Input-Indistinguishable Computation

Silvio Micali
MIT CSAIL
silvio@csail.mit.edu

Rafael Pass[*]
Cornell University
rafael@cs.cornell.edu

Alon Rosen
Harvard DEAS
alon@eecs.harvard.edu

Abstract

We put forward a first definition of general *secure computation that,* without any trusted set-up,

- *handles an* arbitrary *number of* concurrent *executions; and*
- *is implementable based on* standard complexity assumptions.

In contrast to previous definitions of secure computation, ours is not simulation-based.

1. Introduction

General secure computation should enable any number of players, each having his own secret input, to evaluate any desired function f on their inputs in a way that is both private and correct. This desideratum was originally formalized by [19] in a very stringent way. Intuitively, the players should be *absolutely indifferent* between (1) securely computing f and (2) privately handing their secret inputs to a trusted party, who then computes f and privately returns the results. This formalization of secure computation is now referred to as the *simulation-based* approach [19, 4, 21, 31, 7].

A crucial property of this definition is its *ready implementability,* based on standard complexity assumptions. Indeed, as proven in [19] (based on the prior two-party result of [42]), the existence of trap-door permutations implies the existence of general secure computation in the *stand-alone* setting. That is, when a single function evaluation is envisaged.

1.1. Pure Concurrent Security

Often, however, the players may need to securely evaluate multiple functions (on matching sets of secret inputs), by running the proper protocols in an asynchronous environment (like the Internet). We refer to this as the *concurrent setting.* In this more complex environment, privacy and correctness should be preserved for arbitrarily many protocols, and for any possible scheduling of their executions. (Put it in another way, players executing a secure protocol should never worry about their executing other secure protocols.) We refer to this intuitive desideratum as *concurrent security.* Much effort has been devoted in the last two decades to develop a suitable notion of concurrent security. Ideally, such a notion should satisfy two crucial properties. Namely,

1. **Pure Complexity.** Informally: for any function f, the existence of a protocol securely evaluating f should be guaranteed by standard complexity-assumptions alone. That is,

   Security should not *arise from some form of trusted infrastructure, but solely from the players' inability of performing super-polynomial computation.*

2. **Independent Design.** Informally, it should be possible to design a protocol P in *total isolation,* so as to satisfy the chosen notion of security, and rest assured that privacy and correctness will continue hold when P is concurrently executed with other protocols *satisfying the same security notion.* That is,

   In order to guarantee concurrent security, protocol designers should not need to coordinate their efforts in any way beyond adopting the same notion of security.

We consider these two desiderata to be intrinsic to true concurrency, and call *Pure Concurrent Security* a notion of concurrent security that satisfies both of them.

1.2. Current State of Concurrent Security

Currently, the main approach to define concurrent general secure computation is via simulation.

The strongest such notion is universal composibility (UC), first achieved in the perfect setting by Dodis and Micali [12] —under the name "parallel reducibility"— and then in the computational setting by Canetti [8] and Pfitzman and Waidener [37]. The UC approach, however, can only be exemplified by relying either on the assumption that the majority of the players are honest [5, 11, 40], or on some trusted set-up assumption —such as the existence of a

[*] Work mostly done while at MIT.

public string trusted to have been sampled from a specified distribution [10]; or a public-key infrastructure in which every player is guaranteed to know the secret key corresponding to his public key [2].) Such restrictions for the implementability of the UC notion have been proved to be necessary [9, 29]. In particular, *all general two-party UC protocols require some trusted set-up assumptions.* In a sense, therefore, UC does not satisfy Pure Complexity in the two-party case. However, it does satisfy Independent Design.[1]

To circumvent impossibility results regarding UC protocols, two veins of research have been investigated in the literature: 1) weaker concurrency and 2) weaker security.

Weaker Concurrency: Building on initial results by [28, 34, 1], [32] constructs general multi-party protocols that remain secure as long as there exists an a-priori bound on the number of concurrent executions at any given point, a.k.a *bounded-concurrency* (the model of bounded-concurrency was first considered in [1]).[2] As such, the bounded-concurrency model allows us to regain Pure Complexity, while sacrificing Independent Design.

Extending [14], [24] obtain general multi-party protocols in the timing model (first introduced in [14]). In this model it is assumed that all honest parties have access to clocks which proceed at approximately the same rate; this assumption can be used to enforce scheduling constraints on the execution of concurrent protocols. As such, protocols in the timing-model sacrifice both Pure Complexity and Independent Design (but arguably sacrifice Pure Complexity to a lesser extent than UC protocols, and Independent Design to a lesser extent than bounded-concurrent protocols).

Weaker Security: [33] suggests to weaken the standard simulation-based definition of security by allowing for a *super polynomial-time simulation* (SPS). [3] and [39], based on initial results by [33], obtain general multi-party protocols that are concurrently SPS secure. These protocols, however, have only been exemplified under non-standard complexity assumptions: a conspicuously non standard one in [39]; and the existence of sub-exponentially hard to invert functions in [3]. Thus, the SPS approach does not quite satisfy Pure Complexity. More problematically, however, from a conceptual view point, the SPS approach violates Independent Design. By this we do *not* mean that running SPS secure protocols may endanger other protocols which are "insecure" —i.e., not satisfying the SPS approach. (This is a recognized potential problem for SPS protocols. But the case can be made that every one has an obligation of designing protocols according to the best possible standards, and SPS designers cannot be responsible for protocols not adhering to their standards.) We instead mean that the SPS approach requires that all SPS protocols, designed *anywhere* by *anyone*, adopt essentially the same security parameters in order to be secure when concurrently run.[3] Such an adoption would constitute an extraordinary cooperation among SPS designers: who is to say that —for example— 2,000-bit keys are sufficient for every one? This results in a difficult state of affairs: if one financial institution decided to boost its security by adopting 1M-bit keys for its main protocol P (a quite legitimate and autonomous decision), then in the SPS approach, such a decision may actually render insecure all 1K-bit-key SPS protocols that may happen to be run concurrently with P. In a sense, therefore, *All known SPS protocols achieve concurrent security at the expense of Independent Design.*

In sum, therefore, no *general* notion of Pure Concurrent Security exist today.

1.3. Our Contribution

We put forward a new notion of security for the concurrent setting that *relaxes* simulation-based security, but brings us closer to simultaneously achieving both Pure Complexity and Independent Design. We call our security notion *Input Indistinguishability.* This notion restricts what a subset m of the n players can learn about the inputs of the other $n-m$ players in a secure computation of a *deterministic* function f. Expressing this in full generality quickly becomes very complex. Below we sketch our notion when $m = 1$ and n is arbitrary, but in this extended abstract we further restrict ourselves to the case when $n = 2$ (which, as we shall see, is complex enough). Then, in the stand-alone setting, input indistinguishability means that a player cannot tell which inputs the other parties might have used. To even be slightly more precise we need a minimum of notation.

[1] In fact, UC protocols satisfy a seemingly stronger property: namely, they do not affect the security of any other protocols —even not UC ones— concurrently run with them; this is called *concurrent general composition*. However, as showed in [30], in the context of standard simulation-based definitions this property is actually equivalent to only guaranteeing that secure protocol do not affect the security of other secure protocol; this is called *concurrent self-compostion*.

[2] The earlier results of [28, 34, 1] only consider two-party protocols. More importantly, however, these earlier protocols only remain secure as long as the same two parties execute a single protocol with so called *fixed roles*, i.e., the same party always plays the same role in the protocol.

[3] An SPS protocol has several interrelated running times associated with it. Let us single out two of them: informally, T_1, representing the the amount of "free computational help" that an adversary can obtain by virtue of participating in the protocol, and T_2, the time necessary to violate the input privacy. Thus, T_1 must be very large, but much smaller than T_2. Now, if the SPS protocol is run concurrently with a second one whose main running times are T_1' and T_2', then it better be that $T_1 < T_2'$. (Else, by participating in an execution of the first protocol, an adversary might be able to utilize "knowledge that is only computable in T_1 steps" when attacking the privacy of the second protocol run concurrently with the first.) But this condition entails that the security parameters of the two protocols, which actually control their two fundamental running times, be very close. Indeed, because the complexity gap in sub-exponential simulation is much tighter than that between polynomial and exponential, these parameters —if the underlying complexity assumptions are the same— can be taken to be roughly the same at a first approximation.

Denote the set of players other than i by $-i$. If V a vector indexed by the players, denote by V_{-i} the sub-vector whose indices belong to $-i$. Consider a distributed protocol P as a vector of individual interactive Turing machines, $P = (P_1, \ldots, P_n)$, and a n-input n-output function f as a vector $(f_1, \ldots, f_n)$, where f_i is the function consisting of evaluating f and returning the i^{th} component of its output. Say that input x_i *splits* the input sub-vectors x_{-i} and x'_{-i} if $f(x_i, x_{-i}) \neq f(x_i, x'_{-i})$; else, say that it *agrees* with them.

Then, a cryptographic protocol P for computing f is (one-player) *input indistinguishable* if, informally speaking, the following is true. For any (bad) player i, for any possible input sub-vectors x_{-i} and x'_{-i} (of the good players), and for any polynomial-time cheating protocol P_i' that i may run, at the end of a random execution of (P_{-i}, P_i) in which the input vector of the good players is with probability 1/2 x_{-i} and with probability 1/2 x'_{-i}, player i cannot tell which of the two sub-vectors were actually used with probability noticeably greater than 1/2, unless he chooses an input x_i for himself that splits x_{-i} and x'_{-i}.

A Minimal Definition. In this work, we focus on a minimal version of input indistinguishability, both for clarity and space constraints. Input indistinguishability can be enriched – with some advantage – with additional desiderata such as *input-awareness* (which will make it closer to simulation-based notions, while still avoiding its pitfalls). These enrichments will be the subject of forthcoming work.

1.4. Main Results

We prove the following properties about Input Indistinguishability. First, that an input indistinguishable protocol $P = (P_1, P_2)$ exists (and can in fact be readily found) for any 2-input 2-output function f, based on standard complexity assumptions and without relying on trusted set-up. Second, for any secret input vectors $\vec{x}^1, \vec{x}^2, \ldots$ the players can concurrently execute P and *provably maintain* Input Indistinguishability. Roughly, this means that if a player i knew beforehand that the inputs of the other player were $x^1_{-i}, x^2_{-i}, \ldots$ with prob. 1/2 and $x^1_{-i}{}', x^2_{-i}{}', \ldots$ with prob. 1/2, he could not disambiguate which is the case better than at random, unless, for some execution j, he deliberately chooses an input x^j_i for himself that splits x^j_{-i} and $x^j_{-i}{}'$.

Theorem 1 *Let f be a deterministic two-party function. Suppose there exists a collection of enhanced trapdoor permutations, and a family of claw-free permutations.*[4] *Then, there exists a protocol (P_1, P_2) that computes f in a concurrent input-indistinguishable way.*

Thus, Input Indistinguishability relaxes simulation-based security, but enables us to obtain a protocol for secure computation that under standard complexity assumptions remains secure under an unbounded number of concurrent executions. This may be considered progress so long as our relaxation is meaningful. (Which of course crucially depends on the precise details of our definition!)

We point out that in analogy with the notion of Witness Indistinguishability[5] (more on Witness Indistinguishability below), Input Indistinguishability does not provide any privacy guarantees when considering functions f for which the input of a player is *fully determined* by the input and output of the other player (even if this input is "hard" to compute). On the other hand, when this is not the case (i.e., when the input of a player is not fully determined by the input and output of the other player), input indistinguishability indeed provides stronger privacy guarantees. As such, our results further the undertanding of what types of functions can be securely and concurrently computed under standard complexity assumptions.

Towards Pure Concurrent Security. As our protocol does not rely on any trusted set-up and only relies on standard complexity assumption it satisfies our goal of Pure Complexity. It furthermore acheives independence of design, though in a very basic way. A minimal level of coordination for choosing the length of security parameters needs to be present: in essence, they must be polynomially related. In practice, this is going to be automatically satisfied, and in any case, requires significantely less coordination than that arising when relying on complexity leveraging; such techniques, in fact, require a very precise choice of security parameters that are unlikely to occur if this choice is made by independent designers. (Furthermore, to guarantee concurrent security, we additionally require all protocol designers to adhere to our construction paradigm — that is, security of a protocol is only guaranteed if concurrently executed with other secure protocols.[6])

1.5. Comparison to the Notion of Witness Indistinguishability

The closest source of inspiration of our definition is the notion of a Witness-Indistinguishable ($\mathcal{WI}$) protocol. And in a sense (though comparing apples and oranges), Input Indistinguishability can be meaningful as a form of secure computation as Witness Indistinguishability can be meaningful as a form of Zero Knowledge. One should keep in mind, however, that the two notions are defined in extremely different settings:

[4] Both assumptions in the hypothesis of Theorem 1 follow from the assumption that factoring Blum integers is hard. Enhanced trapdoor permutations are required for obtaining (semi-honest) Oblivious Transfer. Claw-free permutations are required for obtaining *perfectly* hiding-commitment, as well as collision resistant hashing.

[5] Recall that Witness Indistinguishability is trivially satisfied by any interative proof for a language with unique witnesses.

[6] This is sometimes called concurrent self-composition [29].

(1) In general two-party computation both parties obtain an input, whereas in the case of $\mathcal{WI}$ the verifier has no input. As a result, the adversary has some "control" on what information he can obtain just by playing with his input.

(2) The output of a general protocol is not a single bit. This is in contrast to $\mathcal{WI}$ in which the verifier's output is merely an ACCEPT/ REJECT bit.

(3) $\mathcal{WI}$ is defined with respect to fixed roles (i.e., the adversary can corrupt either provers or verifiers, but not both), whereas input indistinguishability allows arbitrary corruptions.

Being defined in a more challenging setting, input indistinguishable protocols are more difficult to construct (and to define!). Indeed, unlike witness indistinguishability, input indistinguishability is not closed under concurrent composition. At a high level, the difficulty in composition stems from the potential *malleability* of the protocol at hand [13].

Theorem 2 *Suppose that there exist one-way functions. Then, there exists a protocol that is stand-alone input indistinguishable, but is not concurrent input indistinguishable.*

Thus, to achieve a meaningful result in the context of composition, one must show *directly* that a specific protocol is concurrently input indistinguishable. This is in analogy to the situation in zero-knowledge, which is not closed under concurrent composition [17] (or even sequential if auxiliary inputs are not taken into consideration); yet there exist protocols that remain zero-knowledge even under a concurrent attack [41, 25, 38].

2. Input Indistinguishable Computation

We shall follow the notation of [6] and [23] verbatim. We assume familiarity with cryptographic protocols, and the notion of interactive Turing machines.

2.1. Notation

We consider m concurrent executions of a protocol (P_1, P_2), assuming that no other types of protocols are executed at the same time (a.k.a. self composition). The executions are run concurrently and their messages can be arbitrarily interleaved. Each distinct execution of the protocol is called a *session*. We index the various sessions according to the order in which they are terminated; i.e., for any $i, i' \in [m]$, if $i < i'$ then session i has terminated before session i'.

Inputs and random tapes. For every m, we will let $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_1^m \times \mathcal{D}_2^m$ denote the corresponding vectors of inputs. That is, $\mathbf{x} = (x_1, \ldots, x_m)$ where $x_i \in \mathcal{D}_1$ is P_1's input in session i, and $\mathbf{y} = (y_1, \ldots, y_m)$ where $y_i \in \mathcal{D}_2$ is P_2's input in session i.[7] Random tapes are of the form $\rho_1 = (\rho_1^1, \ldots, \rho_1^m)$ and $\rho_2 = (\rho_2^1, \ldots, \rho_2^m)$, where ρ_1^i serves as random tape for P_1 in session i, and ρ_2^i serves as random tape for P_2 in the same session.

Random executions. For an integer n, we let $\text{EXEC}^{P_1,P_2}(\mathbf{x}, \mathbf{y}; 1^n)$ denote the random variable obtained by (a) randomly and independently selecting random tapes $\rho_1 = (\rho_1^1, \ldots, \rho_1^m)$ for P_1 and $\rho_2 = (\rho_2^1, \ldots, \rho_2^m)$ for P_2; (b) for all $i \in [m]$ executing the i^{th} session of (P_1, P_2) with 1^n as common input, x_i, and ρ_1^i as private inputs and random tapes for P_1, and y_i, ρ_2^i as private inputs and random tapes for P_2; and (c) returning the execution so generated.

Views. Let $\mathbf{e}$ be an execution that consists of m concurrent sessions of (P_1, P_2). For a positive integer $i \in [m]$, let M_1^i be the sequence of messages *received* by the first party in session i. The *first-party view* of session i in $\mathbf{e}$, denoted $\mathsf{view}_1^i(\mathbf{e})$, is defined to be (x_i, ρ_1^i, M_1^i). Symmetrically defined is the *second-party view of session* i, $\mathsf{view}_2^i(\mathbf{e})$,

Concurrent adversaries. An m-*concurrent* adversary runs $m = \text{poly}(n)$ many executions of the protocol, and has full control of the scheduling of messages sent and received in the various executions. For simplicity, we assume that the adversary can only corrupt either a subset (or all) of the P_1's or a subset (or all) of the P_2's (but not both). At the cost of more cumbersome notation, our definitions can be extended to handle arbitrary corruptions, assuming each possible participant in the protocols is assigned a unique identity. Our protocols can be shown to be secure even in the latter (more complex) scenario.

Inputs, executions, views and outputs. An m-concurrent adversary P_1^* may ignore the inputs, $\mathbf{x}$, of individual sessions, and replace them with inputs chosen adaptively as a function of the messages it receives. We assume that P_1^* has a single random tape, ρ_1^*, which is used throughout the m concurrent executions of (P_1^*, P_2). A random variable $\text{EXEC}^{P_1^*,P_2}(\mathbf{x}, \mathbf{y}; 1^n)$ is defined accordingly. For a positive integer $i \in [m]$, let $\mathbf{M}_1$ be the sequence of messages *received* by P_1^* in *all* m sessions of $\mathbf{e}$. The *full view of* P_1^* in $\mathbf{e}$, denoted $\mathsf{view}_1^*(\mathbf{e})$, is defined to be $(\mathbf{x}, \rho_1^*, \mathbf{M}_1)$. The output of P_1^*, is determined as a function of its full view, namely $P_1^*(\mathsf{view}_1^*(\mathbf{e}))$. All of the above applies symmetrically to an m-concurrent adversary P_2^*.

Aborts. The adversary may "abort" a specific execution of the protocol at any point during the interaction. This could be done by sending an ill-formed message (i.e., not according to the protocol's prescribed instructions), or by simply refusing to continue. In such a case, the adversary is said to have sent an ABORT message. We assume

[7] For simplicity, the inputs of the honest parties are assumed to be chosen in advance. In the full version we describe how the definition can be extended to handle inputs that are chosen adaptively during the interaction.

that once an ABORT message has been sent in a session, both parties continue exchanging ABORT messages (within the corresponding session) until the session terminates. All other concurrent sessions proceed independently of these aborts. For $(i, j) \in [m] \times [k]$, define a Boolean variable $\mathtt{ABORT}^{(i,j)}(\mathbf{e})$ to be true if and only if session i in $\mathbf{e}$ is aborted by round j.

Output delivery message. The protocols that we consider in our definition will be required to have a designated *output delivery* message (before which no information on the output of the protocol should be revealed). For simplicity assume that output delivery occurs at the k^{th} message. Define a Boolean variable $\mathtt{OUTPUT}_1^i(\mathbf{e})$ to be true if and only if the output delivery message has been sent to party P_1 in session i in $\mathbf{e}$. $\mathtt{OUTPUT}_2^i(\mathbf{e})$ is symmetrically defined.

Extended functions. To capture the unavoidable possibility of an adversary aborting the execution in the beginning/middle of an interaction, we extend the domains and ranges of f so that they include a special $\perp$ symbol. This enables any one of the parties to choose $\perp$ as its local input, thus forcing the output of the protocol to be $\perp$. More specifically, for any two-party function $f : \mathcal{D}_1 \times \mathcal{D}_2 \rightarrow \mathcal{R}_1 \times \mathcal{R}_2$ we consider its *extended version* f': $(\mathcal{D}_1 \cup \{\perp\}) \times (\mathcal{D}_2 \cup \{\perp\}) \rightarrow (\mathcal{R}_1 \cup \{\perp\}) \times (\mathcal{R}_2 \cup \{\perp\})$ that is defined by:

$$f'(x, y) \mapsto \begin{cases} f(x, y) & \text{if both } x \neq \perp \text{ and } y \neq \perp \\ \perp & \text{if either } x = \perp \text{ or } y = \perp \end{cases}$$

2.2. The Definition

The notion of *implicit input*, is central to our definition. It is introduced as a means to "pin down" the actual inputs on which the adversary implicitly performs the computation.

Definition 1 (Implicit input) *Let (P_1, P_2) be a k-round protocol, and let P_1^* be an m-concurrent adversary. Consider a function function, $\textsc{in}_1$, that maps the full view, $\mathsf{view}_1^*(\mathbf{e})$, in an execution $\mathbf{e}$ of (P_1^*, P_2), into a sequence $\mathbf{x}^* = (x_1^*, \ldots, x_m^*) \in (\mathcal{D}_1 \cup \{\perp\})^m$. The function is said to be a* first party implicit input function *for (P_1, P_2) if for any $i \in [m]$ for which $\mathtt{ABORT}^{(i,k-1)}(\mathbf{e})$ is true, the value x_i^* equals $\perp$. The notion of a* second-party implicit input, *$\textsc{in}_2$, is symmetrically defined.*

The definition of input-indistinguishable computation is stated in terms of m-concurrent adversaries. The value of $m = m(n)$ can be taken to be any polynomial in the security parameter n, and the protocol's computational and communication complexities do not depend of the value of m. We refer to the special case where $m = 1$ as *stand alone* input indistinguishability. Protocols that retain their security for any $m = \text{poly}(n)$ are said to be *concurrent* input indistinguishable.

Definition 2 (Input-indistinguishable computation) *Let $f : \mathcal{D}_1 \times \mathcal{D}_2 \rightarrow \mathcal{R}_1 \times \mathcal{R}_2$ be a deterministic function, and let (P_1, P_2) be a fixed-round two-party protocol. We say that (P_1, P_2)* securely computes *f* with respect to the first party and *implicit input function $\textsc{in}_2$, if for every polynomial $m = m(n)$, the following conditions hold:*

1. Completeness: *For every $(\mathbf{x}, \mathbf{y}) \in (\mathcal{D}_1)^m \times (\mathcal{D}_2)^m$, every $n \in N$, and every $i \in [m]$:*

$$\Pr\Big[\ P_1(\mathsf{view}_1^i(\mathbf{e})) = f_1(x_i, y_i)\ \Big] = 1$$

where $\mathbf{e} \stackrel{R}{\leftarrow} \textsc{exec}^{P_1,P_2}(\mathbf{x}, \mathbf{y}; 1^n)$

2. Implicit Computation: *For every efficient m-concurrent ITM P_2^*, there exists a negligible function $\nu : N \rightarrow N$, so that for every $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_1^m \times \mathcal{D}_2^m$, every $n \in N$, and every $i \in [m]$:*

$$\Pr\left[P_1(\mathsf{view}_1^i(\mathbf{e})) = \begin{cases} f_1(x_i, y_i^*) & \mathtt{OUTPUT}_1^i(\mathbf{e}) \\ \perp & \neg\mathtt{OUTPUT}_1^i(\mathbf{e}) \end{cases}\right] > 1 - \nu(n)$$

where $\mathbf{e} \stackrel{R}{\leftarrow} \textsc{exec}^{P_1,P_2^*}(\mathbf{x}, \mathbf{y}; 1^n)$, $\mathbf{y}^* \leftarrow \textsc{in}_2(\mathsf{view}_2^*(\mathbf{e}))$.

3. Input Indistinguishability and Independence: *For every efficient m-concurrent ITM P_2^*, every $\mathbf{x}^1, \mathbf{x}^2 \in \mathcal{D}_1^m$, and every $\mathbf{y} \in \mathcal{D}_2^m$, the following ensembles are computationally indistinguishable:*

   - $\left\{\textsc{expt}^{P_1,P_2^*}(\mathbf{x}^1, \mathbf{x}^2, \mathbf{y}; 1^n)\right\}_{n \in N}$
   - $\left\{\textsc{expt}^{P_1,P_2^*}(\mathbf{x}^2, \mathbf{x}^1, \mathbf{y}; 1^n)\right\}_{n \in N}$

   where the random variable $\textsc{expt}^{P_1,P_2^*}(\mathbf{x}^1, \mathbf{x}^2, \mathbf{y}; 1^n)$ is defined as follows:

```
$\textsc{expt}^{P_1,P_2^*}(\mathbf{x}^1, \mathbf{x}^2, \mathbf{y}; 1^n)$
(a) $\mathbf{e} \stackrel{R}{\leftarrow} \textsc{exec}^{P_1,P_2^*}(\mathbf{x}^1, \mathbf{y}; 1^n)$
(b) $\mathbf{y}^* \leftarrow \textsc{in}_2(\mathsf{view}_2^*(\mathbf{e}))$
(c) If $\exists i \in [m]$ for which $\mathtt{OUTPUT}_2^i(\mathbf{e})$ is true, and
        $f_2(x_i^1, y_i^*) \neq f_2(x_i^2, y_i^*)$,
    then output $\perp$.
(d) Otherwise, output $(\mathbf{y}^*, \mathsf{view}_2^*(\mathbf{e}))$.
```

Secure computation with respect to the second party is symmetrically defined. We finally say that (P_1, P_2) securely computes *f, if there exist implicit input functions $\textsc{in}_1, \textsc{in}_2$ such that (P_1, P_2) securely computes f with respect to* both *the first and the second party, and $\textsc{in}_1, \textsc{in}_2$.*

2.3. Handling arbitrary corruptions

For simplicity the above definition assumes that the adversary only corrupts either a subset (or all) of the players running P_1 or a subset (or all) of the players running P_2, *but not both*. The definition can be extended to handle arbitrary corruptions, assuming each possible participant in the protocols is assigned a unique identity. Our protocols can be shown to be secure even in the latter (more complex) scenario.

2.4. Comments

Non-uniformity. In order to avoid cumbersome notation, we refrain from explicitly addressing the issue of auxiliary inputs. We note that our treatment will carry through even in case that the adversary is given an auxiliary input $z \in \{0,1\}^*$. Our proof does not preserve uniformity, and hence the hardness assumptions that we rely on are non-uniform.

Implicit inputs. For those familiar with secure computation, implicit input should be thought of as a statistically binding commitment that, with overwhelming probability over the coin tosses of the honest party, implicitly determines the inputs used by the adversary. We note that a malicious P_2^* could cause the value of IN_1 not to be well defined (by sending an inappropriate initialization message to the statistically binding commitment in use). Indeed, Def. 2 does not require anything from the value of IN_1 when dealing with a corrupted P_2^*.

Input awareness. Note that Definition 2 does not require the implicit input functions to be efficiently computatable. As such, our definition of input indistinguishable computation does not imply *input awareness*, i.e., that all parties "know" the implicit inputs upon which the protocol performs the computation. In a forthcoming paper we augment the above definition to also incorporate input awareness and discuss implications such an augmentation.

Implicit computation. When P_1 does not have an output (i.e. when $f_1(x,y) = \lambda$ for all x, y), implicit computation doesn't impose anything on implicit input (since it holds vacuously). However, when P_1 does have an output, implicit computation is an important feature. In particular, it guarantees that the implicit input of P_2^* is "consistent" with the output of P_1.

Aborts. Definition 2 does not impose restrictions on the value of the implicit input y_i^* in case that session i is aborted before P_2^* receives its output. Note that in such a case, none of the parties receives any outputs. Since whenever one of the inputs to f equals $\perp$ then so does its output, we could thus think of an adversary that aborts session i before it receives its output, as if it has a-priori "chosen" to have $y_i^* = \perp$ (thus forcing the output of both parties to be $\perp$).

Fairness. We do not guarantee fairness. However, by conditioning on the value of $\mathsf{OUTPUT}^i(\mathbf{e})$ in the implicit computation condition, we can guarantee that the party who gets the output first in the protocol always gets the value of $f(x_i, y_i^*)$ (resp. $f(x_i^*, y_i)$).

Gradual output release. Stand alone input indistinguishability rules out gradual release of the output. That is, it only allows protocols for which the output is revealed "all at once" in a single message. Loosely speaking, this is because if $\mathsf{OUTPUT}_2^i(\mathbf{e})$ is false, indistinguishability of the view is always guaranteed. This holds regardless of the value of f and *even if the execution is truncated from the output delivery message and on*. Thus, if the implicit input of session i' is defined prior to the output delivery round of session i, then the actual input of session i' is independent of session i's output. Note that the latter is indeed a strong property, and is not guaranteed by many of the previously known definitions.

"Splitting" inputs. If the sequence $\mathbf{y}^*$ distinguishes between $\mathbf{x}^1$ and $\mathbf{x}^2$ "through" the function f (i.e., $f(x_i^1, y_i^*) \neq f(x_i^2, y_i^*)$ for some $i \in [m]$) then input indistinguishability will give no guarantee. However, if it doesn't then the corresponding views (i.e., the one using $\mathbf{x}^1$ and the one using $\mathbf{x}^2$) will be computationally indistinguishable.

Witness indistinguishability. Note that $\mathcal{WI}$ is a special case of Input Indistinguishability: for any $\mathcal{NP}$-relation R, consider the function $f_R((x,w),x) = 1$ iff $(x,w) \in R$).

3. Input Indistinguishability does not Compose

In this section we show that, *unlike witness indistinguishability*, our notion of input indistinguishability is not closed under concurrent composition. We demonstrate this by presenting a protocol that is stand-alone input indistinguishable (i.e., for $m = 1$), but fails to be input independent when as little as two executions take place concurrently. Moreover, the protocol fails to be concurrent input indistinguishable even with *fixed* roles, namely the adversary is either allowed to corrupt parties that play the role of P_1 or parties that play the role of P_2 (but not both).

Let Com be a statistically-binding commitment, and let $\mathcal{ZK}$ be any zero-knowledge proof of knowledge of a committed value. We consider the following two-party protocol (P_1, P_2) for computing the function $f(x,y) = (x \oplus y, x \oplus y)$ (see Fig 1 below). The parties (P_1, P_2) are assumed not to proceed in case a $\mathcal{ZK}$ protocol verified by them is rejected.

Claim 1 (P_1, P_2) *is stand-alone input indistinguishable.*

Claim 1 follows directly from the definition of input indistinguishable computation (proof omitted).

Claim 2 (P_1, P_2) *is* not *concurrent input independent.*

Local inputs: $x \in \mathcal{D}_1$ for P_1, and $y \in \mathcal{D}_2$ for P_2.

Common Input: Security parameter 1^n.

$P_1 \rightarrow P_2$: Pick $\rho \overset{\text{R}}{\leftarrow} \{0,1\}^n$. Send c = Com($x;\rho$).

$P_1 \Leftrightarrow P_2$: Using $\mathcal{ZK}$, prove $\exists x, \rho$ s.t. c = Com($x;\rho$).

$P_2 \rightarrow P_1$: Pick $\sigma \overset{\text{R}}{\leftarrow} \{0,1\}^n$. Send d = Com($y;\sigma$).

$P_2 \Leftrightarrow P_1$: Using $\mathcal{ZK}$, prove $\exists y, \sigma$ s.t. d = Com($y;\sigma$).

$P_1 \Leftrightarrow P_2$: Send x, and using $\mathcal{ZK}$, prove $\exists \rho$ s.t. c=Com($x;\rho$).

$P_2 \Leftrightarrow P_1$: Send y, and using $\mathcal{ZK}$, prove $\exists \sigma$ s.t. d=Com($y;\sigma$).

P_1, P_2: Output $x \oplus y$.

Figure 1. Protocol (P_1, P_2)

The claim is proved by considering a 2-concurrent adversary P_2^* that plays the role of P_2 in both executions, and acts as depicted in Figure 2 (the value of y is arbitrary). Note that the protocol (P_1, P_2) is symmetric, so it is indeed possible for P_2^* to mount the above attack. For lack of space we omit the full details.[8]

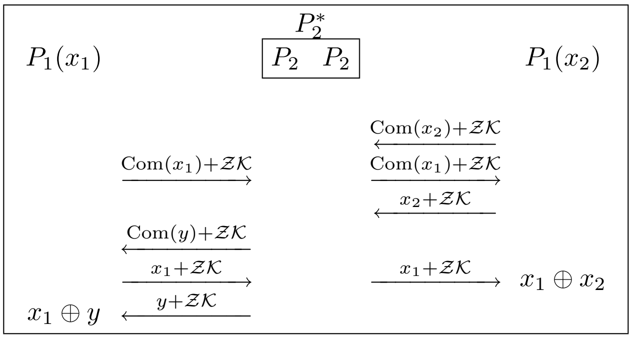

Figure 2. The 2-concurrent adversary P_2^*.

The above attack exemplifies the difficulties encountered with respect to independence of inputs when multiple executions of a protocol for secure computation take place concurrently. On a high level, the source of the problem is the potential *malleability* of the protocols at hand. More specifically, dependencies will be caused by either one of the following two reasons:

Man-in-the middle dependencies: The adversary might be able to create dependencies between its own input in one execution with the other party's inputs in other executions. As in the above example, this could happen if the protocol is symmetric and the adversary copies messages he received in one session and forwards them back in another session as if they were its own messages. Notice that the stand alone setting does not rule out the possibility that a symmetric protocol is input independent. In fact the "typical" protocols (e.g. [19]) are symmetric and indeed do not rule out the copying attack.

Input/Output dependencies: In the concurrent setting the input of the adversary (or even the honest party) in one instance of the protocol may depend on the output in a previous instance. Since we do not require the honest party's inputs to be independent of each other, dependencies through the input/output relation of the functionality that is being computed might occur (and are in fact unavoidable). Indeed, Definition 2 does not rule out the possibility of having such dependencies.

Thus, if we wish to construct protocols that preserve input indistinguishability under concurrent executions, we will have to make sure that they satisfy some form of *non-malleability* (notice that this is not necessary in the case of witness indistinguishability). This is indeed the approach we take. In particular, the zero-knowledge proofs that we use in the "compilation" of our protocols satisfy a strong form of non-malleability.

4. Highlights of our Construction

Our protocol needs to be secure also when the adversary corrupts different parties in different executions. Hence the protocol instructions not only depend on the "role" of the party in the protocol (i.e., if it is a first, or a second party) but also on an identity id $\in [2^n]$ assigned to it.

4.1. Non-malleability

One of the central tools used in our construction are *non-malleable* protocols [13]. Roughly speaking, non-malleable protocols are designed to withstand a "man-in-the-middle" adversary, who fully controls the communication channels between interacting parties. The adversary has the power to omit, insert or modify messages at its choice. It has also full control over the scheduling of the messages, and the honest parties are not necessarily aware to its existence.

Constructions of non-malleable protocols go back to the paper by Dolev, Dwork and Naor [13]. However, the security guarantee provided by these protocols are not strong enough for our purposes (i,e., when many protocols take place concurrently at both sides of the man-in-the-middle attacker). Moreover, these protocols (as well as later ones) were designed only having the fairly basic of case of commitment and zero-knowledge in mind, and do not take into consideration more involved scenarios where the presence of messages from other types of protocols is conceivable.

Until recently it was not clear how to extend the basic results on non-malleability for the case of general secure computation (even for the seemingly trivial case of two executions). This problem has been recently addressed, as-

[8] The actual proof is quite subtle and involves going over all requirements of 2-concurrent input-indistinguishable computation.

suming an a-priori known bound on the number of concurrent executions (a.k.a. bounded concurrency) [32][9]. Our construction and analysis rely on these techniques and their augmentations [35], which eventually found an application also in the context of unbounded concurrency for the specific case of non-malleable commitment [36]. Using and extending ideas from all of these works will enable us to carry out the proof even in our (highly demanding) setting of general concurrent secure computation.

4.2. The protocol

Our starting point is Yao's honest-but-curious protocol for computing a function f, which we denote by (Y_1, Y_2) [42]. Using techniques analogous to the ones of [19], we compile (Y_1, Y_2) into a protocol that is secure against malicious adversaries:

1. We add a set-up phase where both parties commit to their input, and to a truly random tape, using a statistically-binding commitment, denoted Com. The commitment to the random tape is obtained using a "coin-tossing" protocol.

2. Then, both parties execute (Y_1, Y_2) using the pre-committed input and random tape, and additionally prove, after each message of (Y_1, Y_2), using a zero-knowledge proof that they computed this message correctly (w.r.t. pre-committed input and random tape).

Our compilation differs from the one in [19] in three ways.

1. Our compilation uses a specific "coin-tossing." This coin-tossing will ensure that once the first messages, m_1, m_2, in the protocol have been sent, all subsequent (Y_1, Y_2) messages are predetermined. Furthermore, except with negligible probability, an honest party will detect if an adversary does not send these predetermined messages. This property has been called *provable determinism* in [27] and will be important in the analysis of the protocol.

2. Instead of using an arbitrary zero-knowledge proof of knowledge, we will rely on a particular *non-malleable* zero-knowledge protocol; namely, a perfect zero-knowledge variant of the protocol of [32], due to [35]. This protocol takes as additional input an identity $\mathsf{id} \in [2^n]$, and is denoted ZK_{id}. We let $\mathsf{id}_1, \mathsf{id}_2$ denote the identities used by the first and second parties respectively.

3. To enable the analysis, we will additionally have both parties commit to their inputs and randomness (as well as to m_1, m_2) using a *perfectly hiding* commitment scheme, denoted **Com**. At a high level, the role of this commitment is to "soften" the inherent asymmetry between the two parties, given that one of the parties commits to its input and randomness before the other.

The resulting protocol is denoted (P_1, P_2).

4.3. Analysis

The proof of Theorem 1 starts by showing that the protocol (P_1, P_2) described above is *stand-alone* input indistinguishable. It then proceeds by showing how to transform any m-concurrent adversary P_2^* for (P_1, P_2) into a stand-alone adversary for that protocol.

Proposition 1 *Suppose that* Com *is a statistically-binding commitment, that* **Com** *is a perfectly hiding commitment, and that* $ZK_{\mathsf{id}_1}, ZK_{\mathsf{id}_2}$ *are* $\mathcal{ZK}$ *interactive arguments. Then,* (P_1, P_2) *securely computes any deterministic function* f *in a* stand-alone *input-indistinguishable way.*

Proof Sketch: We define the implicit input function, IN_1, for party P_1 in the following way: If the statistically binding commitment to the input, sent by party P_1 is uniquely defined, and if the ZK_{id_1} proof following the commitment is accepting, the implicit input is defined as the value committed to. Otherwise, it is defined as $\perp$. The implicit input IN_2 is symmetrically defined.

The proof that (P_1, P_2) is stand-alone input indistinguishable is based on a fairly standard simulation argument (comparing a real execution with an ideal execution). The actual proof, however, requires some augmentations to the standard simulation-based proofs. Most notably, it crucially relies on the fact that the outputs of the parties are delivered in "one shot" at a given message of the protocol (P_1, P_2) (a property inherited from (Y_1, Y_2)). It also requires to augment the outcome of the ideal and real executions with the implicit input IN_i of the corrupted party P_i^*. ■

Proposition 2 *Suppose that* Com *is a statistically-binding commitment, that* **Com** *is a perfectly hiding commitment, and that* $ZK_{\mathsf{id}_1}, ZK_{\mathsf{id}_2}$ *are* $\mathcal{ZK}$ *interactive arguments. Then,* (P_1, P_2) *securely computes any deterministic function* f *in a concurrent input-indistinguishable way.*

Proof Sketch: Concurrent implicit computation and completeness are established using the same implicit input functions IN_1, IN_2 as defined in the proof sketch above. Here we rely on the fact that IN_1, IN_2 are fully determined by the transcript of the concurrent interaction (i.e., the random-coins of the adversary are not needed). Using this fact we

[9] We mention that in contrast to [32], earlier work on bounded-concurrent two-party computation by [28, 34] do not provide security against man-in the middle attacks (although non-malleability of protocols is an issue in those solutions as well).

can then reduce concurrent implicit computation to stand-alone implicit computation.

To establish concurrent input indistinguishability and independence, we assume for contradiction that there exist input sequences $\mathbf{x}^1, \mathbf{x}^2, \mathbf{y}$ and an m-concurrent adversary P_2^* that violate the concurrent input-indistinguishability of (P_1, P_2). We then show how to transform P_2^* into a stand-alone adversary $P_2^{\mathsf{STA}*}$ that violates the stand-alone input-indistinguishability of (P_1, P_2), in contradiction to Prop. 1.

The stand-alone adversary $P_2^{\mathsf{STA}*}$ generates the entire concurrent view of P_2^*, except for one (the i^{th}) execution, the messages of which are forwarded externally to P_1. The “internal” generation of P_2^*'s concurrent view is facilitated by the fact that all the inputs of the participating honest parties are known to us, and consist of $x_1^1, \ldots, x_{i-1}^1, x_{i+1}^2, \ldots, x_m^2$ (this can be guaranteed using a somewhat non-standard hybrid argument).

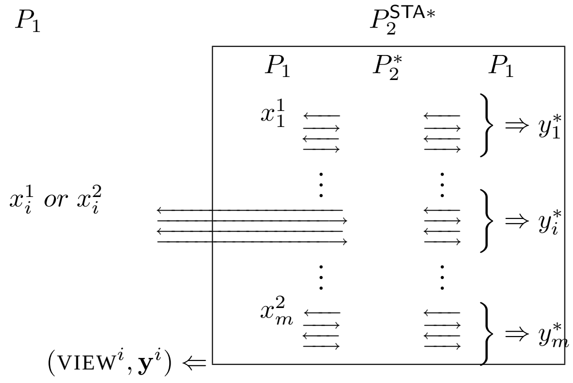

Figure 3. The stand alone adversary $P_2^{\mathsf{STA}*}$.

The task of constructing $P_2^{\mathsf{STA}*}$ then reduces to simulating the i^{th} left interaction, that corresponds to the external execution and for which we do not know the input (it is either x_i^1 or x_i^2). On top of simulating one session, it will be required to extract *multiple* implicit inputs $y_1^*, \ldots, y_m^*$ concurrently from P_2^*. These inputs will be then fed into a machine that presumably distinguishes between $\text{EXPT}(\mathbf{x}^1, \mathbf{x}^2, \mathbf{y}, 1^n)$ and $\text{EXPT}(\mathbf{x}^2, \mathbf{x}^1, \mathbf{y}, 1^n)$ (which should exist by our contradiction assumption).

The extraction of $y_1^*, \ldots, y_m^*$ is done by means of “rewinding” individual sessions one by one (cf. [28]). The property that enables the successful executions of all of these tasks is called *one-many simulation extractability*. As shown in [36], this property is satisfied by the $\mathcal{ZK}$ protocols from [32, 35] (and these are indeed used in the compilation).

In [36] it was shown how to use simulation extractability to obtain concurrent non-malleable commitments. Due to the more demanding setting of the current paper, the analysis does not go through in a modular way, and several new ideas are required. The main source of complication stems from the fact that the external protocol is *interactive* (its messages consist of the commitments to the inputs and randomness, as well as corresponding messages of (Y_1, Y_2)). This means that, unlike the case of commitments, we need to deal with multiple external messages. This causes trouble when rewinding the interaction in order to extract P_2^*'s inputs. Specifically, the rewinding might require us to rewind the external interaction as well (which is of course impossible). This is where the provable determinism property comes into play. We can assume that the messages forwarded to the external party remain unchanged, while the messages forwarded back to P_2^* internally are ”faked” by the simulator of our zero-knowledge protocol.

One thing that provable determinism cannot solve, however, is a case in which P_2^* rewinds its interaction beyond the start of the external protocol. In such a case, the external interaction has to start from scratch with a possibly different input. To get around this difficulty we simultaneously run two extraction procedures in parallel (with different inputs). We then argue that at least one of these extractions will succeed. ■

5. Acknowledgements

We wish to thank Johan Håstad for helpful discussions at an early stage of this research, and the anonymous reviewers for their careful reading and thoughtful comments.

References

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

[2] B. Barak, R. Canetti, J. B. Nielsen, R. Pass: Universally Composable Protocols with Relaxed Set-Up Assumptions. In *FOCS 2004*, pages 186-195.

[3] B. Barak and A. Sahai. How To Play Almost Any Mental Game Over The Net - Concurrent Composition via Super-Polynomial Simulation. FOCS 2005: 543-552

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

[5] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In 20th *STOC*, pp. 1-10, 1988.

[6] M. Blum, A De Santis, S. Micali, G. Persiano. Noninteractive Zero-Knowledge. *SICOMP* 20(6) pp. 1084-1118, 1991.

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

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

[9] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. In *Eurocrypt 2003*, Springer-Verlag (LNCS 2656), pages 68–86, 2003.

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

[11] D. Chaum, C. Crepeau, I. Damgård. Multiparty Unconditionally Secure Protocols. In *20th STOC*, pages. 11–19, 1988.

[12] Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. *CRYPTO'00*, pp. 74–92, 2000.

[13] D. Dolev, C. Dwork and M. Naor. Non-Malleable Cryptography. *SICOMP*, Vol. 30(2), pages 391–437, 2000.

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

[15] U. Feige. Ph.D. thesis, Alternative Models for Zero Knowledge Interactive Proofs. Weizmann Institute of Science, 1990.

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

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

[18] 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*, 38(1), pp. 691–729, 1991.

[19] 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.

[20] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Jour. of Cryptology*, Vol. 7, No. 1, pages 1–32, 1994.

[21] 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.

[22] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SICOMP*, Vol. 18(1), pp. 186–208, 1989.

[23] S. Goldwasser, S. Micali and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks. *SICOMP*, Vol. 17, No. 2, pp. 281–308, 1988.

[24] Y. Kalai, Y. Lindell, M. Prabhakaran. Concurrent Composition of Secure Protocols in the Timing Model. In *37th STOC*, pages 644–653, 2005.

[25] J. Kilian and E. Petrank. Concurrent and Resettable Zero-Knowledge in Poly-logarithmic Rounds. In *33rd STOC*, pages 560–569, 2001.

[26] J. Kilian, E. Petrank and C. Rackoff. Lower Bounds for Zero-Knowledge on the Internet. In *39th FOCS*, pp. 484–492, 1998.

[27] M. Lepinski, S. Micali, C. Peikert, A. Shelat. Completely fair SFE and coalition-safe cheap talk. PODC04, pp. 1-10, 2004.

[28] Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In *34th STOC*, 2003.

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

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

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

[32] R. Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. *36th STOC*, pp. 232-241, 2004.

[33] R. Pass. Simulation in Quasi-Polynomial Time and its Application to Protocol Composition. In *EuroCrypt 2003*. Springer LNCS 2656, pages 160–176, 2003

[34] R. Pass and A. Rosen. Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In *34th FOCS*, pages 404-413, 2003.

[35] R. Pass and A. Rosen. New and Improved Constructions of Non-Malleable Cryptographic Protocols. In *36th STOC*, pages 533-542, 2005.

[36] R. Pass and A. Rosen. Concurrent Non-Malleable Commitments. In *46th FOCS*, pages 563-572, 2005.

[37] B. Pfitzmann and M. Waidner: A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. IEEE Symp on Sec. and Priv., pages 184-193, 2001.

[38] M. Prabhakaran, A. Rosen and A. Sahai. Concurrent Zero-Knowledge with Logarithmic Round Complexity. In *43rd FOCS*, pages 366-375, 2002.

[39] M. Prabhakaran and A.Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC 2004*, pages 242-251.

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

[41] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt99*, Springer LNCS 1592, pages 415–431, 1999.

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