# Impossibility Results for Static Input Secure Computation

Sanjam Garg[1], Abishek Kumarasubramanian[1], Rafail Ostrovsky[1], and Ivan Visconti[2]

[1] UCLA, Los Angeles, CA
{sanjamg,abishekk,rafail}@cs.ucla.edu,
[2] University of Salerno, Italy
visconti@dia.unisa.it

**Abstract.** Consider a setting of two mutually distrustful parties Alice and Bob who want to securely evaluate some function on *pre-specified* inputs. The well studied notion of *two-party secure computation* allows them to do so in the *stand-alone* setting. Consider a deterministic function (e.g., 1-out-of-2 bit OT) that Alice and Bob can not evaluate trivially and which allows only Bob to receive the output. We show that Alice and Bob can not securely compute any such function in the *concurrent* setting even when their inputs are *pre-specified*. Our impossibility result also extends to all deterministic functions in which both Alice and Bob get the same output. Our results have implications in the *bounded*-concurrent setting as well.

## 1 Introduction

Consider a setting of two mutually distrustful parties Alice and Bob who want to securely evaluate a function $f$. The well studied notion of *two-party secure computation* [1, 2] allows them to do so. However this notion is only relevant to the *stand-alone* setting where security holds only if a single protocol session is executed in isolation. Additionally these secure computation protocols are *interactive* and Alice and Bob are expected to preserve *state* information during the protocol execution.

*What if Alice and Bob want to evaluate multiple functions concurrently?* This problem has drawn a lot of attention in the literature. A large number of secure protocols (in fact under an even stronger notion of security called UC security) based [3–12] on various trusted setup assumptions have been proposed. To address the problem of concurrent security for secure computation in the *plain model*, a few candidate definitions have been proposed, including input-indistinguishable security [13, 14] and super-polynomial simulation [15–17, 9, 18]. Both of these notion, although very useful in specialized settings, *do not* suffice in general. Additionally other models that limit the level of concurrency have also been considered [19, 20] or allow simulation using additional outputs from

the ideal functionality [21]. Among these models the model of $m$-bounded concurrency [22, 19] which allows for $m$ different protocol executions to overlap has received a lot of attention in the literature [22, 19, 23, 24]. Unbounded concurrent oblivious transfer in the restricted model where all the inputs in all the executions are assumed to be independent has been constructed [25].

At the same time, impossibility results ruling out the existence of secure protocols in the concurrent setting have been shown. UC secure protocols for most functionalities of interest have been ruled out in [3, 26]. Concurrent self-composability [23] for a large class of interesting functionalities (i.e., bit transmitting functionalities) has been ruled out however only in a setting in which the honest parties choose their inputs *adaptively* (i.e., "on the fly"). Unfortunately, in the very natural setting of *static* (pre-specified) inputs only the $\mathcal{F}_{ZK+OT}$ functionality (i.e., a mix of the zero-knowledge and the oblivious transfer functionalities) and the functionality that evaluates a pseudorandom function on a committed key [20] have been ruled out.

This leaves a large gap between the very few functionalities (e.g., $ZK + OT$) for which the impossibility [27, 20] has been proved, and the achieved results on concurrent ZK [28–31, 27]. In this paper, we ask the following *basic* question:

*What functions can Alice and Bob concurrently securely compute in the plain model in the natural setting of static (i.e., pre-specified) inputs?*

### 1.1 Our Results

In this paper we give the following two results.

**Impossibility results for concurrent self computation.** We show (assuming one-way functions) that no two-party protocol concurrent securely realizes any symmetric (where both parties get the same output) or asymmetric (only one party gets the output) deterministic functionality [32–34] that is *complete*[3] in the stand-alone setting. Our impossibility results hold even in the *very restricted* setting of *static* inputs (inputs of honest parties are pre-specified) and *fixed roles* (i.e, the adversary can corrupt only one party who plays the same role across all executions).

We additionally show that an $m$-bounded concurrently secure protocol for any symmetric deterministic functionality must have a communication complexity of at least $\frac{m}{k^c}$ bits where $c \geq 0$ (depending on the functionality) is a constant and $k$ is the security parameter. This bound corresponds to $m$ bits for the case of string OT.

Independent of our work, exciting results concerning the impossibility of concurrently secure computation have been obtained recently by Agrawal et. al. We refer the reader to [35], in these proceedings, for more details.

[3] A functionality is said to be complete if it can be used to securely realize any other functionality.

**Impossibility results for stateless two-party secure computation.** The question of *general* secure computation with stateless parties has been studied in [36, 37]. It might seem that secure computation with stateless parties should imply secure computation in the concurrent but stateful setting. However interestingly, this is not true as the definition of secure computation among stateless parties *only* requires the existence of a simulator that can additionally benefit from the stateless nature of honest parties (technically speaking, the ideal-world adversary can rewind the functionality). Goyal and Maji [37] showed a positive result for stateless secure computation for a large class of (in particular for all functionalities except the ones which behave as a weak form of pseudorandom generators) functionalities. In this work, we show unconditionally that there exists a functionality that can not be securely realized between two stateless parties. Similar results have been obtained concurrently and independently by Goyal and Maji [38].

### 1.2 Technical Overview - Impossibility of Concurrent Self Composition

Consider two parties Alice and Bob executing a two-party secure computation protocol. Now consider a real-world adversary that corrupts either Alice or Bob and is allowed to participate in any arbitrary (still polynomial) number of executions of the protocol. In this setting we construct a real-world adversary that interacts with the honest party in an execution of the protocol, referred to as the *main* execution that can not be simulated in the ideal world. The starting point for realizing such an adversary is the idea that the adversary has secure computation at its disposal and it can use it to its advantage. More specifically, an adversary can at will interact with the honest party in multiple *additional* executions of the secure computation protocol and use the "information" obtained in these additional executions to complete the *main* execution. In order to realize the impossibility we need to establish what this "information" is and how can this be obtained. Looking ahead this "information" is going to be the very messages that the adversary needs to send in the main execution and it is going to be obtained via secure computation with the honest party. In this paper we extend this intuition to show impossibility of concurrent secure computation for a very general class of functionalities. However, in order to build intuition we start by considering the impossibility for the special case of 1-out-of-2 string oblivious transfer (OT).

*String OT.* Consider two parties Alice and Bob executing an instance of the string OT protocol in which Alice plays the role of the sender and Bob plays the role of the receiver. We refer to this execution as the *main* execution. Now in order to complete the main execution adversary needs to execute some function securely with Alice.

First we begin by addressing the issue of providing Bob with the ability to execute some function exactly once. Observe that providing malicious Bob with a garbled circuit allows him to evaluate (once) any function securely as

long as he can obtain the right secret keys for evaluating the garbled circuit. However obtaining such keys is not a problem as malicious Bob has access to a string OT channel with Alice. More specifically, if we set garbled circuit keys as the inputs of Alice then malicious Bob can obtain the keys of his choice and evaluate any function securely. Very roughly we have established that malicious Bob can evaluate any function securely allowing it with an access to a source of "information" of its choice.

The next question to ask is "What is the right information?" Observe that malicious Bob needs to send messages as the receiver in the main execution of the protocol.[4] Now note that the "information" that malicious Bob receives may very well be the very messages that it needs to send to Alice in the main execution. Bob obtains the messages it sends to Alice using secure evaluation. This allows us to argue that no ideal-world simulator can simulate the view of this "virtual" receiver. In arguing this we crucially rely of the fact that the ideal-world simulator is limited and can obtain *only* one key corresponding to each input wire of the garbled circuit. Additionally, we would like to stress *two* points:

- Observe that malicious Bob obtains the keys from Alice in a very straightforward and well specified manner. On the other hand a simulator trying to simulate malicious Bob is not required to follow this strategy. In particular the simulator could potentially obtain keys in an *adaptive* manner that depends on the garbled circuit it gets as input. To deal with this problem, we use a construction of Yao's garbled circuit secure against *malicious* adversaries [39].
- Since we are in the *static* input setting all the inputs of Alice must be specified before any execution of the protocol is started. In our setting the inputs of the honest Alice consist of the keys for the garbled circuit, and can always be specified before the execution of the protocol. On the other hand malicious Bob can follow any arbitrary polynomial-time strategy. In particular the adversary can choose its inputs adaptively.

*Bit OT.* The intuition described above crucially relies on the fact that the adversary can execute multiple instances of the string OT protocol in order to obtain the desired keys for input wires of the garbled circuit and thereby evaluate the circuit. Note that the adversary described in the setting of string OT will continue to work when provided with an access to a protocol that can be used to evaluate bit OTs. However unfortunately, we will not be able to rule out simulators that obtain potentially different bits of the key strings. We solve this problem using the string OT construction from bit OT of [40]. The key idea of the construction in [40] is to "encode" the keys of the garbled circuit in a manner such that the partial information obtained by any simulator on the encoded keys is essentially "useless." An important feature of the [40] technique is that it allows us to specify inputs of Alice before any bit OT protocol is executed.

[4] Our proof builds upon the intuitive application of *chosen protocol attack* as in [27].

*Extending to general functionalities.* In extending the impossibility result to general functionalities the challenge lies in realizing a form of bit OT but without the use of adaptive inputs for honest parties. The key idea in achieving this is to exploit the "uncertainty" in the input of honest Alice that must exist in the eyes of any simulator that is constrained to keep the outputs of the real-world execution and the ideal-world execution indistinguishable. In the case of *symmetric* functionalities (which are complete in the stand-alone malicious setting) the outputs of honest Alice plays a crucial role in ensuring this. In particular in order to make sure that Alice gets the correct output a simulator simulating malicious Bob is forced into leaving some "uncertainty" in the input of honest Alice. On the other hand *asymmetric* functionalities (again, which are complete in the stand-alone malicious setting) do not provide any output to honest Alice but posses a more general structure. In particular these functionalities have a structure that leaves some "uncertainty" in the input of the honest Alice regardless of the input of malicious Bob. This allows us to argue our impossibility result.

### 1.3 Technical Overview - Impossibility of Stateless Two-Party Computation

Consider three stateless parties Alice, Bob and Charlie executing two instances of a two-party secure computation protocol. Malicious Bob[5] interacts with an honest Alice in the first instance and with an honest Charlie in the second instance. In this setting malicious Bob can evaluate any function securely with Charlie and use it as source of "information." Just like in the concurrent security case we can ask the question: "What is the right information?" Observe that the adversary needs to send messages in the execution of the protocol with Alice. Now note that the "information" that malicious Bob receives from Charlie may very well be these next messages that it needs to send to Alice. This allows us to construct a "virtual" party that malicious Bob securely implements with the help of honest Charlie. Now observe that a simulator might have non-black box access to malicious Bob, however, this is essentially useless because malicious Bob acts just like a "message forwarding machine." Therefore simulator only has black-box access to malicious Bob which alone of course does not help the simulator because a malicious Bob on being rewound can always reset honest Charlie as it is stateless. Therefore we can conclude that no ideal-world adversary can simulate the view of malicious Bob because an ability to do so will contradict the security of the underlying protocol itself.

## 2 Preliminaries and Definitions

Let $k$ denote a security parameter. We say that a function is *negligible* in the security parameter $k$ if it is asymptotically smaller than the inverse of any fixed

[5] We stress that a malicious Bob is not required to be stateless.

polynomial. Otherwise, the function is said to be *non-negligible* in $k$. We say that an event happens with *overwhelming* probability if it happens with a probability $p(k) = 1 - \nu(k)$ where $\nu(k)$ is a negligible function of $k$. In this section, we recall the definitions of basic primitives studied in this paper. Let $\stackrel{c}{\equiv}$ stand for computational indistinguishability of two distributions.

Our definitions of concurrent security are judiciously borrowed from the work of Lindell [24]. Some of the text has been taken verbatim from [24], however, the definitions have been tailored and simplified to suit our requirements.

**Two-party computation.** A two-party protocol problem is cast by specifying a function that maps pairs of inputs to pairs of outputs (one input/output for each party). We refer to such a map as a *functionality* and denote it by $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$, i.e. for every pair of inputs $(x, y)$, we have a pair of outputs $(f_1(x, y), f_2(x, y))$. Thus for example, the oblivious transfer functionality is denoted by $\mathcal{F}_{OT}((m_0, m_1), b) = (\perp, m_b)$.

In the context of concurrent composition each party actually uses many inputs. These are represented by vectors of inputs $\bar{x}, \bar{y}$ for the two parties. Furthermore, let $\ell = |\bar{x}| = |\bar{y}|$ denote the number of sessions the two parties interact in. In this work we consider the setting of *static* inputs as opposed to *adaptive* inputs. More specifically in the setting of static inputs, the inputs of both parties are specified before the execution of any protocol. This differs from the more general setting of adaptive inputs, in which honest parties choose inputs for sessions on the fly (possibly using the information obtained in previous sessions).

**Adversarial Behavior.** Throughout this paper we only consider a malicious and static adversary. In a two-party interaction, such an adversary chooses one of the parties before the protocol begins (who is referred to as a corrupted party) and may then interact with the honest party on behalf of the corrupted party while arbitrarily deviating from the specified protocol. Furthermore, in this work, we consider secure computation protocols with *aborts* and no *fairness*. This notion is well studied in literature [41]. More specifically, the adversary can abort at any point and the adversary can decide when (if at all) the honest parties will receive their output even when it receives its own output. The scheduling of the message delivery is decided by the adversary.

Note that we study the security of the protocols in a setting where multiple instances of a two-party protocol (between $P_1$ and $P_2$) are being executed. In order to obtain stronger impossibilities, we look at the setting in which the same party plays the role of $P_1$ (and similarly $P_2$) across all executions of the protocol. We refer to this as the setting of *fixed roles*. In this setting, an adversary can corrupt only one party that consistently plays the role of $P_1$ (or $P_2$) across all sessions.

Observe that we consider a very restricted adversary (in terms of what it can do) and provide impossibility results in this paper. Furthermore, all the impossibility results in this setting directly translate to more demanding settings.

**Security of protocols (informal).** We give an informal description of the definition here and refer the reader to the full version for formal definitions. We start by giving the definition in the *stand-alone* case, in which the parties exe-

cute only one instance of the protocol. The security of a protocol is defined by comparing what an adversary can do in the real-world execution of the protocol to what it could have done in an ideal scenario. The real-world execution of the protocol corresponds to the actual interaction of the adversary with honest parties. The real execution of $\rho$ (with security parameter $k$, initial inputs $(x, y)$, and auxiliary input $z$ to the adversary $\mathcal{A}$), denoted $\text{EXEC}_{\rho,\mathcal{A}}(k, x, y, z)$, is defined as the output pair of the honest party and $\mathcal{A}$, resulting from the above process. The ideal scenario is formalized by considering an ideal 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. Then the ideal execution of $f$ (with security parameter $k$, initial inputs $(x, y)$ and auxiliary input $z$ to $\mathcal{S}$), denoted by $\text{EXEC}_{f,\mathcal{S}}(k, x, y, z)$ is denoted as the output pair of the honest party and the adversary $\mathcal{S}$ from the above execution. Informally, we require that executing a protocol in the real world roughly emulates the ideal process.

Next we extend the definition to the concurrent setting. Unlike in the case of stand-alone computation, in the concurrent setting the real-world protocol can be executed multiple times. Correspondingly, in the ideal world, the trusted party computes the functionality many times, each time upon the received inputs. Loosely speaking, a protocol is secure if any adversary interacting in the real-world protocol (where no trusted third party exists) can do no more harm than if it was involved in a corresponding ideal computation, as described above.

**Bounded Concurrency.** The notion of concurrent self composition allows for an unbounded (though, still polynomial) number of executions of a protocol. We sometimes refer to this notion of concurrent self composition as *unbounded* concurrency, in order to distinguish it from $m-$bounded concurrency that we describe next. In the setting of $m$-bounded concurrency, the number of concurrent executions is a priori bounded by $m$ (a fixed polynomial). More specifically, we require that in the interval between the beginning and termination of any given execution, the number of different sessions executed is at most $m$. Furthermore, the protocol design and hence the running time of each party (in a single session) can depend on this bound.

**Resettability.** The notion of resettably secure computation [36, 37] is also defined by comparing what an adversary can do in the real-world execution of the protocol to what it could have done in an ideal scenario, however there are two crucial differences. The real-world adversary in the resettable setting is allowed to reset (or, "rewind") honest parties. In a similar way, the ideal-world adversary in the resettable setting can query the ideal functionality on behalf of the adversary multiple times, each time with a different input of its choice (while the input of the honest party remains the same).

## 3 Impossibility of Static Input Concurrency

In this section we prove that the string OT functionality can not be concurrently and securely realized even in the setting of *static* inputs and against adversaries

that corrupt only one party that plays a *fixed role*. Next we generalize this and provide a general theorem allowing us to argue impossibility for a broader class (we do this in Section 4) of functionalities.

### 3.1 The Case of String OT

In this section we start by first giving an impossibility result for the string OT functionality. Roughly speaking, string OT is a two-party functionality between a Sender $S$, with input $(m_0, m_1)$ and a Receiver $R$ with input $b$ which allows $R$ to learn $m_b$ without learning anything about $m_{1-b}$. At the same time the Sender $S$ learns nothing about $b$. More formally string OT functionality $\mathcal{F}_{OT} : (\{0,1\}^{p(k)} \times \{0,1\}^{p(k)}) \times \{0,1\} \rightarrow \{0,1\}^{p(k)}$ is defined as, $\mathcal{F}_{OT}((m_0, m_1), b) = m_b$, where $p(\cdot)$ is any polynomial and only $R$ gets the output. We show that for some polynomial $p(\cdot)$ (to be fixed later), there does not exist a protocol $\pi$ that concurrently securely realizes the $\mathcal{F}_{OT}$ functionality. More specifically we show that there exists an adversary $\mathcal{A}$ who starts $\ell(k)$ sessions (to be fixed later) of the protocol $\pi$ with honest parties (with pre-specified inputs drawn from a particular distribution $\mathcal{D}$) such that no ideal-world adversary whose output is computationally indistinguishable from the output of real-world adversary $\mathcal{A}$ exists. We stress that the adversary (we construct) corrupts only one of the two parties – the Sender $S$ or the Receiver $R$ in all the $\ell(k)$ sessions. In other words we are in the setting of *fixed roles* (and our results of course extend also to the setting where the adversary plays different roles).

**Theorem 1** *(impossibility of static input concurrent-secure string OT) Let $\pi$ be any protocol which implements*[6] *the $\mathcal{F}_{OT}$ functionality for a particular (to be determined later) polynomial $p(k)$. Then, (assuming one-way functions exist) there exists a polynomial $\ell(k)$ and a distribution $\mathcal{D}$ over $\ell(k)$-tuple of inputs and an adversarial strategy $\mathcal{A}$ such that for every probabilistic polynomial-time simulation strategy $\mathcal{S}$, definition of concurrent security* cannot *be satisfied when the inputs of the parties are drawn from $\mathcal{D}$.*

***Proof Intuition.*** We start by giving an informal outline of the proof. We postpone the full proof to the full version.

1. **Setting up a Chosen protocol attack:** As stated earlier our goal is to construct a real-world adversary that can achieve something in the real world that is infeasible for any ideal-world adversary to accomplish in the ideal world. The starting point for realizing such an adversary is to consider a *chosen protocol attack* [42, 43, 27] and adapt it to our setting. Let us start by considering a protocol $\pi$ that concurrently securely realizes the $\mathcal{F}_{OT}$ functionality. The key attack methodology of chosen protocol attack (first considered in the context of zero-knowledge protocols) is to consider a specific protocol $\pi'$ (which may depend on $\pi$ and hence the name "chosen protocol

[6] We say that a protocol implements a functionality if the protocol allows two parties to evaluate the desired function. This protocol however may not be secure.

attack") which when concurrently executed with $\pi$ renders $\pi$ completely insecure. Next we present the chosen protocol attack in our specific setting. Consider a setting with four parties $A, \tilde{B}, \tilde{C}, D$. (Looking ahead parties $\tilde{B}$ and $\tilde{C}$ will be corrupted by the adversary.) In this setting $A$ and $\tilde{B}$ execute an instance of the OT protocol $\pi$ where $A$ plays the role of the Sender (with inputs say $(m_0, m_1)$) and $\tilde{B}$ plays the role of the Receiver (with some choice bit $c$ as input). At the same time, parties $\tilde{C}$ (with inputs say $(m'_0, m'_1)$) and $D$ (with a bit $b \in \{0, 1\}$ and values $m_b, w$ as input) execute the following chosen protocol $\pi'$ (which depends on $\pi$) and is meant to render $\pi$ insecure. We now describe this protocol $\pi'$. $\pi'$ involves first an execution of $\pi$ between $\tilde{C}$ and $D$ in which $\tilde{C}$ plays the role of the sender (with inputs say $(m'_0, m'_1)$) and $D$ plays the role of the receiver. Now note that $D$ playing the role of the receiver obtains the value $m'_b$ in the execution of $\pi$. $D$ now checks if $m'_b = m_b$ and sends $w$ to $\tilde{C}$ if this is indeed the case. Observe that in the above setting $A$ is provided with $(m_0, m_1)$ as input and $D$ is provided with $b$ and the corresponding $m_b$ as input.

Next we show how concurrent execution of protocol $\pi$ with the protocol $\pi'$ renders $\pi$ insecure. Towards this goal, consider a real-world adversary $\mathcal{A}$ that corrupts $\tilde{B}$ and $\tilde{C}$, ignores their inputs and proceeds as follows[7]. Our adversary $\mathcal{A}$ merely acts as an intermediary who forwards the messages he receives from honest $A$ to honest $D$ on behalf of $\tilde{C}$ and similarly forwards the messages it receives from honest $D$ to honest $A$ on behalf of $\tilde{B}$[8]. Our adversary $\mathcal{A}$ will be able to complete the execution of $\pi$ with Sender $A$ and the execution of $\pi$ (executed as a part of $\pi'$) with receiver $D$. This setting roughly amounts to $A$ and $D$ executing a protocol $\pi$ in which $A$ has inputs $(m_0, m_1)$ and $D$ has input $b$. Additionally in this execution, $D$ will always obtain the value $m_b$ which will match its input $m_b$ and therefore it will always send the value $w$ to $\mathcal{A}$. Our adversary outputs this value $w$ as its output. On the other hand, executing merely an ideal version of $\pi$ and obtaining its output will not help $\tilde{B}, \tilde{C}$ to successfully complete an interaction with $D$ and thus the adversary does not output $w$. This results in breaking the indistinguishability between the real world and the ideal world, contradicting the security definition of $\pi$.

Now that, we have described an adversarial strategy in a four party scenario where $\mathcal{A}$ interacts in an execution of $\pi$ and an execution of $\pi'$. We would like to move to a setting with only concurrent executions of the protocol $\pi$.

2. **Using Yao [1] for simulating parties $\tilde{C}, D$ in the head:** With the goal of completely removing the execution of $\pi'$ between the adversary and an

[7] Recall that we are in setting of static inputs. This setting requires that the inputs of the honest parties be pre-specified before any execution of the protocol happens. On the other hand our adversary $\mathcal{A}$ may deviate arbitrarily from the protocol specification based on his input and auxiliary input. In particular, it could adaptively decide on the messages it sends in the protocol ignoring the input it obtains completely.

[8] We remark that in case the underlying protocol uses identities, then we will think of $\tilde{C}$ as using the identity of $A$ and $D$ as using the identity $B$. As we will see later, this continues to work in our setting.

external party $D$ we provide the adversary with a garbled circuit to simulate the execution of the protocol $\pi'$ "in the head". Just like the previous scenario consider an adversary $\mathcal{A}$, who interacts with $A$ in an execution of $\pi$. In this execution, $A$ with input $(m_0, m_1)$ acts as the Sender and $\mathcal{A}$ acts as the Receiver. We will refer to this execution as the *main* execution of $\pi$. However, we need this adversary $\mathcal{A}$ to send messages to $A$ on behalf of a receiver. Our approach for generation of these messages is provide the adversary $\mathcal{A}$ with a garbled circuit as an auxiliary input that it can use to evaluate the next messages on behalf of $D$ in $\pi$, in effect simulating the interaction of $\tilde{C}$ with $D$ "in its head". In other words $\mathcal{A}$ now acts on behalf of the party $\tilde{B}$ and interacts with $A$ in an execution of protocol $\pi$ and executes the chosen protocol via evaluating the garbled circuit. This allows us to simulate the protocol $\pi'$. However, note that in order to securely evaluate the garbled circuit, the adversary $\mathcal{A}$ will need keys corresponding to the input wires. Furthermore, since we will rely on the security of the garbled circuits itself we will need to ensure that only one key per wire can be obtained. Looking ahead this will be crucial when considering ideal-world adversaries and try to reach a contradiction. Loosely speaking this will guarantee that garbled circuits will be useful for a one-time evaluation only, revealing only the input/output behavior of the underlying function.
Next, we show how to achieve this task.

*Garbled circuit keys:* In the above setting note that $\mathcal{A}$ needs to obtain keys for secure evaluation of the garbled circuit. We achieve this by using other invocations of the OT protocol $\pi$ at our disposal. Note that corresponding to each input wire of the garbled circuit we will have two keys and the adversary $\mathcal{A}$ needs to learn one of them in order to evaluate the garbled circuit correctly. Corresponding to each input wire of the garbled circuit we will provide the two keys to honest party $A$. The adversary $\mathcal{A}$ can choose and obtain the appropriate keys in multiple executions of the OT protocol $\pi$. We refer to these executions as the *additional* executions of $\pi$. Note that this will involve an executions of the OT protocol that is interleaved with the main invocation of the OT protocol. The benefits achieved here are two fold. Firstly, we are now able to transfer the appropriate garbled circuit keys to the adversary $\mathcal{A}$ via executions of $\pi$. This makes the chosen protocol attack in the setting where only multiple instance of $\pi$ are executed concurrently possible. Secondly, loosely speaking, the adversary $\mathcal{A}$ for each input wire can obtains only one of the two keys that the honest party $A$ holds. This intuitively follows from the sender security of the OT protocol $\pi$. Further note here that since the honest party $A$ always plays the role of the sender, the adversary $\mathcal{A}$ is corrupting only the receiver in all executions of $\pi$. This concludes the construction of our real-world adversary.

*Remark on static nature of inputs.* Observe that the inputs provided to the honest party $A$ include the values $m_0$, $m_1$ and the garbled circuit keys which can all be fixed in advance. Recall that since we are in the *static* input setting

this is required for our adversary. Additionally a garbled circuit, generated as above, is provided as an auxiliary input to the adversary.

3. **The contradiction:** Now that we have roughly specified the details of our real-world adversary $\mathcal{A}$ we will now provide the key idea behind why no simulator (or the ideal-world adversary) $\mathcal{S}$ can simulate the view of the adversary $\mathcal{A}$ in all executions of $\pi$ given access to the ideal functionality $\mathcal{F}_{OT}$ only. Observe that the real-world adversary $\mathcal{A}$ is a deterministic procedure that uses garbled circuits to securely evaluate the messages it needs to send to $A$. From the security of garbled circuits, the messages sent by the adversary $\mathcal{A}$ in the main session roughly amount to be the messages generated by an external honest party. In particular this means that $\mathcal{S}$ essentially has only black-box access to the source of these messages and it can not rewind it. Therefore the simulator $\mathcal{S}$ can not simulate the view of the adversary in the main session allowing us to reach a contradiction.

***Implications for bounded concurrency.*** Observe that the attack described in the above proof (in the unboundend concurrent setting) has natural implications in the bounded setting as well. In particular, the number of sessions that our adversary executes, or the "extent" of concurrency used by the adversary in the proof above in order to arrive at a contradiction is bounded by the communication complexity of the protocol. More specifically the adversary needs to make one additional OT call for every bit that the Sender sends in the protocol. This yields the following corollary:

**Corollary 1** *Let $\pi$ be any protocol that securely realizes the $\mathcal{F}_{OT}$ functionality under $m$-bounded concurrent self-composition. Then (assuming one-way functions) the communication complexity of (a single session of) $\pi$ is at least $m$ bits.*

***Implications in the setting of very limited concurrency.*** Observe that the attack described in the above proof (in the setting of arbitrary concurrent composition) has natural implications even if we restrict ourselves to a very limited concurrent composition. In particular, the adversary in the proof above only interleaves the main session with the rest of the sessions all of which are executed just sequentially.

### 3.2 General Theorem for Impossibility Results

In order to extend our results to more general functionalities we present a general theorem that allows us to argue impossibility for any functionality which satisfies certain special properties. We next state our theorem. In Section 4 we will use this theorem to argue impossibility for large classes of functionalities.

**Theorem 2** *Let $\mathcal{F}$ be any two-party functionality between a Sender, $S$ and a Receiver, $R$. Consider a protocol $\pi$ that securely realizes $\mathcal{F}$ in the concurrent setting. Then (assuming one-way functions) at least one of the following is* not *true.*

1. ***Key Transmission.*** *There exists a real-world adversary $\mathcal{A}_1$ and a distribution $(\bar{x}, z) \leftarrow \mathcal{D}_1(X_0, X_1)$ with inputs $X_0, X_1$ (each one bit long) such that the following holds. $\mathcal{A}_1$ on input $b, z$ interacting with an honest $S$ with input $\bar{x}$ in concurrent executions of $\pi$ outputs $X_b$ such that every ideal-world adversary $\mathcal{S}_1$ running on input $(b, z)$ that simulates $\mathcal{A}_1$ can be simulated* [9] *by querying an oracle that holds $X_0$ and $X_1$ for only one of the values except with negligible probability.*
2. ***Chosen Protocol Attack.*** *There exists a chosen protocol $\pi'$ for $\pi$ and a distribution $(x, y) \leftarrow \mathcal{D}_2$ such that there exists a real-world adversary $\mathcal{A}_2$ (with auxiliary input $z$) that interacts with an honest sender $A$ of $\pi$ with input $x$ and a honest receiver $D$ of $\pi'$ with input $y$ and that there exists no corresponding simulator $\mathcal{S}_2$. Namely, for every hybrid-world adversary $\mathcal{S}_2$ interacting with the ideal functionality $\mathcal{F}$ (that talks to $A$) and a honest receiver $D$ of $\pi'$ we have*

$$\text{EXEC}_{\mathcal{F},\pi',\mathcal{S}_2}(k, x, y, z) \stackrel{c}{\not\equiv} \text{EXEC}_{\pi,\pi',\mathcal{A}_2}(k, x, y, z)$$

We start by giving the intuition. The key difference between the theorem as stated above (beside the natural generalization) from Theorem 1 is that the above theorem requires transmission of bits only. More specifically, the adversary gets to choose one bit among $X_0$ and $X_1$ and gets $X_b$ as the response. On the other hand in the case of string OT these values were actually strings. We deal with this problem by using a specialized garbled circuit (see the full version for more details) construction in which all the keys are bits. This construction involves applying an encoding function to the keys of the garbled circuit thereby obtaining encoded key bits. These encoded key bits are then transferred via bit OTs. The security guarantee is that each key for every input wire is encoded in a manner such that even an adversary that obtains bits of its choice can not learn more than one key per input wire. We prove the above theorem in the full version.

### 3.3 Example: The Case of Bit OT

In this section we give an impossibility result for the bit OT functionality. Roughly speaking, bit OT is a two-party functionality between a sender $S$, with input bits $(m_0, m_1)$ and a receiver $R$ with input $b$ which allows $R$ to learn $m_b$ without learning anything about $m_{1-b}$. At the same time the sender $S$ learns nothing about $b$. More formally bit OT functionality is defined as $\mathcal{F}_{\mathsf{Bit\text{-}OT}} : (\{0,1\} \times \{0,1\}) \times \rightarrow \{0,1\}$ where $\mathcal{F}_{\mathsf{Bit\text{-}OT}}((m_0, m_1), b) = m_b$ and only $R$ gets the output. We stress that we are in the setting of *static inputs* and *fixed roles*.

[9] Note that there are two levels of simulation happening here. First, any adversary $\mathcal{A}_1$ who runs $\pi$ with honest party input drawn from $\mathcal{D}(X_0, X_1)$ is being simulated by an ideal world adversary $\mathcal{S}_1$ with access to only the ideal functionality $\mathcal{F}$. Second, such a simulator $\mathcal{S}_1$ learns only one of the two values $X_0$ or $X_1$. That is, his ideal-world interaction can actually be simulated by querying for only one of the two values (to generate the honest party input).

**Lemma 1.** *(impossibility of static input concurrency - bit OT) Let $\pi$ be any protocol which implements the $\mathcal{F}_{\mathsf{Bit\text{-}OT}}$ functionality. Then, (assuming one-way functions) there exists a polynomial $\ell(k)$ and a distribution $\mathcal{D}$ over $\ell(k)$-tuple of inputs and an adversarial strategy $\mathcal{A}$ such that for every probabilistic polynomial-time simulation strategy $\mathcal{S}$, definition of concurrent security* cannot *be satisfied when the inputs of the parties are drawn from $\mathcal{D}$.*

**Proof (Informal).** Lemma 1 follows by a direct application of Theorem 2 and the ideas developed in Theorem 1. Observe that Theorem 2 requires us to prove two properties, namely, key transmission and chosen protocol attack. Key transmission property requires us to construct an adversary $\mathcal{A}_1$ and a distribution $\mathcal{D}_1$ that generates inputs for the honest party $S$ and auxiliary input $z$ for the adversary $\mathcal{A}_1$. Note that our adversary will run only one execution of $\pi$. In particular this means that $\nu(k) = 1$. We first specify our distribution $\mathcal{D}_1$. Distribution $\mathcal{D}_1$ on input $X_0, X_1$ outputs $X_0, X_1$ for the honest party $S$ and it outputs an empty string $z$ for $\mathcal{A}_1$. Observe that $\mathcal{A}_1$ on input $b$ can easily obtain $X_b$ in one execution of the protocol $\pi$ playing as the receiver.

Next note that the chosen protocol attack for the case of bit OT functionality is the same as the string OT functionality. Note that our goal here is to just give intuition for Theorem 2 and since we prove general theorems for all functionalities (in Section 4) we skip the details here. ■

## 4 The Case of General Functionalities

In this section we give impossibility results for general functionalities. We first consider symmetric functionalities in which both parties get the same output and then we consider functionalities in which the two parties get different outputs. We stress that all our results are in the setting of *static inputs* and *fixed roles*. Surprisingly, completeness theorems (and constructions) for secure computation of such functions are known in the stand-alone two-party setting [33, 32, 34, 2].

### 4.1 The Case of Symmetric General Functionalities

Consider a two-party functionality $\mathcal{F}_{sym}$ between a sender $S$ with input $x$ and a receiver $R$ with input $y$ which allows both $S$ and $R$ to learn $f(x, y)$ (and nothing else). More formally, let $f : X \times Y \to Z$ be any finite function[10] then a symmetric functionality $\mathcal{F}_{sym} : X \times Y \to Z \times Z$ is defined as, $\mathcal{F}_{sym}(x, y) = (f(x, y), f(x, y))$ where both $S$ and $R$ get $f(x, y)$. For any *complete* symmetric function $f$, as defined below, we show that there does not exist a protocol $\pi$ that concurrently securely realizes the $\mathcal{F}_{sym}$ functionality.

Loosely speaking, a symmetric functionality that contains an *embedded-OR* is complete. We know that $\mathcal{F}_{sym}$ is *complete* [44] (both in the setting of *semi-honest* and *malicious* adversaries) in the stand-alone setting of information-theoretically

[10] A function is said to be finite if both the domain and the range are of finite size.

secure computation [11] iff $\exists a_0, a_1, b_0, b_1$ such that $f(a_0, b_0) = f(a_0, b_1) = f(a_1, b_0) \neq f(a_1, b_1)$.

**Theorem 3** *(impossibility of static input concurrent security for symmetric complete functionalities) Let $\mathcal{F}_{sym}$ be a functionality that is complete in the stand-alone setting and $\pi$ be any protocol which implements $\mathcal{F}_{sym}$. Then, (assuming one-way functions) there exists a polynomial $\ell(k)$ and a distribution $\mathcal{D}$ over $\ell(k)$-tuple of inputs and an adversarial strategy $\mathcal{A}$ such that definition of concurrent security* cannot *be satisfied when the inputs of the parties are drawn from $\mathcal{D}$.*

Our argument in the proof of the theorem above specifically relies on the fact that the output distribution of the honest party between a real-world and an ideal-world execution cannot change. We give a proof of the above theorem in the full version. The impossibility of secure computation for complete, symmetric functionalities in the unbounded concurrent setting has natural implications in the bounded setting as well. In particular, the number of sessions, or the "extent" of concurrency used by the adversary in the proof of theorem above in order to arrive at a contradiction is polynomially related to the communication complexity of the protocol. This yields the following corollary:

**Corollary 2** *Let $\pi$ be any protocol that securely realizes $\mathcal{F}_{sym}$ functionality under $m$-bounded concurrent self-composition for any polynomial $m$. Then (assuming one-way functions) there exists a constant $c \geq 0$ such that for sufficiently large $k$, the communication complexity of (a single session of) $\pi$ is at least $\frac{m}{k^c}$-bits.*

### 4.2 The Case of Asymmetric General Functionalities

Consider a two-party functionality $\mathcal{F}_{asym}$ between a sender $S$, with input $x$ and a receiver $R$ with input $y$ which allows $R$ to learn $f(x, y)$ (and nothing else) and where $S$ learns nothing. More formally, let $f : X \times Y \rightarrow Z$ be any finite function then an asymmetric functionality $\mathcal{F}_{asym}$ is defined as, $\mathcal{F}_{asym}(x, y) = (\perp, f(x, y))$ where $S$ gets nothing and $R$ gets $f(x, y)$. We show that there does not exist a protocol $\pi$ that concurrently securely realizes any *complete* $\mathcal{F}_{asym}$ functionality as defined below.

We know that $\mathcal{F}_{asym}$ is *complete* [33] in the setting of stand-alone two-party computation in the presence of *malicious* adversaries iff $\forall b_0, \exists b_1, a_0, a_1$ such that

$$f(a_0, b_0) = f(a_1, b_0) \wedge f(a_0, b_1) \neq f(a_1, b_1).$$

**Theorem 4** *(impossibility of static input concurrent security for asymmetric complete functionalities) Let $\pi$ be any protocol which implements any $\mathcal{F}_{asym}$*

[11] Note that the setting of stand-alone and information-theoretic security is used only to define the class of functions. We deal with the concurrent and computational security of this class of functions in this work.

*functionality that is complete in the stand-alone setting. Then, (assuming one-way functions) there exists a polynomial $\ell(k)$ and a distribution $\mathcal{D}$ over $\ell(k)$-tuple of inputs and an adversarial strategy $\mathcal{A}$ such that definition of concurrent security,* cannot *be satisfied when the inputs of the parties are drawn from $\mathcal{D}$.*

The key difference when considering asymmetric functionalities as opposed to the case of symmetric functionalities is that for asymmetric functionalities only one party gets the output. Observe that our arguments in Theorem 3 specifically relied on the fact that the output distribution of either of the parties between a real-world and ideal-world execution (note that this output contains the outputs of both the honest party and the adversarial party) cannot change. However, complete asymmetric functionalities possess a larger structure than what is available to complete symmetric functionalities. We crucially use this additional structure in our proof. We give the full proof of the above theorem in the full version.We also note that a direct analogue of Corollary 2 holds in the asymmetric setting as well.

## 5 Impossibility of Stateless Two-Party Computation

In this section, we show the existence of a deterministic two-party functionality for which there does not exist any stateless secure protocol. We start by giving some intuition about our impossibility result. The key observation behind our impossibility result is that even a deterministic adversary in interaction with an honest party can come up with honest looking messages on behalf of an adversarial party by obtaining them from other honest interactions. However, loosely speaking, since these messages are obtained from other honest parties, the simulator only has black box access to the source of these messages. The simulator does have non-black box access to the adversary itself, however, it is essentially useless because the adversary acts just like a "message forwarding machine." Therefore simulator essentially only has black-box access to the adversary which alone of course does not help the simulator because a real-world adversary can also reset honest parties.

*Functionalities considered.* Before we move on to a formal claim, we introduce description of some notation that will be useful in this work. Let us start by describing a general functionality. Let $\mathcal{F}_{universal} : (\{0,1\}^{p_1(k)} \times \{0,1\}^{q_1(k)}) \times (\{0,1\}^{p_2(k)} \times \{0,1\}^{q_2(k)}) \rightarrow (\{0,1\}^{r(k)} \times \{0,1\}^{r(k)})$ be the following two-party (between $P_1$ and $P_2$) functionality, $\mathcal{F}_{universal}((C,x),(C',y))$, where $\mathcal{F}_{universal}$ obtains the input $(C,x)$ from $P_1$ and $(C',y)$ from $P_2$. The functionality outputs $\perp$ to both $P_1$ and $P_2$ if $C \neq C'$ and it outputs $C(x,y)$ to both $P_1$ and $P_2$ otherwise, where $p_1(\cdot), q_1(\cdot), p_2(\cdot), q_2(\cdot), r(\cdot)$ are polynomials. We stress that we consider a functionality which outputs the same value to both parties. This is consistent with the definition of [37]. Let $\pi$ be a protocol that resettably securely realizes the functionality $\mathcal{F}_{universal}$. Next we describe two circuits that will be useful in our context.

**Equality testing:** Let $C_{eq}(x, y)$ be a circuit that outputs 1 if $x = y$ and 0, otherwise. In an execution of the ideal functionality $\mathcal{F}_{universal}$, in which the parties $P_1$ and $P_2$ input $(C_{eq}, x)$ and $(C_{eq}, y)$ respectively corresponds to the *equality testing* functionality.

**Next message function of $P_2$:** Let $C_\pi$ be the next message function of $P_2$ in the protocol $\pi$. In other words, $C_\pi$ is a non-interactive algorithm that gets as an input, the input and random tape of $P_2$ and the history of messages sent by $P_1$, and outputs the next message that $P_2$ would send in a real execution of $\pi$ in which it sees this message history. More formally, $C_\pi$ takes as input a sequence of messages $(h_1, h_2 \dots h_t)$ and $P_2$'s input $(C, y)$ and random coins $r$ and generates the next message of $P_2$, i.e. the message that $P_2$ sends in the execution with the history $h_1, h_2 \dots h_t$.

**Theorem 5** *(impossibility of static input resettability) There does not exist any stateless secure protocol for the $\mathcal{F}_{universal}$ functionality. (unconditionally)*

Now we give a brief outline of the proof. The full proof is provided in the full-version. For the sake of contradiction, assume that there exists a protocol $\pi$, with round complexity $r$, that resettably securely realizes the functionality $\mathcal{F}_{universal}$. We give an outline of the proof before giving all the details.

1. We consider the "first scenario" of two parties $P_1$ and $P_2$ executing $\pi$. In this setting we consider an adversary that corrupts $P_2$ and interacts honestly with $P_1$, in the first incarnation. However, it does not generate the honest responses on its own. In fact it obtains these responses from $P_1$ itself (via different incarnations of $P_1$). In this setting, as per the definition of resettability there must exist a simulator (plausibly non-black box in this adversary), which can "extract" the input used by the adversary, on behalf of $P_2$, in interaction with the first incarnation of $P_1$.
2. Next we consider the "second scenario" of two parties $P_1$ and $P_2$ executing $\pi$. In this setting we construct a real-world adversary that corrupts $P_1$ and interacts with an honest $P_2$. This adversary internally uses the ideal-world adversary constructed in the "first scenario" to extract the input of $P_2$. Finally, we observe that no ideal-world adversary in this "second scenario" can achieve the same, thereby reaching a contradiction.

## 6 Acknowledgements

The work is supported in part by NSF grants 0830803, 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. The work of the fourth author

has been done while visiting UCLA and is supported in part by the European Commission through the FP7 programme under contract 216676 ECRYPT II.

We thank Hemanta K. Maji, Akshay Wadia, Vipul Goyal and Amit Sahai for valuable discussions.

## References

1. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS, IEEE Computer Society (1986) 162–167
2. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In Aho, A.V., ed.: STOC, ACM (1987) 218–229
3. Canetti, R., Fischlin, M.: Universally composable commitments. In: CRYPTO. LNCS, Springer (2001) 19–40
4. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC. (2002) 494–503
5. Barak, B., Canetti, R., Nielsen, J., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: FOCS. (2004) 186–195
6. Canetti, R., Pass, R., Shelat, A.: Cryptography from sunspots: How to use an imperfect reference string. In: FOCS. (2007) 249–259
7. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Eurocrypt. (2007)
8. Chandran, N., Goyal, V., Sahai, A.: New constructions for uc secure computation using tamper-proof hardware. EUROCRYPT (2008)
9. Lin, H., Pass, R., Venkitasubramaniam, M.: A unified framework for concurrent security: universal composability from stand-alone non-malleability. In: STOC. (2009) 179–188
10. Groth, J., Ostrovsky, R.: Cryptography in the multi-string model. In: CRYPTO. (2007) 323–341
11. Goyal, V., Katz, J.: Universally composable multi-party computation with an unreliable common reference string. In: TCC. (2008) 142–154
12. Garg, S., Goyal, V., Jain, A., Sahai, A.: Bringing people of different beliefs together to do uc. In: TCC. (2011) 311–328
13. Micali, S., Pass, R., Rosen, A.: Input-indistinguishable computation. In: FOCS. (2006) 367–378
14. Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently secure computation in constant rounds. In: EUROCRYPT. (2012)
15. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. (2003)
16. Prabhakaran, M., Sahai, A.: New notions of security: achieving universal composability without trusted setup. In: STOC. (2004) 242–251
17. Barak, B., Sahai, A.: How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In: FOCS, IEEE Computer Society (2005) 543–552
18. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: FOCS. (2010) 541–550
19. Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. (2004) 232–241

20. Goyal, V.: Positive results for concurrently secure computation in the plain model. Cryptology ePrint Archive, Report 2011/602 (2011) http://eprint.iacr.org/.
21. Goyal, V., Jain, A., Ostrovsky, R.: Password-authenticated session-key generation on the internet in the plain model. In: CRYPTO. (2010) 277–294
22. Pass, R., Rosen, A.: Bounded-concurrent secure two-party computation in a constant number of rounds. (2003)
23. Lindell, Y.: Lower bounds for concurrent self composition. In Naor, M., ed.: TCC. LNCS, Springer (2004) 203–222
24. Lindell, Y.: Lower bounds and impossibility results for concurrent self composition. J. Cryptology **21**(2) (2008) 200–249
25. Garay, J.A., MacKenzie, P.D.: Concurrent oblivious transfer. In: FOCS. (2000) 314–324
26. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. J. Cryptology **19**(2) (2006) 135–167
27. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: FOCS. (2006) 345–354
28. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC. (1998) 409–418
29. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. In: EUROCRYPT. (1999) 415–431
30. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In: STOC. (2001) 560–569
31. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS. (2002) 366–375
32. Kilian, J.: Founding cryptography on oblivious transfer. In Simon, J., ed.: STOC, ACM (1988) 20–31
33. Kilian, J.: More general completeness theorems for secure two-party computation. In: STOC '00, New York, NY, USA, ACM (2000) 316–324
34. Beimel, A., Malkin, T., Micali, S.: The all-or-nothing nature of two-party secure computation. In Wiener, M.J., ed.: CRYPTO. LNCS, Springer (1999) 80–97
35. Agrawal, S., Goyal, V., Jain, A., Prabhakaran, M., Sahai, A.: New impossibility results for concurrent composition and a non-interactive completeness theorem for secure computation. In: CRYPTO. (2012)
36. Goyal, V., Sahai, A.: Resettably secure computation. In: EUROCRYPT. (2009) 54–71
37. Goyal, V., Maji, H.K.: Stateless cryptographic protocols. In: FOCS. (2011) http://research.microsoft.com/en-us/people/vipul/gm11.pdf.
38. Goyal, V., Maji, H.K.: Personal communication. (2012)
39. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In Wagner, D., ed.: CRYPTO. LNCS, Springer (2008) 39–56
40. Brassard, G., Crépeau, C., Santha, M.: Oblivious transfers and intersecting codes. IEEE Transactions on Information Theory **42**(6) (1996) 1769–1780
41. Goldwasser, S., Lindell, Y.: Secure computation without agreement. In Malkhi, D., ed.: DISC. LNCS, Springer (2002) 17–32
42. Kelsey, J., Schneier, B., Wagner, D.: Protocol interactions and the chosen protocol attack. In: Security Protocols Workshop. (1997) 91–104
43. Lindell, Y.: General composition and universal composability in secure multi-party computation. In: FOCS, IEEE Computer Society (2003) 394–403
44. Kilian, J.: A general completeness theorem for two-party games. In Koutsougeras, C., Vitter, J.S., eds.: STOC, ACM (1991) 553–560