concurrent non-malleable zero knowledge. In *TCC*, pages 535–552, 2010.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *Eurocrypt*, 2003.

[PPV08] Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. Adaptive one-way functions and applications. In *CRYPTO*, pages 57–74, 2008.

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

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

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

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

[PTV08] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent zero knowledge: Simplifications and generalizations (manuscript), 2008. `http://hdl.handle.net/1813/10772`.

[PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In *TCC*, pages 553–570, 2008.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, 2001.

[RK99] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, pages 415–432, 1999.

[Ros04] Alon Rosen. *The Round-Complexity of Black-Box Concurrent Zero-Knowledge*. PhD thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 2004.

[Sah99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proc.* 40*th FOCS*, pages 543–553, 1999.

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

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proc.* 27*th FOCS*, pages 162–167, 1986.

A Generalized Version of the Robust Extraction Lemma

In the generalized version of the lemma, we allow A^* to open sessions of the PRS preambles that are *statistically hiding*. At the start of the preamble, it is already understood whether it is statistically-binding or statistically-hiding. Since statistically-hiding preambles are only computationally-binding, we require that A^* is a PPT machine. Furthermore, we make following new adjustments to the robust-concurrent attack:

At the successful completion of a *statistically-hiding* PRS preamble s, when A^* receives the string α_s from $\mathcal{E}$, it can choose to respond with an opening of the committed value v_s. It does so by sending appropriate decommitment strings (v_s, d_s). Furthermore, the scheduling of this message is decided by A^*.

Other than this, the attack remains unchanged. We now present the generalized version of the lemma. The essence of the lemma is still the same as before. We only need to add conditions to deal with the statistically-hiding preambles. For such preambles, the validity constraint requires that α_s be equal to the opened value v_s (if any), except with negligible probability.

Lemma 6 (Robust Extraction: General Version)**.** *There exists an interactive Turing machine S ("robust simulator"), such that for every* PPT *A^*, for every $\Pi := \langle B, A \rangle$, there exists a party $\mathcal{E}$ ("online extractor"), such that for every $n \in \mathbb{N}$, for every $x \in \mathrm{dom}_B(n)$, and every $z \in \{0,1\}^*$, the following conditions hold:*

1. **Validity constraint.** *For every output ν of $\mathsf{REAL}^{A^*}_{\mathcal{E},\Pi}(n, x, z)$, we have:*

   (a) for every statistically-binding preamble s (appearing in ν) with transcript τ_s, if there exists a unique *value $v \in \{0,1\}^n$ in the commitment-transcript τ_s, then $\alpha_s = v$,*

   (b) for every statistically-hiding preamble s (appearing in ν) with transcript τ_s, if there exists a valid opening (v_s, d_s) in the view ν, then $\alpha_s = v_s$,

   where α_s is the value $\mathcal{E}$ sends at the completion of s.

2. **Statistical simulation.** *If $k = k(n)$ and $\ell = \ell(n)$ denote the round complexities of Π and the* PRS *preamble respectively, then the statistical distance between distributions $\mathsf{REAL}^{A^*}_{\mathcal{E},\Pi}(n, x, z)$ and $\mathsf{OUT}_{\mathrm{S}}\left[B(1^n, x) \leftrightarrow S^{A^*}(1^n, z)\right]$ is given by:*

$$\Delta(n) \leq 2^{-\Omega(\ell - k \cdot \log T(n))},$$

   where $T(n)$ is the maximum number of total PRS *preambles between A^* and $\mathcal{E}$.*[15] *Further, the running time of S is $\mathrm{poly}(n) \cdot T(n)^2$.*

The proof of this general version of the lemma is identical to the proof of the original lemma, presented in appendix 5. We only need to show that condition 1(b) also holds. We show that if it does not then we can violate the computational binding of the statistically-hiding PRS. Suppose that Com_{sh} is the underlying commitment scheme of the statistically hiding PRS. Then, if α_s is not equal to v_s, look at the two opened challenges in the execution of recurse whose XOR results in α_s; let they belong to slot i of this session. Both of these strings must have been decommitted to correctly (for the scheme Com_{sh}). Further, since opening v_s requires opening of all slots of PRS such that pairs in each slot XOR to the same v_s, we have that all pairs of slot i must have been correctly opened to strings that are different from what recurse learned. Therefore, we must have an instance of Com_{sh} with correct openings to two different values. The details are standard and omitted. We note that the value of $\Delta(n)$ in the second condition does not change since both $\mathcal{E}$ and S extract value α_s identically for the statistically-hiding PRS.

[15] The lemma allows for exponential $T(n)$ as well. However, if it is too large—e.g., $T(n) = 2^{2n}$, the PRS preamble should be modified suitably. For example, the value v as well as the challenges in each slot, must be of length at least $n + 2 \log T(n)$.

B Proof of Security for CCA-Com*

We do not present a full proof of security for this protocol here, since it follows closely the structure of proof for CCA-Com (based on collision-resistant hash functions). Instead, we present here the main changes to the proof, and how we proceed, during simulation.

We first start with the protocol CoinFlip and show that it actually has a robust simulator as well (i.e., the one that does not rewind the external party B of protocol Π), so long as the number of rounds are appropriately high.

A robust simulator for protocol CoinFlip. We now show that protocol CoinFlip admits a robust "simulator" w.r.t. k-round protocols, so long as $q - k \in \omega(1)$. Let us denote the protocol CoinFlip := $\langle P_1, P_2 \rangle$ where P_1 follows the step 2(a) whereas P_2 follows the step 2(b) defined in figure 4. Recall that protocol has $q \in \omega(1)$ rounds, where in round i P_1 commits to $\log n$ size string u_i, and P_2responds with a $\log n$ size string σ_i.

1. We say that P_2 *wins* if it can set $u_i = \sigma_i$ for every $i \in [q]$ with noticeable probability. Note that since $q \in \omega(1)$ no cheating prover P_1^*, even with unbounded running time, can win this game. The probability of winning the game for any P_1^* is at most n^{-q}.

2. We say that a machine S_{CF} is a simulator for protocol CoinFlip if for every P_2^*, if its output is computationally indistinguishable from the view of P_2^* in a real execution with P_1; furthermore, if P_2^* completes the protocol successfully, then S_{CF} outputs a trapdoor (u, d) such that except with negligible probability: $u_i = \sigma_i$ and d_i is a correct decommitment-string for round i commitment c_i to the value u_i. We require that S_{CF} be a strict polynomial time machine.

3. First observe that for the protocol CoinFlip such a simulator S_{CF} indeed does exist: trivial and standard. Simulator S_{CF} simply rewinds P_2^* in each slot-i until it succeeds in setting up $u_i = \sigma_i$, up to a maximum of n times. The simulator fails in output a view only with probability n^{-q}.

4. We now explain a robust version of this simulator. Suppose that P_2^*, in addition to interacting with P_1, also interacts with a party B of some protocol Π (which has k rounds). We would like a simulator which satisfies the same conditions as our simulator above S_{CF} above, but without "rewinding" party B. Therefore the new S_{CF} will interact with B as participant of Π, yet be able to output a view for P_2^* such that the "trapdoor" and the indistinguishability conditions are satisfied as above. Call such an S_{CF} robust.

5. It is easy to obtain a robust S_{CF} for CoinFlip. Modify S_{CF} during n rewinds of any given slot, if it encounters a message of Π—a breakpoint—as before, it simply gives up that rewind, and goes on to the next. It is easy to see that with this modification, S_{CF} is indeed robust. It fails only with probability n^{q-k} which is negligible since $q - k \in \omega(1)$.

Composing the two robust simulators. Having defined the robust S_{CF} for CoinFlip we now show how to modify our proof by composing S_{CF} with our robust-concurrent simulator S. The modifications follow:

1. As a first step, we view CoinFlip as a "replacement" for the statistically hiding phase PRS_2. Therefore, the simulator will always to look for $\alpha_s = (u, d)$ coming from outside (just like before where the PRS-secret was being provided from outside). Party $\mathcal{E}$ simply returns a random value, whereas simulator S, will expect it to come from outside, and be included in the transcript when CoinFlip ends.

2. In order to do this, without rewinding B, simulator S will run with respect to a modified protocol B' which runs both B as well as $\mathsf{CoinFlip}$. This will have a total of $q + k$ messages, and by choice of ℓ this will still keep the statistical distance negligible.

3. Now the simulator S is an adversary for the protocol $\mathsf{CoinFlip}$ and S_{CF} will simulate its view without rewinding B. Messages of B will be forwarded to S by S_{CF} unaltered. It is not hard to see that although the probabilities of aborting add up, they still remain negligible with our choice of parameters.

4. Therefore, the composed simulator $S_{\mathsf{CF}} \circ S$ is now a robust simulator which provides the trapdoor for $\mathsf{CoinFlip}$ as well.

The rest of the proof is now goes through trivial changes: instead of using S used the composed simulator $S_{\mathsf{CF}} \circ S$ and modify internal syntax to deal with this change. Details are repetitive and easy to fill in.

C Robust Non-malleable Commitments

We recall the notion of a robust non-malleable commitment scheme [LP09]. We only need a non-malleable commitment scheme that is robust with respect to constant round protocols. The notion of k-robust non-malleability, roughly speaking, requires non-malleability w.r.t. every k-round protocol, in addition to itself.

Formally, let n be the security parameter, $\langle C, R\rangle$ be a tag-based statistically-binding commitment scheme for identities of length $l(n)$. Consider a man-in-the-middle adversary A^* that, on inputs n and (advice) z, participates in one left and one right interaction simultaneously. In the left interaction, A^* interacts with C, receiving a commitment to value $v \in \{0,1\}^n$ using identity id of its choice. In the right interaction A^* interacts with R attempting to commit to a related value $\tilde{v}$, again using identity $\tilde{\mathsf{id}}$ of its choice. If the right commitment is invalid, or undefined, its value is set to $\perp$. In addition, if $\mathsf{id} = \tilde{\mathsf{id}}$, the value $\tilde{v}$ is again set to $\perp$. That is, if A^* copies the left identity id, it is not a valid adversary. Let $\mathsf{nmc}^{A^*}_{\langle C,R\rangle}(n, v, z)$ denote a random variable that describes the value $\tilde{v}$ *and* the view of A^* in this experiment. Then, following [DDN91, LPV08]: We say that $\langle C, R\rangle$ is *non-malleable w.r.t. itself*, if for every PPT A^*, and every pair of n-bit strings (v_1, v_2), and every $z \in \{0,1\}^*$, variables $\mathsf{nmc}^{A^*}_{\langle C,R\rangle}(n, v, z)$ and $\mathsf{nmc}^{A^*}_{\langle C,R\rangle}(n, v', z)$ are computationally indistinguishable.

Let $\langle C, R\rangle$ be commitment scheme as above that is non-malleable w.r.t. itself. Further, let $\Pi := \langle B, A\rangle$ be a protocol of $k(n)$ rounds, and $\mathrm{dom}_B(n)$ denote the set of valid inputs for B. We say that Π is an *interesting* protocol if for every PPT $\widetilde{A}$, every $n \in \mathbb{N}$, every pair of inputs $x_1 \in \mathrm{dom}_B(n)$ and $x_2 \in \mathrm{dom}_B(n)$, and every $z \in \{0,1\}^*$, we have that the following two random variables are computationally indistinguishable: $\mathsf{OUT}_{\widetilde{A}}[B(1^n, x_1) \leftrightarrow \widetilde{A}(1^n, z)]$ and $\mathsf{OUT}_{\widetilde{A}}[B(1^n, x_2) \leftrightarrow \widetilde{A}(1^n, z)]$.

We consider a man-in-the-middle A^* who participates in a left interaction–with the honest party $B(1^n, x)$ for $x \in \mathrm{dom}_B(n)$–as well as in a right interaction with R (as before) committing to $\tilde{v}$ using identity $\tilde{\mathsf{id}}$ (defined and adaptively chosen as before). Denote by $\mathsf{nmc}^{B,A^*}_{\langle C,R\rangle}(n, x, z)$ the random variable that describes the value $\tilde{v}$ along with the view of A^* in this experiment. Then, following [LP09]: We say that $\langle C, R\rangle$ is $k()n$*-robust*, if for every PPT A^*, every *interesting* protocol Π of round-complexity $k(n)$, every $n \in \mathbb{N}$, every pair of inputs (x_1, x_2) chosen from $\mathrm{dom}_B(n)$, and every $z \in \{0,1\}^*$, variables $\mathsf{nmc}^{B,A^*}_{\langle C,R\rangle}(n, x_1, z)$ and $\mathsf{nmc}^{B,A^*}_{\langle C,R\rangle}(n, x_2, z)$ are computationally indistinguishable. Constant round protocols that are non-malleable and 4-robust are now known [GOLV12, LP11, Goy11, Wee10, LP09].