What Information is Leaked under Concurrent Composition?[*]

Vipul Goyal
Microsoft Research India and
vipul@microsoft.com

Divya Gupta[†]
University of California, Los Angeles
divyag@cs.ucla.edu

Abhishek Jain[‡]
Massachusetts Institute of Technology
and Boston University,
abhishek@csail.mit.edu

Abstract

Achieving security under concurrent composition is notoriously hard. Indeed, in the plain model, far reaching impossibility results for concurrently secure computation are known. On the other hand, some positive results have also been obtained according to various weaker notions of security (such as by using a super-polynomial time simulator). This suggest that somehow, "not all is lost in the concurrent setting."

In this work, we ask what and exactly how much private information can the adversary learn by launching a concurrent attack? "Can he learn all the private inputs in all the sessions? Or, can we preserve the security of some (or even most) of the sessions fully while compromising (a small fraction of) other sessions? Or is it the case that the security of all (or most) sessions is (at least partially) compromised? If so, can we restrict him to learn an arbitrarily small fraction of input in each session?" We believe the above questions to be fundamental to the understanding of concurrent composition. Indeed, despite a large body of work on the study of concurrent composition, in our opinion, the understanding of *what exactly is it that goes wrong in the concurrent setting and to what extent* is currently quite unsatisfactory.

Towards that end, we adopt the knowledge-complexity based approach of Goldreich and Petrank [STOC'91] to quantify information leakage in concurrently secure computation. We consider a model where the ideal world adversary (a.k.a simulator) is allowed to query the trusted party for some "leakage" on the honest party inputs. We obtain both positive and negative results, depending upon the nature of the leakage queries available to the simulator. Informally speaking, our results imply the following: in the concurrent setting, "significant loss" of security (translating to high leakage in the ideal world) in some of the sessions is unavoidable if one wishes to obtain a general result. However on the brighter side, one can make the fraction of such sessions to be an *arbitrarily small polynomial* (while *fully* preserving the security in all other sessions). Our results also have an implication on secure computation in the bounded concurrent setting [Barak-FOCS'01]: we show there exist protocols which are secure as per the standard ideal/real world notion in the bounded concurrent setting. However if the actual number of sessions happen to exceed the bound, there is a *graceful degradation* of security as the number of sessions increase. (In contrast, prior results do not provide *any* security once the bound is exceeded.)

In order to obtain our positive result, we model concurrent extraction as the classical *set-covering problem* and develop, as our main technical contribution, a new *sparse* rewinding strategy. Specifically, unlike previous rewinding strategies which are very "dense", we rewind "small

[*]This paper is the full version of [GGJ13] that appeared at Crypto 2013.
[†]Work done in part while visiting Microsoft Research India.
[‡]Work done in part while visiting Microsoft Research India.

intervals" of the execution transcript and still guarantee extraction. This yields other applications as well, including improved constructions of precise concurrent zero-knowledge [Pandey et al.-Eurocrypt'08] and concurrently secure computation in the multiple ideal query model [Goyal et al.-Crypto'10].

In order to obtain our negative results, interestingly, we employ techniques from the regime of leakage-resilient cryptography [Dziembowski-Pietrzak-FOCS'08].

Keywords: Secure Computation, Concurrent Composition

1 Introduction

Concurrently Secure Computation. Traditional security notions for cryptographic protocols such as secure computation [Yao86, GMW87] were defined for a *stand-alone* setting, where security holds only if a single protocol session is executed in isolation. Today's world, however, is driven by networks – the most important example being the Internet. In a networked environment, several protocol instances may be executed *concurrently*, and an adversary may be able to perform coordinated attacks across sessions by corrupting parties in various sessions. As such, a protocol that is secure in the classical standalone setting may become completely insecure in the network setting.

Towards that end, over the last decade, a tremendous amount of effort has been made to obtain protocols with strong composability guarantees under concurrent execution. Unfortunately, a sequence of works have demonstrated far reaching impossibility results for designing secure protocols in the concurrent setting [CF01, CKL03, Lin03b, Lin03a, Lin04, BPS06, Goy12, AGJ$^+$12, GKOV12]. In particular, these works have ruled out secure realization of essentially all non-trivial functionalities even in very restricted settings such as where inputs of honest parties are fixed in advance (rather than being chosen adaptively in each session), and where the adversary is restricted to corrupting parties with specific roles.

What Information is Getting Leaked to the Adversary? Many of these impossibility results work by designing an explicit "chosen protocol attack". Such an attack shows that there exists some information the concurrent adversary can learn in the real world which is impossible to obtain for the ideal adversary (a.k.a the simulator). Nevertheless, subsequent to these impossibility results, several prior works have in fact obtained positive results for concurrently secure computation according to various relaxed notions of security such as super-polynomial simulation [Pas03, PS04, BS05, CLP10, GGJS12, LP12], input indistinguishable computation [MPR06, GGJS12], multiple-ideal query model [GJO10], etc.[1] These results suggest that somehow, "not all is lost in the concurrent setting" (in the plain model).

Given the above, the following natural questions arise:

What and exactly how much private information can the adversary learn by launching a concurrent attack? Can he learn all the private inputs in all sessions? Or, can we preserve the security of some (or even most) of the sessions fully while compromising (a small fraction of) other sessions? Or is it the case that the security of all (or most) sessions is (at least partially) compromised? If so, can we restrict him to learn an arbitrarily small fraction of input in each session?

We believe the above questions are very natural to ask and fundamental to the understanding of concurrent composition. Indeed, despite a large body of research on the study of concurrent composition, in our opinion, the understanding of "*what exactly is it that goes wrong in the concurrent setting, and, to what extent*" is currently unsatisfactory. The current paper represents an attempt towards improving our understanding of this question.

[1]There has also been a rich line of works on designing secure computation with some type of "setup" where, e.g., a trusted party publishes a randomly chosen string [CLOS02, BCNP04, Kat07, CGS08]. However the focus of the current work is the *plain model*.

A Knowledge-complexity based Approach. We adopt the knowledge-complexity based approach of Goldreich and Petrank [GP91] to quantify the information leakage to the adversary in concurrently secure computation. Specifically, generalizing the approach of [GP91], we consider a modification of the standard real/ideal paradigm where in the ideal world experiment, the simulator is allowed to query the trusted party for some "leakage" on the honest party inputs. The underlying intuition (as in [GP91]) is that the amount of leakage observed by the simulator in order to simulate the view of an adversary represents an upper bound on the amount of information potentially leaked to the real adversary during the concurrent protocol executions.

More concretely, we consider two notions of "leaky ideal world" in this work, described as follows.

Ideal world with individual leakage. Let there be m concurrent sessions with the honest party input in the i^{th} session denoted by x_i. In the *individual* leakage model, in every session i, as usual the simulator is allowed to query the trusted party once and get an output on the input of its choice. In *addition*, (in session i), the simulator can query with an efficiently computable leakage function L_i and get $L_i(x_i)$ in return. The constraint is that in every session i, the length of $L_i(x_i)$ is at most $\epsilon|x_i|$. If this is the case, we say that the protocol is secure with an ϵ-individual ideal leakage simulator. Intuitively, if such a security guarantee is satisfied, then it means that the security of all the sessions is only partially compromised; in particular, every session still has a non-trivial level of security guarantee.

As we discuss below, in this model, our main result is a negative one. This brings us to our next model.

Ideal world with joint leakage. In the *joint* leakage model, first, the honest party sends its input for every session to the trusted party. Next, in every session, the simulator is allowed to do the following exactly once at any point: query with an input and get the resulting output (as usual), and *additionally* query with an efficiently computable leakage function L_i and get $L_i(\vec{x})$ in return (where $\vec{x} = (x_1, \ldots, x_m)$). The constraint is that throughout the simulation, the total number of bits leaked $\sum L_i(\vec{x})$ is at most $\epsilon|\vec{x}|$. If this is the case, we say that the protocol is secure with an ϵ-joint ideal leakage simulator. In this model, our main result is a positive one, as we discuss below.

On the Relation to Leakage-resilient Interactive Protocols. We remark that our ideal model resembles those considered in the line of works concerned with constructing leakage-resilient secure computation protocols [GJS11, BCH12, BGJ$^+$12]. However we stress that in our setting, there is *no* physical leakage in the real world and instead there are just an (unbounded) polynomial number of concurrent sessions. Indeed, as in classical security definitions, the adversary in our security model is only given *black-box* access to the system, and as such, achieving security against physical (non-black-box) attacks is outside the scope of this work. Nevertheless, comparing our work to [GJS11, BCH12, BGJ$^+$12], we find it interesting that there is a parallel between the ideal world guarantees considered in two unrelated settings: leaky real world, and, concurrent real world.

1.1 Our Results

We consider the setting of unbounded concurrent composition in the plain model. As with several previous works [BPS06, GJO10, Goy12], we consider static corruptions with fixed inputs (i.e., where the inputs of honest parties are fixed in advance rather than being chosen adaptively). Below, we describe our results in each of the two leaky ideal world models (as discussed above), along with additional applications.

I. Positive Result in the Joint Leakage Model. We obtain the following main result in the joint leakage model:

Theorem 1 *(Informally stated.) Assuming 1-out-of-2 (semi-honest) oblivious transfer, for every polynomial poly(n), there exists a protocol which is secure with an $\epsilon = \frac{1}{poly(n)}$ joint leakage simulator.*

The round complexity of our protocol is $\frac{\log^6 n}{\epsilon}$. We show that this is *almost optimal* w.r.t. a black-box simulator: we rule out protocols with round complexity $\frac{O(\log n)}{\epsilon}$ proven secure using a black-box simulator.

Fully preserving the security of most sessions. We note that the simulator for our positive result, in fact, satisfies the following additional property: rather than leaking a small fraction of the input in each session, it leaks the entire input of a small (i.e., ϵ) fraction of sessions while *fully* preserving the security of the remaining sessions. Hence, we get the following interesting corollary:

Theorem 2 *(Informally stated.) Assuming 1-out-of-2 (semi-honest) oblivious transfer, for every polynomial poly(n), there exists a protocol s.t. under unbounded concurrent self-composition, the security of at most a $\frac{1}{poly(n)}$ fraction of the sessions may be compromised. For the remaining sessions, the standard ideal/real world security notion can be satisfied.*

In fact, our negative result in the independent leakage model (discussed below) indicates that for a general positive result, the above security guarantee is essentially optimal.

Bounded concurrent secure computation with graceful security degradation. Going further, observe that by choosing $\epsilon < \frac{1}{m|x|}$, we get a construction where the simulator is allowed *no leakage* at all if the number of sessions is up to m. This is because the maximum number of bits simulator is allowed to leak will be $\epsilon m|x|$ which is less than 1. Hence, positive results for bounded concurrent secure computation [Lin03a, PR03, Pas04] follow as a special case of our result. However if the actual number of sessions just slightly exceed m, the simulator is allowed some small leakage on the input vector (i.e., total of only 1 bit up to $2m$ sessions, 2 bits up to $3m$ sessions, and so on). Thus, the leakage allowed grows slowly as the number of sessions grow. This phenomenon can be interpreted as *graceful degradation of security* in the concurrent setting.

Theorem 3 *(Informally stated.) Assuming 1-out-of-2 (semi-honest) oblivious transfer, there exist protocols which are secure as per the standard ideal/real world notion in the bounded concurrent setting. However if the actual number of sessions happen to exceed this bound, there is graceful degradation of security as the number of sessions increase.*

A set-cover approach to concurrent extraction and a new rewinding strategy. In order to obtain our positive result, we take a generic "cost-centric" approach to rewinding in the concurrent setting. For example, in our context, the amount of leakage required by the simulator to simulate the protocol messages during the rewindings can be viewed as the "cost" of extraction. Thus, the goal is to perform concurrent extraction with minimal cost. With this view, we model concurrent extraction as the classical *set-covering problem* [CLRS09] and develop, as our main technical contribution, a new *sparse* rewinding strategy. Very briefly, unlike known concurrent rewinding techniques [RK99b, KP01, PRS02, PPS$^+$08] that are very "dense", we rewind "small intervals" of the execution transcript, while still guaranteeing extraction in all of the sessions. As we explain in Section 1.2, which intervals of the execution transcript are rewound (and how many times), has a direct consequence on the cost of extraction. In particular, rewinding small intervals (only a few times) is crucial to minimizing the cost and obtaining our positive result.

Our sparse rewinding strategy also yields other interesting applications that we discuss below. Thus, we believe it to be of independent interest.

II. Negative Result in the Individual Leakage Model. In the individual leakage model, our main result is negative, ruling out even *non-black-box* simulation. Specifically, we give an impossibility result for the oblivious transfer (OT) functionality where the ideal leakage allowed is $(1/2-\delta)$ fraction of the input length (for every positive constant δ). Note that this is the maximum possible leakage bound such that the ideal adversary still does not learn the entire input of the honest parties (which would otherwise result in a trivial positive result): indeed, if the fraction of leakage allowed is $1/2$, the ideal adversary can learn one of the sender inputs by making use of leakage, and, the other by making use of the "official" trusted party call.

Leakage-resilient One-Time Programs. Of independent interest, the techniques used in our negative result also yield a new construction of one-time programs [GKR08] where the adversary can query the given hardware tokens once (as usual), and *additionally* leak on the secrets stored in each token in any arbitrary adaptively chosen manner (e.g., the leakage function for the next token may even depend upon the leakage and the "official" outputs seen so far, and so on). Our key technical tool in constructing such a gadget is the intrusion-resilient secret sharing protocol of Dziembowski and Pietrzak [DP07].

Put together, results (I) and (II) show that in the concurrent setting, "significant loss" of security in some of the sessions is unavoidable if one wishes to obtain a general positive result. However on the brighter side, one can make the fraction of such sessions to be an *arbitrarily small polynomial* (while *fully* preserving the security in all other sessions).

III. Other Applications. As alluded to above, along the way to developing our main positive result, we develop a new *sparse rewinding strategy* that leads to interesting applications in the context of precise zero-knowledge [MP06, PPS$^+$08] and concurrently secure computation in the multiple ideal query model [GJO10]. We discuss them below.

Improved precise concurrent zero knowledge. In the traditional notion of zero-knowledge, the simulator may run in time which is any polynomial factor of the (worst-case) running time of the adversarial verifier. This means that any information learned by the adversary in the protocol could also be computed directly in *polynomial* time. The notion of precise zero-knowledge [MP06] deals with studying how low this polynomial can be. In particular, can one design protocols where the running time of the simulator is only slightly higher than the actual running time of the adversary? Besides being a fundamental question on its own, the notion of precise zero-knowledge has found applications in unrelated settings such as designing leakage-resilient zero-knowledge [GJS11], designing concurrently secure protocols in the multiple ideal query model [GJO10], etc.

Pandey et al. [PPS$^+$08] study the problem of precise *concurrent* zero-knowledge and give a protocol with the following parameters. Let t be the actual running time of the verifier. Then, the protocol of [PPS$^+$08] has round complexity n^δ (for any constant $\delta \leq 1$) and knowledge precision $c \cdot t$ where c is a large constant depending upon the adversary. That is, the running time of the simulator is *linear* in the running time of the adversary.[2]

Our sparse rewinding strategy directly leads to a new construction of precise concurrent zero-knowledge, improving upon [PPS$^+$08] *both* in terms of round complexity as well as knowledge precision.

Theorem 4 *(Informally stated.) Assuming one way functions, there exists a concurrent zero-knowledge protocol with poly-logarithmic ($\mathcal{O}(\log^6 n)$, to be precise) round-complexity and almost optimal knowledge precision of $(1+\delta)t$ (for any constant δ).*

Improved concurrently secure computation in the MIQ model. In the quest for positive results for concurrently secure computation, Goyal, Jain and Ostrovsky proposed the multiple ideal query (MIQ) model, where for every session in the real world, the simulator is allowed to query

[2][PPS$^+$08] also give a construction requiring only $\omega(\log n)$ rounds, however, the knowledge precision achieved in this case is super-linear.

the ideal functionality for the output *multiple* times (as opposed to only *once*, as in the standard definition of secure computation). They construct a protocol in this model whose security is proven w.r.t. a simulator that makes a total of $c \cdot m$ number of ideal queries in total (and c queries per session, *on an average*), where c is a large constant that depends on the adversary and m is the number of sessions.

We note that our security model is intimately connected to the multiple ideal query (MIQ) model of Goyal, Jain and Ostrovsky [GJO10, GJ13] for concurrently secure computation. To observe this connection, note that the additional output queries in the MIQ model can be viewed as leakage observed by the simulator in our model. One can make a more accurate comparison between the models based on the output length of functionality (that we wish to compute securely):

1. For general functionalities whose output length is at least a constant fraction of the input length, the result of [GJO10] does *not* yield any non-trivial result in our model. This is because the average leakage required per session would be $c|y|$ (where y is the function output) which is as much as leaking the *entire* input.
2. For functionalities whose output is much shorter than the input (e.g. boolean functions), the result of [GJO10] does yield a non-trivial result in our model. For example, in the case of boolean functions (or functions with constant bit outputs), [GJO10] yields a positive result in our model with a leakage fraction of $\epsilon = \frac{\mathcal{O}(1)}{|x|}$. However, by a direct study of our model, we obtain a much better leakage bound with $\epsilon = \frac{1}{\mathrm{poly}(n)}$ (for any polynomial $\mathrm{poly}(n)$).

Looking at the other direction, the techniques for obtaining our main result also yield a new positive result in the MIQ result with only $(1 + \frac{1}{\mathrm{poly}(n)})$ number of ideal queries per session (*on an average*), thus improving upon the result of [GJO10]. We stress, however, that even this improved result in the MIQ model does not directly translate to a *general* result in our model with the leakage bounds that we achieve by a direct study. Again, this is because for functionalities whose output is much larger than the input, even one extra query per session would translate to very high leakage on the input. On the other hand, for functionalities whose output length is similar to the input length, the security guarantees in the two models are indeed similar.

Indeed, one way of interpreting our positive results is a new technique to obtain better results in the MIQ model.

Theorem 5 *(Informally stated.) Assuming 1-out-of-2 (semi-honest) oblivious transfer, there exists a concurrently secure protocol in the MIQ model with* $(1+\frac{1}{poly(n)})$ *number of ideal queries per session (on an average).*

Open Problem. We note that while our positive result in the leaky ideal world model is essentially tight for general functionalities (in keeping with our negative result), the MIQ model provides a hint that it is not tight if one is interested in achieving positive results for *specific* functionalities. In particular, in the MIQ model, even if the simulator requires multiple queries for some sessions, some meaningful security for those sessions may still be plausible (*depending on the specification of the functionality*). In light of this, one can consider the following more general question: is it possible to construct protocols such that under unbounded concurrent self-composition, security of most sessions is fully preserved (as per the standard real/ideal paradigm) while in the remaining sessions, some meaningful security guarantee is still provided, e.g., w.r.t. a game based security notion. We leave this as an exciting open question for future work.

1.2 Our Techniques

Here we give an overview of the underlying techniques used in our positive result. Due to lack of space, we defer discussion on our negative results to the technical sections.

A Starting Approach. A well established approach to constructing secure computation protocols against malicious adversaries in the standalone setting is to use the GMW compiler [GMW87]: take a semi-honest secure computation protocol and "compile" it with zero-knowledge arguments. Then, a natural starting point to construct a concurrently secure computation protocol is to follow the same principles in the concurrent setting: somehow compile a semi-honest secure computation protocol with a concurrent zero-knowledge protocol (for security in more demanding settings, compilation with concurrent non-malleable zero-knowledge [BPS06] may be required). Does such an approach (or minor variants) already give us protocols secure according to the standard ideal/real world definition in the plain model?

The fundamental problem with this approach is the following. Note that known concurrent zero-knowledge simulators (in the fully concurrent setting) work by rewinding the adversarial parties. In the concurrent setting, the adversary is allowed to control the scheduling of the messages of different sessions. Then the following scenario might occur:

- Between two messages of a session s_1, there might exist another entire session s_2.
- When the simulator rewinds the session s_1, it may rewind past the beginning of session s_2. Hence throughout the simulation, the session s_2 may be executed multiple times from the beginning.
- Every time the session s_2 is executed, the adversary may choose a different input (e.g., the adversary may choose his input in session s_2 based on the entire transcript of interaction so far). In such a case, the simulator is required to leak additional information about the input of the honest party (e.g., in the form of an extra output as in [GJO10]).

Indeed, some such problem is rather inherent as indicated by various impossibility results [Lin04, BPS06, Goy12, AGJ$^+$12, GKOV12].[3] As stated above, our basic idea will be to use leakage on the inputs of the honest parties in order to continue in the rewindings (or look-ahead threads). Our simulator would simply request the ideal functionality for the entire input of the honest party in such a session. Subsequent to this, *such a session can appear on any number of look-ahead threads*: we can simply use the leaked input and use that to proceed honestly.

Main Technical Problem. The key technical problem we face is the following. All previous rewinding strategies are too "dense" for our purposes. These strategies do not lead to any non-trivial results in our model: the simulator will simply be required the leak the honest party input in *each session*. For example, in the oblivious rewinding strategies used in [KP01, PRS02, PPS$^+$08, GJO10], the "main" thread of protocol execution is divided into various blocks (2 blocks in [KP01, PRS02] and n blocks in [PPS$^+$08, GJO10]). Each given block is rewound that results in a "look-ahead thread". Each session on the main thread will also appear on these look-ahead threads (in fact, on multiple look-ahead threads). Hence, it can be shown that our strategy of leaking inputs of sessions appearing in look-ahead threads will result in leakage of inputs in all sessions. For the case of adaptive rewinding strategies [RK99a, PV08, DGS09], the problem is even more pronounced. Any given block (or an interval) of the transcript may be rewound any polynomial number of times (each time to solve a different session).

Thus, the known rewinding strategies do not yield any non-trivial results in our model (let alone allow leakage of any arbitrarily small polynomial fraction of inputs).

Main Idea: Sparse Rewinding Strategies. In order to address the above problem, we develop a new "cost-based" rewinding strategy. In particular, our main technical contribution is the development of what we call *sparse rewinding strategies* in the concurrent setting. In a sparse rewinding strategy, the main idea is to choose various *small intervals* of the transcript and rewind *only* those intervals. The main technical challenge is to show that despite rewinding only only few locations

[3]We stress that these results rule out non-black-box simulation as well.

of the transcript, extraction is still guaranteed for every session (regardless of where it lies on the transcript).

In more detail, our rewinding strategy bears similarities with the oblivious recursive rewinding strategies used in [KP01, PRS02]. Our main contribution lies in showing that a "significantly stripped down" version of their strategy is still sufficient to guarantee extraction in all sessions. More specifically, recall that the recursive rewinding strategies in [KP01, PRS02] have various threads of executions (also called blocks) which are at different "levels" and have different sizes. We carefully select only a small subset of these blocks and carry them out as part of our rewinding schedule (while discarding the rest). The leakage parameter ϵ and the resulting round complexity (which we show to be almost optimal w.r.t. a black-box simulator) determines what fraction of blocks (and at what levels) are picked to be carried out in the rewinding schedule. Given such a strategy, we reduce the problem of covering all sessions to a *set cover problem*: pick sufficiently many blocks (each block representing a set of sessions which are "solved" when that block is carried out as part of the rewinding schedule) such that every session is covered (i.e., extraction is guaranteed) while still keeping the overall leakage (more generally, the "cost") to be low. Indeed, this cost-centric view is what also allows us to improve upon the precision guarantees in [PPS$^+$08].

Additional Challenges. To convert the above basic idea into an actual construction, we encounter several difficulties. The main challenge is to argue extraction in all sessions. Recall that the *swapping arguments* in prior works [KP01, PRS02, PTV12, PPS$^+$08] crucially rely on "symmetry" between the main thread of execution and the look-ahead threads (i.e., execution threads created view rewinding). In particular, to argue extraction, [PRS02, PTV12] define *swap* and *undo* procedures w.r.t. execution threads that allow to transform a "bad" random tape of the simulator (that leads to extraction failure) into a "good" random tape (where extraction succeeds) and back. The idea being to show that every bad random tape, there exist super-polynomially many good random tapes; as such, with overwhelming probability, the simulator must choose a good random tape.

In our setting, using such swapping arguments becomes non-trivial. First off, note that we cannot directly employ the standard greedy strategy for set-cover problem to choose which blocks must be rewound. Very briefly, this is because once one swaps two blocks (one on the main thread, and the corresponding one on a look-ahead thread), the choice of set of blocks which should be chosen might completely change (this is because the associated "costs" of blocks may change after swapping). Indeed, any such "biased" strategy seems to be doomed for failure against adversaries that choose the schedule adaptively. Towards this end, we use a *randomized* strategy for choosing which blocks to rewind, with the goal of still keeping the extraction cost minimal. Nevertheless, despite the randomized approach, the sparse nature of our block choosing strategy still results in significant "asymmetry" across the entire rewinding schedule. This leads to difficulties in carrying out the swap and undo procedures as in [PRS02, PTV12]. We resolve these difficulties by using a careful "localized" swapping argument (see technical sections for details).

Our final protocol is based on compilation with concurrent non-malleable zero-knowledge [BPS06]. We recall that there are several problems that arise with such a compilation. First, the security of the [BPS06] construction is analyzed only for the setting where all the statements being proven by honest parties are fixed in advance. Secondly, the extractor of [BPS06] is unsuitable for extracting inputs of the adversary since it works after the entire execution is complete on a *session-by-session* basis. Fortunately, these challenges were tackled in the work of Goyal et al. [GJO10]. Indeed, Goyal et. al. presented an approach which can be viewed as a technique to correctly compile a semi-honest secure protocol with [BPS06]. We adopt their approach to construct our final protocol.

Due to lack of space, the above discussion is significantly oversimplified. We refer the reader to the technical sections for details.

Organization. The paper is organized as follows: We begin by describing our security models, namely Joint Leaky Ideal World model and Individual Leaky Ideal World model, in Section 2. Next we describe our concurrently extractable commitment scheme and our cost based sparse rewinding strategy in Section 3. For a formal proof of extraction and cost analysis of our rewinding strategy refer to Appendix B. Finally, we give our main theorem statements in Section 4. In Appendix A, we describe all the primitives required for our protocol as well the negative results. Detailed description of the concurrently secure MPC protocol in joint leaky ideal world model is given in Appendix C. We give our improvement in concurrent precise zero-knowledge in Appendix D. Next, we give the impossibility result for individual leaky ideal world model in Appendix E. Finally, we give the round complexity black box lower bound for joint leaky ideal world model in Appendix F.

2 Our Model

In this section, we present our security model. Throughout this paper, we denote the security parameter by κ.

We define our security model by extending the standard real/ideal paradigm for secure computation. Roughly speaking, we consider a relaxed notion of concurrently secure computation where the ideal world adversary, aka, the simulator) is allowed to leak on the inputs of the honest parties. Intuitively, the amount of leakage obtained by the simulator in order to simulate the view of a concurrent adversary corresponds to the "information leakage" under concurrent composition.

We define two models of security:

- In the first model, the simulator is allowed to obtain *joint* leakage on the inputs of the honest parties across all the sessions. Our main result in this model is a positive one.
- In the second model, the simulator is only allowed to leak on the honest party's input in each session *individually.* In Appendix E, we essentially rule out achieving security in this model.

We now describe each of these two models separately. We first state some points that are common to both of these models.

In this work, we consider a malicious, static adversary. The scheduling of the messages across the concurrent executions is controlled by the adversary. We do not require fairness and hence in the ideal model, we allow a corrupt party to receive its output in a session and then optionally block the output from being delivered to the honest party, in that session. We consider a static adversary that chooses whom to corrupt before execution of the protocol. Finally, we consider *computational* security only and therefore restrict our attention to adversaries running in probabilistic polynomial time. We denote computational indistinguishability by $\overset{c}{\equiv}$.

Concurrently Secure Computation in the Joint Leaky Ideal World Model. We first describe the "joint" leakage model. To model joint leakage across the ideal world executions, we allow the simulator to make queries of the form L to the trusted party, where L is the circuit representation of an efficiently computable function. On receiving such a query, the trusted party (who controls all the ideal world sessions) computes L over the honest parties' inputs in all the sessions and returns the response to the adversary.

We now proceed to describe the ideal and real world experiments and then give our security definition.

IDEAL MODEL. We first define the ideal world experiment, where there is a trusted party for computing the desired two-party functionality $\mathcal{F}$. Let there be two parties P_1 and P_2 that are involved in multiple sessions, say $m = m(\kappa)$. An adversary may corrupt either of the two parties. As in the standard ideal world experiment for concurrently secure computation, the parties send their inputs to the trusted party and receive the output of f evaluated on their inputs. The main difference from the standard ideal world experiment is that the ideal adversary is allowed to make leakage queries to the trusted party on the honest parties' inputs in all the sessions at any time during the experiment. In more detail, the ideal world execution proceeds as follows.

I. Inputs: P_1 and P_2 obtain a vector of m inputs, denoted $\vec{x}$ and $\vec{y}$ respectively. The adversary is given auxiliary input z, and chooses a party to corrupt. Without loss of generality, we assume that the adversary corrupts P_2 (when the adversary controls P_1, the roles are simply reversed). The adversary receives the input vector $\vec{y}$ of the corrupted party.

II. Initiation: The adversary initiates all the sessions by sending a $\mathsf{start\text{-}session}$ message to the trusted party. Upon receiving this message, the trusted party sends $\mathsf{start\text{-}session}$ to the honest party P_1.

III. Honest parties send inputs to trusted party: Upon receiving $\mathsf{start\text{-}session}$ from the trusted party, honest party P_1 sends the input vector $\vec{x}$ to the trusted party.

IV. Adversary sends input to trusted party and receives output: Whenever the adversary wishes, it may send a message (i, y'_i) to the trusted party for any y'_i of its choice. Upon sending this pair, it receives back $(i, \mathcal{F}(x_i, y'_i))$ where x_i is the input of P_1 for session i.

V. Adversary instructs trusted party to answer honest party: When the adversary sends a message of the type (output, i) to the trusted party, the trusted party sends $(i, \mathcal{F}(x_i, y'_i))$ to P_1, where x_i and y'_i denote the respective inputs sent by P_1 and adversary for session i.

VI. Adversary asks Leakage Queries: At any time during the ideal world experiment, adversary may send leakage queries of the form L to the trusted party. On receiving such a query, the trusted party computes $L(\vec{x})$ and returns it to the adversary.

VII. Outputs: The honest party P_1 always outputs the values $\mathcal{F}(x_i, y'_i)$ that it obtained from the trusted party. The adversary may output an arbitrary (probabilistic polynomial-time computable) function of its auxiliary input z, input vector $\vec{y}$ and the function outputs obtained from the trusted party.

The $\mathsf{ideal\ execution}$ of a function $\mathcal{F}$ with security parameter κ, input vectors $\vec{x}$, $\vec{y}$ and auxiliary input z to $\mathcal{S}$, denoted $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\kappa, \vec{x}, \vec{y}, z)$, is defined as the output pair of the honest party and $\mathcal{S}$ from the above ideal execution.

Definition 1 (ϵ-Joint-Ideal-Leakage Simulator) *Let $\mathcal{S}$ be a non-uniform probabilistic (expected)* PPT *ideal-model adversary. We say that $\mathcal{S}$ is a ϵ-joint-ideal-leakage simulator if it leaks at most ϵ fraction of the input vector of the honest party.*

REAL MODEL. We now consider the real model in which a real two-party protocol is executed (and there exists no trusted third party). Let $\mathcal{F}$ be as above and let Π be a two-party protocol for computing $\mathcal{F}$. Let $\mathcal{A}$ denote a non-uniform probabilistic polynomial-time adversary that controls either P_1 or P_2. The parties run concurrent executions of the protocol Π, where the honest party follows the instructions of Π in all executions. The honest party initiates a new session i with input x_i whenever it receives a $\mathsf{start\text{-}session}$ message from $\mathcal{A}$. The scheduling of all messages throughout the executions is controlled by the adversary. That is, the execution proceeds as follows: the adversary sends a message of the form (i, msg) to the honest party. The honest party then adds msg to its view of session i and replies according to the instructions of Π and this view. At the conclusion of the protocol, an honest party computes its output as prescribed by the protocol. Without loss of generality, we assume the adversary outputs exactly its entire view of the execution of the protocol.

The real concurrent execution of Π with security parameter κ, input vectors $\vec{x}$, $\vec{y}$ and auxiliary input z to $\mathcal{A}$, denoted $\text{REAL}_{\Pi,\mathcal{A}}(\kappa, \vec{x}, \vec{y}, z)$, is defined as the output pair of the honest party and $\mathcal{A}$, resulting from the above real-world process.

SECURITY DEFINITION. Having defined the ideal and real models, we now give our security definitions.

Definition 2 (Concurrently Secure Computation in the Joint Leaky Ideal World Model) *A protocol* Π *evaluating a functionality* $\mathcal{F}$ *is said to be* ϵ-secure in the joint leaky ideal world model *if for every real model non-uniform* PPT *adversary* $\mathcal{A}$*, there exists a non-uniform (expected)* PPT *ϵ-joint-ideal-leakage simulator $\mathcal{S}$ such that for every polynomial $m = m(\kappa)$, every pair of input vectors $\vec{x} \in X^m$, $\vec{y} \in Y^m$, every $z \in \{0,1\}^*s$,*

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\kappa, \vec{x}, \vec{y}, z)\}_{\kappa\in\mathbb{N}} \overset{c}{\equiv} \{\text{REAL}_{\Pi,\mathcal{A}}(\kappa, \vec{x}, \vec{y}, z)\}_{\kappa\in\mathbb{N}}$$

Concurrently Secure Computation in the Individual Leaky Ideal World Model. We now describe the "individual" leakage model. In this model, the ideal and real world experiments are defined in the same as in the joint leaky ideal world model. The only difference is how the leakage queries are handled in the ideal world experiment. Specifically, in the individual leaky ideal world model, at any point during the ideal world experiment, the ideal adversary may send a leakage query of the form (i, L) to the trusted party. On receiving such a query, the trusted party first checks whether a leakage query of the form $(i, \cdot)$ was earlier made by the adversary. If this is the case, then the trusted party ignores the leakage query. Otherwise, it computes $L(x_i)$ (where x_i is the input of the honest party in session i) and returns the response to the adversary.

Definition 3 (ϵ-Individual-Ideal-Leakage Simulator) *Let $\mathcal{S}$ be a non-uniform probabilistic (expected)* PPT *ideal-model adversary. We say that $\mathcal{S}$ is a ϵ-individual-ideal-leakage simulator if it leaks at most ϵ fraction of the the honest party input in each session.*

Definition 4 (Concurrently Secure Computation in the Individual Leaky Ideal World Model) *A protocol* Π *evaluating a functionality* $\mathcal{F}$ *is said to be* ℓ-secure in the leaky ideal world model against joint leakage *if for every real model non-uniform* PPT *adversary* $\mathcal{A}$*, there exists a non-uniform (expected)* PPT *ϵ-individual-ideal-leakage simulator $\mathcal{S}$ such that for every polynomial $m = m(\kappa)$, every pair of input vectors $\vec{x} \in X^m$, $\vec{y} \in Y^m$, every $z \in \{0,1\}^*s$,*

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\kappa, \vec{x}, \vec{y}, z)\}_{\kappa\in\mathbb{N}} \overset{c}{\equiv} \{\text{REAL}_{\Pi,\mathcal{A}}(\kappa, \vec{x}, \vec{y}, z)\}_{\kappa\in\mathbb{N}}$$

3 Framework for Cost-based rewinding

3.1 Setting

Consider two players P_1 and P_2 running concurrent execution of a two party protocol Π. Π may consists of multiple executions of the extractable commitment scheme $\langle C, R\rangle$ (Section 3.2) and some other protocol messages. These other protocol messages will depend upon our underlying applications. In particular we will consider two main applications. In our application of concurrently secure computation in joint leaky ideal world model, protocol Π is simply the secure computation protocol. In precise concurrent zero-knowledge protocol, Π will be a zero-knowledge protocol.

Moreover, each message in the protocol will have an associated fixed non-zero cost based on the application. In case of concurrent execution of the secure computation protocol, any message from the adversary which causes our simulator to make an output query to the trusted functionality in the ideal world is considered a "heavy" message. All other messages are almost "free". In case of concurrent precise zero-knowledge, cost of a message is the time taken by the adversary to generate that message. All messages of the honest prover are unit cost.

We consider the scenario when exactly one of the parties is corrupted. We begin by describing the extractable commitment scheme $\langle C, R\rangle$.

3.2 Extractable Commitment Protocol $\langle C, R\rangle$

Let $\textsc{com}(\cdot)$ denote the commitment function of a non-interactive perfectly binding string commitment scheme (as described in Section A). Let κ denote the security parameter. Let $\ell = \omega(\log \kappa)$. Let $N = N(\kappa)$ which is fixed based on the application. The commitment scheme $\langle C, R\rangle$, where the committer commits to a value σ (referred to as the *preamble secret*), is described as follows.

Commit Phase:
Stage Init: To commit to a κ-bit string σ, C chooses $(\ell \cdot N)$ independent random pairs of κ-bit strings $\{\alpha^0_{i,j}, \alpha^1_{i,j}\}^{\ell,N}_{i,j=1}$ such that $\alpha^0_{i,j} \oplus \alpha^1_{i,j} = \sigma$ for all $i \in [\ell], j \in [N]$. C commits to all these strings using $\textsc{com}$, with fresh randomness each time. Let $B \leftarrow \textsc{com}(\sigma)$, and $A^0_{i,j} \leftarrow \textsc{com}(\alpha^0_{i,j})$, $A^1_{i,j} \leftarrow \textsc{com}(\alpha^1_{i,j})$ for every $i \in [\ell], j \in [N]$.
We say that the protocol has reached Start if message in Stage Init is exchanged.

Challenge-Response Stage:
For every $j \in [N]$, do the following:

- Challenge : R sends a random ℓ-bit challenge string $v_j = v_{1,j}, \ldots, v_{\ell,j}$.
- Response : $\forall i \in [\ell]$, if $v_{i,j} = 0$, C opens $A^0_{i,j}$, else it opens $A^1_{i,j}$ by sending the decommitment information.

A **slot**$_j$ of the commitment scheme consists of the receiver's Challenge and the corresponding committer's Response message. Thus, in this protocol, there are N slots.
We say that the protocol has reached End when Challenge-Response Stage is completed and is accepted by R.

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

This completes the description of $\langle C, R\rangle$ which is an $\mathcal{O}(N)$ round protocol. The commit phase is said to the *valid* iff there exists an opening of commitments such that the open phase is accepted by an honest receiver.

Having defined the commitment protocol, we will describe a simulator $\mathcal{S}$ for the protocol Π that uses a rewinding schedule to "simulate" the view of the adversary. For this, we would like to prove an extraction lemma similar to [PRS02, PPS$^+$08] for the protocol Π, i.e., in every execution whenever a *valid* commit phase ends such that the adversary is playing the role of the committer, our simulator (using rewinding) would be able to extract the *preamble secret* with all but negligible probability. Moreover, we would like to guarantee that if the honest execution has total cost[4] C, then the cost incurred by our simulator is only $C(1 + \epsilon(N, \kappa))$, where is ϵ is a small fraction.

3.3 Description of the simulator

We describe a new "cost-based" recursive rewinding strategy. We begin by giving some preliminary definitions that will be used in the rest of the paper.

[4]Cost of an execution is the total cost of all the messages sent and received.

A thread of execution (consisting of the views of all the parties) is a perfect simulation of a prefix of an actual execution. In particular, the *main thread*, is a perfect simulation of a complete execution, and this is the execution thread that is output by the simulator. In addition, our simulator will also make other threads by rewinding the adversary to a previous state and continuing the execution from that state. Such a thread shares a (possibly empty) prefix with the previous thread. We call the execution on this thread which is not shared with any of the previous threads as a *look-ahead thread*.

We now first give an overview of the main ideas underlying our simulation technique and then proceed to give a more formal description.

Overview. Consider the main thread of execution. At a high level, we divide this thread into multiple parts referred to as "sets" consisting of possibly many protocol messages. The way we define our sets is similar to previous rewinding strategies [PRS02, PPS+08]. Essentially if the entire execution has cost c, then we divide the entire main thread into two sets of cost $c/2$ each, where cost of a set is the total cost of the messages contained in that set. Next, we divide each of these sets into two subsets, each of cost $c/4$. We continue this process recursively till we have c sets, each of unit cost[5]. Note that if each message is of unit cost, then this dividing strategy is exactly identical to [PRS02].[6] The novel idea underlying our rewinding technique is that unlike [PRS02, PPS+08], our simulator only rewinds a small subset of these sets while still guaranteeing extraction. In other words, unlike [PRS02, PPS+08], ours is a "sparse" rewinding strategy.

We now describe our rewinding strategy by using an analogy to the set covering problem [CLRS09]. Recall that in set covering problem, there is a universe of elements and sets. Each set contains some elements and has a fixed cost. The goal is to choose a minimum cost collection of these sets which covers all the elements in the universe. In our setting, we think of each session as an element in the universe. If there are m concurrent sessions $\{1, 2, \ldots, m\}$, we have m elements in our universe. Now consider the sets defined above in our setting. A set is said to cover an element i if it contains a complete slot of session i. Recall that the cost of a set is the sum of the cost of messages in this set. We want to consider a minimum cost collection $\mathcal{C}$ of these sets which together covers all the elements in the universe (i.e. all the sessions). Intuitively, we wish to rewind only the sets in $\mathcal{C}$. At a high level, this is the strategy adopted by our simulator. Due to reasons as discussed in Section 1.2, we adopt a slightly modified strategy in which the collection $\mathcal{C}$ is picked via a randomized strategy. Recall that for all i there 2^i sets with cost $c/2^i$. Very briefly, for each collection of sets which have same cost, we pick a fixed small fraction of these sets. We will prove that using this strategy we will cover each session $\omega(\log \kappa)$ times in order to guarantee extraction. As we will see later on, with this strategy, we are able to guarantee that the simulator performs extraction with all but negligible probability while incurring a small overhead.

Formal description of the simulator We begin by introducing some notation and terminology. Let C be the total cost of main execution[7]. Without loss of generality, let $C = 2^x$ for some $x \in \mathbb{N}$. Let $p(\kappa) = \omega(\log \kappa)$, and $q(\kappa) = \omega(1)$. Recall that N is the number of challenge-response slots in $\langle C, R \rangle$.

Thread at recursion level RL. We say that the main thread belongs to recursion level 0. The look-ahead threads which fork off the main thread are said to be at recursion level 1. Recursively, we say that look-ahead threads forking off a thread at recursion level RL belong to recursion level $(\mathsf{RL} + 1)$.

[5]Note that due to this dividing strategy, we allow a message to be "divided" across multiple sets.

[6]If cost of a message is the time taken by the adversary to generate that message, then this dividing strategy is exactly identical to [PPS+08].

[7]This cost C is always bounded by some polynomial in κ, i.e., $C \leq \kappa^\alpha$ for some constant α.

Sets and set levels. Let T be the main thread or a look-ahead thread with cost c at recursion level RL. We define the sets and the set levels of T as follows: The entire thread T is defined as one set at recursion level RL and set level 0 with cost c. We denote it as $\mathsf{set}^{0}_{\mathsf{RL}}$. Now divide $\mathsf{set}^{0}_{\mathsf{RL}}$ into two sets at recursion level RL and set level 1 of cost $c/2$ each. We denote the first set as $\mathsf{set}^{1,1}_{\mathsf{RL}}$ and the second set as $\mathsf{set}^{1,2}_{\mathsf{RL}}$. Let $\mathsf{set}^{i,1}_{\mathsf{RL}}, \mathsf{set}^{i,2}_{\mathsf{RL}}, \ldots, \mathsf{set}^{i,2^i}_{\mathsf{RL}}$ be 2^i sets at set level i, each of cost $c/2^i$, where $\mathsf{set}^{i,j}_{\mathsf{RL}}$ is the j^{th} set at set level i. Divide each set $\mathsf{set}^{i,j}_{\mathsf{RL}}$ into two sets at set level $(i+1)$ each of cost $c/2^{i+1}$. We continue this recursively till we reach set level $\log c$ where each set has cost 1. This way we have $L = \log c + 1$ set levels $(0, 1, \ldots, \log c)$ with total sets $2c - 1$.

For ease of notation, we will denote each set $\mathsf{set}^{i,j}_{\mathsf{RL}}$ as a tuple $(\mathsf{s\text{-}point}, \mathsf{e\text{-}point})$ where $\mathsf{s\text{-}point}$ denotes the cost of the thread T from the start of T till the start of $\mathsf{set}^{i,j}_{\mathsf{RL}}$ and $\mathsf{e\text{-}point}$ to denote the cost of thread from the start of T till the end of $\mathsf{set}^{i,j}_{\mathsf{RL}}$. Thus by definition cost of a set $\mathsf{set}^{i,j}_{\mathsf{RL}}$ is $\mathsf{e\text{-}point} - \mathsf{s\text{-}point}$. This will help us in describing our simulation strategy.

Simulator $\mathcal{S}$. We now proceed to describe our simulation strategy which consists of procedures Simulate, PickSet and SimMsg. More specifically, $\mathcal{S}$ simply runs $\textsc{Simulate}(C, \mathsf{st}_0, \phi, 0, r_m, r_s)$ to simulate the main thread at recursion level 0 with cost C when st_0 is the initial state of $\mathcal{A}$. $\mathcal{S}$ starts with empty history of messages, i.e. $\mathsf{hist} = \emptyset$. Also, $\mathcal{S}$ uses two separate random tapes r_m and r_s to generate messages and choose sets respectively. Finally, $\mathcal{S}$ returns its output as the view of the adversary. We begin by describing these procedures in detail.

```
The $\textsc{Simulate}(c, \mathsf{st}, \mathsf{hist}, \mathsf{RL}, r_m, r_s)$ Procedure.
  1. Compute $\textsc{psets} = \{(\mathsf{s\text{-}point}_j,\ \mathsf{e\text{-}point}_j)\} \leftarrow \textsc{PickSet}(c, r)$, where $r$ is randomness of appropriate size
     from $r_s$. Update $r_s = r_s \backslash r$. Let $J = |\textsc{psets}|$.
  2. Create a list $\tilde{\textsc{psets}}$ from $\textsc{psets}$ as follows: For each entry $(\mathsf{s\text{-}point}_j,\ \mathsf{e\text{-}point}_j) \in \textsc{psets}$, initialize $\mathsf{st}_j = \perp$
     and $\mathsf{hist}_j = \perp$. Insert $(\mathsf{s\text{-}point}_j, \mathsf{e\text{-}point}_j, \mathsf{st}_j, \mathsf{hist}_j)$ into $\tilde{\textsc{psets}}$. We order the list by increasing order of
     $\mathsf{e\text{-}point}$.
  3. If $c = 1$, $(\mathsf{st}', \mathsf{hist}', r'_m, \tilde{\textsc{psets}}) \leftarrow \textsc{SimMsg}(0, 1, \mathsf{st}, \mathsf{hist}, r_m, \tilde{\textsc{psets}})$. Output: $(\mathsf{st}', \mathsf{hist}', r'_m, r_s)$.
  4. Otherwise (i.e., $c > 1$),
     • Initialize $\mathsf{ctr} = 0$.
     • While $(j < J)$
        • $(\mathsf{st}', \mathsf{hist}', r'_m, \tilde{\textsc{psets}}) \leftarrow \textsc{SimMsg}(\mathsf{ctr}, \mathsf{e\text{-}point}_j - \mathsf{ctr}, \mathsf{st}, \mathsf{hist}, r_m, \tilde{\textsc{psets}})$.
        • Set $\mathsf{ctr} = \mathsf{e\text{-}point}_j$, $r^0_m = r'_m$, $r^0_s = r_s$.
        • Let there exist $\ell$ entries $\{(\mathsf{s\text{-}point}_{j_i}, \mathsf{e\text{-}point}_{j_i}, \mathsf{st}_{j_i}, \mathsf{hist}_{j_i})\}_{i\in[\ell]}$ in $\tilde{\textsc{psets}}$ such that $\mathsf{e\text{-}point}_{j_i} =$
          $\mathsf{e\text{-}point}_j$.
        • For each $i \in [\ell]$,
          $(\mathsf{st}'_{j_i}, \mathsf{hist}'_{j_i}, r^i_m, r^i_s) \leftarrow \textsc{Simulate}((\mathsf{e\text{-}point}_{j_i} - \mathsf{s\text{-}point}_{j_i}), \mathsf{st}_{j_i}, \mathsf{hist}_{j_i}, \mathsf{RL}+1, r^{i-1}_m, r^{i-1}_s)$.
        • Set $\mathsf{hist} = \mathsf{hist}' \cup (\bigcup_i \mathsf{hist}'_{j_i})$, $\mathsf{st} = \mathsf{st}'$, $r_m = r^\ell_m$, $r_s = r^\ell_s$ and $j = j + \ell$.
     • If $(\mathsf{ctr} < c)$
        • $(\mathsf{st}', \mathsf{hist}', r'_m, \tilde{\textsc{psets}}) \leftarrow \textsc{SimMsg}(\mathsf{ctr}, c - \mathsf{ctr}, \mathsf{st}, \mathsf{hist}, r_m, \tilde{\textsc{psets}})$.
        • Update $\mathsf{hist} = \mathsf{hist}'$, $\mathsf{st} = \mathsf{st}'$, $r_m = r'_m$.
     • Output: $(\mathsf{st}, \mathsf{hist}, r_m, r_s)$.
```

Figure 1: The cost-based content oblivious simulator Simulate

Procedure SIMULATE. The procedure is used to simulate any thread T at recursion level RL of cost c. It takes the following set of inputs. (a) The cost c of thread T. (b) The state st of the adversary at the beginning of T. (c) The history hist of messages seen so far in simulation. (d) Recursion level RL of T. (e) The random tape r_m which is used to generate messages of the honest party. (f) The random tape r_s used by PICKSET to choose sets.

At a high level, SIMULATE procedure when invoked on a set of inputs $(c, \mathsf{st}, \mathsf{hist}, \mathsf{RL}, r_m, r_s)$ does the following:

1. It invokes PICKSET procedure to choose a list of sets on T, say PSETS, which it will rewind. Here each set will be denotes by the corresponding tuple $(\mathsf{s\text{-}point}, \mathsf{e\text{-}point})$.
2. Next, SIMULATE augments each entry of PSETS with two additional entries to create a new list $\tilde{\text{PSETS}}$ where each entry consists of $(\mathsf{s\text{-}point}, \mathsf{e\text{-}point}, \mathsf{st}, \mathsf{hist})$, where st is the state of the adversary and hist is the history of simulation at $\mathsf{s\text{-}point}$. State st and history hist at $\mathsf{s\text{-}point}$ are populated by the procedure SIMMSG (described below) when simulation reaches $\mathsf{s\text{-}point}$.
3. SIMULATE generates messages for the thread iteratively till the end of the thread is reached as follows:
    1. It invokes the SIMMSG procedure to generate the messages from current point of simulation to the next $\mathsf{e\text{-}point}$ of some set in $\tilde{\text{PSETS}}$.
    2. For each of the sets which end at this point, it calls SIMULATE procedure recursively to create new look-ahead threads at recursion level $\mathsf{RL} + 1$.
    3. Finally, it merges the current history of messages with messages seen on the look-ahead threads.
4. It returns $(\mathsf{st}', \mathsf{hist}', r'_m, r'_s)$, where st' is the state of the adversary at the end of the thread, hist' is the updated collection of messages, r'_m and r'_s are the unused parts of the random tapes r_m and r_s respectively.

The figure 1 gives a formal description of SIMULATE procedure.

Algorithm PICKSET. At a high level, given the main thread or a look-ahead thread T at recursion level RL with cost c, it chooses a fixed fraction of sets across all set levels of T where our simulator would rewind. More formally, on input (c, r), where c is the cost of T and r is some randomness, PICKSET(c, r) returns a list of sets PSETS$= \{(\mathsf{s\text{-}point}_j, \mathsf{e\text{-}point}_j)\}$ consisting of $\left\lfloor \frac{p(\kappa)\cdot q(\kappa)\cdot \log^3 \kappa}{N} \cdot 2^i \right\rfloor$ sets at random at set level i for every $i \in [\log c]$.

Note that the sets picked by PICKSET depend only on the cost c of the thread T and randomness r and not on the protocol messages of T.

Procedure SimMsg. This procedure generates the messages by running the adversary step by step[8], i.e. incurring unit cost at a time. It takes the following set of inputs. (a) The partial cost ctr of the current thread simulated so far. (b) The additional cost c for which the current thread has to be simulated. (c) The current state st of the adversary. (d) The history hist of messages seen so far in simulation. (e) The random tape r_m to be used to generate messages. (f) The list $\tilde{\text{PSETS}}$ of the sets chosen by PICKSET for thread T.

SIMMSG generates messages on thread T one step at a time for c steps as follows:

1. If the next scheduled message is the challenge message in an instance of $\langle C, R\rangle$, it chooses a challenge uniformly at random. Also, if the next scheduled message is some other protocol message from honest party, it uses the honest party algorithm to generate the same.

[8]We will assume that it is possible to run the adversary one step at a time. We elaborate on this in our applications.

2. If the next scheduled message is from $\mathcal{A}$, SimMsg runs $\mathcal{A}$ for one step and updates st and hist. Note that it is possible that $\mathcal{A}$ may not generate a message in one step.
3. If the current point on the thread corresponds to the $\mathsf{s\text{-}point}$ of some sets in $\tilde{\textsc{psets}}$, it updates the corresponding entries with current state st of $\mathcal{A}$ and history hist of messages.

Finally it outputs the final state st of the adversary, updated history hist of messages, unused part r'_m of random tape r_m and updated list $\tilde{\textsc{psets}}$. Procedure SimMsg is described formally in Figure 2.

The SimMsg($\mathsf{ctr}, c, \mathsf{st}, \mathsf{hist}, r_m, \tilde{\textsc{psets}}$) Procedure.
For $i = 1$ to c do the following:

- **Next scheduled message if from honest party to $\mathcal{A}$:** If the next scheduled message is a challenge message of $\langle C, R\rangle$, choose a random challenge message using randomness from r_m. Else, if the next message is some other message from honest party, send this message according to honest party algorithm using randomness from r_m. Feed this message to $\mathcal{A}$.

  Next scheduled message is from $\mathcal{A}$: If the next scheduled message is from $\mathcal{A}$, run $\mathcal{A}$ from its current state st for exactly 1 step. If an output, β, is received and if β is a response message in $\langle C, R\rangle$, store β in hist as a response to the corresponding challenge message. Update st to the current state of $\mathcal{A}$. If it is some other message of the protocol, store it in hist.
- If there exists k entries $\{(\mathsf{s\text{-}point}_{j_y}, \mathsf{e\text{-}point}_{j_y}, \mathsf{st}_{j_y}, \mathsf{hist}_{j_y})\}_{y\in[k]}$ in $\tilde{\textsc{psets}}$ such that $\mathsf{s\text{-}point}_{j_y} = \mathsf{ctr} + i$. For each $y \in [k]$ update $\mathsf{st}_{j_y} = \mathsf{st}$ and $\mathsf{hist}_{j_y} = \mathsf{hist}$.

Let r'_m be the unused part of r_m. Output: $(\mathsf{st}, \mathsf{hist}, r'_m, \tilde{\textsc{psets}})$.

Figure 2: The SimMsg Procedure

In Appendix B we prove the following two lemmas:

Lemma 1 *(Extraction lemma) Consider two parties P_1 and P_2 running polynomially many (in the security parameter) sessions of a protocol Π consisting of possibly multiple executions of the commitment scheme $\langle C, R\rangle$. Also, let one the parties, say P_2, be corrupted. Then there exists a simulator $\mathcal{S}$ such that except with negligible probability, in every thread of execution simulated by $\mathcal{S}$, if honest P_1 accepts a commit phase of $\langle C, R\rangle$ as valid, then at the point when that commit phase is concluded, $\mathcal{S}$ would have already extracted the preamble secret committed by the corrupted P_2.*

Lemma 2 *Let C be the cost of the main thread. Then the cost incurred by our simulator is bounded by $C \cdot (1 + \frac{(\log^* \kappa)^2 \log C \log^4 \kappa}{N})$ when $\langle C, R\rangle$ has $N \geq \log^6 \kappa$ slots.*

4 Our Results

We now state the main results in this paper. Due to lack of space, here we only state the theorem statements, and refer the reader to different sections of the appendix for all the details.

Positive Results. As the main result of this paper, we construct an $\mathcal{O}(N)$ round protocol Π that ϵ-securely realizes any (efficiently computable) functionality $\mathcal{F}$ in the joint leaky ideal world model for any $\epsilon > 0$. More formally, we show the following:

Theorem 6 *Assume the existence of 1-out-of-2 oblivious transfer protocol secure against honest but curious adversaries. Then for every polynomial poly(κ) such that $\epsilon = 1/poly(\kappa)$, for any functionality $\mathcal{F}$, there exists an $\mathcal{O}(N)$ round protocol Π that ϵ-securely realizes $\mathcal{F}$ in the joint leaky ideal world model, where $N = \frac{(\log^6 \kappa)}{\epsilon}$.*

More generally, assume the existence of 1-out-of-2 oblivious transfer protocol secure against honest but curious adversaries and collision resistent hash functions. Then for any $\epsilon > 0$, for any functionality $\mathcal{F}$, there exists an $\mathcal{O}(N)$ round protocol Π that ϵ-securely realizes $\mathcal{F}$ in the joint leaky ideal world model, where $N = \frac{(\log^6 \kappa)}{\epsilon}$.

Protocol Π is essentially the protocol of [GJO10] instantiated with N-round concurrently-extractable commitment scheme described earlier in the paper. The security analysis of the protocol is done using the simulation technique described earlier.

Negative Results. We also present strong impossibility results for achieving security in both the individual and joint leaky ideal world model. First, we prove the following result:

Theorem 7 *There exists a functionality f such that no protocol Π ϵ-securely realizes f in the individual leaky ideal world model for $\epsilon = \frac{1}{2} - \delta$, where δ is any constant fraction.*

Additionally, we prove a lower bound on the round-complexity of protocols for achieving ϵ-security in the joint leaky ideal world model, with respect to black-box simulation. Specifically, we prove the following result:

Theorem 8 *Let ϵ be any inverse polynomial. Assuming dense cryptosystems, there exists a functionality f that cannot be ϵ-securely realized with respect to black-box simulation in the joint leaky ideal world model by any $\frac{\log(\kappa)}{\epsilon}$ round protocol.*

References

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

[BCH12] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In *TCC*, pages 266–284, 2012.

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

[BGJ+12] Elette Boyle, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, and Amit Sahai. Secure computation against adaptive auxiliary information. In *Manuscript*, 2012.

[Blu87a] Manual Blum. How to prove a theorem so no one else can claim it. In *International Congress of Mathematicians*, pages 1444–1451, 1987.

[Blu87b] 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.

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

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

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

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

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Eurocrypt '03*, 2003.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. 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`.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

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

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, 2009.

[DP07] Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *FOCS*, pages 227–237, 2007.

[GGJ13] Vipul Goyal, Divya Gupta, and Abhishek Jain. What information is leaked under concurrent composition? In *CRYPTO*, pages 220–238, 2013.

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

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

[GJ13] Vipul Goyal and Abhishek Jain. On concurrently secure computation in the multiple ideal query model. In *EUROCRYPT*, 2013.

[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.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai. Leakage-resilient zero knowledge. In *CRYPTO*, pages 297–315, 2011.

[GKOV12] Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In *CRYPTO*, 2012.

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

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

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

[GP91] Oded Goldreich and Erez Petrank. Quantifying knowledge complexity. In *FOCS*, pages 59–68, 1991.

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

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

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

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

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

[Lin03a] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *Proc.* 35*th STOC*, pages 683–692, 2003.

[Lin03b] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403, 2003.

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

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

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

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

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

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

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

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *STOC*, pages 232–241, 2004.

[PPS$^+$08] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Precise concurrent zero knowledge. In *EUROCRYPT*, pages 397–414, 2008.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *FOCS*, 2003.

[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.

[PTV12] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent zero knowledge, revisited. In *Manuscript*, 2012.

[PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 553–570. Springer, 2008.

[RK99a] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, pages 415–432, 1999.

[RK99b] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT*, pages 415–431, 1999.

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

A Building Blocks

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

A.1 Statistically Binding String Commitments

In our protocol, we will use a (2-round) statistically binding string commitment scheme, e.g., a parallel version of Naor's bit commitment scheme [Nao91] based on one-way functions. For simplicity of exposition, in the presentation of our results, we will actually use a non-interactive perfectly binding string commitment.[9] Such a scheme can be easily constructed based on a 1-to-1 one way function. Let $\textsc{com}(\cdot)$ denote the commitment function of the string commitment scheme. For simplicity of exposition, in the sequel, we will assume that random coins are an implicit input to the commitment function.

A.2 Statistically Witness Indistinguishable Arguments

In our construction, we shall use a statistically witness indistinguishable (SWI) argument $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$ for proving membership in any **NP** language with perfect completeness and negligible soundness error. Such a scheme can be constructed by using $\omega(\log k)$ copies of Blum's Hamiltonicity protocol [Blu87a] in parallel, with the modification that the prover's commitments in the Hamiltonicity protocol are made using a statistically hiding commitment scheme [NOVY98, HHK$^+$05].

[9] It is easy to see that the construction given in Section C.1 does not necessarily require the commitment scheme to be non-interactive, and that a standard 2-round scheme works as well. As noted above, we choose to work with non-interactive schemes only for simplicity of exposition.

A.3 Semi-Honest Two Party Computation

We will also use a semi-honest two party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ that emulates the functionality $\mathcal{F}$ (as described in section 2) in the stand-alone setting. The existence of such a protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ follows from [Yao86, GMW87, Kil88].

A.4 Concurrent Non-Malleable Zero Knowledge Argument

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

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

In our main construction, we will use a specific CNMZK protocol, denoted $\langle P, V \rangle$, based on the CNMZK protocol of Barak et al. [BPS06] to guarantee non-malleability. Specifically, we will make the following two changes to Barak et al's protocol: (a) Instead of using an $\omega(\log \kappa)$-round extractable commitment scheme [PRS02], we will use the N-round extractable commitment scheme $\langle C, R \rangle$ (described in Section 3.2). (b) Further, we require that the non-malleable commitment scheme being used in the protocol be public-coin w.r.t. receiver[10]. We now describe the protocol $\langle P, V \rangle$.

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

PREAMBLE PHASE.

1. P and V engage in execution of $\langle C, R \rangle$ (Section 3.2) where V commits to a random string σ.

POST-PREAMBLE PHASE.

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

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

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

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

5. P now proves the following statement to V using SZKAOK:
   (a) *either* the value committed to in step 4 is a valid witness to x (i.e., $R(x, w) = 1$, where w is the committed value), *or*
   (b) the value committed to in step 2 is the trapdoor secret σ.

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

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

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

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

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

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

A.5 Intrusion Resilient Secret Sharing Schemes

Our impossibility result in the independent leaky ideal world model makes use of the Intrusion-Resilient Secret Sharing (IRSS) scheme of [DP07]. Here, we give a short description of its syntax and the adversarial model, and the security properties that we will use. For simplicity of exposition, the description below is not as general as in [DP07], and is instead tailored to our specific application.

We start by giving the functional definition of IRSS.

Definition 1 *An* intrusion-resilient secret sharing (IRSS) *scheme* $\mathsf{IRSS}_{n,\ell}$ *is a protocol between a dealer and a set of players* $\mathcal{P} = \{P_0, \cdots, P_{n-1}\}$). *It consists of the following two algorithms:*

- $\mathsf{Share}_{n,\ell}$ *is a randomized algorithm that takes as input a secret value* X *and returns a set of shares* $\{X_1, \cdots, X_n\}$ *(where each* X_i *is of some fixed length* κ*), as well as a "reconstruction" sequence* $R = (r_1, \cdots, r_\ell)$ *(where each* $r_i \in \{1, \ldots, n\}$*. The algorithm is executed by the dealer that later sends* X_i *to player* P_i*.*

- $\mathsf{Rec}_{n,\ell}$ *is a deterministic algorithm that takes as input the shares* $\{X_1, \cdots, X_n\}$ *and the sequence* $R = (r_1, \cdots, r_\ell)$ *(produced by the* Share *algorithm) and returns the secret* X*. The algorithm* $\mathsf{Rec}_{n,\ell}$ *consists of* $\ell - 1$ *rounds, where in the i-th round (for* $i = 1, \cdots, \ell - 1$*) player* P_{r_i} *sends a message to player* $P_{r_{i+1}}$*. The output is computed by* P_{r_ℓ}*.*

Definition 2 *Let* $||$ *denote concatenation. Let* $X = (x_1, \ldots, x_n)$ *be a sequence. We say that* X *is a* subsequence *of sequence* Y*, if there exist (possibly empty) sequences* $Y_1, \ldots, Y_{k+1}$ *such that* $Y = Y_1||x_1||\ldots||Y_k||x_n||Y_{k+1}$*. We also say that* Y *contains* X*.*

We now give the adversarial model for an IRSS from [DP07]. For convenience, we will abuse notation and drop parameters from notation when they are fixed.

Adversarial Model. Let $\mathsf{IRSS}_{n,\ell} = (\mathsf{Share}_{n,\ell}, \mathsf{Rec}_{n,\ell})$ be an IRSS scheme as above. Consider an adversary $\mathcal{A}$ that plays the following game against an oracle Ω. The oracle runs $\mathsf{Share}(X)$ to obtain the shares $X_1, \cdots X_n$. Now, the adversary can issue an (adaptively chosen) sequence $\mathsf{leak}_1, \cdots, \mathsf{leak}_w$ of leakage requests. Each request leak_j is a pair (c_j, L_j), where $c_j \in \{1, \ldots, n\}$, and $L_j : \{0,1\}^n \to \{0,1\}^{s_j}$ is an arbitrary function (represented as a circuit). Note that this circuit can have the history of the adversary hardwired into it as input. On input $\mathsf{leak}_j = (c_j, L_j)$, Ω replies with $L_j(X_{c_j})$. We will say that the adversary chose a leakage sequence $\mathcal{C} = (c_1, \cdots, c_w)$.

An adversary $\mathcal{A}$ is a *valid* adversary if the sequence R (that was output by the share algorithm) is not a subsequence of $\mathcal{C}$. Finally $\mathcal{A}$ outputs a guess Y. We say that valid adversary $\mathcal{A}$ breaks the scheme with advantage γ if $\Pr[Y = X] = \epsilon$. When we consider adversaries in the IRSS scheme, we only consider valid adversaries. We now define a βn bounded adversary.

Definition 3 *An adversary $\mathcal{A}$ is βn-bounded, if the corruption sequence $\mathcal{C} = (c_1, \cdots, c_w)$ chosen by $\mathcal{A}$ satisfies the following: $\sum_{j=1}^{w} s_j \leq \beta n$, where s_j is the output length of L_j.*

We define the security of an IRSS scheme.

Definition 4 *An IRSS scheme $\mathsf{IRSS}_{n,\ell}$ is $(\gamma, \beta n)$-secure if every βn bounded valid adversary $\mathcal{A}$ breaks IRSS_a with advantage at most γ.*

Parameters for Impossibility Result in Section E. For our impossibility result, we will work with parameters $n = 3$, $\ell = 5$. Moreover, for convenience, we simply fix the "reconstruction" sequence R to be $(1, 2, 3, 1, 2)$. We now claim the following simple lemma:

Lemma 3 *Let X_1, X_2, X_3 denote the secret shares of a random secret X output by the* Share *algorithm. Let $\mathcal{A}$ be an adversary who makes (at most) three leakage queries $\mathsf{leak}_1, \mathsf{leak}_2, \mathsf{leak}_3$, where at most one query $\mathsf{leak}_i = (c_i, L_i)$ is such that $L_i(X_{c_i}) = X_{c_i}$ (i.e., L_i is the identity function), while the other queries $\mathsf{leak}_j = (c_j, L_j)$ leak at most a constant (< 1) fraction of the share X_{c_j}. Then, $\mathcal{A}$ is a valid adversary for $\mathsf{IRSS}_{3,5}$.*

The above lemma follows immediately from the description of $\mathcal{A}$. Specifically, since at most one leakage query $\mathsf{leak}_i = (c_i, L_i)$ reveals a secret share entirely, the "residual" reconstruction sequence (obtained after deleting each occurrence of c_i from R) is still of length at least 3. Now, since $\mathcal{A}$ only makes at most 2 more leakage queries, we have that R cannot be a subsequence of $\mathcal{A}$'s leakage sequence. Therefore $\mathcal{A}$ is valid.

In the rest of this paper, we will simply refer to the IRSS scheme as IRSS, with the parameters $n = 3$, $\ell = 5$ fixed, and omitted from the subscript.

A.6 One Time Programs

A one-time program (OTP) [GKR08, GIS+10] for a function f allows a party to evaluate f on a single input x chosen by the party dynamically. Thus, a one-time program guarantees that no efficient adversary, after evaluating the one-time program on some input x, can learn anything beyond $(x, f(x))$. In particular he gains no information about $f(y)$ for any $y \neq x$ outside of the information provided by $f(x)$. OTPs are *non interactive*; an OTP corresponding to some function f, $\mathsf{OTP\text{-}msg}_f$ given to a user by some environment (or user) U allows the user to evaluate f *once* on some input of its choice, with no further interaction with U. It is shown in [GKR08, GIS+10] that OTPs cannot be implemented by software alone. Thus, an OTP is typically implemented as a software plus hardware package, where the hardware is assumed to be secure (that is, can only be accessed in a black box way via its interface), and the software can be read and tampered with. The secure hardware devices that are used are called *one time memory* devices (OTM).

An OTM is a memory device initialized by two keys $\mathsf{key}^0, \mathsf{key}^1$ which takes as input a single bit b, outputs key^b and then "self destructs". OTMs were inspired by the oblivious transfer protocol, and indeed, for our purposes, we will find it necessary to use oblivious transfer in place of one time memory tokens. This approach is not new and has been used in previous works as well [GIS+10]. Thus, for our setting, we will define one-time programs in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model directly for ease of use. We state this definition in terms of an *asynchronous, non-interactive* protocol (as defined above) for an asymmetric "one-time program functionality" $\mathcal{F}_{\mathsf{OTP}}$, which accepts a circuit f from the party P_1 and an input x from the party P_2, and returns $f(x)$ to P_2 (and notifies P_1) (see Figure 3). The protocol is required to be secure only against corruption of P_2. Note that the requirement of being non-interactive makes this non-trivial (i.e., P_2 cannot send its input to P_1 and have it evaluated).

Ideal Functionality of One Time Program
In Figure 3, we describe the ideal functionality of a One Time Program.

Functionality $\mathcal{F}^{\mathsf{OTP}}$

Create Upon receiving $(\mathsf{sid}, \mathsf{create}, P_1, P_2, f)$ from P_1 (with a fresh session ID sid), where f is a circuit for a function, do:

1. Send $(\mathsf{sid}, \mathsf{create}, P_1, P_2, \mathsf{size}(f))$ to P_2, where the size outputs the size of the circuit and the number of input wires it has.
2. Store (sid, P_2, f).

Execute Upon receiving $(\mathsf{sid}, \mathsf{run}, x)$ from P_2, find the stored tuple (sid, P_2, f) (if no such tuple exists, do nothing). Send $(\mathsf{sid}, \mathsf{value}, f(x))$ to P_2 and delete tuple (sid, P_2, f). (If x is not of the right size, define $f(x) = \perp$.) Send $(\mathsf{sid}, \mathsf{evaluated})$ to P_1.

Figure 3: Ideal functionality for One-time Program.

Definition 5 (One-Time Program) *(Adapted from [GKR08, GIS+10]) A* one-time program (OTP) scheme *is an asynchronous two-party non-interactive protocol $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ (in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model) that UC-securely realizes $\mathcal{F}_{\mathsf{OTP}}$ (as defined in Figure 3) when the adversary is allowed to corrupt only P_2.*

In other words, if $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ is a one-time program scheme, then for every probabilistic polynomial time adversary $\mathcal{A}$ corrupting P_2, and adaptively scheduling the $\mathcal{F}_{\mathsf{OT}}$ sessions in the real-model execution of Π, there exists a probabilistic polynomial time ideal-world adversary $\mathsf{Sim}_{\mathsf{OTP}}$ (the simulator), such that for every environment $\mathcal{Z}$, it holds that $\mathsf{Ideal}^{f}_{\mathsf{Sim}_{\mathsf{OTP}},\mathcal{Z}} \overset{c}{\equiv} \mathsf{Real}^{\Pi}_{\mathcal{A},\mathcal{Z}}$, where $\mathsf{Ideal}^{f}_{\mathsf{Sim}_{\mathsf{OTP}},\mathcal{Z}}$ and $\mathsf{Real}^{\Pi}_{\mathcal{A},\mathcal{Z}}$ denote the outputs produced by $\mathsf{Sim}_{\mathsf{OTP}}$ and $\mathcal{A}$ respectively.

We shall refer to an execution of an OTP scheme with input f for P_1 as an OTP *for f.*

We note that OTPs exist if a (standalone secure) Oblivious Transfer protocol exists [GIS+10], which in turn exists if a concurrent secure OT protocol exists. Thus, for proving the impossibility of concurrent secure OT protocols, we can assume the existence of OTPs "for free."

Intuitively, this means that for every probabilistic polynomial time algorithm $\mathcal{A}$ given access to a one-time program (or non interactive protocol Π) for some function f, there exists another probabilistic polynomial time algorithm $\mathsf{Sim}_{\mathsf{OTP}}$ which is given one time black box access to f, i.e. it can request to see the value $f(x)$ for any x of its choice, such that (for any f) the output distributions of $\mathsf{Sim}_{\mathsf{OTP}}$ (denoted by $\mathsf{Ideal}^{f}_{\mathsf{Sim}_{\mathsf{OTP}}}$) and $\mathcal{A}$ (denoted by $\mathsf{Real}^{\Pi}_{\mathcal{A}}$) are computationally indistinguishable, even to a machine that knows f.

OTP simulator : In the real world, the output of the adversary $\mathcal{A}$ consists of a one time program $\mathsf{OTP\text{-}msg}^{\mathsf{Real}}$ for a function f of circuit size ℓ (say), as well as one input to $\mathsf{OTP\text{-}msg}^{\mathsf{Real}}$, denoted by $\mathsf{keys}^{\mathsf{Real}} = \{(\mathsf{key}^{\mathsf{Real}}_1, \ldots \mathsf{key}^{\mathsf{Real}}_k\}$ (for k input wires). Thus, $\mathsf{Real}^{\Pi}_{\mathcal{A},\mathcal{Z}} = \{\mathsf{OTP\text{-}msg}^{\mathsf{Real}}, \mathsf{keys}^{\mathsf{Real}}\}$.

We describe how our OTP simulator (denoted by $\mathsf{Sim}_{\mathsf{OTP}}$) interacts with some environment (or user), say U.

1. U gives $\mathsf{Sim}_{\mathsf{OTP}}$ a circuit size ℓ, $\mathsf{Sim}_{\mathsf{OTP}}$ returns a one time program $\mathsf{OTP\text{-}msg}^{\mathsf{Ideal}}$ for circuit size ℓ.
2. Let us assume that $\mathsf{OTP\text{-}msg}^{\mathsf{Ideal}}$ has k input wires, and denote these inputs by $\mathsf{key}^{\mathsf{Ideal}}_1, \ldots, \mathsf{key}^{\mathsf{Ideal}}_k$. U may query $\mathsf{Sim}_{\mathsf{OTP}}$ for the inputs to be provided to $\mathsf{OTP\text{-}msg}^{\mathsf{Ideal}}$. Then, $\mathsf{Sim}_{\mathsf{OTP}}$ returns inputs $\{\mathsf{key}^{\mathsf{Ideal}}_1, \ldots, \mathsf{key}^{\mathsf{Ideal}}_t\}$ for the first t bits, for some $t < k$ (we refer to the inputs as keys since they are similar to the keys that are input to garbled circuits).
3. After returning t inputs, $\mathsf{Sim}_{\mathsf{OTP}}$ produces a value, say x, and queries U for $f(x)$, i.e. the desired output of $\mathsf{OTP\text{-}msg}^{\mathsf{Ideal}}$ on x.
4. U provides the output, at which point $\mathsf{Sim}_{\mathsf{OTP}}$ returns the values for the remaining input wires $\{\mathsf{key}^{\mathsf{Ideal}}_{t+1}, \ldots, \mathsf{key}^{\mathsf{Ideal}}_k\}$. We will denote $\mathsf{keys}^{\mathsf{Ideal}} = \{\mathsf{key}^{\mathsf{Ideal}}_1, \ldots, \mathsf{key}^{\mathsf{Ideal}}_k\}$.
5. Output $\mathsf{Ideal}^{f}_{\mathsf{Sim}_{\mathsf{OTP}}} = \{\mathsf{OTP\text{-}msg}^{\mathsf{Ideal}}, \mathsf{keys}^{\mathsf{Ideal}}\} \overset{c}{\equiv} \{\mathsf{OTP\text{-}msg}^{\mathsf{Real}}, \mathsf{keys}^{\mathsf{Real}}\} = \mathsf{Real}^{\Pi}_{\mathcal{A},\mathcal{Z}}$.

A.7 Leakage-Resilient Signatures

Our definition of leakage resilient signatures is essentially the standard notion of existentially unforgeability under adaptive chosen message attacks, except that we allow the adversary to specify an arbitrary function $L(\cdot)$ (whose output length is bounded by the leakage parameter), and obtain the value of L applied to the signing key. Let κ be the security parameter, and $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ denote a signature scheme. Let ℓ denote the leakage parameter. In order to define leakage resilient signatures, we consider the following experiment.

1. Compute $(sk, vk) \leftarrow \mathsf{Gen}(1^{safe}, \ell)$ and give vk to the adversary.
2. Run the adversary $\mathcal{A}(1^{safe}, vk, \ell)$. The adversary may make adaptive queries to the signing oracle $\textsc{Sign}_{sk}(\cdot)$ and the leakage oracle $\textsc{Leak}_{sk}(\cdot)$, defined as follows:
   - On receiving the i^{th} query s_i, $\textsc{Sign}_{sk}(s_i)$ computes $\sigma_i \leftarrow \mathsf{Sign}_{sk}(s_i)$ and outputs σ_i.
   - On receiving an input L (where L is a polynomial-time computable function, described as a circuit), $\textsc{Leak}_{sk}(L)$ outputs $L(vk)$ to $\mathcal{A}$. The adversary is allowed only a single query to the leakage oracle. It can choose any function $L(\cdot)$ whose output length is bounded by $\{0,1\}^{\ell\cdot|sk|}$.
3. At some point, $\mathcal{A}$ stops and outputs (s^*, σ^*).

We say that $\mathcal{A}$ *succeeds* if (a) $\mathsf{Verify}_{vk}(s^*, \sigma^*) = 1$, and (b) s^* was never queried to $\textsc{Sign}_{sk}(\cdot)$.

Definition 6 *A signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be ℓ-leakage resilient if all polynomial-time adversaries $\mathcal{A}$ can succeed with only negligible probability in the above experiment.*

Theorem 9 ([KV09]) *Assuming dense public-key cryptosystems, there exists a $(1 - o(1)$-leakage-resilient signature scheme.*

B Proof of Extraction of Trapdoor

In this section, we will prove that if the commit phase of $\langle C, R\rangle$ of any session concludes on any thread of execution such that it is valid, then our simulator would have extracted the preamble secret committed by the adversary in that session with all but negligible probability. In particular, we prove the following lemma.

Lemma 4 *(Extraction lemma) Consider two parties P_1 and P_2 running polynomially many (in the security parameter) sessions of a protocol Π consisting of possibly multiple executions of the commitment scheme $\langle C, R\rangle$. Also, let one the parties, say P_2, be corrupted. Then there exists a simulator $\mathcal{S}$ such that except with negligible probability, in every thread of execution simulated by $\mathcal{S}$, if honest P_1 accepts a commit phase of $\langle C, R\rangle$ as valid, then at the point when that commit phase is concluded, $\mathcal{S}$ would have already extracted the preamble secret committed by the corrupted P_2.*

For simplicity of exposition, we will prove the above lemma assuming that the protocol Π has only one execution of $\langle C, R\rangle$. However, we stress that our analysis is robust to having multiple executions of $\langle C, R\rangle$ in Π. Similar to the analysis in [PRS02], in order to prove the above lemma for any thread and any session, we will prove the claim for a fixed thread of execution T^* of cost c^* and for a fixed session s^* occurring on T^*, which we will henceforth call *target thread* and *target session* respectively, and call all other threads and sessions "auxiliary". Since there are only polynomially many threads and polynomially many sessions (since C is polynomial in κ), Lemma 4 follows by taking a union bound of failure probabilities over all threads of execution and sessions. Also note that if the cost of the main thread is C, then $c^* \leq C$.

We now give an outline of remainder of the proof. The proof has two parts as described below:

1. **Analysis of PickSet.** Recall the analogy of our problem to set covering problem described earlier. In this part, at a high level, we will prove that among the sets chosen by PickSet for the target thread T^* there are $\omega(\log \kappa)$ sets of equal cost which "cover" the target session s^*. In this direction, first note that any thread of execution T can have parts at various recursion levels. We will first prove that there is a "part" of the target thread T^* at recursion level RL, say T^*_{RL}, containing sufficiently many slots of the target session s^*. Next, we will show that there are many sets in T^*_{RL} which contain a slot of the target session s^*. Moreover, we will show that a large number of these sets will infact belong to the same set level, say j^*, and hence have equal cost. Finally, we will prove that since PickSet chooses sets at random, it will pick at least $p(\kappa)$ sets at set level j^* which "cover" the target session s^*.

2. **Swapping Argument.** In this part we will show that by rewinding each of the sets picked by PickSet, the simulator will be able to extract the preamble secret committed by $\mathcal{A}$ in session s^* with all but negligible probability. We will first fix a "part" T^*_{RL} of T^* and a set level j^* and argue successful extraction for session s^* on T^* which has many sets in T^*_{RL} on set level j^*. As shown in first part, in this case, PickSet will choose at least $p(\kappa)$ sets at set level j^* which "cover" the target session s^*. Now, we will use swapping argument similar to [PRS02, PTV12] to show that the simulator is able to extract the preamble secret with all but negligible probability. In particular, we show that for any bad random tape (on which the extraction fails) there are super polynomial number of good tapes (on which the extraction succeeds) and the good tapes corresponding to any two bad tapes are disjoint. In this argument, we crucially use the fact the sets we consider (which cover s^*) lie on the same set level. Finally, by taking a union bound over different parts of T^* and different set levels, we argue successful extraction of preamble secret for the target session s^* on the target thread T^*.

B.1 Analysis of PickSet

We begin by introducing some terminology and proving certain properties about the target session s^* on the target thread T^*. We define a section of thread T^* as follows:

Definition 5 *Let thread T^* consists of a sequence of parts $T_0^*, T_1^*, \ldots, T_D^*$ such that each T_i^* is a prefix of a look-ahead thread at recursion level i and D is the maximum depth of recursion for the target thread. Then we call each T_i^*, a* section *of T^*.*

Note that the slots of the target session can be spread across all of these sections and some slots can even be spread across multiple sections. More precisely, there can be slots, say $\mathsf{slot}_j = (\mathsf{ch}_j, \mathsf{rsp}_j)$, such that ch_j appears on T_ℓ^* and rsp_j appears on $T_{\ell'}^*$, where $\ell \neq \ell'$. By counting argument and the fact that all slots of a session are disjoint, there can be at most D slots which span across multiple sections.

Claim 1 *$\exists$ recursion level RL such that there are $n \geq (N-D)/D$ slots of target session in T_{RL}^*.*

Proof. It follows trivially from pigeon hole principle. ■

Note that T_{RL}^* might not be the complete look-ahead thread at recursion level RL. Let $\widetilde{T_{\mathsf{RL}}^*}$ be the complete look-ahead thread at recursion level RL of which T_{RL}^* is a prefix. Recall that for any thread at recursion level RL with cost c, there are $(\log c)$ set levels, such that at set level j, there are 2^j sets $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j},1}, \ldots, \mathsf{set}_{\mathsf{RL}}^{\mathsf{j},2^{\mathsf{j}}}$ with cost $c/2^j$. We now define the following:

Definition 7 *(*covering *the session s^*) Consider the thread $\widetilde{T_{\mathsf{RL}}^*}$ and the target session s^*. We say that a set $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$ of $\widetilde{T_{\mathsf{RL}}^*}$* covers *the target session s^* if it contains a slot of s^*.*

Lemma 5 *If a session s has γ slots in a set $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$ of $\widetilde{T_{\mathsf{RL}}^*}$, then s is* covered *by at least γ sets of $\widetilde{T_{\mathsf{RL}}^*}$. Moreover these sets are subsets of $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$.*

Proof. We will prove this lemma by induction on γ. If $\gamma = 1$, then s is covered by set $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$. Let us assume that the lemma 5 holds for all $\gamma \leq \gamma'$. Now let $\gamma = \gamma' + 1$. Now since there is more than one slot in $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$, its cost is more than unity. Hence, there exists subsets $\mathsf{set}^{\mathsf{left}}$ and $\mathsf{set}^{\mathsf{right}}$ of $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$ at set level $j+1$ of $\widetilde{T_{\mathsf{RL}}^*}$. Let $\mathsf{set}^{\mathsf{left}}$ contain γ_1 slots and $\mathsf{set}^{\mathsf{right}}$ contain γ_2 slots of s. By induction hypothesis, s is covered by γ_1 subsets of $\mathsf{set}^{\mathsf{left}}$ and γ_2 subsets of $\mathsf{set}^{\mathsf{right}}$ from sets of $\widetilde{T_{\mathsf{RL}}^*}$. Moreover, subsets of $\mathsf{set}^{\mathsf{left}}$ and $\mathsf{set}^{\mathsf{right}}$ are disjoint and both are subsets of $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$. Also, $\gamma_1 + \gamma_2 \geq \gamma - 1$ because while dividing $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$ into two subsets we can destroy at most one slot. Also, note that s is covered by $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$. Hence, s is covered by $\gamma_1 + \gamma_2 + 1 \geq \gamma$ sets of $\widetilde{T_{\mathsf{RL}}^*}$ which are subsets of $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$. ■

Corollary 1 *The target session s^* is* covered *by n sets of $\widetilde{T_{\mathsf{RL}}^*}$.*

Consider any set $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}} = (\mathsf{s\text{-}point}, \mathsf{e\text{-}point})$ of $\widetilde{T_{\mathsf{RL}}^*}$. We say that $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$ is contained in T_{RL}^* if both s-point and e-point are in T_{RL}^*. Now we define the term special cover and prove the following lemma which relates the number of slots of session s^* to the number of sets which special cover the session s^*.

Definition 8 *(*special covering *the session s^*) Consider thread $\widetilde{T_{\mathsf{RL}}^*}$ and session s^*. We say that a set $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$ of $\widetilde{T_{\mathsf{RL}}^*}$* special covers *the session s^* if it contains a slot of s^* but does not contain* Start *or* End *of s^* and $\mathsf{set}_{\mathsf{RL}}^{\mathsf{j,k}}$ is contained in T_{RL}^*.*

Lemma 6 *If the target session session s^* has γ slots in T^*_{RL}, then there are $\gamma' \geq \gamma - 2\log C$ sets of $\widetilde{T^*_{\mathsf{RL}}}$ which* special cover *session s^*, where C is the cost of the main thread.*

Proof. If a session s^* has γ slots in T^*_{RL}, then s has γ slots in $\widetilde{T^*_{\mathsf{RL}}}$. By lemma 5, there exist γ sets of $\widetilde{T^*_{\mathsf{RL}}}$ which cover s^*. Since the cost of any thread is at most C, there are at most $\log C$ set levels of $\widetilde{T^*_{\mathsf{RL}}}$. Among the γ sets of $\widetilde{T^*_{\mathsf{RL}}}$ which cover s^*, at any set level there can be at most two sets satisfying either of the following conditions: (a) The set contains Start or End of s^*. (b) The set is not contained in T^*_{RL}. Summing over all set levels, there are at least $\gamma - 2\log C$ sets of $\widetilde{T^*_{\mathsf{RL}}}$ which special cover s^*. ■

Corollary 2 *The target session s^* is* special covered *by $n' = n - 2\log C$ sets of $\widetilde{T^*_{\mathsf{RL}}}$.*

Lemma 7 *For the target session s^*, $\exists$ set level j^* among set levels of $\widetilde{T^*_{\mathsf{RL}}}$ such that s^* is* special covered *by $n'' \geq \frac{N}{2\log^2 C}$ sets at set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$.*

Proof. For any thread, there are at most $\log C$ set levels. By pigeon hole principle and Corollary 2, there exists a set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$ such that s^* is special covered by $n'' \geq n'/\log C$ sets at set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$. Now substituting values for n', n in terms of N and using the fact that $D < \log C$, we get,

$$n'' \geq \frac{\frac{N-D}{D} - 2\log C}{\log C} \geq \frac{N - \log C - 2\log^2 C}{\log^2 C} \geq \frac{N}{2\log^2 C}$$

The last is meaningful if $N > \log^3 \kappa$ for sufficiently large value of κ. ■

Now recall from the description of the algorithm PickSet that there are some lower set levels (with less number of sets) where PickSet does not pick any set. But we argue in Lemma 8 that for the target session s^* and set level j^*, PickSet will choose at least $p(\kappa)$ sets from set level j^*. Using this, we will prove in Lemma 9 and Corollary 3 that PickSet will choose at least $p(\kappa)$ sets on set level j^* which special cover s^* with all but negligible probability.

Lemma 8 *For the target session s^*, let j^* be as defined in Lemma 7, then* PickSet *will choose at least $p(\kappa)$ sets at set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$.*

Proof. PickSet chooses $\left\lfloor \frac{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)}{N} \cdot 2^{j^*} \right\rfloor$ sets at set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$. Also, by Lemma 7, there are n'' sets at set level j^* which special cover s^*. Assume for contradiction that PickSet chooses less than $p(\kappa)$ sets at set level j^*, then $\frac{N}{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)} > \frac{2^{j^*}}{2 \cdot p(\kappa)}$. Also, since we have 2^{j^*} sets at set level j^*, $2^{j^*} \geq n''$. Combining the two inequaties we have,

$$\frac{N}{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)} > \frac{N}{4 \cdot p(\kappa) \log^2 C} \Rightarrow 4\log^2 C > q(\kappa) \cdot \log^3 \kappa \Rightarrow C > \kappa^{\sqrt{\log \kappa}}$$

But this is a contradiction since C is bounded by some polynomial in κ. ■

Now we divide the sets at set level j^* which special cover the target session into $p(\kappa)$ partitions as follows:

Definition 9 *(Partition) Given a sequence of n'' sets at set level j^* (defined in Lemma 7) which* special cover *the target session s^*, we divide them into $p(\kappa)$ parts $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{p(\kappa)}$ such that each $\mathcal{P}_\ell$ contains $n''/p(\kappa)$ consecutive sets among n'' sets. We call each $\mathcal{P}_\ell$ a partition.*

Next, if we show in lemma 9 that PickSet will pick at least one set from each partition $\mathcal{P}_\ell$ with overwhelming probability, then we have that PickSet will choose $p(\kappa)$ sets which special cover the target session s^* with overwhelming probability. Also, note that since these sets belong to the same set level they will be disjoint. This would be used crucially in the swapping argument in next section.

Lemma 9 *For the target session s^*, let j^* be as defined in Lemma 7, then for any partition $\mathcal{P}_\ell$,* PickSet *will choose at least one set in $\mathcal{P}_\ell$ with overwhelming probability.*

Proof. By Lemma 7, there is a set level j^* such that there are n'' sets at set level j^* which special cover the target session s^*. We have defined partitions for these n'' sets. Hence, all sets of partition $\mathcal{P}_\ell$ belong to set level j^*. PickSet chooses $\left\lfloor \frac{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)}{N} \cdot 2^{j^*} \right\rfloor$ sets at random at set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$. Also, by Lemma 8, $\left\lfloor \frac{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)}{N} \cdot 2^{j^*} \right\rfloor > p(\kappa)$. Let fail denote the event that PickSet does not choose any set in $\mathcal{P}_\ell$. Then,

$$\begin{aligned}
\Pr[\mathsf{fail}] &\leq \left(1 - \frac{n''/p(\kappa)}{2^{j^*}}\right)^{\left\lfloor \frac{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)}{N} \cdot 2^{j^*} \right\rfloor} \leq \left(1 - \frac{N}{2 \cdot p(\kappa) \log^2 C \cdot 2^{j^*}}\right)^{\left(\frac{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)}{N} \cdot 2^{j^*}\right) - 1} \\
&\leq \exp\left(-\frac{N}{2 \cdot p(\kappa) \log^2 C \cdot 2^{j^*}} \frac{\log^3 \kappa \cdot p(\kappa) \cdot q(\kappa) \cdot 2^{j^*}}{N} \frac{1}{2}\right) \\
&\leq \exp\left(-\frac{\log^3 \kappa \cdot q(\kappa)}{4 \log^2 C}\right) \leq \frac{1}{\kappa^{q/4\alpha^2}} \text{ is } \textit{negligible}.
\end{aligned}$$

where $C = \kappa^\alpha$ for some constant α and $q/4\alpha^2$ is $\omega(1)$. ■

Definition 10 *Let* $\text{PSETS}_{\mathsf{RL},j^*}$ *be the sets at set level j^* contained in T^*_{RL} which are chosen by* PickSet *when it is invoked for $\widetilde{T^*_{\mathsf{RL}}}$.*

Corollary 3 *There will be at least $p(\kappa)$ sets in* $\text{PSETS}_{\mathsf{RL},j^*}$ *which* special cover *session s^* with all but negligible probability.*

This follows from Lemma 9 that one set from each partition is picked with overwhelming probability and there are $p(\kappa) = \omega(\log \kappa)$ partitions.

In this section we have concluded that if the target session s^* reaches End on the target thread T^*, then there exists a section of this thread T^*_{RL} and a set level j^* such that there are $n'' \geq N/2 \log^2 C$ sets which special cover session s^*. Now, if the simulator fails to extract the preamble secret for s^* on T^* with noticeable probability then since there are only $\log C$ sections on any thread and $\log C$ set levels, there exists a section T^*_{RL} of target thread T^* and a set level j^* (where the target session has n'' sets which special cover it) such that the simulator fails to extract the preamble secret in target session s^* with noticeable probability. In the following section, we will prove that for any section T^*_{RL} and set level j^* simulator can fail only with negligible probability. By taking union bound over sections and set levels, we get that the simulator can fail to extract for this target session s^* on target thread T^* only with negligible probability. Since number sessions and threads is only a polynomial, Lemma 4 follows by taking a union bound over all sessions and threads.

B.2 Swapping Argument

In this section, we will fix a target thread T^*, a section T^*_{RL} of this thread, a set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$ (as defined in previous section) and a target session s^*. We will argue that simulator will succeed in extracting the preamble secret for session s^* if there are has $n'' \geq N/2 \log^2 C$ sets at set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$ which special cover the session s^*. Recall that we argued that for such a session s^*, there will be $p(\kappa)$ sets in $\text{PSETS}_{\mathsf{RL},j^*}$ which will special cover session s^*. In other words, among these n'' sets which special cover the session s^*, PickSet would have chosen $p(\kappa)$ sets to rewind. In this section we will argue that using only the look-ahead threads made for these sets, simulator would be able to extract the preamble secret committed by $\mathcal{A}$ with all but negligible probability.

More formally, after we have fixed the target thread as T^*, target $\mathsf{section}$ as T^*_{RL}, target set level as j^*, target session as s^*, we want to guarantee extraction for the target session s^* in the following setting:

1. Target session reaches End on the target thread T^*.
2. The commit phase of s^* is "valid".
3. Let $\widetilde{T^*_{\mathsf{RL}}}$ be the look-ahead thread at recursion level RL of which T^*_{RL} is a prefix. Then there are $n'' \geq \frac{N}{2 \cdot \log^2 C}$ sets at set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$ which $\mathsf{special\ cover}$ session s^*.

To extract the preamble secret in such a target session s^*, $\mathcal{S}$ will use the messages seen on the auxiliary threads which complete before the simulation reaches the End of s^*. In order to extract, there should be at least one convincing slot of s^* on an auxiliary thread. Recall that our simulator had two random tapes r_m and r_s. Different parts of r_m were used to generate messages on different threads. Here we will assume that our simulator always uses same amount of randomness to generate any two sequence of messages of same cost. This will be used crucially to argue extraction. Also, r_s was used by PickSet to choose sets on all the threads. Throughout the following discussion we will keep the random tape r_s fixed and talk about different random tapes for r_m. Since, $\mathcal{A}$ can be assumed to be a deterministic machine, the contents of any thread are fully determined by the state of $\mathcal{A}$ and the history of messages at the start of the thread and randomness used to generate messages by the simulator (in particular SimMsg). The forking of various look-ahead threads off any thread T and the messages seen on these look-ahead threads, does not affect the messages seen on T.

In our case, we say that the simulator $\mathcal{S}$ fails in extracting the preamble secret of the target session if with respect to $\widetilde{T^*_{\mathsf{RL}}}$ corresponding to target $\mathsf{section}$ T^*_{RL} there are $n'' \geq N/2\log^2 C$ sets at target set level j^* which $\mathsf{special\ cover}$ session s^* but none of the look-ahead threads made for sets in $\textsc{psets}_{\mathsf{RL},j^*}$ contain a convincing slot of s^*. The random tape r_m which leads to this scenario is called a bad random tape. All other tapes are called good random tapes. We will show (just like in PRS) that for each bad random tape there exist super-polynomially many good random tapes. Also, good tapes corresponding to any two bad tapes are disjoint. This will help us conclude that the probability of using a bad random tape is negligible.

Intuitively, to generate a good tape from a bad tape, we just need to "swap" a convincing slot from a set in target $\mathsf{section}$ into into an auxiliary thread. After the swapping, should the simulation reach the End of the target session on the target thread, the convincing slot on the auxiliary thread, together with the corresponding convincing slot on the target thread, will allow $\mathcal{S}$ to extract the preamble secret. To actually "swap" convincing slots, we modify the random tape r_m of $\mathcal{S}$ used to generate the messages on a set $\mathsf{set}^{\mathsf{j}^*,\mathsf{k}}_{\mathsf{RL}} \in \textsc{psets}_{\mathsf{RL},j^*}$ which $\mathsf{special\ covers}$ s^* with the randomness used for the look-ahead thread of $\mathsf{set}^{\mathsf{j}^*,\mathsf{k}}_{\mathsf{RL}}$, say, $\mathsf{set}^{\mathsf{j}^*,\mathsf{k},\mathsf{la}}_{\mathsf{RL}}$. Since, $\mathsf{set}^{\mathsf{j}^*,\mathsf{k}}_{\mathsf{RL}}$ and $\mathsf{set}^{\mathsf{j}^*,\mathsf{k},\mathsf{la}}_{\mathsf{RL}}$ start from the same state st and history hist, swapping the randomness used to generate the messages on these sets swaps the contents of these sets exactly. After swap, the convincing slot starts appearing on an auxiliary thread. Note that we do not change the randomness used to generate messages on look-ahead threads of $\mathsf{set}^{\mathsf{j}^*,\mathsf{k},\mathsf{la}}_{\mathsf{RL}}$.

Just to emphasize again r_s is fixed for all the discussion in this section. When we talk about a random tape, we mean r_m which is the randomness used to generate messages. Now we give some definitions similar to [PRS02, PTV12]. Some of the text in the following definitions and proofs have been taken verbatim from [PTV12].

Definition 6 *A set* $\mathsf{set}^{\mathsf{j}^*,\mathsf{k}}_{\mathsf{RL}}$ *is called a* block *if* $\mathsf{set}^{\mathsf{j}^*,\mathsf{k}}_{\mathsf{RL}} \in \textsc{psets}_{\mathsf{RL},j^*}$.

Definition 7 *A block B and the corresponding look-ahead thread B' are called* siblings.

Since all the blocks lie on the same recursion level RL and set level j^*, any two blocks are disjoint. Using this fact we define an order on blocks as follows:

Definition 8 *Given two blocks A and B, we say $A > B$ if A occurs before B.*

Now we define composable blocks whose randomness we will swap with their siblings to get good tapes.

Definition 9 *A block B is* composable *if the following two conditions are satisfied:*

Main Block Condition. *B* special covers *the target session s^*, i.e., contains a convincing slot of s^* and does not contain* Start *or* End *of the target session s^* and is contained in target section T^*_{RL}.*

Sibling Condition. *Look-ahead thread of B does not contain* End *of the target session s^*.*

Claim 2 *Let τ be a random tape (not necessarily* bad*). Let B be a* composable *block with sibling B' when $\mathcal{S}$ is executed with random tape τ, and let s be the common start state of B and B'. Further, let τ' be the random tape obtained after swapping the randomness of blocks B and B'. Then:*

1. *Goodness: τ' is a* good *random tape.*
2. *Composability: Any* composable *block A on τ such that $A > B$ is still* composable *on τ'.*

Proof. Recall that the simulation, i.e., the sequence of messages, on blocks B and B' are exchanged exactly after the swapping of randomness used to generate messages on B and B'.

Goodness: When simulation is done using τ', B' will be on the target section T^*_{RL} of the target thread. Since, B' has same cost as B, it will be a set at the same location as B among the sets of $\widetilde{T^*_{\mathsf{RL}}}$. Also, since we have fixed r_s, PICKSET will choose B' as one the sets in $\textsc{psets}_{\mathsf{RL},j^*}$. Since B' does not contain End, the look-ahead thread starting from s will be executed before the simulation reaches the End of the target session. Thus, the convincing slot in B (which is now on the look-ahead thread) and corresponding convincing slot on main thread together form a matching pair of convincing slots which occur before the End of the target session. Hence, τ' is a good random tape.

Composability: Given a composable block A, such that $A > B$ we have that A occurs before B because any two composable blocks are disjoint. Since, swapping the randomness of block B and B' doesn't change anything which occurs prior to B, the contents of the main thread are unchanged from the start of session till start of B under τ'. Also, the contents of the look-ahead thread corresponding to A are unchanged. Since, r_s has not changed, PICKSET on τ' will still pick A, i.e., $A \in \textsc{psets}_{\mathsf{RL},j^*}$ when messages are generated using τ'. Hence, A is still a composable block.

■

Swap **Procedure.** Let τ be a bad random tape. Assume that we are given a set of composable blocks $B_1 > B_2 > \ldots > B_q$. Then we generate a tape τ' by swapping the randomness of the composable blocks with their siblings in the order of $i = q, q-1, \ldots, 1$. Recall that when we swap the randomness of a block B_i with its sibling B_i' we only swap the randomness used to generate messages on B_i and the look-ahead thread of B_i, i.e. B_i'. All other parts of the randomness are left unchanged. By the composability condition, after swapping $B_q, B_{q-1}, \ldots, B_2$, block B_1 is still composable. Hence, by goodness property of B_1, τ' is a good random tape. We denote this by

$\mathsf{Swap}(\tau, B_1, \ldots, B_q) = \tau'$.

Note that throughout this section, we have fixed the target section as T^*_{RL} and target set level as j^*. Hence, all the procedures in this section are aware of all these parameters. Next define Undo procedure, which given a good tape τ' obtained from a bad tape τ as input returns the unique bad tape τ. This procedure will crucially use the fact that all the blocks which were swapped belong to the set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$.

Undo **Procedure.** Let τ' be a good random tape obtained as an output of the Swap procedure on a bad tape τ and a sequence of composable blocks $B_1 > B_2 > \ldots > B_q$ as described above. Let B'_i be the sibling of B_i. Note that after all the swaps, B'_1 lies on the target thread and B_1 lies on the look-ahead thread and contains a convincing slot of the target session s^*. Now we want to define a procedure Undo which on input τ' is able to identify these blocks such that we get a unique bad tape τ by swapping back these blocks in reverse order. If we can show that there is a deterministic procedure Find of identifying B_1 from τ' (a good tape) then we can swap back B_1 and keep doing this recursively till we get τ (a bad tape) and we can no longer apply this procedure. We do the following:

$\mathsf{Find}(\tau')$**.** Call a block A on τ' special if look-ahead of A contains a convincing slot of the target session and A does not contain End of the target session. Pick a special block A^* such that for any other special block X, it holds that $A^* > X$.

Claim 3 $A^* = B'_1$.

Proof. Let τ'' be the tape obtained from τ after swapping $B_q, B_{q-1}, \ldots, B_2$. B_1 is composable on τ''. Hence, B'_1 is special on τ' because B_1 now lies on the look-ahead thread and contains a convincing slot of the target session and by sibling condition for B_1, B'_1 does not contain the End of the target session. Now we need to show that there is no other special block X such that $X > B'_1$.

If $X > B'_1$ then X occurs before B'_1 on τ'. Since τ' has been obtained from the bad tape τ, even after swapping $B_1 > B_2 > \ldots > B_q$, there can be no convincing slot of the target session on a look-ahead thread which starts and ends before B_1. Hence, there cannot be a convincing slot on the look-ahead thread of X. Hence, X is not a special block.

■

Thus, $\mathsf{Find}(\tau') = B'_1$. Now we will swap back B'_1 and B_1 and get back τ''. Doing this recursively, given the good tape τ', we will identify the same blocks $B_1, B_2, \ldots, B_q$, which were swapped on bad tape τ to get τ'. We will denote this by $\mathsf{Undo}(\tau') = (\tau, B_1, \ldots, B_q)$, where τ is a bad random tape. Note that this Undo procedure is deterministic.

The next claim counts the number of good tapes which can be obtained from a bad tape having γ composable blocks.

Claim 4 *Let τ be a* bad *random tape, $\mathcal{B} = \{B_1, \ldots, B_\gamma\}$ be a set of composable blocks for τ. Then, we can generate a set of* good *random tapes, $G(\tau, \mathcal{B})$, by swapping the various* composable *blocks in $\mathcal{B}$, so that the following holds:*

1. $|G(\mathcal{B}, \tau)| \geq 2^\gamma - 1$.

2. *For any* bad *tape $\tau \neq \tau'$ and any set of* composable *blocks $\mathcal{B}'$ for τ', $G(\mathcal{B}, \tau) \cap G(\mathcal{B}', \tau') = \phi$.*

Proof: Without loss of generality, let $B_1 > B_2 \ldots > B_\gamma$. Consider any non-empty subsequence of $1, \ldots, \gamma$, say $u_1, \ldots, u_q$. There are $2^\gamma - 1$ such sequences. Let $\tau_{u_1,\ldots,u_q}$ be the random tape obtained by $\mathsf{Swap}(\tau, B_{u_1}, \ldots, B_{u_q})$. From the description of Swap, it follows that $\tau_{u_1,\ldots,u_q}$ is a good random tape. We further note that given $\tau_{u_1,\ldots,u_q}$, we can recover the blocks $B_{u_1},\ldots,B_{u_q}$ by $\mathsf{Undo}(\tau_{u_1,\ldots,u_q})$. The resulting bad tape will always be τ. Since Undo is deterministic, we must have $\tau_{\vec{u}} \neq \tau_{\vec{v}}$ whenever $\vec{u} \neq \vec{v}$ in order for Undo to recover a different set of blocks on input $\tau_{\vec{u}}$ and $\tau_{\vec{v}}$. Thus, we obtain $2^\gamma - 1$ distinct good random tapes.

Similarly, take any $\alpha \in G(\tau, \mathcal{B})$ and $\beta \in G(\tau', \mathcal{B}')$. Applying Undo on α will result in bad tape τ, while applying the same procedure on β will give τ'. Again, since Undo is deterministic, we have $\alpha \neq \beta$

Number of composable blocks: Given a bad random tape for generation of messages, we know that the target session is special covered by $n'' \geq N/2\log^2 C$ sets at target set level j^* w.r.t. $\widetilde{T^*_{\mathsf{RL}}}$. By Corollary 3, there are at least $p(\kappa)$ sets from these n'' sets which are in $\textsc{psets}_{\mathsf{RL},j^*}$. Lets call the sets picked among n'' sets as $\mathcal{X}$. Each of the sets $\mathsf{set}^{\mathsf{j^*,k}}_{\mathsf{RL}} \in \mathcal{X}$ satisfies the Main Block condition. Since, this is a bad tape, each $\mathsf{set}^{\mathsf{j^*,k}}_{\mathsf{RL}} \in \mathcal{X}$ special covers session s^* but none of their look-aheads contain a convincing slot of s^*. Since, $\mathsf{set}^{\mathsf{j^*,k}}_{\mathsf{RL}}$ does not contain End of s^*, the corresponding look-ahead thread cannot contain the End of s^*. Hence, $\mathsf{set}^{\mathsf{j^*,k}}_{\mathsf{RL}}$ satisfies the Sibling condition. Since $|\mathcal{X}| > p(\kappa)$, there are at least $p(\kappa)$ composable blocks.

Claim 5 *For any fixed r_s, for any thread T^*, any session s^*, any section T^*_{RL} of T^* and any set level j^* of $\widetilde{T^*_{\mathsf{RL}}}$ (defined w.r.t. T^*_{RL}) such that there are $n'' \geq N/2\log^2 C$ sets at set level j^* of T^*_{RL} which* special cover *s^*, for each* bad *random tape, there are at least $O(\kappa^{\omega(1)})$* good *random tapes.*

Claim 6 *For any fixed r_s, for any thread T^*, any session s^*, for each* bad *random tape, there are at least $O(\kappa^{\omega(1)})$* good *random tapes.*

This follows from the previous claim by taking a union bound of bad tapes for all the sections and set levels since there are only at most $\log C$ sections and at most $\log C$ set levels for each section.

Finally, lemma 4 follows from the previous claim by taking a union bound over all sessions and threads since there are at most polynomial number of sessions and threads.

B.3 Cost of Rewinding

In this section, we will bound the total cost of rewinding incurred by our simulation strategy.

Lemma 10 *Let C be the cost of the main thread. Then the cost incurred by our simulator is bounded by*
$C \cdot (1 + \frac{(\log^* \kappa)^2 \log C \log^4 \kappa}{N})$ when $\langle C, R\rangle$ has $N \geq \log^6 \kappa$ slots.

Proof. Let C be the total cost of the output thread. The overhead incurred by our simulator corresponds to the cost of the look-ahead threads created. Hence, we will bound the cost of the sets picked across all recursion levels and across all set levels. We begin by bounding the cost of the look-ahead threads of a thread T of cost c at some recursion level RL.

Any set at set level j of T has cost $c/2^j$. At set level j, we pick $\left\lfloor \frac{\log^3 \kappa \cdot p(\kappa)\cdot q(\kappa)}{N} \cdot 2^j \right\rfloor$ sets. Cost of all the sets picked at set level j is $\left\lfloor \frac{\log^3 \kappa \cdot p(\kappa)\cdot q(\kappa)}{N} \cdot 2^j \right\rfloor \cdot \left(\frac{c}{2^j}\right) \leq \frac{c\log^3 \kappa \cdot p(\kappa)\cdot q(\kappa)}{N}$. Since there are $\log c$ set levels, summing the cost across all set levels, we get total cost of all look-ahead threads of T is at most $\frac{c \log c \log^3 \kappa \cdot p(\kappa)\cdot q(\kappa)}{N}$.

Using the above bound for output thread of cost C, we get that total cost of look-ahead threads at recursion level 1 is $C_1 \leq \frac{C \log C \log^3 \kappa \cdot p(\kappa)\cdot q(\kappa)}{N}$.

Consider a thread at recursion level 1 with cost c_ℓ. Invoking the above bound for this thread, we get that total cost of look-ahead threads for this thread is at most $\frac{c_\ell \log c_\ell \log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)}{N}$. Summing over all threads at recursion level 1 we get that total cost of all sets at recursion level 2 is

$$C_2 \leq \sum_\ell \frac{c_\ell \log c_\ell \log^3 \kappa \cdot p(\kappa) \cdot q(\kappa)}{N} \leq \sum_\ell \frac{c_\ell p(\kappa) q(\kappa) \log C \log^3 \kappa}{N} \leq C \cdot \left(\frac{p(\kappa) q(\kappa) \log C \log^3 \kappa}{N} \right)^2$$

, where ℓ indexes over all threads at recursion level 1.

By a similar analysis, we will get that total cost of all the threads at recursion level i is $C_i \leq C \cdot \left(\frac{p(\kappa) q(\kappa) \log C \log^3 \kappa}{N} \right)^i$.

Summing across all recursion levels, we will get that cost of all look-ahead threads across all recursion levels, i.e., total cost of rewinding, is at most $\left(\frac{2Cp(\kappa) q(\kappa) \log C \log^3 \kappa}{N} \right)$ if $N > 2p(\kappa) q(\kappa) \log C \log^3 \kappa$. Finally, the lemma holds by putting $p(\kappa) = \log \kappa \log^* \kappa$, $q(\kappa) = \log^* \kappa$ and C is polynomial in κ.

■

B.4 Modified Commitment Scheme $\langle C', R' \rangle$.

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

When we use $\langle C', R' \rangle$ in our main construction, we will require the committer C' to prove the "correctness" of the values (i.e., the secret shares) it reveals in the last step of the commitment protocol. In fact, due to technical reasons, we will also require the the committer to prove that the commitments that it sent in the first step are "well-formed". Looking ahead, these proofs will be done via an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ (Section A.2). Below we formalize both these properties in the form of a *validity* condition for the commit phase.

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

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

Precise Concurrent Extraction In the previous section, we had proved the lemma 4 for the concurrent execution of commitment scheme $\langle C, R \rangle$. We would like to have a similar extraction lemma for the modified commitment scheme $\langle C', R' \rangle$. But, we note that in our main construction, commitments sent in the commit phase of an execution of $\langle C', R' \rangle$ are never later opened via the opening phase. However, the above lemma is still applicable to $\langle C', R' \rangle$ as well as long as the proofs of *validity* given along with $\langle C', R' \rangle$ are sound. In our construction, the proofs of validity will be given via executions of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$. We note that for each of these executions, the statement for

$\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$ will have a "trapdoor condition" that will allow our simulator to cheat; however, in our security proof, we will ensure that that the trapdoor condition is *false* for each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$ where the adversary plays the role of the prover. Therefore, by relying on the soundness of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$, we will still be able to use lemma 4.

Moreover, the cost of the rewinding will same as the bound given in Lemma 10. Combining both the commitment scheme we have the following lemma:

Lemma 11 *(Extraction lemma) Consider two parties P_1 and P_2 running polynomially many (in the security parameter) sessions of a protocol Π consisting of possibly multiple executions of the commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$. Also, let one the parties, say P_2, be corrupted. Then except with negligible probability, in every thread of execution output by $\mathcal{S}$, if honest P_1 accepts a commit phase of $\langle C, R\rangle$ or $\langle C', R'\rangle$ as valid, then at the point when that commit phase is concluded, $\mathcal{S}$ would have already extracted the preamble secret committed by the adversarial P_2.*

Moreover, if the total cost of the output thread is C, then simulation by $\mathcal{S}$ incurs a cost of at most $C \cdot \left(1 + \frac{\omega(1)\log C \log^4 \kappa}{N}\right)$.

C Concurrently Secure Computation in the Joint Leaky Ideal World Model

In this section, we will prove the following theorem:

Theorem 10 *Assume the existence of 1-out-of-2 oblivious transfer protocol secure against honest but curious adversaries. Then for every polynomial poly(κ) such that $\epsilon = 1/poly(\kappa)$, for any functionality $\mathcal{F}$, there exists an $\mathcal{O}(N)$ round protocol Π that ϵ-securely realizes $\mathcal{F}$ in the joint leaky ideal world model, where $N = \frac{(\log^6 \kappa)}{\epsilon}$.*

As one can note, when ϵ is $\frac{1}{\text{poly}(\kappa)}$ for some polynomial poly(κ), then our protocol will have polynomial number of rounds. We handle the cases when we want lower round complexity at the cost of fraction of sessions leaked as follows: We assume the existence of collision resistant hash functions to get constant round statistically witness indistinguishable arguments and prove the following theorem in the same way as the previous theorem.

Theorem 11 *Assume the existence of 1-out-of-2 oblivious transfer protocol secure against honest but curious adversaries and collision resistent hash functions. Then for any $\epsilon > 0$, for any functionality $\mathcal{F}$, there exists an $\mathcal{O}(N)$ round protocol Π that ϵ-securely realizes $\mathcal{F}$ in the joint leaky ideal world model, where $N = \frac{(\log^6 \kappa)}{\epsilon}$.*

In section C.1 we describe such a protocol Π. Then in section C.2 we describe the simulator $\mathcal{S}$. In section C.3, we prove that our simulator is a ϵ-Joint-Ideal-Leakage simulator for any $\epsilon > 0$ followed by proof of security in section C.4.

C.1 Our Construction

In this section, we describe our concurrently secure computation protocol Π in joint leaky ideal world model for a general functionality $\mathcal{F}$.

In order to describe our construction, we first recall the notation associated with the primitives that we use in our protocol. Let $\text{COM}(\cdot)$ denote the commitment function of a non-interactive perfectly binding commitment scheme. Let $\langle C, R\rangle$ denote the N-round extractable commitment scheme and $\langle C', R'\rangle$ be its modified version as described in Section B.4. Let $\langle P, V\rangle$ denote the modified version of the CNMZK argument of Barak et al. [BPS06] as described in Section A.4.

Further, let $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ denote a SWI argument (Section A.2) and let $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ denote a *semi-honest* two party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ that securely computes $\mathcal{F}$ in the stand-alone setting as per the standard definition of secure computation (Section A.3).

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

I. Trapdoor Creation Phase.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We now claim the following:

Theorem 12 *The proposed $\mathcal{O}(N)$ round protocol* Π ϵ-securely realizes *the functionality $\mathcal{F}$ in joint leaky ideal world model with $\epsilon = \frac{(\log^* \kappa)^3 \cdot \log^5 \kappa}{N}$, as per the Definition 2.*

We will prove Theorem 12 in the following manner. First, in Section C.2, we will construct a simulator $\mathcal{S}$ for protocol Π that will simulate the view of $\mathcal{A}$ in the ideal world. We will show that $\mathcal{S}$ makes at most ϵ (defined in Theorem 12) leakage queries while simulating the view of $\mathcal{A}$. Looking ahead, in any query to the leakage oracle, $\mathcal{S}$ will leak the inputs of the honest party in some session. Finally, in Section C.4, we will argue that the output distributions of the real and ideal world executions are computationally indistinguishable.

Theorem 10 can be proved as a corollary of theorem 12 by setting $N = (\log^6 \kappa)/\epsilon$.

C.2 Proof of theorem 12

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

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

Consider any session $i \in [m]$ between H and $\mathcal{A}$. Consider the last message from $\mathcal{A}$ before H sends a message to $\mathcal{A}$ during the secure computation phase in session i. Note that this message could either be the first message of the secure computation phase or the last message of the input commitment phase, depending upon whether $\mathcal{A}$ or H sends the first message in the secure computation phase. In the sequel, we will refer to this message from $\mathcal{A}$ as the $\mathsf{special}$ message. Intuitively, this message is important because our simulator will need to leak the honest party's inputs for this session whenever it receives such a message from $\mathcal{A}$ on a look-ahead thread. Looking ahead, in order to bound the number of leakage queries made by our simulator, we will be counting the number of $\mathsf{special}$ messages sent by $\mathcal{A}$ on look-ahead threads during the simulation. This is possible because for an adversary to successfully use different inputs in different executions of the same session, it has to change some of its messages from the start of the session to $\mathsf{special}$ message. Otherwise, the input used by the adversary is fixed due to perfect binding property of COM and soundness of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$.

C.2.1 Simulator $\mathcal{S}$

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

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

Let S_{cec} denote the simulator for the commitment schemes $\langle C, R\rangle$ and $\langle C', R'\rangle$ as described in Section 3.2. The machine S_{ext} is essentially the same as the simulator S_{cec} that interacts with $\mathcal{A}$ in order to extract the committed value in each instance of $\langle C, R\rangle_{H\to\mathcal{A}}$ and $\langle C', R'\rangle_{H\to\mathcal{A}}$. Specifically, in order to perform these extractions, S_{ext} employs the cost-based rewinding strategy of S_{cec} for an "imaginary" adversary, described below. During the simulation, whenever S_{ext} receives a message from $\mathcal{A}$ in an execution of $\langle C, R\rangle_{H\to\mathcal{A}}$ or $\langle C', R'\rangle_{H\to\mathcal{A}}$, then it answers it on its own in the same manner as S_{cec} does (i.e., by sending a random challenge string). However, on receiving any other message, it simply passes it to the main simulator S_{main} (described below), and transfers its response to $\mathcal{A}$. Whenever S_{ext} extracts a committed value from an execution of $\langle C, R\rangle_{H\to\mathcal{A}}$ or $\langle C', R'\rangle_{H\to\mathcal{A}}$ at any point during the simulation, it immediately passes it to S_{main}. Whenever S_{ext} fails to extract any of the committed values from $\langle C, R\rangle_{H\to\mathcal{A}}$ or $\langle C', R'\rangle_{H\to\mathcal{A}}$, then it aborts with the special symbol $\perp$.

MESSAGE GENERATION TIMINGS OF $\mathcal{A}$. We note that in order to employ the cost-based rewinding strategy of S_{cec}, S_{ext} needs to know the cost of each of the message that $\mathcal{A}$ generates in the protocol (see Section 3.2). Our goal is to minimize the number of leakage queries made by $\mathcal{S}$. To this end, We consider an imaginary experiment in which we will assign disproportionately large cost to the messages of $\mathcal{A}$ which force $\mathcal{S}$ to make a leakage query. In particular, we will assign large cost to the $\mathsf{special}$ message in each session. Then the rewinding strategy of S_{ext} is determined by running S_{cec} using the next message generation costs of such an (imaginary) adversary, explained as follows.

Consider all the messages sent by $\mathcal{A}$ during a protocol execution. Assign θ units to the $\mathsf{special}$ message, where θ is the round complexity (linear in number of slots N) of our protocol; any other message is simply assigned one unit. Intuitively, by assigning more weight to the $\mathsf{special}$ message, we ensure that if the cost incurred by our simulator on the look-ahead threads is bounded by ϵ, then the number of $\mathsf{special}$ messages sent by $\mathcal{A}$ on the look-ahead threads during the simulation must be a bounded by ϵ as well. Looking ahead, this in turn will allow us to prove that the number of sessions for which our simulator leaks the honest party's inputs are bounded by ϵ.

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

TRAPDOOR CREATION PHASE. In the first step, instead of committing to bit 0, S_{main} sends com_1 as a commitment to bit 1. Now, recall that S_{ext} interacts with $\mathcal{A}$ during the preamble phase in $\langle P, V\rangle_{H\to\mathcal{A}}$ and extracts the preamble secret $\sigma_{\mathcal{A}}$ from $\mathcal{A}$ at the conclusion of the "valid" preamble with all but negligible probability. Then, on receiving $\sigma_{\mathcal{A}}$ from S_{ext}, S_{main} simulates the *post-preamble* phase of $\langle P, V\rangle_{H\to\mathcal{A}}$ in a straight-line manner (by using $\sigma_{\mathcal{A}}$); in the same manner as explained in Section A.4. In the case when the preamble phase is not valid but S_{ext} extracts some value, then S_{main} uses this value to simulate the post-preamble phase. On the other hand, if S_{ext} returns $\perp$, then S_{main} executes the post-preamble phase of $\langle P, V\rangle_{H\to\mathcal{A}}$ by following the honest party strategy. In the second step of this phase, S_{main} simply uses the honest party strategy to interact with $\mathcal{A}$.

As we have shown in section 3.2, except with negligible probability, S_{ext} always succeeds in extracting the preamble secret $\sigma_{\mathcal{A}}$ in as long as the commitment protocol $\langle C, R\rangle_{\mathcal{A}\to H}$ is "valid". In other words, S_{ext} outputs $\perp$ or a value different from $\sigma_{\perp}$ only if the commitment protocol $\langle C, R\rangle_{\mathcal{A}\to H}$ is not "valid". In this case, $\mathcal{A}$ would fail with probability 1 in successfully decommitting to the preamble secret during the post-preamble phase of $\langle P, V\rangle_{H\to\mathcal{A}}$. As a consequence, S_{main} will abort the session.

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

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

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

First note that a call to the trusted party is defined by the corresponding $\mathsf{special}$ message. If the transcript from the start of the session to the $\mathsf{special}$ message is same for two different trusted party calls, then these calls and their outputs are identical. Now, when S_{sh} makes a call to the trusted party, S_{main} does the following: S_{main} finds the corresponding $\mathsf{special}$ message msg and checks the following: If a trusted party call has been made corresponding to msg, it uses the output received from the trusted party on the query. Note that in this case the input $x_{\mathcal{A}}$ and hence output from trusted party would be the same as before due to perfectly binding property of COM and soundness of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$. Otherwise, it checks if a leakage query has been made corresponding to msg, it uses the output of the leakage query to compute the output. If none of the above holds then S_{main} computes the output as follows: If msg lies on the main thread, S_{main} makes a trusted party call for the session i with input $x_{\mathcal{A}}$. Also, it records that a trusted party query has been made corresponding to msg and stores the output for possible future use. Otherwise, msg lies on some look-ahead thread. In this case, S_{main} sends a leakage query L_i to the trusted party such that L_i takes honest parties' inputs across all sessions as input and outputs the honest party's input for session i. Now S_{main} can compute the output itself by using the honest party's input and $x_{\mathcal{A}}$. S_{main} makes a record of this leakage query. If the query corresponds to the main thread, S_{main} sends the message (output, s) to the trusted party, indicating it to send the output to the honest party in session. [11] The output is passed on to S_{sh}.

[11]Note that $s = \ell$ in this case. We stress that by setting $s = \ell$ for a query on the main thread, S_{main} ensures that

Having received the output from S_{main}, S_{sh} runs further, and finally halts and outputs a transcript $\beta_{H,1}, \beta_{\mathcal{A},1}, \ldots, \beta_{H,t}, \beta_{\mathcal{A},t}$ of the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, and an associated random string $\hat{r}_{\mathcal{A}}$. S_{ext} now performs the following steps.

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

2. Now, in each round $j \in [t]$, S_{main} sends $\beta_{H,j}$. It then engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{H \to \mathcal{A}}$ with $\mathcal{A}$ where it uses the trapdoor witness (deviating from honest party strategy that used the real witness). Next, on receiving $\mathcal{A}$'s next message $\beta_{\mathcal{A},j}$ in the protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$, S_{main} engages in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}$ with $\mathcal{A}$ where it uses the honest verifier strategy. Finally at any stage, if the j^{th} message of the adversary is not $\beta_{\mathcal{A},j}$ and the proof $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle_{\mathcal{A} \to H}$ given immediately after this messages is accepted, then the simulator aborts all communication and outputs $\perp$. (Later, we establish in the proof of Lemma 13 that S_{ext} outputs $\perp$ with only negligible probability.)

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

C.3 Fraction of honest parties' inputs leaked by $\mathcal{S}$

In this section we will prove that $\mathcal{S}$ is a ϵ-joint-ideal-leakage simulator for ϵ defined below.

Lemma 12 *Simulator $\mathcal{S}$ is ϵ-Joint-Ideal-Leakage Simulator for* $\epsilon \leq \frac{(\log^* \kappa)^3 \log^5(\kappa)}{N}$.

Proof. Let m be the total number of sessions of $\Pi = (P_1, P_2)$ being executed concurrently. Let C be the total cost of the output thread in the real execution, as per the cost assignment strategy described in section C.2.1. Let q be the round complexity of Π. Then, as per the cost assignment strategy given in section C.2.1, $C = (\theta - 1 + \theta) \cdot m$ (recall that the $\mathsf{special}$ message is assigned a cost of θ units, while each of the remaining $\theta - 1$ messages is assigned one unit cost). Now by section B.3 cost-based rewinding strategy S_{cec} has following cost for all the look-ahead threads created: $\left(\frac{C \cdot (\log^* \kappa)^2 \log C \log^4 \kappa}{N}\right) = \frac{m \cdot \theta \cdot (\log^* \kappa)^2 (\log m + \log N) \log^4 \kappa}{N}$, where θ is $\mathcal{O}(N)$.

Also, cost of a $\mathsf{special}$ message is θ. Hence, maximum number of $\mathsf{special}$ messages which can be present on look-aheads is bounded by $\Gamma \leq \frac{m \cdot (\log^* \kappa)^2 (\log m + \log N) \log^4 \kappa}{N}$. Recall that we make at most one leakage query corresponding to each distinct $\mathsf{special}$ message. Hence $\mathcal{S}$ makes at most Γ leakage queries.

Each leakage query leaks the honest party's input for a single session. Hence, we leak honest parties' inputs for at most Γ sessions. Lemma follows by using the fact that both m and N are bounded by some polynomials in κ.

■

C.4 Indistinguishability of the Views

We consider two experiments H_0 and H_1, where H_0 corresponds to the real execution of Σ while H_1 corresponds to the ideal computation of $\mathcal{F}$, as described below.

Experiment H_0: The simulator $\mathcal{S}$ is given the inputs of all the honest parties. By running honest programs for the honest parties, it generates their outputs along with $\mathcal{A}$'s view. This corresponds to the real execution of the protocol. The output of the hybrid corresponds to the outputs of the honest parties and the view of the adversary $\mathcal{A}$.

the honest party in session ℓ receives the correct output. (Note that an honest party does not receive any output for an output query on a look-ahead thread.)

Experiment H_1: $\mathcal{S}$ simulates all the sessions without the inputs of the honest parties (in the same manner as explained in the description of $\mathcal{S}$) and outputs the view of $\mathcal{A}$. Each honest party outputs the response it receives from the trusted party. Again the output of the hybrid corresponds to the outputs of the honest parties and the view of the adversary $\mathcal{A}$.

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

Lemma 13 $v^0 \overset{c}{\equiv} v^1$

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

C.4.1 Getting Started

We will prove Lemma 13 by contradiction. Suppose that the hybrids H_0 and H_1 are distinguishable in polynomial time, i.e., there exists a PPT distinguisher D that can distinguish between the two hybrids with a non-negligible probability. We will now consider a series of hybrid experiments $\mathcal{H}_{i:j}$, where $i \in [1, 2m]$, and $j \in [1, 6]$. We define two additionally hybrids – first, a dummy hybrid $\mathcal{H}_{0:6}$ that represents the real world execution (i.e., H_0, as defined above), and second, an additional hybrid $\mathcal{H}_{2m+1:1}$ that corresponds to the simulated execution in the ideal world (i.e., H_1, as defined above). For each intermediate hybrid $\mathcal{H}_{i:j}$, we define a random variable $\nu_{i:j}$ that represents the output (including the view of the adversary and the outputs of the honest parties) of $\mathcal{H}_{i:j}$.

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

In the sequel, we will make use of the notation described in Section C.2. In particular, whenever necessary, we will augment our notation with a super-script that denotes the session number. We now describe some additional notation that will be used in the proof.

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

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

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

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

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

Lemma 14 *Let $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ and $\pi^{\ell}_{\mathcal{A}}$ be as described above corresponding to the real execution. Then, for each session $\ell \in [m]$, if the honest party does not abort the session (before the first message of the Secure Computation Phase is sent) in the view $\nu_{0:6}$, then $\rho^{\ell}_{0:6}$ is a valid witness to the statement $\pi^{\ell}_{\mathcal{A}}$, except with negligible probability.*

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

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

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

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

C.4.2 Description of the Hybrids

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

Experiment $\mathcal{H}_{i:1}$: Same as $\mathcal{H}_{i-1:6}$, except that $\mathcal{S}$ performs rewindings *upto* LM_i (as described in Section C.2). Specifically, the rewindings are performed with the following restrictions:

- No new-look ahead threads are created beyond LM_i on the main thread (i.e., the execution is straight-line beyond LM_i).
- Consider any look-ahead thread that is created before the execution reaches LM_i on the main-thread. Then, any such look-ahead thread is terminated as soon as the execution reaches the i^{th} LM *on that thread*[12].

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

Additionally, $\mathcal{S}$ extracts and records the committed value in each execution of $\langle C, R\rangle_{\mathcal{A}\to H}$ and $\langle C', R'\rangle_{\mathcal{A}\to H}$ that concludes before LM_i. $\mathcal{S}$ outputs the abort symbol $\perp$ if it "gets stuck". Otherwise, it outputs the view of the adversary in the main thread of this simulation as $\nu_{i:1}$.

We now claim that,

$$\nu_{i-1:6} \overset{c}{\equiv} \nu_{i:1} \tag{1}$$

$$\forall \ell \quad \rho^{\ell}_{i-1:6} \overset{c}{\equiv} \rho^{\ell}_{i:1} \tag{2}$$

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

$$\nu_{i-1:6} \overset{s}{\equiv} \nu_{i-1:6:1} \tag{3}$$

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

$$\forall \ell \quad \rho^{\ell}_{i-1:6} \overset{c}{\equiv} \rho^{\ell}_{i-1:6:1} \tag{4}$$

Let us assume that equation 4 is false. That is, $\exists \ell \in [m]$ such that $\rho^{\ell}_{i-1:6}$ and $\rho^{\ell}_{i-1:6:1}$ are distinguishable by a probabilistic polynomial time (PPT) distinguisher. In this case, we can create an unbounded adversary that extracts the value contained in the non-malleable commitment inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ and is then able to distinguish between the main threads in $\mathcal{H}_{i-1:6}$ and $\mathcal{H}_{i-1:6:1}$, which is a contradiction.

We now argue that in hybrid $\mathcal{H}_{i-1:6:1}$, $\mathcal{S}$ is able to extract (except with negligible probability) the committed value in each execution of $\langle C, R\rangle_{\mathcal{A}\to H}$ and $\langle C', R'\rangle_{\mathcal{A}\to H}$ that concludes before LM_i. Towards this, we first note that by construction, simulator's strategy in this experiment is identical for each thread, irrespective of whether it is the main-thread or a look-ahead thread. Now consider an imaginary adversary who aborts once the execution reaches LM_i on any thread. Note that lemma 4 holds for such an adversary (i.e. the probability that the simulator fails to extract the committed value of a "concluded" commitment $\langle C, R\rangle$ or $\langle C', R'\rangle$ is negligible). Then, if the adversary does not abort (as is the case with $\mathcal{A}$), the probability that the simulation successfully extracts the committed values must be only higher. Hence our claim follows for case 1.

For case 2, we note that lemma 4 is applicable if we can argue that the *soundness condition* holds (specifically, we require that the trapdoor condition is false for each instance of SWI in $\langle C, R\rangle^{\ell}_{\mathcal{A}\to H}$ if $\langle C, R\rangle^{\ell}_{\mathcal{A}\to H}$ concludes before LM_i). Note that this is already implied by equation 4. Hence, our claim follows for case 2 as well.

Proving Equations 1 *and* 2. Note that the only difference between $\mathcal{H}_{i-1:6:1}$ and $\mathcal{H}_{i:1}$ is that $\mathcal{S}$ outputs the abort symbol $\perp$ if S_{cec} "gets stuck". We have shown that this event happens only with negligible probability. Hence our claim follows.

Experiment $\mathcal{H}_{i:2}$: Same as $\mathcal{H}_{i:1}$, except that if LM_i is of type I, then $\mathcal{S}$ simulates the post-preamble phase of $\langle P, V\rangle^{s(i)}_{H\to\mathcal{A}}$ in a *straight-line* manner, as explained in Section A.4. For completeness, we recall it below. Recall that no look-ahead threads are started once the execution reaches LM_i on the main thread. Thus, all the changes in the main thread, as explained below, are performed *after* LM_i.

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

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

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

Now we prove that,

$$\nu_{i:1} \overset{c}{\equiv} \nu_{i:2} \tag{5}$$

$$\forall \ell \quad \rho^{\ell}_{i:1} \overset{c}{\equiv} \rho^{\ell}_{i:2} \tag{6}$$

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

We now claim the following:

$$\nu_{i:1} \overset{c}{\equiv} \nu_{i:1:1} \tag{7}$$

$$\forall \ell \quad \rho^{\ell}_{i:1} \overset{c}{\equiv} \rho^{\ell}_{i:1:1} \tag{8}$$

$$\nu_{i:1:1} \overset{c}{\equiv} \nu_{i:1:2} \tag{9}$$

$$\forall \ell \quad \rho^{\ell}_{i:1:1} \overset{c}{\equiv} \rho^{\ell}_{i:1:2} \tag{10}$$

$$\nu_{i:1:2} \overset{c}{\equiv} \nu_{i:1:3} \tag{11}$$

$$\forall \ell \quad \rho^{\ell}_{i:1:2} \overset{c}{\equiv} \rho^{\ell}_{i:1:3} \tag{12}$$

Note that equation 5 follows by combining the results of equations 7, 9, and 11. Similarly, equation 6 follows by combining the results of equations 8, 10, and 12. We now prove the above set of equations.

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

Proving Equations 7 and 8. We first note that SCOM and SZKAOK can together be viewed as a statistically hiding commitment scheme. Let $\overline{\text{SCOM}}$ denote this new commitment scheme. Then, equation 7 simply follows from the hiding property of $\overline{\text{SCOM}}$.

In order to prove equation 8, let us first assume that the claim is false, i.e., $\exists \ell \in [m]$ such that $\rho^{\ell}_{i:1}$ and $\rho^{\ell}_{i:1:1}$ are distinguishable by a PPT distinguisher D. We will create a standalone machine M^* that is identical to $\mathcal{H}_{i:1}$, except that instead of simply committing to a string of all zeros using $\overline{\text{SCOM}}$ in $\langle P, V\rangle^{s(i)}_{H\to\mathcal{A}}$, M^* takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$. Additionally, M^* "exposes" the NMCOM in $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Let us describe the interaction between M^* and C in more detail. M^* first sends the preamble secret $\sigma^{m^{\text{BPS}}}_{\mathcal{A}} s(i)$ to C. Now, when C starts the execution of $\overline{\text{SCOM}}$ in $\langle P, V\rangle^{s(i)}_{H\to\mathcal{A}}$, M^* forwards the messages from C to $\mathcal{A}$; the responses from $\mathcal{A}$ are forwarded externally to C. Note that if C commits to a string of all zeros in the $\overline{\text{SCOM}}$ execution, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:1:1}$. On the other hand, if C commits to the preamble secret $\sigma_{\mathcal{A}}^{m\text{BPS}} s(i)$, then the (C, M^*, R) system is equivalent to $\mathcal{H}_{i:1:2}$. We will now construct a computationally unbounded distinguisher D' that distinguishes between these two executions, thus contradicting the statistically hiding property of $\overline{\text{SCOM}}$. D' simply extracts the value inside the NMCOM received by R and runs D on this input. D' outputs whatever D outputs. By our assumption, D's output must be different in these two experiments; this implies that D' output is different as well, which is a contradiction.

Proving Equations 9 and 10. Equation 9 simply follows due to the witness indistinguishability property of SZKAOK. Equation 10 follows from the fact that SZKAOK is *statistically* witness indistinguishable. The proof details are almost identical to the proof of equation 8 and therefore omitted.

Proving Equations 11 and 12. Equation 11 simply follows from the hiding property of NMCOM. To see this, we can construct a standalone machine M that internally runs $\mathcal{S}$ and $\mathcal{A}$ and outputs the view generated by $\mathcal{S}$. M is identical to $\mathcal{H}_{i:1:2}$ except that in phase IV of $\langle P, V\rangle^{s(i)}_{H\to\mathcal{A}}$, instead of simply committing (using NMCOM) to a valid witness (to the proof statement $y^{s(i)}$), it takes this commitment from an external sender C and "forwards" it internally to $\mathcal{A}$.

In order to prove equation 12, we will use the non-malleability property of NMCOM. Let us assume that equation 12 is false, i.e., $\exists \ell \in [m]$ such that $\rho^{\ell}_{i:1:2}$ and $\rho^{\ell}_{i:1:3}$ are distinguishable by a PPT machine. We will construct a standalone machine M^* that is identical to the machine M described above, except that it will "expose" the non-malleable commitment inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ to an external receiver R by relying on the public-coin property of NMCOM, as described earlier. Now, if E commits to the witness to y^{ℓ}, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:1:2}$, whereas if E commits to a random string, then the (C, M^*, R) system is identical to $\mathcal{H}_{i:1:3}$. From the non-malleability property of NMCOM, we establish that the value committed by M^* to R must be computationally indistinguishable in both cases.

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

We now claim that,

$$\nu_{i:2} \overset{c}{\equiv} \nu_{i:3} \tag{13}$$

$$\forall \ell \quad \rho^{\ell}_{i:2} \overset{c}{\equiv} \rho^{\ell}_{i:3} \tag{14}$$

Proving Equations 13 and 14. Equation 13 simply follows from the (computationally) hiding property of the commitment scheme COM.

In order to prove equation 14, we will leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in $\langle P, V\rangle$. Let us first assume that equation 14 is false, i.e., $\exists \ell \in [m]$ such that $\rho^{\ell}_{i:2}$ and $\rho^{\ell}_{i:3}$ are distinguishable by a PPT distinguisher. Note that it cannot be the case that the NMCOM inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ concludes *before* $\mathcal{S}$ sends the non-interactive commitment $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$ in session $s(i)$, since in this case, the execution of NMCOM is independent of $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$. Now consider the case when the NMCOM inside $\langle P, V\rangle^{\ell}_{\mathcal{A}\to H}$ concludes *after* $\mathcal{S}$ sends $\Pi^{s(i)}_{\text{COM},H\to\mathcal{A}}$.

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

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

We now claim that,

$$\nu_{i:3} \overset{c}{\equiv} \nu_{i:4} \tag{15}$$
$$\forall \ell \quad \rho^{\ell}_{i:3} \overset{c}{\equiv} \rho^{\ell}_{i:4} \tag{16}$$

Proving Equations 15 and 16. Equation 15 simply follows from the witness indistinguishability of SWI by a standard hybrid argument.

In order to prove equation 16, let us first consider the simpler case where $\mathcal{S}$ uses the trapdoor witness only in the *first* instance (in the order of execution) of SWI in session $s(i)$ where the honest party plays the role of the prover. In this case, we can leverage the "statistical" nature of the witness indistinguishability property of SWI in a similar manner as in the proof of equation 10. Then, by a standard hybrid argument, we can extend this proof for multiple SWI.

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

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

We now claim that,

$$\nu_{i:4} \overset{c}{\equiv} \nu_{i:5} \tag{17}$$
$$\forall \ell \quad \rho^{\ell}_{i:4} \overset{c}{\equiv} \rho^{\ell}_{i:5} \tag{18}$$

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

We now claim the following:

$$\nu_{i:4} \overset{c}{\equiv} \nu_{i:4:1} \tag{19}$$
$$\forall \ell \quad \rho^{\ell}_{i:4} \overset{c}{\equiv} \rho^{\ell}_{i:4:1} \tag{20}$$
$$\nu_{i:4:1} \overset{c}{\equiv} \nu_{i:4:2} \tag{21}$$
$$\forall \ell \quad \rho^{\ell}_{i:4:1} \overset{c}{\equiv} \rho^{\ell}_{i:4:2} \tag{22}$$

Note that equation 17 follows by combining the results of equations 19 and 21. Similarly, equation eq:b45 follows by combining the results of equations 20 and 22. We now prove the above set of equations.

Proving Equations 19 and 20. Equation 19 simply follows from the (computational) hiding property of the commitment scheme COM.

In order to prove equation 20, let us first consider the simpler case where $\mathcal{S}$ only modifies the first commitment in the commit phase in $\langle C, R\rangle_{H\to\mathcal{A}}^{s(i)}$. In this case, we can leverage the hiding property of COM and the extractability property of the non-malleable commitment scheme in $\langle P, V\rangle$ in a similar manner as in the proof of equation 14. Then, by a standard hybrid argument, we can extend this proof to the case where $\mathcal{S}$ modifies all the commitments in the commit phase in $\langle C, R\rangle_{H\to\mathcal{A}}^{s(i)}$.

Proving Equations 21 and 22. Note that the main-thread is identical in hybrids $\mathcal{H}_{i:4:1}$ and $\mathcal{H}_{i:4:2}$ since we are only changing some random strings to other random strings; furthermore, the strings being changed are not used elsewhere in the protocol. Equations 21 and 22 follow as a consequence.

Experiment $\mathcal{H}_{i:6}$: Same as $\mathcal{H}_{i:5}$, except that if LM_i is of type II, $\mathcal{S}$ "simulates" the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$ in session $s(i)$, in the following manner. Let S_{sh} be the simulator for the semi-honest two party protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$ used in our construction. $\mathcal{S}$ internally runs the simulator S_{sh} for the semi-honest two party protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$ on $\mathcal{A}$'s input in session $s(i)$ that was extracted earlier. When S_{sh} makes a query to the trusted party with some input, $\mathcal{S}$ selects a session index s' and forwards the query to the trusted party in the same manner as explained earlier in Section C.2.1. The response from the trusted party is passed on to S_{sh}. Further, $\mathcal{S}$ decides whether the output must be sent to the honest party in the same manner as explained earlier. S_{sh} finally halts and outputs a transcript of the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$, and an associated random string for the adversary.

Now, $\mathcal{S}$ forces this transcript and randomness on $\mathcal{A}$ in the same manner as described in section C.2.1. We claim that during the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$, each reply of $\mathcal{A}$ must be consistent with this transcript, except with negligible probability. Note that we have already established from the previous hybrids that the *soundness condition* holds (except with negligible probability) at this point. This means that the trapdoor condition is false for each instance of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to H}^{s(i)}$. Then our claim follows from the soundness property of SWI used in our construction.

We now claim that:

$$\nu_{i:5} \overset{c}{\equiv} \nu_{i:6} \tag{23}$$

$$\forall \ell \quad \rho_{i:5}^{\ell} \overset{c}{\equiv} \rho_{i:6}^{\ell} \tag{24}$$

Proving Equation 23. Informally speaking, equation 23 follows from the semi-honest security of the two-party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$ used in our construction. We now give more details.

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

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

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

In the first case, the (C, M) system is identical to $\mathcal{H}_{i:5}$, while in the second case, the (C, M) system is identical to $\mathcal{H}_{i:6}$. By the (semi-honest) security of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$, we establish that the output of M must be indistinguishable in both the cases, except with negligible probability. This proves equation 23.

Proving Equation 24. We will leverage the semi-honest security of the two-party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$ and the extractability property of the non-malleable commitment scheme in $\langle P, V\rangle$ to prove equation 24.

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

D Precise Concurrent Zero-Knowledge

Precise zero knowledge introduced by Micali and Pass (STOC'06) guarantees that the view of any verifier V can be simulated in time closely related to the *actual* (as opposed to worst-case) time spent by V in the generated view. [PPS+08] gave the first constructions of precise concurrent zero-knowledge protocols. In this section we improve upon there results both in terms of round complexity and precision of simulation by using our cost-based rewinding strategy described in section 3.2.

Many parts of this section have been taken verbatim from [PPS+08].

Notation. Let L denote an NP-language and R_L the corresponding NP-relation. Let $(\mathcal{P},\mathcal{V})$ denote an interactive proof(argument) system where $\mathcal{P}$ and $\mathcal{V}$ are the prover and verifier algorithms respectively. By $\mathcal{V}^*(x, z, \bullet)$ we denote a non-uniform *concurrent adversary* verifier with common input x and auxiliary input (or advice) z whose random coins are fixed to a sufficiently long string chosen uniformly at random; $\mathcal{P}(x, w, \bullet)$ is defined analogously where $w \in R_L(x)$.

Note that $\mathcal{V}^*$ is a *concurrent adversary* verifier. Formally, it means the following. Adversary $\mathcal{V}^*$, given an input $x \in L$, interacts with an unbounded number of independent copies of $\mathcal{P}$ (all on common input x)[13]. An execution of a protocol between a copy of $\mathcal{P}$ and $\mathcal{V}^*$ is called a *session*. Adversary $\mathcal{V}^*$ can interact with all the copies at the *same* time (i.e., concurrently), interleaving messages from various sessions in any order it wants. That is, $\mathcal{V}^*$ has control over the scheduling of messages from various sessions. In order to implement a scheduling, $\mathcal{V}^*$ concatenates each message with the session number to which the next scheduled message belongs. The convention is that the prover copy corresponding to the session number specified in the last message, sends the next message (as per the specifications of the protocol). The *view* of concurrent adversary $\mathcal{V}^*$ in a

[13]We remark that instead of a single fixed theorem x, $\mathcal{V}^*$ can be allowed to adaptively choose provers with different theorems x'. For ease of notation, we choose a single theorem x for all copies of $\mathcal{P}$. This is not actually a restriction and our results hold even when $\mathcal{V}^*$ adaptively chooses different theorems.

concurrent execution is: the common input x, followed by the sequence of prover and verifier messages exchanged during the interaction, followed by the contents of the random tape of $\mathcal{V}^*$.

Let $\text{VIEW}_{\mathcal{V}^*(x,z,\bullet)}$ be the random variable denoting the view of $\mathcal{V}^*(x, z, \bullet)$ in a *concurrent* interaction with the copies of $\mathcal{P}(x, w, \bullet)$. Let $\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}(x,z,\bullet)}$ denote the view output by the simulator. When simulator's random tape is *fixed* to r, its output is instead denoted by $\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)}$. Finally, let $T_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)}$ denote the *steps* taken by the simulator and let $T_{\mathcal{V}^*}(\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)})$ denote the steps taken by $\mathcal{V}^*$ in the view $\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)}$.

Definition 10 (Precise Concurrent Zero Knowledge) *Let $p : N \times N \to N$ be a monotonically increasing function. We say that $(\mathcal{P},\mathcal{V})$ is* concurrent zero knowledge *proof (argument) system with* precision p*, if for every non-uniform* probabilistic polynomial time $\mathcal{V}^*$*, the following conditions hold:*

1. *For all $x \in L$, $z \in \{0,1\}^*$, the following distributions are computationally indistinguishable over L:*

$$\left\{\text{VIEW}_{\mathcal{V}^*(x,z,\bullet)}\right\} \text{ and } \left\{\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}(x,z,\bullet)}\right\}$$

2. *For all $x \in L$, $z \in \{0,1\}^*$, and every sufficiently long $r \in \{0,1\}^*$, it holds that:* $T_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)} \leq p(|x|, T_{\mathcal{V}^*}(\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)}))$.

D.1 Protocol Description

We describe the protocol in Figure 4. This protocol is same as that of [PPS+08] with the modification of using $\langle C, R\rangle$ extractable commitment scheme with N rounds.

PCZK(N): A Protocol for Precise Concurrent Zero Knowledge Arguments.

(Stage 1)
$\mathcal{P}$ and $\mathcal{V}$ engage in execution of $\langle C, R\rangle$ with N challenge-response rounds (Section 3.2) where $\mathcal{V}$ commits to a random string σ.
(Stage 2)

- p1: Run κ parallel and *independent* copies of BH-prover (Figure 5) and send the κ prover messages $\widehat{\text{p1}}$ to the verifier.
- v1: Reveal the challenge σ and send decommitment information for *all* commitments which are unopened so far. Each bit of σ can be thought of as verifier's challenge in Step $\widehat{\text{v1}}$ of BH-protocol.
- p2: Prover verifies that all the decommitments are proper and that $\sigma = \sigma^0_{i,j} \oplus \sigma^1_{i,j}$. If yes, execute the step $\widehat{\text{p2}}$ for each of the κ parallel copies of the BH-protocol.
- v2: Verify each of the κ parallel proofs as described in $\widehat{\text{v2}}$. If all κ $\widehat{\text{v2}}$ steps are accepting, accept the proof, otherwise reject the proof.

Figure 4: Our Precise Concurrent Zero Knowledge Protocol.

D.2 Simulator description

The simulator $\mathcal{S}_{\mathcal{V}^*}$ would be same as S_{cec} described in Section C.2. There we have described how to handle the messages of Stage 1. For handling the Stage 2 messages we describe the PROVE procedure in Figure 6.

The BLUM-HAMILTONICITY(BH) Protocol [Blu87b].

$\widehat{\text{p1}}$: Choose a random permutation π of vertices V. Commit to the adjacency matrix of the permuted graph, denoted $\pi(G)$, and the permutation π, using a *perfectly binding* commitment scheme. Notice that the adjacency matrix of the permuted graph contains a 1 in position $(\pi(i), \pi(j))$ if $(i, j) \in E$. Send both the commitments to the verifier.

$\widehat{\text{v1}}$: Select a bit $\sigma \in \{0, 1\}$, called the *challenge*, uniformly at random and send it to the prover.

$\widehat{\text{p2}}$: If $\sigma = 0$, send π along with the decommitment information of *all* commitments. If $\sigma = 1$ (or anything else), decommit all entries $(\pi(i), \pi(j))$ with $(i, j) \in C$ by sending the decommitment information for the corresponding commitments.

$\widehat{\text{v2}}$: If $\sigma = 0$, verify that the revealed graph is identical to the graph $\pi(G)$ obtained by applying the revealed permutation π to the common input G. If $\sigma = 1$, verify that all the revealed values are 1 and that they form a cycle of length n. In both cases, verify that all the revealed commitments are correct using the decommitment information received. If the corresponding conditions are satisfied, accept the proof, otherwise reject the proof.

Figure 5: The Blum-Hamiltonicity protocol used in PCZK

Lemma 15 (Concurrent Zero Knowledge) *The ensembles* $\{\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *and* $\{\text{VIEW}_{\mathcal{V}^*}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *are computationally indistinguishable over* L.

This follows from the Lemma 4 and proof of indistinguishibility in [PPS+08] (Lemma 5).

D.3 Precision of simulation

The main theorem of [PPS+08] is the following:

Theorem 13 *Assuming the existence of one-way functions, for every* $\varepsilon > 0$*, there exists an* $O(n^{\varepsilon})$*-round concurrent zero knowledge argument for all languages in* $\mathcal{NP}$ *with precision* $p(t) = O(t 2^{\frac{2}{\epsilon} \log_n t})$.

In our case, by using cost-based rewinding strategy for the extractable commitment scheme $\langle C, R \rangle$ we obtain the following precision guarantee:

Theorem 14 *(Precision) The protocol in Figure 4 is an* $\mathcal{O}(N)$ *round concurrent zero-knowledge argument for all languages in NP with precision* $p(t) = t \cdot \left(1 + \frac{(\log^* \kappa)^3 \log t \log^4 \kappa}{N}\right)$.

Using total time of execution t as cost C in lemma 10 we get $p(t) = t \cdot \left(1 + \frac{(\log^* \kappa)^2 \log t \log^4 \kappa}{N}\right)$.

Lemma 16 *There exists an* $\mathcal{O}(\log^6 \kappa)$ *round concurrent zero-knowledge argument for all languages in NP with precision* $p(t) = t \cdot (1 + \epsilon)$*, for any constant* ϵ.

It can be proved as a consequence of the above theorem by observing that t is polynomial in κ.

E Impossibility Result for Individual Leaky Ideal World Model

In this section, we prove an impossibility result for concurrently secure computation in the individual leaky ideal world model. We start by formally stating the main result in this section:

The PROVE Procedure.

Let $s \in [m]$ be the session for which the prove procedure is invoked. The procedure outputs either p1 or p2, whichever is required by $\mathcal{S}_{\mathcal{V}^*}$. Let hist denote the set of messages exchanged between $\mathcal{S}_{\mathcal{V}^*}$ and $\mathcal{V}^*$ in the *current* thread. The PROVE procedure works as follows.

1. If the verifier has aborted in any of the N first stage messages of session s (i.e., hist contains Vj=Abort for $j \in [N]$ of session s), abort session s.
2. Otherwise, search the table $\mathcal{H}$ to find values $\sigma^0_{i,j}, \sigma^1_{i,j}$ belonging to session s, for some $i \in [\ell], j \in [N]$. If no such pairs are found, output $\perp$ (indicating failure of the simulation). Otherwise, extract the challenge $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ as $\sigma^0_{i,j} \oplus \sigma^1_{i,j}$, and proceed as follows.
   (a) If the next scheduled message is p1, then for each $h \in [n]$ act as follows. If $\sigma_h = 0$, act according to Step $\widehat{\text{p1}}$ of BH-protocol. Otherwise (i.e., if $\sigma_h = 1$), commit to the entries of the adjacency matrix of the complete graph K_n and to a random permutation π.
   (b) Otherwise (i.e., the next scheduled message is p2), check (in hist) that the verifier has properly decommitted to all relevant values (and that the h^{th} bit of $\sigma^0_j \oplus \sigma^1_j$ equals σ_h for all $j \in [N]$) and abort otherwise.
   For each $h \in [\kappa]$ act as follows. If $\sigma_h = 0$, decommit to all the commitments (i.e., π and the adjacency matrix). Otherwise (i.e., if $\sigma_h = 1$), decommit only to the entries $(\pi(i), \pi(j))$ with $(i, j) \in C$ where C is an arbitrary Hamiltonian cycle in K_n.

Figure 6: The PROVE Procedure used by SIMULATE for Stage 2 messages

Theorem 15 *There exists a functionality f such that no protocol Π ϵ-securely realizes f in the individual leaky ideal world model for $\epsilon = \frac{1}{2} - \delta$, where δ is any constant fraction.*

We will prove the above theorem for the oblivious transfer (OT) functionality. Recall that for a two-party protocol Π to ϵ-securely realize a functionality f in the individual leaky ideal world model, it must hold that for *every* pair of input vectors for f and *every* auxiliary input to the adversary, a real world concurrent attack that is successful with non-negligible probability (say) ϵ implies an ideal world attack that succeeds with probability at least $\epsilon - \mathsf{negl}$. Now, suppose towards contradiction, we are given a protocol Π that $\epsilon(= \frac{1}{2} - \delta)$-securely realizes the OT functionality in the individual leaky ideal world model. We will exhibit a set of inputs for the honest sender and an auxiliary input for the adversarial receiver and show that the adversarial receiver can perform a concurrent attack to learn a secret value, called secret with probability 1. We will then prove that no adversarial receiver in the ideal world can learn secret with high enough probability, thus arriving at a contradiction.

Our proof builds upon the works of [AGJ+12, GKOV12] who prove the impossibility of concurrently secure OT in the plain model w.r.t. the standard definition of secure computation. Below, we recall the high-level proof outline of [AGJ+12] and then discuss the crucial differences with our setting:

1. Let Π be a protocol that implements the OT functionality. The first step is to construct an instance of the so called *chosen protocol attack* for Π. That is, given Π, we construct a protocol $\widehat{\Pi}$ such that protocol Π is insecure when executed concurrently with $\widehat{\Pi}$. There are three parties in the system: Alice and Eve running Π (as sender and receiver respectively); Eve and David running the "chosen protocol" $\widehat{\Pi}$ (as sender and receiver respectively). Then, by choosing appropriate inputs for Alice and David, we show that in this scenario, the man-in-the-middle Eve can learn a secret value secret with probability 1 (violating the security of Π).

2. The next step is to use *one time programs* (OTPs) [GKR08, GIS$^+$10] to eliminate David. In more detail, Eve simply gets (as auxiliary input) a set of one-time programs implementing the next message function of David. To execute these one-time programs, Eve is required to carry out a number of Π invocations. In these invocations, Alice is given the required keys and acts as the sender. Thus, now there are only Alice and Eve running a number of concurrent executions of Π. One can argue that Eve can still learn the secret value secret. This gives us a real world concurrent attack against Π.

3. The final goal is then to show the infeasibility of an adversarial receiver in the ideal world to learn secret with probability $1 - \mathsf{negl}$. Very roughly, this is done by relying on the security guarantee of the OTPs and the stand-alone security of Π against a cheating sender.

Now, note that in our individual leaky ideal world model, the ideal world adversary is allowed to obtain (bounded) leakage on the input of the honest party in each session. In the case of OT, this means that in each session, an adversarial receiver can legitimately receive one of the input strings of the honest sender in each session and additionally learn the other string "partially". Then, in the above approach, it is not immediately clear how to argue the infeasibility of a successful ideal world adversary. For example, we cannot directly rely on the security of OTP due to leakage on the secret keys.

We resolve the above problem by making use of the intrusion-resilient secret sharing (IRSS) scheme of [DP07] (see Section A.5). Very roughly, we secret-share each OTP wire key into three shares using the scheme of [DP07]. Then, for every OTP wire key pair $(\mathsf{key}^0, \mathsf{key}^1)$, the honest receiver is now required to engage in 3 executions of Π (instead of only one), where the input of the honest sender in each execution corresponds to one of the three secret shares of $(\mathsf{key}^0, \mathsf{key}^1)$. We then rely on the security of the IRSS scheme to argue that despite bounded leakage in ideal world session, an ideal world adversary cannot recover one of the keys in each key pair, and therefore must fail to output secret with non-negligible probability. We now give more details.

E.1 Chosen Protocol Attack

Let Π be a protocol that ϵ-securely realizes the OT functionality in the individual leaky ideal world model. Let s^0 and s^1 be two random strings (of appropriate length).

Consider the following scenario involving three parties, namely, Alice, Eve and David. Alice holds input values (s^0, s^1), while David holds (b, s^b) (where b is a random bit), as well as a random string secret. On the "left", Alice and Eve are involved in an execution of Π where Alice plays the role of the sender with inputs (s^0, s^1) and Eve plays the role of the receiver (with any input of its choice). On the "right", Eve and David are involved in an execution of the "chosen" protocol $\widehat{\Pi}$, described as follows. $\widehat{\Pi}$ consists of an execution of Π, where Eve plays the role of the sender with any inputs of its choice while David plays the role of the receiver with input bit b. At the end of the execution of Π, David checks whether his output $s' = s^b$. If this is the case, then David sends secret to Eve.

Now, it is easy to see that a malicious Eve can launch a "man-in-the-middle" attack, playing simultaneously with Alice and David. Specifically, by merely forwarding Alice's messages to David and David's back to Alice, she can learn the value secret with probability 1. However, if the "left" execution of Π between Alice and Eve is replaced with leaky ideal world for the OT functionality, then the attack does not carry through. (To avoid repetition, we skip proof details here and instead present them later more formally).

E.2 From General to Self-Composition

Note that the above attack is valid in the setting of concurrent general composition. However, we are interested in the setting of concurrent self composition, where only the OT protocol Π is executed concurrently. Towards this end, in this section, we describe how to eliminate David from the above scenario. In other words, we describe how the "right" interaction interaction between Eve and David can be replaced with multiple executions of Π between Alice and Eve.

Eliminating David. Let $\mathcal{F}^{\mathsf{David}}$ denote the (reactive) next message functionality of David in the protocol $\widehat{\Pi}$. The functionality $\mathcal{F}^{\mathsf{David}}$ is described as follows: $\mathcal{F}^{\mathsf{David}}$ has the pair (b, s^b) and a random string secret hardwired. On receiving as input a message a_i from the sender, $\mathcal{F}^{\mathsf{David}}$ outputs David's i^{th} message d_i in $\widehat{\Pi}$. Alternatively, let n be the round complexity of protocol Π. Let $N = n + 1$ be the number of messages sent by David in protocol $\widehat{\Pi}$. Then, note that we can think of N next-message functions $\mathcal{F}_1^{\mathsf{David}}, \ldots, \mathcal{F}_N^{\mathsf{David}}$, where each $\mathcal{F}_i^{\mathsf{David}}$ emulates $\mathcal{F}^{\mathsf{David}}$ for the $i^t h$ outgoing message of David in $\widehat{\Pi}$.

Prepare one time programs $\mathsf{OTP\text{-}msg}_1, \ldots, \mathsf{OTP\text{-}msg}_N$ where $\mathsf{OTP\text{-}msg}_i$ computes $\mathcal{F}_i^{\mathsf{David}}$. Let $\{\mathsf{key}_{i,k}^0, \mathsf{key}_{i,k}^1\}_{k=1}^K$ denote the set of secret keys for $\mathsf{OTP\text{-}msg}_i$. Now, for every key $\mathsf{key}_{i,k}^\ell$, run the secret sharing algorithm Share of the intrusion-resilient secret sharing scheme $\mathsf{IRSS} = (\mathsf{Share}, \mathsf{Rec})$ to compute shares $(\mathsf{key}_{i,k}^\ell[1], \mathsf{key}_{i,k}^\ell[2], \mathsf{key}_{i,k}^\ell[3]) \leftarrow \mathsf{Share}(1^\kappa)$. Let $\vec{\mathsf{key}}_{i,k}^\ell = (\mathsf{key}_{i,k}^\ell[1], \mathsf{key}_{i,k}^\ell[2], \mathsf{key}_{i,k}^\ell[3])$.

Inputs for Alice and Eve. Eve is given as auxiliary input, the one-time programs $\mathsf{OTP\text{-}msg}_1, \ldots, \mathsf{OTP\text{-}msg}_n$, as described above. Alice is given as input the values (s^0, s^1), and $(\vec{\mathsf{key}}_{i,k}^0, \vec{\mathsf{key}}_{i,k}^1)$, where $i \in [N]$, $k \in [K]$.

We now consider two kinds of executions of Π being carried out between Alice and Eve : one "main" OT execution where Alice uses (s^0, s^1) as her input, and $3k(N-1)$ "auxiliary" OT executions that allow Eve to obtain one set of secret keys to evaluate the one time programs. For clarity of exposition, we will refer to the "main" executions as Π^{main} and each of the auxiliary ones as Π^{David}.

E.3 Attack in the Real World

We now describe a real world concurrent attack on protocol Π. We describe the algorithms for the honest sender Alice and an adversarial receiver $\mathsf{Eve}^{\mathsf{real}}$.

Alice's program: Alice receives the values (s^0, s^1) as input for Π^{main} session, and the values $(\vec{\mathsf{key}}_{i,k}^0, \vec{\mathsf{key}}_{i,k}^1)$ as inputs for Π^{David} sessions. Here $i \in [N]$, and $k \in [K]$.

Alice plays the role of the honest sender (with appropriate inputs) in each execution of Π initiated by $\mathsf{Eve}^{\mathsf{real}}$.

$\mathsf{Eve}^{\mathsf{real}}$**'s program:** $\mathsf{Eve}^{\mathsf{real}}$ is given as auxiliary input the one time programs $\{\mathsf{OTP\text{-}msg}_i\}_{i=1}^N$. For $i = 1, \ldots, N$, do:

1. Upon receiving the i^{th} message from Alice in the execution of Π^{main}, say a_i, suspend (temporarily) the ongoing Π^{main} session. Parse $a_i = a_{i,1}, \ldots, a_{i,K}$.

2. For $k = 1, \ldots, K$, do:

   (a) Start three new Π^{David} sessions $\Pi^{\mathsf{David}}[1], \Pi^{\mathsf{David}}[2], \Pi^{\mathsf{David}}[3]$, in parallel, with Alice. In each session $\Pi^{\mathsf{David}}[\ell]$, $\mathsf{Eve}^{\mathsf{real}}$ plays the honest receiver with bit $a_{i,k}$ as input, while Alice plays the honest sender with input strings $(\mathsf{key}_{i,k}^0[\ell], \mathsf{key}_{i,k}^1[\ell])$.

   (b) When the three sessions are completed, compute $\mathsf{key}_{i,k}^{a_{i,k}} = \mathsf{Rec}(\mathsf{key}_{i,k}^{a_{i,k}}[1], \ldots, \mathsf{key}_{i,k}^{a_{i,k}}[3])$.

3. Run $\mathsf{OTP\text{-}msg}_i$ with keys $(\mathsf{key}_{i,1}^{a_{i,1}}, \ldots, \mathsf{key}_{i,K}^{a_{i,K}})$ and obtain output d_i as David's response to message a_i.

4. If $i \cdot j \leq N-1$, resume the suspended Π^{main} protocol and send d_i back to Alice as response. Otherwise, output the value secret received from $\mathsf{OTP\text{-}msg}_N$.

Lemma 17 $\mathsf{Eve}^{\mathsf{real}}$ *outputs* secret *with probability* 1.

The above lemma follows from the description of $\mathsf{Eve}^{\mathsf{real}}$.

E.4 Infeasibility of Ideal World Attack

We now consider the experiment where all the executions of the OT protocol Π between Alice and Eve are replaced with the corresponding ideal world executions for the OT functionality in the independent leaky ideal world model. We claim the following:

Lemma 18 *There exists no ideal world adversary Eve with auxiliary input* $\{\mathsf{OTP\text{-}msg}_i\}_{i=1}^{N}$*, that outputs* secret *with probability* $1 - \mathsf{negl}$.

Towards a contradiction, let us assume that there exists an ideal world attacker $\mathsf{Eve}^{\mathsf{ideal}}$ that successfully outputs secret with probability $1 - \mathsf{negl}$. $\mathsf{Eve}^{\mathsf{ideal}}$ has oracle access to the ideal OT functionality $\mathcal{F}_{\mathsf{OT}}$ in the independent leaky ideal world model, which for notational clarity we will (as before) divide into two kinds of OTs – $\mathcal{F}_{\mathsf{OT}}^{\mathsf{Main}}$ and $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}$. Then, we have two possible cases:

Case 1: With probability at least $\frac{1}{2} + \mathsf{non\text{-}negl}$, $\mathsf{Eve}^{\mathsf{ideal}}$ queries $\mathcal{F}_{\mathsf{OT}}^{\mathsf{Main}}$ for the same bit b that is hardwired in $\{\mathsf{OTP\text{-}msg}_i\}_{i=1}^{N}$. In this case, we construct a stand-alone cheating sender $\widehat{\mathcal{S}}$ that executes OT protocol Π with an honest receiver $\mathcal{R}$, and uses $\mathsf{Eve}^{\mathsf{ideal}}$ to learn $\mathcal{R}$'s secret input bit b with probability at least $\frac{1}{2} + \mathsf{non\text{-}negl}$. This will imply that Π is standalone insecure, a contradiction.

Case 2: With probability at most $\frac{1}{2} + \mathsf{negl}$, $\mathsf{Eve}^{\mathsf{ideal}}$ queries the same bit as in the OTPs. That is, with probability $\frac{1}{2} - \mathsf{negl}$, $\mathsf{Eve}^{\mathsf{ideal}}$ queries different bit. In this case, we will show via an information theoretic argument, that with probability at least $\frac{1}{4} - \mathsf{negl}$, $\mathsf{Eve}^{\mathsf{ideal}}$ cannot output secret. Since by assumption $\mathsf{Eve}^{\mathsf{ideal}}$ succeeds with probability at least $1 - \mathsf{negl}$, this is a contradiction.

These cases are analyzed below.

Case 1: We first consider the case where $\mathsf{Eve}^{\mathsf{ideal}}$ queries the same bit as in the OTPs with probability at least $\frac{1}{2} + \mathsf{non\text{-}negl}$. In this case, we will construct a stand-alone cheating sender $\widehat{\mathcal{S}}$ that can learn an honest receiver $\mathcal{R}$'s secret bit with probability at least $\frac{1}{2} + \mathsf{non\text{-}negl}$.

The cheating sender $\widehat{\mathcal{S}}$ works as follows:

Adversary $\widehat{\mathcal{S}}$:

1. Run the OTP simulator $\mathsf{Sim}_{\mathsf{OTP}}$ to generate simulated programs $\mathsf{OTP\text{-}msg}_1, \ldots, \mathsf{OTP\text{-}msg}_n$, where $\mathsf{OTP\text{-}msg}_i$ corresponds to the i^{th} next message function of David as described before. Run $\mathsf{Eve}^{\mathsf{ideal}}$ with auxiliary input $\mathsf{OTP\text{-}msg}_1, \ldots, \mathsf{OTP\text{-}msg}_N$.

2. $\mathsf{Eve}^{\mathsf{ideal}}$ may make leakage and output queries to each instance of $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}$. Without loss of generality, consider the queries made by $\mathsf{Eve}^{\mathsf{ideal}}$ for $\mathsf{OTP\text{-}msg}_i$ for some $i \in [N]$. For every $k \in [K]$:

- Let $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[1], \mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[2], \mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[3]$ denote the three ideal functionalities corresponding to the k-th key pair of $\mathsf{OTP\text{-}msg}_i$. Sample random strings $(\widehat{\mathsf{key}}_{i,k}^{0}[1], \ldots, \widehat{\mathsf{key}}_{i,k}^{0}[3])$, and $(\widehat{\mathsf{key}}_{i,k}^{1}[1], \ldots, \widehat{\mathsf{key}}_{i,k}^{1}[3])$. Let $q_1, \ldots, q_6$ denote the queries made by $\mathsf{Eve}^{\mathsf{ideal}}$ to the functionalities $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[1], \mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[2], \mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[3]$, where exactly three queries correspond to output queries (one each to $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[\ell]$) and the remaining three correspond to leakage queries (one each to $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[\ell]$). $\widehat{\mathcal{S}}$ does the following:
  (a) On receiving an output query $q_j = (\ell, a)$ such that exactly one previous output query $q_{j'}$ is of the form $q_{j'} = (\ell', a)$, query (i, k, a) to $\mathsf{Sim}_{\mathsf{OTP}}$ and receive a key $\mathsf{key}_{i,k}^{a}$ for the k-th wire of $\mathsf{OTP\text{-}msg}_i$. Now, compute secret shares $(\mathsf{key}_{i,k}^{a}[1], \ldots, \mathsf{key}_{i,k}^{a}[3]) \leftarrow \mathsf{Share}(\mathsf{key}_{i,k}^{a})$ such that the share values are "consistent" with the values learnt by $\mathsf{Eve}^{\mathsf{ideal}}$ so far (either via output or leakage queries). Update $(\widehat{\mathsf{key}}_{i,k}^{a}[1], \ldots, \widehat{\mathsf{key}}_{i,k}^{a}[3])$ with $(\mathsf{key}_{i,k}^{a}[1], \ldots, \mathsf{key}_{i,k}^{a}[3])$. Return $\widehat{\mathsf{key}}_{i,k}^{a}[\ell]$ to $\mathsf{Eve}^{\mathsf{ideal}}$.

  (b) Answer any other output query or leakage query q_j using the values $(\widehat{\mathsf{key}}_{i,k}^{0}[1], \ldots, \widehat{\mathsf{key}}_{i,k}^{0}[3])$ and $(\widehat{\mathsf{key}}_{i,k}^{1}[1], \ldots, \widehat{\mathsf{key}}_{i,k}^{1}[3])$.
- At some point, after returning t keys (for some $t < K$) for $\mathsf{OTP\text{-}msg}_i$, $\mathsf{Sim}_{\mathsf{OTP}}$ makes its one-time input query with some value x and requests the value $\mathcal{F}_i^{\mathsf{David}}(x)$.
- Parse this query as a message $\mathsf{msg}_i^{\widehat{\mathcal{S}}}$ and send it to $\mathcal{R}$.
- In response, $\mathcal{R}$ sends some message $\mathsf{msg}_i^{\mathcal{R}}$ back to $\widehat{\mathcal{S}}$. Upon receiving $\mathsf{msg}_i^{\mathcal{R}}$, $\widehat{\mathcal{S}}$ provides $\mathsf{msg}_i^{\mathcal{R}}$ as a response to $\mathsf{Sim}_{\mathsf{OTP}}$. Now, continue responding to $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}$ queries of $\mathsf{Eve}^{\mathsf{ideal}}$ in the same manner as above. Let us denote all the keys returned by $\mathsf{Sim}_{\mathsf{OTP}}$ for $\mathsf{OTP\text{-}msg}_i$ as keys_i. Then, we have that $\mathsf{OTP\text{-}msg}_i(\mathsf{keys}_i) = \mathsf{msg}_i^{\mathcal{R}}$.

3. If $\mathsf{Eve}^{\mathsf{ideal}}$ makes a query to $\mathcal{F}_{\mathsf{OT}}^{\mathsf{Main}}$ with input bit b, $\widehat{\mathcal{S}}$ outputs this bit b and halts.

Claim 7 *$\widehat{\mathcal{S}}$ outputs $\mathcal{R}$'s secret bit with probability at least* $\frac{1}{2} + \mathsf{non} - \mathsf{negl}$.

Proof (Sketch). It follows from the above description and the security of the IRSS scheme $\mathsf{IRSS} = (\mathsf{Share}, \mathsf{Rec})$ that $\widehat{\mathcal{S}}$ successfully mirrors its interaction with $\mathcal{R}$ by the interaction between $\mathsf{Eve}^{\mathsf{ideal}}$ and $\{\mathsf{OTP\text{-}msg}\}_{i=1}^{N}$. In particular, note that since $\mathsf{Eve}^{\mathsf{ideal}}$ is only allowed one output query and one leakage query to each of the functionalities $\mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[1], \mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[2], \mathcal{F}_{\mathsf{OT}}^{\mathsf{David}}[3]$ (as described above), we have that for every $i \in [N]$, $k \in [K]$, there exists bit a such that $\mathsf{Eve}^{\mathsf{ideal}}$ makes at most one output query q of the form (ℓ, a). Then, from Lemma 3, it follows that $\mathsf{Eve}^{\mathsf{ideal}}$ is a valid adversary for IRSS. Thus, we have that the wire secret key $\mathsf{key}_{i,k}^{\bar{a}}$ is indistinguishable from random to $\mathsf{Eve}^{\mathsf{ideal}}$.

Summing up, we have that the secret bit b in the simulated OTPs correspond to the input bit b of the receiver $\mathcal{R}$. Then, since (by our assumption) $\mathsf{Eve}^{\mathsf{ideal}}$ queries the same bit b as in the OTPs with probability at least $\frac{1}{2} + \mathsf{non\text{-}negl}$, Claim 7 follows.

Case 2: Now consider the case where $\mathsf{Eve}^{\mathsf{ideal}}$ queries the same bit as in the OTP with probability at most $\leq \frac{1}{2} + \mathsf{negl}$. This implies that with probability at least $\frac{1}{2} - \mathsf{negl}$, $\mathsf{Eve}^{\mathsf{ideal}}$ queries a different bit. In this case, we simply run $\mathsf{Eve}^{\mathsf{ideal}}$ with simulated OTPs $\mathsf{OTP\text{-}msg}_1, \ldots, \mathsf{OTP\text{-}msg}_N$. If at any point, $\mathsf{Eve}^{\mathsf{ideal}}$ makes a leakage query L to $\mathcal{F}_{\mathsf{OT}}^{\mathsf{Main}}$, then we simply answer it with a random string. Now, in this case, despite $(\frac{1}{2} - \epsilon)$ fraction of leakage learnt by $\mathsf{Eve}^{\mathsf{ideal}}$, we have that the string s_b is still information-theoretically hidden from $\mathsf{Eve}^{\mathsf{ideal}}$. Thus $\mathsf{Eve}^{\mathsf{ideal}}$ must *fail* to guess s_b correctly with probability at least $\frac{1}{4} - \mathsf{negl}$ and hence cannot output secret with probability at least $\frac{1}{4} - \mathsf{negl}$. Since we assume that $\mathsf{Eve}^{\mathsf{ideal}}$ succeeds with probability at least $1 - \mathsf{negl}$, this is a contradiction.

F Round Complexity Lower Bound for Joint Leaky Ideal World Model

In this section, we prove a lower bound on the round-complexity of protocols for achieving ϵ-security in the joint leaky ideal world model, with respect to black-box simulation. Specifically, we prove the following result:

Theorem 16 *Let ϵ be any inverse polynomial. Assuming dense cryptosystems, there exists a functionality f that cannot be ϵ-securely realized with respect to black-box simulation in the joint leaky ideal world model by any $\frac{\log(\kappa)}{\epsilon}$ round protocol.*

We prove the above theorem in the rest of this section. We start by giving an outline of our proof:

I. We first define a specific functionality f_{sig} for which our negative result will hold. Looking ahead, f_{sig} is the blind (leakage-resilient) signature functionality.

II. Now, assume for contradiction that there exists a $\frac{\log(\kappa)}{\epsilon}$-round protocol that ϵ-securely realizes f_{sig} with respect to black-box simulation in the joint leaky ideal world model. We will construct a specific real-model adversary $\mathcal{A}$ who corrupts party P_2 and interacts with honest P_1 in a fixed polynomial number of concurrent sessions of Π with a specific static schedule of messages. In particular, we will prove some specific properties of the adversarial schedule.

III. Finally, we show that every black-box ϵ-joint-ideal-leakage simulator must fail to simulate the view of $\mathcal{A}$.

We now proceed to give details. We first fix some notation that will be used in the proof. Let $\epsilon = \frac{1}{p(\lambda)}$. Let $q(\lambda) = \log(\lambda)p(\lambda)$ be the round-complexity of Π.

Part I. The blind (leakage-resilient) signature functionality. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ denote a $(1 - o(1))$-leakage resilient signature scheme, as defined in Section A.7. Recall from Theorem 9 that such a signature scheme exists assuming dense cryptosystems. The blind (leakage-resilient) signature functionality f_{sig} is defined as follows:

Inputs: Party P_1 gets a signing key sk as input, while P_2 gets the corresponding verification key vk and a randomly chosen message $s \in \{0,1\}^\kappa$ as input.

Outputs: P_1 gets no output, while P_2 gets $\sigma \leftarrow \mathsf{Sign}_{sk}(s)$.

Thus, the functionality f_{sig} is essentially $f_{\mathsf{sig}}(sk, s) = (\perp, \mathsf{Sign}_{sk}(s))$.

Part II. Adversary $\mathcal{A}$. Adversary $\mathcal{A}$ is provided as auxiliary input a random string $z \in \{0,1\}^\kappa$, as well as verification keys $vk_1, \ldots, vk_m$ as defined above. $\mathcal{A}$ schedules the protocol messages of $m = m(\kappa)$ sessions of Π in the following manner.

Adversarial Schedule. Recall that $q(\lambda)$ is the round complexity of protocol Π. That is, Π consists of q **slots**, where a slot is defined as a message from P_1 followed by a message from P_2. The adversarial schedule consists of $N = q^3$ "outer" sessions $\Pi_1^{\mathsf{out}}, \ldots, \Pi_N^{\mathsf{out}}$, and $K = q^3$ "inner" sessions $\Pi_1^{\mathsf{in}}, \ldots, \Pi_K^{\mathsf{in}}$, defined as follows.

Outer Sessions. The outer sessions $\Pi_1^{\mathsf{out}}, \ldots, \Pi_N^{\mathsf{out}}$ are scheduled in the following manner: Define q^2 **sets** $\mathsf{set}_1, \ldots, \mathsf{set}_{q^2}$, where each set set_i consists of multiple slots of outer sessions being executed in *parallel.* Specifically, for every outer session Π_i^{out}, choose q (out of q^2) sets uniformly at random. Let $\mathsf{set}_{i_1}, \ldots, \mathsf{set}_{i_q}$ denote the chosen sets, in their order of execution. Then, slot j of session Π_i^{out} is executed in set set_{i_j} (in parallel with the slots of other sessions for which set set_{i_j} is chosen).

Inner Sessions. Let $K = k \cdot q^2$, i.e., $k = q$. The "inner" sessions $\Pi_1^{\mathsf{in}}, \ldots, \Pi_K^{\mathsf{in}}$ are scheduled in the following manner: In every set set_i, $\mathcal{A}$ places sequential executions of k complete sessions $\Pi_{i \cdot k+1}^{\mathsf{in}}, \ldots, \Pi_{(i+1)\cdot k}^{\mathsf{in}}$ between the slots of the outer sessions being executed in parallel.

Thus, in total, we have exactly $m = N + K = 2q^3 = \mathrm{poly}(\kappa)$ number of sessions.

Adversary's Strategy. We now describe the strategy of $\mathcal{A}$ for the outer and inner sessions.

Outer Sessions. In every outer session Π_i^{out}, $\mathcal{A}$ behaves honestly (running the code of honest P_2) with a random message s_i^{out} as its input.

Inner Sessions. At the start of any inner session Π_i^{in}, $\mathcal{A}$ collects the partial transcript T_i of the protocol messages (across all sessions) generated so far and computes a message $s_i^{\mathsf{in}} \leftarrow F_K(T_i)$ by applying a pseudo-random function (PRF) F with a random key K to the transcript.[14] $\mathcal{A}$ then behaves honestly (running the code of honest P_2) with message s_i^{in} as its input in protocol Π_i^{in}.

Let $\Pi_1, \ldots, \Pi_m$ denote the m sessions, in the order of execution. When any session Π_i is completed, $\mathcal{A}$ checks whether the output σ_i is such that $\mathsf{Verify}_{vk_i}(s_i, \sigma_i) = 1$, where s_i is $\mathcal{A}$'s input in session Π_i, computed in the manner as described above (depending upon whether Π_i is an inner or outer session). If it is not the case, then $\mathcal{A}$ sends $\perp$, outputs all the output values received from the protocol sessions completed so far, and aborts. On the other hand, if $\mathcal{A}$ receives valid outputs in all the m sessions, then it outputs `accept`, along with all its inputs and output values from all the sessions.

This completes the description of adversary $\mathcal{A}$. We make the following two claims about the adversary $\mathcal{A}$:

Claim 8 *$\mathcal{A}$ outputs* `accept` *with probability* 1 *in a real-world concurrent execution with honest P_1.*

The above claim follows immediately from the description of $\mathcal{A}$. We now prove an important claim about the adversarial scheduling.

Claim 9 *Consider the sets $\mathsf{set}_1, \ldots, \mathsf{set}_{q^2}$ as defined above. We say that a set set_i covers an outer session Π_j^{out} (where $j \in [N]$) if set_i contains a slot of Π_j^{out}. Then, there exists a way of scheduling outer sessions (putting each session into q sets) such that there does not exist any collection of $(q \log q/4)$ sets that covers more than $(0.1/\sqrt{q})$ fraction of outer sessions*

Proof. We will prove the claim by probabilistic methods. To start with, consider *any* collection $\mathcal{C}$ consisting of $q \log q/4$ out of these q^2 sets. Below, we will denote an outer session s_i^{out} as an element e. We will say that an element e is put in a set if a slot of session s_i^{out} is allocated to that set. Now, take any element and put it randomly in exactly q out of these q^2 sets.

Claim 9.1 Probability that element e is not covered by sets in $\mathcal{C}$ is at least $1/\sqrt{q}$.

Proof.

$$\begin{aligned}
\Pr[e \text{ not covered by } \mathcal{C}] &= \left(1 - \frac{q \log q/4}{q^2}\right) \cdots \left(1 - \frac{q \log q/4}{q^2 - q + 1}\right) \\
&\geq \left(1 - \frac{q \log q}{4(q^2 - q + 1)}\right)^q \geq \left(1 - \frac{q \log q}{2q^2}\right)^q \\
&= \left(\left(1 - \frac{\log q}{2q}\right)^{\frac{2q}{\log q}}\right)^{\frac{\log q}{2}} > \frac{1}{\sqrt{q}}
\end{aligned}$$

[14]The key K for the PRF must be included in the auxiliary input of $\mathcal{A}$. We omit it from the description for simplicity of exposition.

The last inequality follows from the fact that for any $\delta > 0$, there exists a N_0 such that for all $n > N_0$ $|(1 - \frac{1}{n})^n| < \delta$. ■

Claim 9.2 Probability that more than $(0.1/\sqrt{q})$ fraction of outer sessions are covered by $\mathcal{C}$ is at most $e^{q^{2.5/50}}$.

Proof: Since the sets for the slots of each session/element are selected independently and indentically, we can apply the following chernoff bounds:

$$\Pr[X < (1-\delta)\mu] < e^{-\frac{\delta^2 \mu}{2}}$$

In our case, $\mu = N/\sqrt{q} = q^{2.5}$ and setting $\delta = 0.1$, we get the above claim.

Completing the proof: Taking the union bound over all collections of size $(q \log q/4)$, we get the following:

$$\Pr[\text{More than } (0.1N/\sqrt{q}) \text{ outer sessions are covered}] \leq C(q^2, q \log q/4) \cdot e^{q^{2.5/50}}$$

Now using the following inequality

$$C(n,k) \leq \left(\frac{en}{k}\right)^k$$

to get the following:

$$\Pr[\text{More than } (0.1N/\sqrt{q}) \text{ outer sessions are covered}] \leq \left(\frac{eq^2}{q \log q/4}\right)^{q \log q/4} \cdot e^{q^{2.5/50}} \leq e^{2q \log^2 q} . e^{q^{2.5/50}} < 1$$

The last inequality holds if $q >$??.

Hence, we get that there exists a way of putting these N elements into q^2 sets (putting each element into q sets) such that *no* choice of the collection $\mathcal{C}$ having $q \log q/4$ sets covers more than $0.1N/\sqrt{q}$ outer sessions. ■

Part III. Ruling out Black-Box Simulation for $\mathcal{A}$. We will now argue that any black-box simulator cannot successfully simulate the view of $\mathcal{A}$ with only ϵ-fraction of leakage on honest party P_1's inputs across the m sessions.

Lemma 19 *Except with negligible probability, there does not exist a ϵ-joint-ideal-leakage black-box simulator such that $\mathcal{A}$ outputs* `accept`.

Proof (Sketch). Recall that a simulator $\mathcal{S}$ works by extracting the input x of the adversary $\mathcal{A}$ in each session and then querying the ideal functionality with x to receive the correct output; this output is then used to complete the simulation of $\mathcal{A}$'s view. Now, further recall that the only advantage that a black-box simulator has over the real adversary is the ability to *rewind.* In other words, a black-box simulator extracts the inputs of $\mathcal{A}$ by rewinding. Also note that for sessions in which the simulator does not succeed in extracting the input of the adversary, the only way the simulator can complete the simulation is via using the additional leakage available in certain way.

Now, to extract the input of $\mathcal{A}$ in any outer session Π_i^{out}, $\mathcal{S}$ must rewind at least one slot of Π_i^{out} at at least once. Then, it follows from Claim 9 that if $\mathcal{S}$ rewinds at most $q \log q/4$ sets, then it will fail to extract in at least $0.9N/\sqrt{q}$ number of outer sessions. Simulator will need to use the leakage queries available in order to complete the simulation of uncovered outer sessions.

Also note that whenever a set set_j is rewound, all the k inner sessions $\Pi_{j_1}^{\mathsf{in}}, \ldots, \Pi_{j_k}^{\mathsf{in}}$ contained in set_j are executed again. Again $\mathcal{S}$ will have to use the leakage queries available in order complete the simulation in the rewound executions.

Below we will prove that the leakage available to the simulator is not even sufficient to complete the execution in even either one of the above described cases. More formally, we have the following claims.

Claim 10 *The simulator cannot successfully complete the execution of* $0.9N/\sqrt{q}$ *number of outer sessions in a straighline manner using* $\epsilon \cdot m$ *leakage bits.*

Proof. $\mathcal{S}$ must generate signatures for $0.9N/\sqrt{q}$ messages (each w.r.t., different verification key) with only ϵ fraction of leakage on the $m = N + K$ signing keys. Plugging in the values, we have that $\mathcal{S}$ must generate $0.9q^{2.5}$ different signatures (each w.r.t. different verification key), with only $\frac{\log(\lambda)}{q}$ fraction of leakage on $2q^3$ signing keys. Thus, it follows that there exists an inner session where $\mathcal{S}$ can only leak at most a constant fraction of the signing key. Then, if $\mathcal{S}$ still succeeds in generating all the signatures, we can construct a forger for the leakage-resilient signature scheme, which is a contradiction.

■

Claim 11 *The simulator cannot successfully complete the execution of all inner sessions and rewound executions when* $q \log q/4$ *sets are rewound and* $\epsilon \cdot m$ *is the allowed leakage.*

Proof. As noted above, whenever a set set_j is rewound, all the k inner sessions $\Pi^{\mathsf{in}}_{j_1}, \ldots, \Pi^{\mathsf{in}}_{j_k}$ contained in set_j are executed again. Let $\tilde{\Pi}^{\mathsf{in}}_{j_1}, \ldots, \tilde{\Pi}^{\mathsf{in}}_{j_k}$ denote the new executions. Then, it follows from the collision-resistance property of the pseudo-random function family F that for every j_ℓ, the input messages $s^{\mathsf{in}}_{j_\ell}, \tilde{s}^{\mathsf{in}}_{j_\ell}$ of $\mathcal{A}$ in $\Pi^{\mathsf{in}}_{j_\ell}, \tilde{\Pi}^{\mathsf{in}}_{j_\ell}$ respectively, are different (i.e., $s^{\mathsf{in}}_{j_\ell} \neq \tilde{s}^{\mathsf{in}}_{j_\ell}$). Since there are $q \log q/4$ rewound executions, we have that in order to make $\mathcal{A}$ output `accept`, $\mathcal{S}$ must generate signatures for $2k \cdot q \log q/4$ different messages, while only having access to $k \cdot q \log q/4$ output queries to f_{sig}, and ϵ fraction of leakage on the $m = N + K$ signing keys. Removing the output queries (and subtracting $k \cdot q \log q$ signatures), we have that $\mathcal{S}$ must generate signatures for $k \cdot q \log q$ messages (each w.r.t., different verification key) with only ϵ fraction of leakage on the $m = N + K$ signing keys. Plugging in the values, we have that $\mathcal{S}$ must generate $q^2 \log q$ different signatures (each w.r.t. different verification key), with only $\frac{\log(\lambda)}{q}$ fraction of leakage on $2q^3$ signing keys. Thus, it follows that there exists an inner session where $\mathcal{S}$ can only leak at most a constant fraction of the signing key. Then, if $\mathcal{S}$ still succeeds in generating all the signatures, we can construct a forger for the leakage-resilient signature scheme, which is a contradiction.

■