What Information is Leaked under Concurrent Composition?

Vipul Goyal[1], Divya Gupta[2⋆], and Abhishek Jain[3⋆⋆]

[1] Microsoft Research, India, vipul@microsoft.com
[2] UCLA, divyag@cs.ucla.edu
[3] MIT and Boston University, abhishek@csail.mit.edu

Abstract. A long series of works have established far reaching impossibility results for concurrently secure computation. 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 an adversary learn by launching a concurrent attack?* Inspired by the recent works on leakage-resilient protocols, we consider a security 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. (Intuitively, the amount of leakage required by the simulator upper bounds the security loss in the real world.)
We show for the first time that in the concurrent setting, it is possible to achieve *full security* for "most" of the sessions, while incurring significant loss of security in the remaining (fixed polynomial fraction of total) sessions. We also give a lower bound showing that (for general functionalities) this is essentially optimal. Our results also have interesting implications to bounded concurrent secure computation [Barak-FOCS'01], as well as to precise concurrent zero-knowledge [Pandey et al.-Eurocrypt'08] and concurrently secure computation in the multiple ideal query model [Goyal et al.-Crypto'10]
At the heart of our positive results is a new simulation strategy that is inspired by the classical set covering problem. On the other hand, interestingly, our negative results use techniques from leakage-resilient cryptography [Dziembowski-Pietrzak-FOCS'08].

1 Introduction

Concurrently Secure Computation. Traditional security notions for cryptographic protocols such as secure computation [38, 16] were defined for a *standalone* setting, where security holds only if a single protocol session is executed

⋆ Work done in part while visiting Microsoft Research, India.

⋆⋆ Supported by NSF Contract CCF-1018064 and DARPA Contract Number: FA8750-11-2-0225. The author also thanks RISCS (Reliable Information Systems and Cyber Security) Institute. Work done in part while visiting Microsoft Research, India.

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 [8, 9, 26, 25, 27, 3, 19, 1, 15]. 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 [31, 4, 10, 13, 24], input indistinguishable computation [29, 13], multiple-ideal query model [20], etc.[4] These results suggest that somehow, *not all security is lost in the concurrent setting.* 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 we "measure" the amount of security loss that must *occur in a concurrent session? Can we achieve* full security *in some (or even most) of the sessions fully while incurring security loss in the remaining sessions?*

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.

A Leaky-Ideal World Approach. We adopt the "leaky-ideal world" approach of Goldreich and Petrank [17] (recently used in works on leakage-resilient protocols; see below) to quantify the information leakage to the adversary in concurrently secure computation. Specifically, generalizing the approach of [17], we

[4] 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 [7, 2, 22]. However the focus of the current work is the *plain model*.

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 [17]) 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 private information potentially leaked to the real adversary during the concurrent protocol executions.

We remark that the our ideal model resembles that considered in the recent works on leakage-resilient secure computation protocols [14, 5, 6]. 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, while [14, 5, 6] use the leaky ideal world approach to bound the security loss in the real world due to leakage attacks, we use the leaky ideal world approach to bound the security loss in the real world due to concurrent attacks. Nevertheless, 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.

We now describe our security model in more detail. Concretely, we consider two notions of leaky ideal world, described as follows.

Ideal world with joint leakage. Let there be m concurrent sessions with the honest party input in the i^{th} session denoted by x_i. In the *joint* leakage model, the simulator is allowed to query the trusted party with efficiently computable leakage functions L_i and get $L_i(\boldsymbol{X})$ in return (where $\boldsymbol{X} = (x_1, \ldots, x_m)$). The constraint is that throughout the simulation, the total number of bits leaked $\sum L_i(\boldsymbol{X})$ is at most $\epsilon|\boldsymbol{X}|$. If this is the case, we say that the protocol is ϵ-secure in the joint leakage model. In this model, our main result is a positive one, as we discuss below.

Ideal world with individual leakage. In the *individual* leakage model, in every 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 ϵ-secure in the individual leakage model. As we discuss below, in this model, our main result is a negative one. This brings us to our next model.

1.1 Our Results

We consider the setting of unbounded concurrent composition in the plain model. We allow for static corruptions and assume that the inputs of honest parties are a priori fixed. We now describe our main results along with some applications.

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

Theorem 1. *(Informally stated.) Let f be any functionality. Assuming 1-out-of-2 oblivious transfer (OT), for every polynomial poly(n), there exists a protocol that ($\epsilon = \frac{1}{poly(n)}$)-securely realizes f in the joint leakage model.*

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. *Let f be any functionality. Assuming 1-out-of-2 OT, for every polynomial poly(n), there exists a protocol that ($\epsilon = \frac{1}{poly(n)}$)-securely realizes f in the joint leakage model s.t. the security of at most ϵ fraction of the sessions is compromised, while the remaining sessions are fully secure.*

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 $\epsilon < \frac{1}{m|\boldsymbol{X}|}$, 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 $\epsilon m|\boldsymbol{X}|$ which is less than 1. Hence, positive results for bounded concurrent secure computation [25, 33, 32] 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.) Let f be any functionality. Assuming 1-out-of-2 oblivious transfer, there exists a protocol that securely realizes f 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. 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* and develop, as our main technical contribution, a new **sparse rewinding strategy**. Very briefly, unlike known concurrent rewinding techniques [37, 23, 36, 30] that are very "dense", we rewind "small intervals" of the execution transcript, while still guaranteeing extraction in all of the sessions. Very roughly, by rewinding small intervals (only a few times), we are able to minimize the cost and obtain our positive result.

Our sparse rewinding strategy also yields other interesting applications that we discuss below in (III).

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 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).[5]

Leakage-resilient One-Time Programs. Of independent interest, the techniques used in our negative result also yield a new construction of one-time programs [18] where the adversary can query the given hardware tokens once (as usual), and *additionally* leak once on the secrets stored in each token in any arbitrary manner (as long as the total leakage is a constant fraction of the secrets). Our key technical tool in constructing such a gadget is the intrusion-resilient secret sharing scheme of [12]. In an independent work, Jain et al. [21] also consider the problem of constructing leakage-resilient OTPs. See the full version for details.

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 discussed above, along the way to developing our main positive result, we develop a new *sparse rewinding strategy* that leads to other interesting applications. 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. The notion of precise zero-knowledge [28] 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 leakage-resilient zero-knowledge [14], concurrently secure protocols [20], etc.

Pandey et al. [30] study the problem of precise *concurrent* zero-knowledge (cZK) and give a protocol with the following parameters. Let t be the actual running time of the verifier. Then, their protocol has round complexity n^{δ} (for

[5] 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.

any constant $\delta \leq 1$) and knowledge precision $c \cdot t$ where c is a large constant depending upon the adversary.[6]

Our sparse rewinding strategy directly leads to a new construction of precise cZK, improving upon [30] *both* in terms of round complexity as well as knowledge precision.

Theorem 4. *Assuming one way functions, there exists a cZK protocol with poly-log round-complexity and 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 et al. proposed the multiple ideal query (MIQ) model, where for every session in the real world, the simulator is allowed to query 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 MIQ model since the additional output queries in this model can simply be viewed as leakage observed by the simulator in our model. Indeed, our positive result described in (I) can be stated as an improved result in the MIQ model since leaking the function output (multiple times) is "no worse" than leaking the entire secret input of the honest party. We defer further discussion to the full version due to lack of space.

Theorem 5. *(Informally stated.) Let* f *be any functionality. Assuming 1-out-of-2 OT, 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).*

1.2 Our Techniques

Here we give an overview of the underlying techniques used in our positive result.

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 [16]: 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 [3] 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?

[6] [30] also give a construction requiring only $\omega(\log n)$ rounds, however, the knowledge precision achieved in this case is super-linear.

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 may exist entire other 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 [20]).

Indeed, some such problem is rather inherent as indicated by various impossibility results [27, 3, 19, 1, 15]. 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 [23, 36, 30, 20], the "main" thread of protocol execution is divided into various blocks (2 blocks in [23, 36] and n blocks in [30, 20]). 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 [37, 35, 11], 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 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 [23, 36]. 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 [23, 36] 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 [30].

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 [23, 36, 34, 30] 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, [36, 34] 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 [36, 34]. 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 [3]. We recall that there are several problems that arise with such a compilation. First, the security of the [3] construction is analyzed only for the setting where all the statements being proven by honest parties are fixed in advance. Secondly, the extractor of [3] 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. [20]. Indeed, Goyal et. al. presented an approach which can be viewed as a technique to correctly compile a semi-honest secure protocol with [3]. We adopt their approach to construct our final protocol.

2 Our Model

In this section, we present a brief overview of our security model, with details deferred to the full version. 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.

In this work, we consider a malicious, static adversary. The scheduling of the messages across the concurrent executions is controlled by the adversary. We allow the adversary to start arbitrarily polynomial number of concurrent session. Also, we consider the fixed input setting, i.e. the inputs of the honest party across all sessions is fixed in advance. Finally, we consider *computational* security only and therefore restrict our attention to adversaries running in probabilistic polynomial time.

We consider two security models that differ in the nature of ideal world leakage available to the simulator. In both of these security models, the real world is the same as in the standard security model for concurrently secure computation. The real concurrent execution of Π with security parameter κ, input vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and auxiliary input z to $\mathcal{A}$, denoted $\text{REAL}_{\Pi,\mathcal{A}}(\kappa, \boldsymbol{x}, \boldsymbol{y}, z)$, is defined as the output pair of the honest party and $\mathcal{A}$, resulting from the above real-world process. Also, in each of the ideal world experiments described below, the ideal execution of a function $\mathcal{F}$ with security parameter κ, input vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and auxiliary input z to $\mathcal{S}$, denoted $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\kappa, \boldsymbol{x}, \boldsymbol{y}, z)$, is defined as the output pair of the honest party and $\mathcal{S}$ from the ideal execution.

Concurrently Secure Computation in the Joint Leaky Ideal World Model. In this model, 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 $L(\boldsymbol{x})$ over honest party inputs $\boldsymbol{x}$ across all sessions and returns it to the adversary.

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.*

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 $\boldsymbol{x} \in X^m$, $\boldsymbol{y} \in Y^m$, every $z \in \{0,1\}^*s$,*

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

Concurrently Secure Computation in the Individual Leaky Ideal World Model. In this model, for every session i, the ideal adversary may send one leakage query of the form (i, L) to the trusted party and learn $L(x_i)$ (where x_i is the input of the honest party in session i).

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 $\boldsymbol{x} \in X^m$, $\boldsymbol{y} \in Y^m$, every $z \in \{0,1\}^*s$,*

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

3 Framework for Cost-based rewinding

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 3.1) 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 $\langle C, R \rangle$.

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

Let $\text{COM}(\cdot)$ denote the commitment function of a non-interactive perfectly binding string commitment scheme. 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 COM, with fresh randomness each time. Let $B \leftarrow \text{COM}(\sigma)$, and $A^0_{i,j} \leftarrow \text{COM}(\alpha^0_{i,j})$, $A^1_{i,j} \leftarrow \text{COM}(\alpha^1_{i,j})$ for every $i \in [\ell], j \in [N]$.
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 $\mathbf{slot}_j$ 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 [36,

30] 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[7] C, then the cost incurred by our simulator is only $C(1 + \epsilon(N, \kappa))$, where is ϵ is a small fraction.

3.2 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.

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 [36, 30]. 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[8]. Note that if each message is of unit cost, then this dividing strategy is exactly identical to [36].[9] The novel idea underlying our rewinding technique is that unlike [36, 30], our simulator only rewinds a small subset of these sets while still guaranteeing extraction. In other words, unlike [36, 30], ours is a "sparse" rewinding strategy.

We now describe our rewinding strategy by using an analogy to the classical set covering problem. Recall that in the 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

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

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

[9] If cost of a message is the time taken by the adversary to generate that message, then this dividing strategy is exactly identical to [30].

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[10]. 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 $(\mathsf{RL} + 1)$.

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 $\text{SIMULATE}(C, \mathsf{st}_0, \phi, 0, r_m, r_s)$ 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

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

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 ℓ 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: $(\mathsf{st}, \mathsf{hist}, r_m, r_s)$.

Fig. 1. The cost-based content oblivious simulator $\textsc{Simulate}$

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.

Procedure $\textsc{Simulate}$. 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 $\textsc{PickSet}$ to choose sets.

At a high level, $\textsc{Simulate}$ 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 $\textsc{PickSet}$ procedure to choose a list of sets on T, say $\textsc{psets}$, which it will rewind. Here each set will be denotes by the corresponding tuple $(\mathsf{s\text{-}point}, \mathsf{e\text{-}point})$.
2. Next, $\textsc{Simulate}$ augments each entry of $\textsc{psets}$ with two additional entries to create a new list $\tilde{\textsc{psets}}$ 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 $\textsc{SimMsg}$ (described below) when simulation reaches $\mathsf{s\text{-}point}$.
3. $\textsc{Simulate}$ 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[11], 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.
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 $\tilde{\text{PSETS}}$, it updates the corresponding entries with current state st of $\mathcal{A}$ and history hist of messages.

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

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 $\text{PS}\tilde{\text{E}}\text{TS}$. Procedure SimMsg is described formally in Figure 2.

The SimMsg($\mathsf{ctr}, c, \mathsf{st}, \mathsf{hist}, r_m, \text{PS}\tilde{\text{E}}\text{TS}$) 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 $\text{PS}\tilde{\text{E}}\text{TS}$ 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: $(\mathsf{st}, \mathsf{hist}, r'_m, \text{PS}\tilde{\text{E}}\text{TS})$.

Fig. 2. The SimMsg Procedure

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.

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 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 the case when $\epsilon = 1/\text{poly}(\kappa)$, we do not need to assume the existence of collision resistent hash functions. Protocol Π is essentially the protocol of [20] 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

1. Agrawal, S., Goyal, V., Jain, A., Prabhakaran, M., Sahai, A.: New impossibility results on concurrently secure computation and a non-interactive completeness theorem for secure computation. In: CRYPTO (2012)
2. Barak, B., Canetti, R., Nielsen, J., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: FOCS (2004)
3. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: FOCS (2006)
4. Barak, B., Sahai, A.: How to play almost any mental game over the net - concurrent composition using super-polynomial simulation. In: Proc. 46th FOCS (2005)
5. Bitansky, N., Canetti, R., Halevi, S.: Leakage-tolerant interactive protocols. In: TCC (2012)
6. Boyle, E., Garg, S., Jain, A., Kalai, Y.T., Sahai, A.: Secure computation against adaptive auxiliary information. In: CRYPTO (2013)
7. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC (2002)
8. Canetti, R., Fischlin, M.: Universally composable commitments. In: CRYPTO (2001)
9. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Eurocrypt (2003)
10. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: FOCS (2010)
11. Deng, Y., Goyal, V., Sahai, A.: Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In: FOCS (2009)
12. Dziembowski, S., Pietrzak, K.: Intrusion-resilient secret sharing. In: FOCS (2007)

13. Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently secure computation in constant rounds. In: Eurocrypt (2012)
14. Garg, S., Jain, A., Sahai, A.: Leakage-resilient zero knowledge. In: CRYPTO (2011)
15. Garg, S., Kumarasubramanian, A., Ostrovsky, R., Visconti, I.: Impossibility results for static input secure computation. In: CRYPTO (2012)
16. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC (1987)
17. Goldreich, O., Petrank, E.: Quantifying knowledge complexity. In: FOCS (1991)
18. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: CRYPTO (2008)
19. Goyal, V.: Positive results for concurrently secure computation in the plain model. In: FOCS (2012)
20. Goyal, V., Jain, A., Ostrovsky, R.: Password-authenticated session-key generation on the internet in the plain model. In: CRYPTO (2010)
21. Jain, A., Prabhakaran, M., Sahai, A., Wadia, A.: Oblivious transfer from any leaky functionality. In: Personal Communication (2013)
22. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Eurocrypt (2007)
23. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In: STOC (2001)
24. Lin, H., Pass, R.: Black-box constructions of composable protocols without set-up. In: CRYPTO (2012)
25. Lindell, Y.: Bounded-concurrent secure two-party computation without setup assumptions. In: STOC (2003)
26. Lindell, Y.: General composition and universal composability in secure multi-party computation. In: FOCS (2003)
27. Lindell, Y.: Lower bounds for concurrent self composition. In: TCC (2004)
28. Micali, S., Pass, R.: Local zero knowledge. In: STOC (2006)
29. Micali, S., Pass, R., Rosen, A.: Input-indistinguishable computation. In: FOCS (2006)
30. Pandey, O., Pass, R., Sahai, A., Tseng, W.L.D., Venkitasubramaniam, M.: Precise concurrent zero knowledge. In: Eurocrypt (2008)
31. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Eurocrypt (2003)
32. Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: STOC (2004)
33. Pass, R., Rosen, A.: Bounded-concurrent secure two-party computation in a constant number of rounds. In: FOCS (2003)
34. Pass, R., Tseng, W.L.D., Venkitasubramaniam, M.: Concurrent zero knowledge, revisited. In: Manuscript (2012)
35. Pass, R., Venkitasubramaniam, M.: On constant-round concurrent zero-knowledge. In: TCC (2008)
36. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS (2002)
37. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. In: Eurocrypt (1999)
38. Yao, A.C.C.: How to generate and exchange secrets. In: FOCS (1986)