Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority

Rafael Pass
Royal Institute of Technology
Stockholm, Sweden
rafael@nada.kth.se

ABSTRACT

We show how to securely realize any multi-party functionality in a way that preserves security under an a-priori bounded number of concurrent executions, regardless of the number of corrupted parties. Previous protocols for the above task either rely on set-up assumptions such as a Common Reference String, or require an honest majority. Our constructions are in the plain model and rely on standard intractability assumptions (enhanced trapdoor permutations and collision resistant hash functions).

Even though our main focus is on feasibility of concurrent multi-party computation we actually obtain a protocol using only a constant number of communication rounds. As a consequence our protocol yields the first construction of constant-round *stand-alone* secure multi-party computation with a dishonest majority, proven secure under standard (polynomial-time) hardness assumptions; previous solutions to this task either require logarithmic round-complexity, or subexponential hardness assumptions.

The core of our protocol is a novel construction of (concurrently) simulation-sound zero-knowledge protocols, which might be of independent interest.

Finally, we extend the framework constructed to give a protocol for secure multi-party (and thus two-party) computation for any number of corrupted parties, which remains secure even when *arbitrary subsets of parties* concurrently execute the protocol, possibly with *interchangeable* roles. As far as we know, for the case of two-party or multi-party protocols with a dishonest majority, this is the first positive result for *any* non-trivial functionality which achieves this property in the plain model.

Categories and Subject Descriptors

F.1.2 [**Theory of Computation**]: Interactive and reactive computation

General Terms

Theory

STOC'04, June 13–15, 2004, Chicago, Illinois, USA.

Keywords

concurrent composition, secure multi-party computation, simulation-sound zero-knowledge, constant-round protocols

1. INTRODUCTION

The task of *secure multi-party computation* involves n parties $P_1, ..., P_n$, that wish to jointly and securely compute a functionality $f(\bar{x}) = (f_1(\bar{x}), ..., f_n(\bar{x}))$ of their corresponding private inputs $\bar{x} = x_1, ..., x_n$, such that party P_i receives the value $f_i(\bar{x})$. This functionality may be probabilistic in which case $f_i(\bar{x})$ is a random variable. Loosely speaking, the security requirements are that the parties learn nothing more from the protocol than their prescribed output, and that the output of each party is distributed according to the prescribed functionality. This should hold even in case that a subset of the parties maliciously deviates from the protocol. Shortly after its conceptualization, very strong results were established for secure multi-party computation. Specifically, it was shown that *any* probabilistic polynomial-time computable multi-party functionality can be securely computed, regardless of the number of corrupted parties, assuming the existence of enhanced trapdoor permutations [38, 22].

1.1 Concurrent Multi-Party Computation

The original setting in which secure multi-party protocols were investigated allowed the execution of a single instance of the protocol at a time. This is the so called *stand-alone* setting. A more realistic setting, however, is one which allows the *concurrent* execution of protocols. In the concurrent setting (originally introduced in the context of zero-knowledge [18, 17]), many protocols are executed at the same time. This setting presents the new risk of a coordinated attack in which an adversary interleaves many different executions of a protocol and chooses its messages in each instance based on other partial executions of the protocol.

It has been noticed that security of a specific protocol in the stand-alone setting does not necessarily imply its security in the, more demanding, concurrent setting. It is thus of great relevance to examine whether the original feasibility results for multi-party computation in the stand-alone setting still hold when many copies of the protocol are executed concurrently.

Multi-party computation with honest majority. For the case of an honest majority, Canetti [9], based on the works of Ben-Or, Goldwasser and Wigderson [6], and Rabin and Ben-Or [34], show that a protocol for secure multi-party

computation that composes concurrently, can be achieved. (In fact, the protocol of [9] achieves the even stronger property of Universal Composability. See [9] for more details.) The proof of [9] relies on the fact that in the multi-party setting with an honest majority a specific simulation technique called *one-pass black-box simulation* (that is black-box simulation without the use of *rewinding*) can be performed. We note that this technique is inherently bound to fail in the case of a dishonest majority [11].

Multi-party computation with a dishonest majority. For the case of a dishonest majority, on the other hand, the situation is less satisfactory. The only previous positive results in the plain model (without set-up assumptions) are in the special-case of two-party computation and can be summarized as follows:

- Lindell demonstrated the feasibility of m-bounded concurrent two-party computation using $O(m)$ communication rounds. [27]
- Recently, Pass and Rosen showed that the notion of m-bounded concurrent two-party computation could be achieved using only $O(1)$ communications rounds. [32].

The notion of m-bounded concurrent composition, first considered by Barak [1] in the context of zero-knowledge, relates to a restricted form of concurrent composition, where an a-priori bound m on the number of concurrent executions is specified before the protocol is constructed. Certainly the notion of unbounded concurrency, i.e, security is guaranteed even under an unbounded number of concurrent executions, is more appealing than that of bounded concurrency. However, a recent, surprising, result by Lindell [29] shows that unbounded concurrent two-party computation can not be achieved in general.[1] On the other hand, in the special case of zero-knowledge proofs, security under unbounded concurrent composition can be achieved [35].

We note that the lower bound by Lindell also extends to the multi-party setting with a dishonest majority. The upper bounds of [27] and [32], however, do not seem to directly extends to the multi-party setting, leaving the following important question unanswered:

Does there exist a protocol for secure multi-party computation with a dishonest majority whose security is preserved under an a-priori bounded number of concurrent executions?

This is the question that we address in the current work.

Concurrent composition with set-up assumptions. Before we state our results, let us mention a few words about the use of set-up assumption in order to deal with concurrency. The literature discusses a wide array of set-up assumptions, most notably the timing assumption [17], the Public-Key models [10, 15] and the Common Reference String (CRS) model [9, 13]. Whereas the former assumptions have been primarily used to obtain concurrent composition of zero-knowledge proofs, the latter assumption has been also used to obtain strong composition theorems for secure multi-party computation. Specifically, in the model assuming the existence of a Common Reference String, it has been shown how to achieve the very strong notion of Universal Composability for both two-party and multi-party secure computation [9, 13]. In particular, universal composability implies security under unbounded concurrent composition.

Nevertheless, although very strong result have been achieved in the CRS model it is still of both theoretical and practical interest to investigate the possibility of constructing protocols in the plain model, without extra set-up assumption, that compose concurrently. Firstly, in some settings it might be difficult to obtain the trusted set-up phase needed in order to set up a common reference string. Secondly, recent work seems to indicate that the CRS model might not reflect the security concerns than one would like to address in reality [31]. Loosely speaking, standard simulation-based definitions of security in the CRS model do not cover certain "natural" security properties that are satisfied in the plain model (e.g., *deniability* of protocols). Moreover, known impossibility results for composition in the plain model (e.g., [12, 27]) will apply to any protocol that satisfies these "natural" security properties in the CRS model.

1.2 Our Results

In this work we address the above mentioned question and present a result of a positive nature. Specifically, we show that for any n-party functionality (two-party functionalities being a special case) there exist a protocol that is secure under bounded-concurrent composition, regardless of the number of corrupted parties.

The security of our protocol relies on the existence of enhanced trapdoor permutations [19], as well as on the existence of collision resistant hash functions. For an integer m, the notion of m-bounded concurrent composition refers to a setting in which at most m copies of a protocol are executed concurrently (typically, $m = m(k)$ is a fixed polynomial in the security parameter k). Our results are stated below.

THEOREM 1. *Assume the existence of enhanced trapdoor permutations and the existence of collision-resistant hash functions. Then, for any n-party functionality f and for any integer m, there exists a protocol Π that securely computes f under m-bounded concurrent composition, regardless of the number of corrupted parties. Moreover, the number of communication rounds is constant.*

Although our main focus is on feasibility of multi-party computation with a dishonest majority that composes concurrently, we actually achieve the best one could hope for in terms of round complexity. By doing so we get the following corollary which is interesting in its own:

COROLLARY 1. *Assume the existence of enhanced trapdoor permutations and the existence of collision-resistant hash functions. Then, for any n-party functionality f there exists a protocol Π that securely computes f, regardless of the number of corrupted parties. Moreover, the number of communication rounds is constant.*

Corollary 1 gives the first constant-round construction for (stand-alone) secure multi-party computation under standard polynomial-time hardness assumptions.[2] The only pre-

[1] Note that the lower bound of [29] relies on the fact that the honest party chooses its input adaptively. It might still be possible to construct a protocol for unbounded concurrent two-party computation for the case when the honest party chooses its inputs ahead of the protocol executions.

[2] In the case of stand-alone security our security definition

vious constant-round solution is a protocol by Katz, Ostrovsky and Smith [26], which relies on the assumption of dense cryptosystems and collision resistant hash functions, both secure against subexponential sized circuits, as well as enhanced trapdoor permutations. The protocol of Katz et al, in fact, relies on the non-malleable zero-knowledge protocol of Barak [2] and inherits both the subexponential hardness assumptions and the complexity of this protocol. Katz et al. also give a protocol based on polynomial-time hardness assumptions, this time relying on the technique of Chor and Rabin [14], although this protocol requires a logarithmic round-complexity.

We further note that an important extra feature of our approach is the relative simplicity of the solution in comparison to the works of [27, 32, 26, 2].

On the model of computation: In this work we consider a *malicious static* computationally bounded (i.e., probabilistic polynomial-time) adversary that is allowed to corrupt an arbitrary number of parties. That is, before the beginning of the interaction the adversary corrupts a subset of the players that may deviate arbitrarily from the protocol.

The parties are assumed to be connected through a point-to-point *asynchronous* public network, without guaranteed delivery of messages.

Following the standard of the area [19], we assume that the communication channels are *authenticated*.

The focus of this work is not on fairness, and we therefore present a definition where the adversary always receives the output of the functionality first and then decides which of the honest parties that will receive its output (c.f. *Secure computation with abort and no fairness* in the taxonomy of [24]).

The concurrent setting considered is a generalization of Lindell's treatment [27, 29] for concurrent secure two-party computation, which, in turn, is adapted from previous works on concurrent zero-knowledge [17]. We consider a *single set* of n parties that are running many concurrent executions of the same multi-party protocol, each time playing the same roles. The honest parties may adaptively choose their inputs to the protocol based on outputs received from previous executions. However, in each instance of the protocol the honest parties are required to act independently. The adversary, on the other hand, may arbitrarily coordinate its actions between the different executions.

The restriction on m-bounded concurrency is formalized by requiring that during the execution of a single instance of the protocol, messages from no more than $m-1$ other executions are exchanged.

New techniques: Already in the case of (bounded) concurrent two-party computation, it was noticed that the main technical difficulty consists of constructing two zero-knowledge protocols that are *simulation-sound* with respect to each other [27, 32]. Simulation-soundness (introduced by Sahai in a different context [37]) means that the soundness of each of the protocols is preserved even when the other protocol is simulated at the same time with the roles of the prover and verifier reversed. The problem is related to the notion of *non-malleability* [16]. Unfortunately, since soundness, here, should hold also when *concurrently* simulating many zero-knowledge protocols, known techniques for achieving non-malleability, such as [16, 2], do not seem to apply. The works of [27, 32] constructed protocols satisfying this notion. We note that their techniques inherently make use of the fact that in the two-party setting only *two* protocols need to be constructed. In fact, both [27] and [32] construct two asymmetric protocols and rely on different methods for proving simulation-soundness in each direction.

In the multi-party setting, on the other hand, the difficulty consists of constructing a family of *polynomially* many zero-knowledge protocols that are simulation-sound with respect to each other under concurrent executions. The main technical contribution of this work is the construction of a family of constant-round protocols satisfying this property. Although our techniques partly rely on techniques from [27] and [32], the methods for achieving simulation-soundness are, in fact, quite different.

Interestingly, the problem of constructing a family of polynomially many zero-knowledge arguments, such that are the protocols are simulation-sound with respect to each other under *parallel* executions, has occurred before in the context of stand-alone secure multi-party computation [26]. Previously, the only known constant-round protocols satisfying this (potentially weaker) property were the non-malleable zero-knowledge protocols of Barak [2] (with extensions in [26]). Our construction improves upon these protocols both in terms of assumptions (we rely on standard polynomial-time hardness assumptions, while the protocols of [2] rely on subexponential hardness assumptions), and in terms of simplicity.

We believe that our techniques for achieving simulation-soundness, and the zero-knowledge protocols constructed, are of independent interest and might find applications elsewhere.

1.3 Extensions to Arbitrary Sets of Parties

Until now we have focused on concurrent composition of multi-party protocols for a *single set of parties*. In this setting two (or more) parties are executing multiple concurrent executions of the same protocol, each time playing the same roles (i.e., using *fixed roles*). The setting originated in works on concurrent zero-knowledge [18, 17], where in particular, one party acts as prover in all concurrent executions and another party acts as verifier in all executions. Alternatively, this setting can be seen as modeling many different provers and verifiers with the restriction that the adversary may only corrupt either parties acting as provers or parties acting as verifiers. In other words, the adversary may not corrupt both a party acting as prover and a party acting as verifier.

A more general setting considers concurrent composability of a protocol with *arbitrary subsets of parties* [28]. In this setting, arbitrary, and possibly intersecting, sets of parties concurrently execute the same protocol. In particular, there is no restriction on what role a specific party plays in the protocol, and the roles might be *interchanged*. This means

reduces down to *secure computation with abort and no fairness* in the taxonomy of [24]. As was noted in [24], in a setting with a broadcast channel, any protocol for secure computation with abort can be transformed (in a round-preserving way) into a protocol secure according to the more standard definition of secure computation with *unanimous abort*. This is obtained by letting the parties broadcast whether they received output or not, at the end of the protocol with abort.

that a party may play the role of P_1 in one execution and simultaneously play the role of P_2 in a different execution. In the case of zero-knowledge proofs, this would mean that a party may act as prover in some of the concurrent executions and act as verifier in the others.

In Section 3.3 we show that the framework developed in this paper can be extended to give a protocol for secure multi-party (and thus also two-party) computation for any number of corrupted parties, which remains secure under bounded-concurrent composition with arbitrary subsets of parties. As far as we know, for the case of two-party or multi-party protocols with a dishonest majority, this is the first positive result for *any* non-trivial functionality (zero-knowledge proofs being an important special-case) which achieves this property. We mention that our solution relies on the assumption that the participating parties have unique identities.

1.4 Subsequent Work and Open Problems

Unbounded concurrency. The elegant lower bound of Lindell [29] shows that unbounded concurrent composition can not be achieved in the model that we are considering (even for a single set of parties). Nevertheless, it might still be possible to achieve unbounded concurrency in a weaker model, where for example the inputs of the honest parties are fixed ahead of the protocol execution (i.e., inputs are chosen in a non-adaptive way).

Non-malleable cryptographic primitives. In subsequent work [33], our zero-knowledge protocols are used in order to construct simple *constant-round* protocols for non-malleable cryptographic primitives, such as commitments and zero-knowledge, under standard cryptographic assumptions. Previous constant-round constructions relied on sub-exponential hardness assumptions [2].

Cryptographic assumptions. We have shown that the notion of (bounded-concurrent) secure multi-party computation with a dishonest majority can be achieved in a constant number of rounds based on standard polynomial-time hardness assumptions, such as the existence of enhanced trapdoor permutations and collision resistant hash-functions. In contrast, stand-alone secure multi-party computation with polynomial round-complexity can be based on the existence of only enhanced trapdoor permutations [21]. This gives rise to two interesting open problems:

- Is it possible to construct a constant-round protocol for stand-alone secure multi-party computation with a dishonest majority, based on only enhanced trapdoor permutations?
- Is it possible to construct a protocol (which might have a non-constant round-complexity) for bounded-concurrent secure two-party/multi-party computation with a dishonest majority based on only enhanced trapdoor permutations?

1.5 Organization of the Paper

Formal definitions of bounded-concurrent secure multi-party computation can be found in Section 2. Section 3 contains a high level description of our main proof, as well as a description of our new zero-knowledge protocols.

2. DEFINITIONS

In this section we present the definition for m-bounded concurrent secure multi-party computation. The definitions below are generalizations of the definition of secure multi-party computation with abort and no fairness of Goldwasser and Lindell [24] and the definition of concurrent secure two-party computation with adaptive inputs of Lindell [29]. Parts of the definition have been taken almost verbatim from [24] and [29].

The basic description and definition of secure computation follows [23, 30, 5, 8]. We denote computational indistinguishability by $\stackrel{c}{\equiv}$, and the security parameter by k. For notational simplicity, we let the lengths of the parties' inputs be k.

Multi-party computation. A multi-party protocol problem for n parties $P_1, ..., P_n$ is cast by specifying a random process that maps vectors of inputs to vectors of outputs (one input and one output for each party). We refer to such a process as a n-ary functionality and denote it $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where $f = (f_1, ..., f_n)$. That is, for every vector of inputs $\bar{x} = (x_1, ..., x_n)$, the output-vector is a random variable $(f_1(\bar{x}), ..., f_n(\bar{x}))$ ranging over vectors of strings. The output of the i'th party (with input x_i) is defined to be $f_i(\bar{x})$. In the context of concurrent composition, each party actually uses many inputs (one for each execution) and these may be chosen adaptively based on previous outputs. The fact that m-bounded concurrency is considered relates to the allowed scheduling of messages by the adversary in the protocol executions; see the description of the real model below.

Adversarial behavior. In this work we consider a malicious, static adversary. That is, at the beginning of the execution the adversary is given a set I of corrupted parties which it controls and through which it interacts with the honest parties while arbitrarily deviating from the specified protocol. The focus of this work is not on fairness. We therefore present a definition where the adversary always receives its own output and can then decide when (if at all) the honest parties will receive their output. The scheduling of message delivery is decided by the adversary.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario. The ideal scenario is formalized by considering an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output.

Unlike in the case of stand-alone computation, here the trusted party computes the functionality many times, each time upon different inputs. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation.

Concurrent execution in the ideal model. Let $I \subset [n]$ denote the subset of corrupted parties. An ideal execution with an adversary who controls the parties I proceeds as follows:

Inputs: The inputs of the parties $P_1, ..., P_n$ are respectively determined by probabilistic polynomial-time Turing machines $M_1, ..., M_n$ and the initial inputs $x_1, ..., x_n$ to these machines. As will be described below, these Turing machine determine the input values to be used by the different parties in the protocol executions. These input values are computed from the initial input, the current session number and outputs that were obtained from executions that have already concluded. Note that the number of previous outputs ranges from zero (when no previous outputs have been obtained) to some polynomial in k that depends on the number of sessions initiated by the adversary.

Session initiation: The adversary initiates a new session by sending a (start-session, i) to the trusted party. If $i \in [n] - I$ (that is P_i is an honest party) the trusted party sends (start-session, j) to P_i, where j is the index of the session (i.e., this is the j'th session to be started by P_i).

Honest parties send inputs to trusted party: Upon receiving (start-session, j) from the trusted party, each honest party P_i applies its input-selecting machine M_i to its initial input x_i, the session number j and its previous outputs, and obtains a new input $x_{i,j}$. In the first session $x_{i,1} = M_i(x, 1)$. In later sessions j, $x_{i,j} = M_i(x, j, \alpha_{i,1}...\alpha_{i,\omega})$ where ω sessions have concluded and the outputs of P_i were $\alpha_{i,1}, ..., \alpha_{i,\omega}$. Each honest party P_i then sends $(j, x_{i,j})$ to the trusted party.

Corrupted parties send inputs to trusted party: Whenever the adversary wishes it may ask a corrupted party P_i to send a message $(j, x'_{i,j})$ to the trusted third party, for any $x'_{i,j} \in \{0,1\}^n$ of its choice. A corrupted party P_i can send the pairs $(j, x'_{i,j})$ in any order it wishes and can also send them *adaptively* (i.e., choosing inputs based on previous outputs). The only limitation is that for any j, at most one pair indexed by j can be sent to the trusted party.

Trusted party answers corrupted parties: When the trusted third party has received messages $(j, x'_{i,j})$ from all parties (both honest and corrupted) it sets $\bar{x}'_j = (x'_{1,j}, ..., x'_{n,j})$. It then computes $f(\bar{x}'_j)$ and sends $(j, f_i(\bar{x}'_j))$ to P_i for every $i \in I$.

Trusted party answers honest parties: When the adversary sends a message of the type (send-output, j, i) to the trusted party, the trusted party directly sends $(j, f_i(x'_j))$ to party P_i. If all inputs for session i have not yet been received by the trusted party the message is ignored. If the output has already been delivered to the honest party, or i is the index of a corrupted party, the messages is ignored as well.

Outputs: Each honest party P_i always outputs the vector $(f_i(\bar{x}'_1), f_i(\bar{x}'_2), ...)$ of outputs that it received from the trusted party. The corrupted parties may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the messages obtained from the trusted party.

Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a n-ary functionality, where $f = (f_1, ..., f_n)$. Let S be a non-uniform probabilistic polynomial-time machine (representing the ideal-model adversary) and let $I \subset [n]$ (the set of corrupted parties) be such that for every $i \in I$, the adversary S controls P_i. Then the ideal execution of f with security parameter k, input-selecting machines $M = M_1, ..., M_n$, initial inputs $\bar{x} = (x_1, ..., x_n)$ and auxiliary input z to S, denoted $\text{IDEAL}_{f,I,S,M}(k, \bar{x}, z)$, is defined as the output vector of the parties and the adversary S resulting from the ideal process described above.

Note that the definition of the ideal model does not include any reference to the bound m on the concurrency. This is because this bound is relevant only to the scheduling allowed to the adversary in the real model; see below. However, the fact that a concurrent setting is considered can be seen from the above-described interaction of the adversary with the trusted party. Specifically, the adversary is allowed to obtain outputs in any order that it wishes, and can choose its inputs adaptively based on previous outputs. This is inevitable in a concurrent setting where the adversary can schedule the order in which all protocol executions take place. In particular, the adversary can schedule the executions sequentially, thereby learning previous outputs before defining the next input.

Execution in the real model. We next consider the real model in which a real multi-party protocol is executed (and there exists no trusted third party).

We consider a scenario where the parties communicate through an *asynchronous* fully connected point-to-point network, without guaranteed delivery of messages. The communication lines between the parties are assumed to be authenticated and thus the adversary cannot modify messages sent by honest parties.

Let f, I be as above and let Π be a multi-party protocol for computing f. Furthermore, let A be a non-uniform probabilistic polynomial-time machine such that for every $i \in I$, the adversary A controls P_i. Then, the real m-bounded concurrent execution of Π with security parameter k, input-selecting machines $M = M_1, ..., M_n$, initial inputs $\bar{x} = (x_1, ..., x_n)$ and auxiliary input z to A, denoted $\text{REAL}^m_{\Pi,I,A,M}(k, \bar{x}, z)$, is defined as the output vector of the honest parties and the adversary A resulting from the following process. The parties run concurrent executions of the protocol, where every party initiates a new session whenever it receives a start-session from the adversary. The honest parties then apply their input-selecting machines to their initial input, the session number and their previously received outputs, and obtain the input for this new session. 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 (j, x) to an honest party. The honest party then adds x to the view of its j'th execution of Π and replies according to the instructions of Π and this view. Note that the honest parties runs each execution of Π obliviously to the other executions. (Thus, this is stateless composition.) The fact that the network is asynchronous means that the adversary has complete control over the timing and delivery of messages. In particular, the adversary may decide never to deliver certain messages. This is modeled as follows: The parties send messages by placing them in a "out-box" and the adversary delivers (or does not deliver) them at will.

The fact that we are considering m-bounded concurrency means that the scheduling by the adversary must fulfill the following condition: for every execution i, from the time that the i'th execution begins until the time that it ends, messages from at most m different executions can be sent.

Note that this definition of concurrency covers the case that m executions are run simultaneously. However, it also includes a more general case where many more than m executions take place, and they possibly all overlap.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure multi-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that for every real-model adversary there exists an ideal model adversary that can simulate an execution of the secure real-model protocol.

DEFINITION 1. (security in the malicious model): *Let $m = m(k)$ be a polynomial and let f and Π be as above. Protocol Π is said to t-*securely compute *f* under *m-*bounded concurrent composition *if for every real-model non-uniform probabilistic polynomial-time adversary A, there exists an ideal-model non-uniform probabilistic expected polynomial-time adversary S, such that for all input-selecting machines $M = M_1, ..., M_n$, every $z \in \{0,1\}^*$, every $\bar{x} = (x_1, ..., x_n)$, where $x_1, ..., x_2 \in \{0,1\}^*$ and every $I \subset [n]$ with $|I| < t$,*

$$\left\{\text{IDEAL}_{f,I,S,M}(k, \bar{x}, z)\right\}_{k \in N} \stackrel{c}{\equiv} \left\{\text{REAL}^{m}_{\Pi,I,A,M}(k, \bar{x}, z)\right\}_{k \in N}$$

That is, concurrent executions of Π with A cannot be distinguished from concurrent invocations of f with S in the ideal model.

3. PROOF OUTLINE

In this section we give a high-level outline of our proof, as well as a description of our new zero-knowledge protocols.

The starting point of our proof is the constant-round secure multi-party computation protocol for semi-honest adversaries of Beaver, Micali and Rogaway [7, 36]. Our goal is to compile this protocol into a protocol that is secure against a malicious adversary, in a way that preserves security under an a-priori bounded number of concurrent executions. Goldreich, Micali and Wigderson [21] presented a general compilation technique for the stand-alone setting, showing the usefulness of zero-knowledge proofs in order achieve such a compilation. Unfortunately, it is not known if the technique of [21] is secure in the concurrent setting.

Inspired by the central role of zero-knowledge proofs in the compilation of [21], we show a compilation for the concurrent setting in the following two step:

1. Identify a suitable notion of "zero-knowledge" proofs, called special-purpose zero-knowledge, and show that the compilation task reduces down to the construction of such protocols.

2. Implement special-purpose zero-knowledge.

The high-level structure of our proof follows that of Pass and Rosen [32] which in turn builds on the work of Lindell [27].

3.1 Reducing the Problem to Special-Purpose Zero-Knowledge

Idealized Functionalities. Our starting point is the protocol of Canetti et al. [13]. They show that in a setting where all parties have access to a specific *idealized functionality*, protocols secure against a semi-honest adversary can be compiled into protocols secure against a malicious adversary.[3] The notion of ideal functionalities is a central tool in the framework of universal composability [9], and can be thought of as the introduction of a trusted third party that is designed to perform a specific task. In the case of Canetti et al. [13], the protocol requires usage of the *ideal one-to many zero-knowledge proof of knowledge* functionality.

The $\mathsf{IdealZK}^{1:M}$ functionality: Informally, the ideal one-to-many zero-knowledge proof of knowledge functionality (called $\mathsf{IdealZK}^{1:M}$) for a language L is specified as follows: The prover sends an instance-witness pair (x, w) to the ideal functionality which, in turn, sends $(x, 1)$ to all the other parties if w is a valid witness for $x \in L$ and $(x, 0)$ otherwise. (That is one prover proves a statement to many verifiers). The setting in which the parties have access to (multiple) ideal one-to-many zero-knowledge functionalities is called the $\mathsf{ZK}^{1:M}$-Hybrid model. Our motivation is to remove the usage of the ideal functionality and implement a protocol that performs the same task but in the real model. In order to achieve this, we show a general transformation that transforms *any* protocol $\tilde{\Pi}$ in the $\mathsf{ZK}^{1:M}$-Hybrid model into a protocol Π in the real model (without any ideal functionalities or set-up assumptions). This is what we call *realizing* the $\mathsf{ZK\text{-}Hybrid}^{1:M}$ model. Jumping ahead, we mention that this part also includes the construction of the special-purpose zero-knowledge protocols (as will be described in Section 3.2). We further show that the transformation satisfies the following properties:

Concurrency: The protocol's security is preserved under (bounded) concurrent composition.

Round-Efficiency: The protocol's round-complexity is preserved (up to a constant factor).

In particular the transformation can be applied to the protocol of [13] (when in turn applied to the protocol of [7]) in order to obtain a constant-round bounded concurrent secure multi-party computation protocol.

How to Realize any Protocol in the $\mathsf{ZK}^{1:M}$-Hybrid Model. The above transformation is performed in three steps:

1. Reduce the usage of the $\mathsf{IdealZK}^{1:M}$ functionality to the usage of the more standard two-party ideal zero-knowledge proof of knowledge functionality, $\mathsf{IdealZK}$. Whereas the $\mathsf{IdealZK}^{1:M}$ functionality models interaction between a single prover and many verifiers, the $\mathsf{IdealZK}$ functionality models interaction between a single prover and single verifier. The idea underlying the reduction is straight-forward and similar to the idea

[3] The result by Canetti et al. [13] is in fact stronger, as they construct a protocol that is universally composable [9]. Nevertheless, in this work we only require composition in the (less demanding) concurrent setting.

used by Goldwasser and Lindell [24] to implement a broadcast channel. Roughly, whenever a prover wishes to prove a statement x using the $\mathsf{IdealZK}^{1:M}$ functionality, he instead gives an individual proof of x to each other party using the IdealZK functionality. Whenever a party receives a proof of a statement x it sends an acknowledgment to all other parties. Finally, when a party has received both a proof (from the IdealZK functionality) and an acknowledgment from all other parties it accepts the proof of the instance x. More details in the full version.

2. Reduce the usage of the IdealZK functionality to the usage of a weaker ideal functionality, MemberZK. Whereas the IdealZK functionality models a proof of knowledge, i.e., the prover not only proves the validity of a statement but also that it possesses a witness for it, the MemberZK functionality models simply a proof. Such a reduction was shown in [32], making use of a general transformation of zero-knowledge proofs into zero-knowledge proofs of knowledge due to [4]. We remark that the same reduction goes through also in the multi-party setting.

3. Show how to transform any protocol using the MemberZK functionality (we call this setting the MemberZK-Hybrid model) into a protocol in the real model by "plugging in" the special-purpose zero-knowledge protocols. We note that such a transformation was shown in [27] and in [32] for the special case of two-party protocols. Unfortunately, these transformations do not seem to directly extend to the multi-party setting.

The proof of the last step essentially boils down to the construction of a family of n (one for each party) constant-round zero-knowledge protocols, $c\mathcal{Z}\mathcal{K}_1, ..., c\mathcal{Z}\mathcal{K}_n$. These protocols are used to instantiate calls to the MemberZK functionality, in the following way: a call to the MemberZK functionality initialized by a party P_i (acting as prover) is instantiated with an execution of the protocol $c\mathcal{Z}\mathcal{K}_i$. In order for the instantiation to preserve security, the family of protocols needs to satisfy the following requirements:

- The protocols are bounded-concurrent zero-knowledge.
- The protocols compose concurrently with respect to arbitrary protocols with an a-priori bounded communication complexity.
- The protocols compose concurrently (an a-priori bounded number of times) with respect to themselves, with the roles of the prover and the verifier reversed. That is, simulation of the protocols can be performed even while simultaneously verifying them.
- Simulation of the protocols does not make use of rewinding.
- The protocols are *simulation-sound* with respect to each other. That is, the soundness of each one of the protocols is preserved even when the all the other protocols are simulated at the same time with the role of the prover and verifier reversed.

3.2 The Special-Purpose ZK Protocols

As in [32], our special purpose zero-knowledge protocols are based on the bounded-concurrent zero-knowledge protocol of Barak [1]. Before describing our protocols we therefore give a brief description of this protocol with an emphasis on the simulation technique, which will be useful in our setting. (Parts of the following paragraph are taken almost verbatim from [32]).

Barak's protocol relies on the existence of collision resistant hash functions. Other tools used in the protocol are perfectly binding bit-commitments and a witness-indistinguishable universal argument (WI UARG) [3]. More details on Barak's protocol can be found in [1]. The underlying idea behind Barak's protocol is the usage of an $NTIME(T(k))$ relation denoted R_{sim}. This relation is described in Figure 1.

Barak's $NTIME(T(k))$ Relation R_{Sim}

Input: A triplet $\langle h, c, r\rangle$.

Witness: A program Π, a string $y \in \{0,1\}^{(|r|-k)}$, and string s.

Relation: $R_{\mathsf{Sim}}(\langle h, c, r\rangle, \langle \Pi, s, y\rangle) = 1$ if and only if:

1. $c = \text{Com}\ (h(\Pi); s)$.
2. $\Pi(c, y) = r$ within $T(k)$ steps.

Figure 1: Barak's $NTIME(T(k))$ relation R_{sim}.

Let $T : \mathcal{N} \to \mathcal{N}$ be a "nice" function that satisfies $T(k) = k^{\omega(1)}$. Let $T' : \mathcal{N} \to \mathcal{N}$ be a function such that $T'(k) = T(k)^{\omega(1)}$. Suppose there exist a $T'(k)$-collision resistant hash functions ensemble $\{\mathcal{H}_k\}_{h\in\{0,1\}^k}$ where h maps $\{0,1\}^*$ to $\{0,1\}^k$. Barak's protocol is described in Figure 2.

Barak's Non Black-Box ZK Protocol for $\mathcal{NP}$

Common Input: an instance x of a language L, security parameter 1^k.

Length parameter: $\ell(k)$.

Stage 1:

$V \to P$: Send $h \stackrel{\text{R}}{\leftarrow} \mathcal{H}_k$.

$P \to V$: Send $c = \text{Com}(0^k)$.

$V \to P$: Send $r \in \{0,1\}^{\ell(k)}$.

Stage 2: (Proof Body)

$P \leftrightarrow V$: A *WI UARG* proving the OR of the following two statements:

1. $\exists w \in \{0,1\}^{\text{poly}(|x|)}$ s.t $R_L(x, w) = 1$.
2. $\exists \langle \Pi, s\rangle$ s.t $R_{\mathsf{Sim}}(\langle h, c, r\rangle, \langle \Pi, s\rangle) = 1$.

Figure 2: Barak's bounded-concurrent zero-knowledge argument for $\mathcal{NP}$.

Before we continue discussing the security of the protocol, let us remark that the assumption of hash functions collision resistant for slightly super-polynomial time can be relaxed to the standard assumption of hash functions collision resistant for polynomial time [3]. Looking ahead, we note that the same relaxation can be done also to our protocols.

As shown in [1], Barak's protocol is computationally sound. Moreover, given access to the verifier's code (or, equivalently, to the verifier's next message function), the protocol can be simulated without making use of rewinding. To simulate a specific session, the simulator commits to the verifier's next-message function (instead of committing to zeros). The verifier's next message function is then a program whose output depends on all the prover messages sent between messages c and r in the relevant session. Now, if the length of all these messages is bounded by $\ell(k) - k$ the simulator will have a valid witness for stage 2 of the protocol. The zero-knowledge property then follows (with some work) from the witness indistinguishable property of stage 2.

Barak [1] has shown that the length of the prover's messages in a single execution can be bounded by $2k^2$. Thus, for m concurrent executions, taking $\ell(k) = 2m \cdot k^2 + k$ would do. We note that the length of the verifier's messages, *not counting r's*, can also be bounded by $2k^2$.

An important extra feature of the protocol is that it is sufficient that the simulator has a "short" description of the prover's messages. The protocol is thus simulatable even if the total length of the prover's messages is greater than $\ell(k) - k$, as long as the simulator has a description of a program for generating the prover's messages that is shorter than $\ell(k) - k$.

Our protocols: Suppose there exist a collision resistant hash functions ensemble $\{\mathcal{H}_k\}_{h \in \{0,1\}^k}$ as required by Barak's protocol. We are ready to describe our special purpose protocols, $c\mathcal{ZK}_1, ..., c\mathcal{ZK}_n$ (depicted in Figure 3).

Protocol i - $c\mathcal{ZK}_i$

Common Input: an instance x of a language L, security parameter 1^k.

Length parameter: $\ell(k)$.

Stage 0 (Set-up):

$V \rightarrow P$: Send $h \stackrel{R}{\leftarrow} \mathcal{H}_k$.

Stage 1 (Slot 1):

$P \rightarrow V$: Send $c_1 = \mathrm{Com}(0^k)$.

$V \rightarrow P$: Send $r_1 \stackrel{R}{\leftarrow} \{0,1\}^{i\ell(k)}$

Stage 2 (Slot 2):

$P \rightarrow V$: Send $c_2 = \mathrm{Com}(0^k)$.

$V \rightarrow P$: Send $r_2 \stackrel{R}{\leftarrow} \{0,1\}^{(n+1-i)\ell(k)}$

Stage 3: (Proof Body)

$P \leftrightarrow V$: A *WI UARG* proving the OR of the following two statements:

1. $\exists w \in \{0,1\}^{\mathrm{poly}(|x|)}$ s.t $R_L(x, w) = 1$.
2. $\exists \langle \Pi, s, j \rangle$ s.t $R_{\mathsf{Sim}}(\langle h, c_j, r_j \rangle, \langle \Pi, s \rangle) = 1$.

Figure 3: The *i*'th special-purpose zero-knowledge protocol – $c\mathcal{ZK}_i$.

The difference of the protocols $c\mathcal{ZK}_i$ from the protocol of Barak is that the prover (simulator) is given two opportunities to guess the verifier's next message (called the challenge). We call each such opportunity a slot. Messages that are exchanged between the message c_j and the message r_j are said to be contained in slot j. Note that it is sufficient to have a "short" description of the messages contained in one of the two slots to succeed in the simulation of $c\mathcal{ZK}_i$.

Note that the sequence of protocols satisfies the following two properties:

- the length of the first challenge, r_1, is strictly *increasing*,
- the length of the second challenge, r_2, is strictly *decreasing*.

Thus, for any two protocols $c\mathcal{ZK}_i, c\mathcal{ZK}_j$ it holds that either the length of the first or the second challenge in $c\mathcal{ZK}_i$ is strictly larger than the corresponding challenge in $c\mathcal{ZK}_j$. This property will be essential to us when proving that the protocols are simulation sound with respect to each other.

Simulating the protocols: Let $\ell(k) = m(\gamma \cdot 4k^2 + \mathtt{length}(\Pi')) + k$, where Π' denotes the protocol in the MemberZK-Hybrid model (that we wish to realize) and γ is the total number of ideal zero-knowledge calls in one execution of Π'. We note that the total length of messages sent by a party, not including the challenges r_1, r_2, is bounded by $m(\gamma \cdot 2k^2 + \mathtt{length}(\Pi'))$, as both the length of the prover messages and verifier messages in the protocols $c\mathcal{ZK}_1, ..., c\mathcal{ZK}_n$ are bounded by $2k^2$. It follows that if the simulator can give a description of the various challenges r_1, r_2 (each being of length at least $\ell(k)$) that is shorter than k^2 it will always succeed in the simulation of $c\mathcal{ZK}_1, ..., c\mathcal{ZK}_n$, even if simultaneously verifying these protocols. As in [32], this is done by letting the simulator use a pseudorandom generator[4] in order to generate verifier messages (and in particular the verifier's challenges r_1, r_2) when playing the role of the verifier in other protocols.

Note that the simulator has two opportunities to succeed in the simulation, that is by either using slot 1 or 2. As a convention, we let the simulator always use the first slot to commit the verifier's code and acts as the honest prover (that is committing to zeros) in the second slot.

Soundness of the stand-alone protocols: It follows using the same argument as in [1] that the protocols are computationally sound.

Simulation soundness. The last step is to guarantee that the soundness of each one of the protocols $c\mathcal{ZK}_1, ..., c\mathcal{ZK}_n$ is not violated when simulating any (or all) of the other protocol with the roles of the prover and verifier reversed. To do this we show how to transform a cheating prover in this simulation/cheating scenario into a cheating prover for the stand alone case.

Suppose there exists a cheating prover P^* that manages to violate the soundness of one instance of protocol $c\mathcal{ZK}_i$ while it is verifying the simulation of multiple concurrent instances of the other protocols $c\mathcal{ZK}_j$, where $j \neq i$.

Using a standard argument, we show how to construct a cheating prover P^{**} for a single instance of $c\mathcal{ZK}_i$ by forwarding the messages of this "cheating" instance of $c\mathcal{ZK}_i$ to an external honest verifier V (the instance in which P^* is

[4]Note that the existence of pseudorandom generators follows from our assumptions [25].

actually cheating can be "guessed" by picking one of the instances of $c\mathcal{ZK}_i$ at random). V's replies are then forwarded by P^{**} back to P^* as if they were generated by the simulator. Since P^* is assumed to be cheating in this specific instance of $c\mathcal{ZK}_i$, and since the verifier messages used by P^{**} are indistinguishable from the messages used by V, the (stand-alone) soundness of $c\mathcal{ZK}_i$ is presumably violated.

One problem that arises is that the code of the external verifier V is not available to us. This means that the straightforward simulation of the protocols $c\mathcal{ZK}_j$, where $j \neq i$, cannot be completed as it is, since it explicitly requires possession of a "short" description of the corresponding verifier messages. To overcome this problem we resort to an alternative simulation technique.

The alternative simulator. For all protocol $c\mathcal{ZK}_j$ where $j \neq i$ we resort to the following alternative simulation technique.

We start by noting that except for the "long" challenges r_1, r_2 sent by the verifier of $c\mathcal{ZK}_i$ we do have a description of all messages sent to the adversary that is shorter than $\ell(k) - k$. In order to show that we can still perform a simulation, even in the presence of these messages (to which we do not have a short description), we use the fact that it is sufficient to have a short description of the messages sent in *one* of the slots of $c\mathcal{ZK}_i$.

We separate between two different schedulings:

Both the first and the second challenge r_1, r_2 in $c\mathcal{ZK}_i$ are contained in the same slot in $c\mathcal{ZK}_j$. It follows that there is always a "free" slot in $c\mathcal{ZK}_j$ which can be used to perform the simulation. The WI property of the $UARG$ guarantees that the the presumed cheating prover will still succeed in convincing the honest verifier of protocol $c\mathcal{ZK}_i$ with roughly the same probability.

The messages r_1, r_2 in $c\mathcal{ZK}_i$ occur in slot 1, 2 respectively in $c\mathcal{ZK}_j$. By the construction of the protocols it follows that the length of either the first or the second challenge in $c\mathcal{ZK}_j$ is at least $\ell(k)$ bits longer than the corresponding challenge in $c\mathcal{ZK}_i$. Thus there exist a slot in $c\mathcal{ZK}_j$ such that even if we include the verifier's challenge, from the protocol $c\mathcal{ZK}_i$, in whole, in the description, we still have $\ell(k) - k$ to describe the other messages, which means that simulation can be performed. As in the previous case, the WI property of the $UARG$ guarantees that the presumed cheating prover will still succeed in convincing the honest verifier of protocol $c\mathcal{ZK}_i$ with roughly the same probability.

3.3 Arbitrary Sets of Parties

Until this point we have focused on concurrent composition of secure multi-party computation for a *single set* of parties, i.e., n parties that concurrently are running many executions of the same protocol, each time playing the same roles in the protocol. A more general setting considers *arbitrary sets of parties* [28]. In this setting, arbitrary, and possibly intersecting, sets of parties concurrently execute the same n-party protocol. In particular, there is no restriction on what role a specific party plays in the protocol. This means that a party may play the role of P_1 in one execution and simultaneously play the role of P_2 in a different execution.

It is quite easy to see that unless the parties have identities it is impossible to construct a protocol for secure two-party (and thus also multi-party) computation in this general setting. This follows since without identities a man-in-the-middle attack can not be prevented. Let us therefore focus on a setting where all parties have unique identities.

First, assume that the participating parties have identities of length $O(\log k)$ (note that this is enough to model $poly(k)$ parties). In this case, essentially the same construction as previously described can be used. The only difference is that when instantiating calls to the $\mathsf{MemberZK}$ functionality with a special-purpose zero-knowledge protocol, this is done by using the protocol $c\mathcal{ZK}_{\mathsf{id}}$, where id is the identity of the prover initiating the zero-knowledge proof. Note that, whereas in the case of a single set of parties we instantiated the zero-knowledge calls according to the *role* of the prover in the the main protocol, the above-described instantiation is independent of the proving party's role and, instead, only depends on its *identity*. Using this construction the same proof as in the case of a single set of parties goes through.[5]

A more standard setting considers parties with identities of length k (i.e., 2^k possible identities). In this setting, the above-mentioned construction does not directly work. In fact, in order to make the same proof work, we need to construct a family of 2^k (instead of only $O(k)$) protocols that are simulation-sound with respect to each other. Below we outline how the special-purpose zero-knowledge protocols given in Figure 3 can be modified in order to achieve this property. Unfortunately, this modification increases the round-complexity.

For simplicity, let the identity id be given in base k. Let id_i denote the i'th digit of id in base k. Note that the length of id, denoted $|\mathsf{id}|$, is $k/\log k$. We show how the special-purpose protocols can be modified, using $2|\mathsf{id}|$ slots, instead of 2 slots, in order to be simulation-sound for 2^k identities. The protocol for a specific identity id is specified as follows. The first half of the slots will decode the identity id, i.e., the length of the challenge r_i in slot i is $\mathsf{id}_i \cdot l(k)$. The second half will be "mirror-slots" (in the same way that slot 2 was a mirror-slot to slot 1 for the special-purpose protocols in Figure 3), i.e, the challenge $r_{|\mathsf{id}|+i}$ in slot $|\mathsf{id}| + i$ is of length $(n+1)l(k) - r_i$. It follows, using the same argument as was used for the protocols in Figure 3, that the above construction yields a family of 2^k zero-knowledge protocols that are simulation-sound with respect to each other.

4. ACKNOWLEDGMENTS

I am very grateful to Johan Håstad for many helpful conversations and great advice. In particular, Johan pointed out that the method described in the paper could be used to obtain constant-round protocols. I am also very grateful to Alon Rosen. This paper could not have been written without our endless conversation on the subject of protocol composition and zero-knowledge. Thanks also to Yehuda Lindell for helpful comments.

[5]It is shown in [13] that the compilation in the $\mathsf{ZK}^{1:\mathrm{M}}$-Hybrid model preserves security also in this setting. (The definition of the $\mathsf{ZK}^{1:\mathrm{M}}$ functionality needs to be slightly changed, as in [13], in order to specify what parties that receive the proof). The proofs of the reductions from the $\mathsf{ZK}^{1:\mathrm{M}}$-Hybrid model to the MemberZK-Hybrid model are also unchanged. Thus the only point that needs to be addressed is the instantiation of the $\mathsf{MemberZK}$ calls.

5. REFERENCES

[1] B. Barak. How to go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pages 106–115, 2001.

[2] B. Barak. Constant-Round Coin-Tossing or Realizing the Shared-Random String Model. In *43rd FOCS*, pages 345-355, 2002.

[3] B. Barak and O. Goldreich. Universal Arguments and their Applications. *17th CCC*, pages 194–203, 2002.

[4] B. Barak and Y. Lindell. Strict Polynomial-Time in Simulation and Extraction. In *34th STOC*, pages 484–493, 2002.

[5] D. Beaver. Foundations of Secure Interactive Computing. In *Crypto'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[6] D. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20'th STOC*, pages 1–10, 1988.

[7] D. Beaver, S. Micali and P. Rogaway The Round Complexity of Secure Protocols. In *22'th STOC*, pages 503–513, 1990.

[8] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[9] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *34th STOC*, pages 494–503, 2002.

[10] R. Canetti, O. Goldreich, S. Goldwasser and S. Micali. Resettable Zero-Knowledge. In *32nd STOC*, pages 235–244, 2000.

[11] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. In *EuroCrypt'03*, Springer LNCS 2656, pages 68–86, 2003.

[12] R. Canetti, J. Kilian, E. Petrank and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires (almost) Logarithmically Many Rounds. *SIAM Jour. on Computing*, Vol. 32(1), pages 1–47, 2002.

[13] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multy-Party Computation. In *34th STOC*, pages 494–503,2002.

[14] B. Chor, M. Rabin. Achieving Independence in Logarithmic Number of Rounds. In *6th PODC*, pages 260-268, 1987.

[15] I. Damgård. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In *EuroCrypt'00*, LNCS 1807, pages 418–430, 2000.

[16] D. Dolev, C. Dwork and M. Naor. Non-Malleable Cryptography. *SIAM Jour. on Computing*, Vol. 30(2), pages 391–437, 2000.

[17] C. Dwork, M. Naor and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.

[18] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.

[19] O. Goldreich. Draft of a Chapter on general Protocols. Available at: `http://www.wisdom.weizmann.ac.il/oded/PSBookFrag/prot.ps`

[20] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Jour. on Computing*, Vol. 25(1), pages 169–192, 1996.

[21] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, Vol. 38(1), pp. 691–729, 1991.

[22] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC*, pages 218–229, 1987.

[23] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Crypto'90,* Springer-Verlag (LNCS 537), pages 77–93, 1990.

[24] S. Golwasser and Y. Lindell Secure Computation without Agreement. In *16th DISC*, Springer-Verlag (LNCS 2508), pages 17-32, 2002.

[25] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Jour. on Computing*, Vol. 28 (4), pages 1364–1396, 1999.

[26] J. Katz, R. Ostrovsky and A. Smith. Round Efficiency of Multi-Party Computation with a Dishonest Majority, In *EuroCrypt'03*. Springer LNCS 2656 pages 578–595, 2003.

[27] Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In *35th STOC*, pages 683-692, 2003.

[28] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In *44th FOCS*, pages 394–403, 2003

[29] Y. Lindell. Lower Bounds for Concurrent Self Composition. In *1st TCC*, pages 203–222, 2004.

[30] S. Micali and P. Rogaway. Secure computation. Unpublished manuscript, 1992. Preliminary version in *Crypto'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[31] R. Pass. On Deniability in the Common Reference String and Random Oracle Model. In *Crypto'03*. Springer LNCS 2729, pages 316–337, 2003.

[32] R. Pass, A. Rosen. Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In *44th FOCS*, pages 404–413, 2003.

[33] R. Pass, A. Rosen. New and Improved Constructions of Non-malleable Cryptographic Primitives. Manuscript in preparation.

[34] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In *21st STOC*, pages 73–85, 1989.

[35] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt'99*, Springer LNCS 1592, pages 415–431, 1999.

[36] P. Rogaway. The Round Complexity of Secure Protocols. PhD thesis, MIT, 1991.

[37] A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. In *40th FOCS*, pages 543-553, 1999.

[38] A. Yao. How to Generate and Exchange Secrets. In *27th FOCS*, pages 162–167, 1986.