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.

Keywords: Impossibility, Static Input Concurrent Self-Composition.

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* [Yao86,GMW87] 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 on [CF01,CLOS02,BCNP04,CPS07,Kat07,CGS08,LPV09,GO07,GK08,GGJS11] 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 [MPR06,GGJS12] and super-polynomial simulation [Pas03,PS04,BS05,LPV09,CLP10]. Both of these notions, although very useful in specialized settings, *do not* suffice in general. Additionally other models that limit the level of concurrency have also been considered [Pas04,Goy11] or allow simulation using additional outputs from the ideal functionality [GJO10]. Among these models the model of m-bounded concurrency [PR03,Pas04] which allows for m different protocol executions to overlap has received a lot of attention in the literature [PR03,Pas04,Lin04,Lin08]. Unbounded concurrent oblivious transfer in the restricted model where all the inputs in all the executions are assumed to be independent has been constructed in [GM00].

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 [CF01,CKL06]. Concurrent self-composability [Lin04] 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"). Instead, in the very natural setting of *static* (pre-specified) inputs only the $\mathcal{F}_{ZK+OT}$ functionality [BPS06] (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 [Goy11] have been ruled out.

This leaves a large gap between the very few functionalities (e.g., $ZK + OT$) for which the impossibility [BPS06,Goy11] has been proved, and the functionalities for which there exist secure protocols (e.g., concurrent zero knowledge [DNS98,RK99,KP01,PRS02,BPS06]). 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 [Kil88,Kil00,BMM99] 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.

Independently 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 [AGJ$^+$12], 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 [GS09,GM11a]. 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 [GM11a] 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 [GM12].

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 [GKR08].

[4] Our proof builds upon the intuitive application of *chosen protocol attack* as in [BPS06].

- 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 [BCS96]. The key idea of the construction in [BCS96] 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 [BCS96] technique is that it allows us to specify inputs of Alice before any bit OT protocol is executed.

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 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

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

$p(k) \geq 1 - \nu(k)$ where $\nu(k)$ is a negligible function of k. In this section, we recall the definitions of basic primitives studied in this paper. Let $\overset{c}{\equiv}$ stand for computational indistinguishability of two distributions.

Our definitions of concurrent security are judiciously borrowed from the work of Lindell [Lin08]. Some of the text has been taken verbatim from [Lin08], 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 $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, 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 [GL02]. 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 Appendix A for formal definitions. We start by giving the definition in the *stand-alone* case, in which the parties execute 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 ρ corresponds to the actual interaction of the adversary with honest parties. The execution of ρ (with security parameter k, initial inputs (x, y), and auxiliary input z to the adversary $\mathcal{A}$), denoted $\mathrm{EXEC}_{\rho,\mathcal{A}}(k, x, y, z)$, 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 $\mathrm{EXEC}_{f,\mathcal{S}}(k, x, y, z)$ 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 [GS09,GM11a] 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). We give a formal definition of this notion in Appendix A.

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 $\mathcal{F}_{OT} : (\{0,1\}^{p(k)} \times \{0,1\}^{p(k)}) \times \{0,1\} \to \{0,1\}^{p(k)}$ 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 π 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 π 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 π 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 (Definition 1) 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 and give the full proof after that.

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* [KSW97,Lin03,BPS06] and adapt it to our setting. Let us start by considering a protocol π 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 π' (which may depend on π and hence the name "chosen protocol attack") which when concurrently

[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.

executed with π renders π 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 π 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 π' (which depends on π) and is meant to render π insecure. We now describe this protocol π'. π' involves first an execution of π 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 π. 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 π with the protocol π' renders π 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 π with Sender A and the execution of π (executed as a part of π') with receiver D. This setting roughly amounts to A and D executing a protocol π 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 π 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 π.

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

2. **Using Yao [Yao86] for simulating parties $\tilde{C}, D$ in the head:** With the goal of completely removing the execution of π' between the adversary and an external party D we provide the adversary with a garbled circuit to simulate the execution of the protocol π' "in the head". Just like the previous scenario consider an adversary $\mathcal{A}$, who interacts with A in an execution of π. 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 π. However, we need this adversary $\mathcal{A}$ to send messages to A on behalf of a receiver. Our approach for the generation of these messages is to 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 π, 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 π and executes the chosen protocol via evaluating the garbled circuit. This allows us to simulate the protocol π'. 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 π 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

[7] Recall that we are in setting of static inputs. This setting requires that the inputs of the honest parties be prespecified 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 $\tilde{B}$ as using the identity of D. As we will see later, this continues to work in our setting.

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 π. We refer to these executions as the *additional* executions of π. 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 π. This makes the chosen protocol attack in the setting where only multiple instance of π 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 π. Furthermore 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 π. 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 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 π 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. (This involves a number of technicalities and we refer the reader to Appendix B for details.) 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 unbounded 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 π 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) π 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.

Full Proof of Theorem 1. We start by recalling and building some notation that we will use in our proof. Next we will consider specifics of our setting. This in particular will include the details on the specifications of inputs of all parties. Then we will define formally the strategy of the real-world adversary. Finally, we will argue that the output of this real-world adversary can not be computationally indistinguishable from the output of any ideal-world adversary.

Notation. Let π be a two-party protocol between a Sender S with inputs (m_0, m_1) and a Receiver R with a choice bit b. Without loss of generality we assume that S sends both the first and the last message in the protocol. Further assume that S sends exactly n messages in the execution of this protocol π. Therefore R sends $n-1$ messages. For the sake of contradiction let us assume that π concurrently securely realizes the

$\mathcal{F}_{OT}$ functionality. Let π' be a protocol between a sender S with inputs (m_0, m_1) and a receiver R with a choice bit b and additional inputs $\tilde{m}$ and w. In the protocol π', S and R first proceed by executing π with inputs (m_0, m_1) and b respectively. At the end of the execution of π, R obtains m_b. R checks to see if $\tilde{m} = m_b$ and sends w to S if this is indeed the case. Otherwise, it just sends $\perp$ to S. Note that R sends n messages in the execution of this protocol π'.

Now, consider the next message function $F_i(b, m, w, r, M_1, M_2 \ldots, M_i)$ where $i \in [n]$ of R for the protocol π'. More specifically, $F_i(b, m, w, r, M_1, M_2, \ldots, M_i)$ generates the ith message that an honest R on input (b, m, w) and random coins r would generate corresponding to the execution in which $M_1, M_2 \ldots M_i$ are sent to R for the protocol π'. Let $\mathcal{F}[b, m, w, r]$ be a reactive functionality parameterized by b, m, w, r that can be invoked n times. The ith invocation of $\mathcal{F}[b, m, w, r]$ expects an input $M_1, \ldots, M_i$ and outputs the ith message that R would have sent in π'. The functionality has the values b, m, w, r which are built into the functionality itself. Let $(\mathtt{Yao}_1, \mathtt{Yao}_2)$ be an implementation of the garbled circuit technique as defined in Definition 5. Roughly speaking $\mathtt{Yao}_1$ is an algorithm that takes the description of a reactive functionality as input and outputs a garbled circuit and the associated keys. Note that there are two keys for every input wire. $\mathtt{Yao}_2$ on the other hand takes as input the garbled circuit and a key corresponding to each wire and outputs an evaluation of the garbled circuit. Since, we are in the malicious case the adversary can obtain the keys for input wire adaptively. We deal with this issues using a technique from [GKR08]. We refer the reader to Appendix B.2 for details.

Our setting. Now we are ready to describe our setting. After we describe our setting we will be ready to describe our real-world adversary $\mathcal{A}$. Consider a setting of two parties S and R executing $\ell(k)$ invocation of π. S plays the role of the sender in all these executions and on the other hand R plays the role of the receiver. Since we are in the *static* input setting we need to specify the inputs of all parties. More specifically we specify the distribution $\mathcal{D}$ according to which the inputs of the parties S and R are sampled.

– Let $m_0, m_1 \leftarrow \{0,1\}^{p(k)}, b \leftarrow \{0,1\}, r \leftarrow \{0,1\}^*, w \leftarrow \{0,1\}^k, (GC, \mathbf{Z}) = \mathtt{Yao}_1(\mathcal{F}[b, m_b, w, r])$. Let $\mu(k)$ be the number of input wires in the garbled circuit GC. Then $\ell(k) = \mu(k) + 1$. Also $\mathbf{Z}$ contains two garbled circuit keys corresponding to each wire. More specifically it consists of values $\binom{Z_{1,0}, Z_{2,0} \ldots Z_{\mu(k),0}}{Z_{1,1}, Z_{2,1} \ldots Z_{\mu(k),1}}$ where $Z_{i,0}, Z_{i,1}$ correspond to the keys for the ith input wire. We will distinguish the sessions into two categories. We will refer to one of the sessions as the *main session*. Rest of the sessions are referred to as *additional sessions*.

– **Input to Honest Sender:** Let (m_0, m_1) be the input of S in the main session. For each $i \in [\mu(k)]$, let $(Z_{i,0}, Z_{i,1})$ be the input of the honest sender in the ith additional session.

– **Input to Receiver:** Let $\bar{y}$ be a vector of $\ell(k)$ bits all chosen randomly. Set $\bar{y}$ as the input of the receiver.

Description of our real-world adversary $\mathcal{A}$. The real-world adversary corrupts the receiver R and receives the garbled circuit GC generated in the above described sampling procedure (distribution $\mathcal{D}$) as auxiliary input. Our adversary on input the garbled circuit GC proceeds as follows. It ignores the message $\mathcal{D}$ generates for the honest receiver. Let the messages that S sends to R in the main session be $M_1, M_2, \ldots, M_n$ (recall that n is the number of messages S sends to R in the protocol π). Upon receiving M_i where $i \in [n]$ from S, the adversary obtains its response to be sent in the main session by evaluating the garbled circuit GC on input $M_1, M_2, \ldots M_i$. Let B denote the concatenation of $M_1, M_2, \ldots M_i$. In order to achieve this, the adversary needs the keys Z_{j,B_j} where $j \in |B|$. Note that among these some of the keys have previously already been received. R obtains the ones that have not been obtained previously by initiating multiple, concurrent, OT protocols to which, by construction, the sender provides $(Z_{i,0}, Z_{i,1}), 1 \leq i \leq \mu(k)$ as inputs. On obtaining these keys, $\mathcal{A}$ invokes $\mathtt{Yao}_2$, and computes the output. For every $i \in [n-1]$ it responds to S in the main session using the obtained output. Finally for $i = n$ it outputs the obtained value as its output.

Next we claim that the real-world adversary described above always (except with negligible probability) succeeds in outputting the value w generated by the sampling procedure $\mathcal{D}$.

Adversarial behavior in the ideal world. Recall that we started by assuming that the protocol π is concurrently secure. Therefore there exists an ideal-world adversary, $\mathcal{S}$ that outputs a distribution that is computationally indistinguishable from the one that our real-world adversary described above generates. Let us recall that $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{OT}$ in a main session and a sequence of additional sessions. The additional sessions however are used just to obtain garbled circuit keys. For notational convenience let $\mathcal{K}$ denote the oracle that is used to obtain garbled circuit keys. More specifically, $\mathcal{S}$ obtains the garbled circuit keys from the key oracle $\mathcal{K}$ rather than the ideal functionality $\mathcal{F}_{OT}$. Next we will convert the simulator $\mathcal{S}$ into an algorithm M that interacts with the $\mathcal{F}_{OT}$ ideal functionality corresponding to the main session only. However, unlike $\mathcal{S}$, M does not obtain a garbled circuit and it does not query the key oracle $\mathcal{K}$. Instead it interacts with an ideal functionality $\mathcal{F}[b, m_b, w, r]$.

Removing the garbled circuit. According to the security definition of garbled circuits (Definition 5), there exists a simulator $\texttt{YaoSim}_{\mathcal{S}}$ which interacts with $\mathcal{S}$ in a black-box manner and the ideal functionality $\mathcal{F}[b, m_b, w, r]$ whose outputs are computationally indistinguishable from that of $\mathcal{S}$. Algorithm M is described as follow: M is essentially the algorithm $\texttt{YaoSim}_{\mathcal{S}}$. It internally executes the simulator $\texttt{YaoSim}_{\mathcal{S}}$. Whenever a query is made by $\texttt{YaoSim}_{\mathcal{S}}$ to the ideal functionality $\mathcal{F}[b, m_b, w, r]$, M performs it and also whenever a query is made to $\mathcal{F}_{OT}$, M performs that too.

Note that the machine M just constructed interacts with $\mathcal{F}[b, m_b, w, r]$ and with $\mathcal{F}_{OT}$ for the main session (in which the honest sender talking to $\mathcal{F}_{OT}$ has input (m_0, m_1)) and is such that its output and the output or the real-world adversary are computationally indistinguishable. This in particular means that the output of M always (except with negligible probability) includes the value w. However, in order to be able to obtain w from $\mathcal{F}[b, m_b, w, r]$, M must query $\mathcal{F}_{OT}$ (for the main session) with the bit b. Otherwise, M will fail to output w with probability at least $\frac{1}{2}$. This follows information theoretically as M has no access to m_b and w which are strings chosen uniformly at random unless it queries $\mathcal{F}_{OT}$ with b. In particular this means that M can be used to extract the choice bit b of an external honest receiver R. This is a contradiction based on the stand-alone security of the receiver.

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 π 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 π 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 π' for π 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 π with input x and a honest receiver D of π' 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 π' we have*

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

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

[9] Note that there are two levels of simulation happening here. First, any adversary $\mathcal{A}_1$ who runs π 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).

a specialized garbled circuit (see Appendix C 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 next prove this theorem.

Proof of Theorem 2 As pointed out earlier this theorem is a generalization of Theorem 1 with some additional technicalities. For the sake of completeness (at the cost of some repetition) we will reprove this theorem from scratch.

We start by recalling and building some notation that we will use in our proof. Next we will consider specifics of our setting. This in particular will include the details on the specifications of inputs of all parties. Then we will define formally the strategy of the real world adversary. Finally, we will argue that the output of this real world adversary can not be computationally indistinguishable from the output of any ideal world adversary.

Notation. Let π be a two party protocol between a sender S with input x and a receiver R with input y. For the sake of contradiction lets assume that π concurrently securely realizes the functionality $\mathcal{F}$ and also assume that both (1) and (2) from the statement of Theorem 2 hold. Let π' be the chosen protocol (as required by the theorem) between a sender S with input x' and a receiver R with input y'. Without loss of generality we assume that S sends the first message in the protocol π' and R sends the last message in the protocol π'. Further assume that R sends exactly n messages in the execution of this protocol π'.

Now, consider the next message function $F(y', r, M_1, M_2, \ldots, M_i)$ where $i \in [n]$ of R for the protocol π'. More specifically, $F_i(y', rM_1, M_2, \ldots, M_i)$ generates the ith message that an honest R on input y' and random coins r would generate corresponding to the execution in which $M_1, M_2, \ldots, M_i$ are sent to R in the execution of π'. Let $\mathcal{F}[y', r]$ be a reactive functionality parameterized by y', r that can be invoked n times. The ith invocation of $\mathcal{F}[y', r]$ expects an input M_i and outputs the ith message that R would would have sent in π'. The functionality has the values y', r which are built into the functionality itself.

In this proof we will use a garbled circuit construction with the special feature (see Appendix C for more details) that the keys in this construction consist of bits rather than strings. Roughly speaking $\texttt{Yao}_1$ is an algorithm that takes the description of a reactive functionality as input and outputs a garbled circuit and the associated keys. $\texttt{Yao}_2$ on the other hand takes as input the garbled circuit and multiple key bits corresponding to each input wire (that allow it to evaluate the key corresponding to any input wire) and outputs an evaluation of the garbled circuit.[10]

Our setting. Now we are ready to describe our setting. After we describe our setting we will be ready to describe our real-world adversary $\mathcal{A}$. Consider a setting of two parties S and R executing $\ell(k)$ invocation of π. S plays the role of the sender in all these executions and on the other hand R plays the role of the receiver. Since we are in the *static* input setting we need to specify the inputs of all parties. More specifically we specify the distribution $\mathcal{D}$ according to which the inputs of the parties S and R are sampled.

- Let $(x_{main}, y_{main}) \leftarrow \mathcal{D}_2$ ($\mathcal{D}_2$ is a distribution as required by the theorem), $(GC, \mathbf{Z}) = \texttt{Yao}_1(\mathcal{F}[y_{main}, r])$. Specifically $\mathbf{Z}$ is the value $\binom{Z_{1,0}, Z_{2,0}, \ldots, Z_{\mu(k),0}}{Z_{1,1}, Z_{2,1}, \ldots, Z_{\mu(k),1}}$ where $Z_{i,0}, Z_{i,1}$ are needed for the evaluation of the garbled circuit GC. Let $\mu(k)$ be the number of pairs of pairs of bits in $\mathbf{Z}$. Then $\ell(k) = \mu(k)\nu(k) + 1$ ($\nu(k)$ is specified by the theorem). We will distinguish the sessions into two categories. We will refer to one of the sessions as the *main session*. The remaining sessions are referred to as *additional sessions*.
- **Input to Honest Sender:** Let x_{main} be the input of S in the main session. For each $i \in \mu(k)$, let $\left(\binom{x_{1,i,0}, x_{2,i,0}, \ldots, x_{\nu(k),i,0}}{x_{1,i,1}, x_{2,i,1}, \ldots, x_{\nu(k),i,1}}, z_i\right) \leftarrow \mathcal{D}_1(Z_{i,0}, Z_{i,1})$ ($\mathcal{D}_1$ is a distribution as required by the theorem). Finally, for every $i \in [\mu(k)], j \in [\nu(k)]$, let $(x_{j,i,0}, x_{j,i,1})$ be the input of the honest sender S in the $(i-1) \cdot \nu(k) + j^{th}$ additional session.

[10] Just like in the setting of string OT we use the technique from [GKR08] to deal with issues of adaptivity. We refer the reader to Appendix B.2 for details. Additionally this construction of garbled circuits (see Appendix C for more details) allows us to "encode" the keys for the garbled circuit in a way such that the keys now correspond to bits rather than strings. We use the encoding of [BCS96].

- **Input to Receiver:** Let $\bar{y}$ be a vector of $\ell(k)$ inputs of R all chosen randomly (from the appropriate distribution). Set $\bar{y}$ as the input of the receiver.

Description of our real-world adversary $\mathcal{A}$. The real world adversary $\mathcal{A}$ corrupts the receiver R and receives the garbled circuit GC and z_i for all $i \in \mu(k)$ generated in the above described sampling procedure (distribution $\mathcal{D}$) as auxiliary input. $\mathcal{A}$ ignores the input that $\mathcal{D}$ generates for the honest receiver and proceeds as follows. Our adversary $\mathcal{A}$ internally executes $\mathcal{A}_2$ (required by the theorem). Looking ahead we will reach a contradiction by constructing a simulator for the adversary $\mathcal{A}_2$. The adversary $\mathcal{A}_2$ expects to interact with an honest sender of π and an honest receiver of the protocol π'. The adversary $\mathcal{A}$ simulates the honest sender of π (for $\mathcal{A}_2$) using the messages it obtains from the external honest party S. On the other hand in order to simulate the honest receiver of the protocol π', our adversary $\mathcal{A}'$ uses the garbled circuit GC as follows. Let the messages that $\mathcal{A}_2$ sends for R in the execution of π' be $M_1, M_2, \ldots, M_n$ (recall that n is the number of messages R sends to S in the protocol π'). Upon receiving M_i where $i \in [n]$ from S, the adversary obtains its response to be sent in the main session by evaluating the garbled circuit GC on input M_i. In order to achieve this, it needs multiple values $Z_{i,b}$ for an $i \in [\mu(k) \cdot \nu(k)]$ and a bit b of its choice. Note that the adversary $\mathcal{A}$ will need multiple bits for this evaluation. We will explain how it can obtain $Z_{i,b}$. $\mathcal{A}$ can obtain all other inputs in an analogous manner. In order to obtain $Z_{i,b}$, $\mathcal{A}$ starts an execution of $\mathcal{A}_1$ with input (b, z_i). By assumption (1) from the theorem we have that $\mathcal{A}_1$ can always obtain $Z_{i,b}$. In doing so $\mathcal{A}_1$ executes $\nu(k)$ execution of π. In particular it will execute π for the sessions $\{(i-1) \cdot \nu(k) + 1, (i-1) \cdot \nu(k) + 2, \ldots, i \cdot \nu(k)\}$. On obtaining these keys, $\mathcal{A}$ invokes $\mathtt{Yao}_2$ (note that $\mathtt{Yao}_2$ first decodes the actual keys for the input wires of the garbled circuit from the obtained key bits) and computes the output. This allows $\mathcal{A}$ to successfully (except with negligible probability) execute $\mathcal{A}_2$. Our adversary outputs the output generated by $\mathcal{A}_2$.

Adversarial behavior in the ideal world. Recall that we started by assuming that the protocol π is concurrently secure. Therefore there exists an ideal world adversary, $\mathcal{S}$ that outputs a distribution that is computationally indistinguishable from the one that our real world adversary $\mathcal{A}$ described above generates. Let us recall that $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}$ in the main session and a sequence of additional sessions. The additional sessions however are used just to obtain bits that are in turn used to evaluate garbled circuit keys. For notational convenience let $\mathcal{K}$ denote the oracle that is used to provide these bits. More specifically, $\mathcal{S}$ obtains the key bits from the key oracle $\mathcal{K}$ rather than the ideal functionality $\mathcal{F}$. Next we will convert this simulator $\mathcal{S}$ into an algorithm M that interacts with the ideal functionality $\mathcal{F}$ corresponding to the main session only. However, unlike $\mathcal{S}$ it does not obtain a garbled circuit and it does not query the key oracle $\mathcal{K}$. Instead it interacts with an ideal functionality $\mathcal{F}[y_{main}, r]$.

Removing the garbled circuit. Consider an algorithm M that internally executes the simulator $\mathcal{S}$. According to the security definition of garbled circuits (Definition 5), there exists a simulator $\mathtt{YaoSim}_{\mathcal{S}}$ which interacts with $\mathcal{S}$ in a black-box manner and the ideal function $\mathcal{F}[y_{main}, r]$ and outputs a computationally indistinguishable distribution. Note that $\mathcal{F}[y_{main}, r]$ corresponds exactly to an honest receiver of the protocol π' with input y_{main}. Additionally note that $\mathtt{YaoSim}_{\mathcal{S}}$ internally executes the simulator of the OT protocol of (Appendix C) in order to simulate the bit queries of $\mathcal{S}$ using one key query (corresponding to an input wire of the circuit) only.

Note that the machine M just constructed interacts with with $\mathcal{F}$ for the main session (where the honest party S has input x_{main}) and an honest receiver of the protocol π' with input y_{main}. Further more the output of M is computationally indistinguishable from the output generated by real world adversary $\mathcal{A}_2$. This contradicts (2) from the theorem.

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 $\mathcal{F}_{\textsf{Bit-OT}} : (\{0,1\} \times \{0,1\}) \times \to \{0,1\}$ where $\mathcal{F}_{\textsf{Bit-OT}}((m_0, m_1), b) = m_b$ and only R gets the output. We stress that we are in the setting of *static inputs* and *fixed roles.*

Lemma 1. *(impossibility of static input concurrency - bit OT) Let π be any protocol which implements the $\mathcal{F}_{\textsf{Bit-OT}}$ 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 (Definition 1) 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 π. 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 π playing as the receiver.

Next note that the chosen protocol attack for the case of the 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 further details. ■

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 [Kil00,Kil88,BMM99,GMW87].

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[11] 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,[12] as defined below, we show that there does not exist a protocol π 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* [Kil91] (both in the setting of *semi-honest* and *malicious* adversaries) in the stand-alone setting of information-theoretically secure computation [13] 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 π 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 (Definition 1) of concurrent security* can not *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 can not change. 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:

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

[12] Recall that a functionality is said to be complete if it can be used to securely realize any other functionality.

[13] 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.

Corollary 2 *Let $\mathcal{F}_{sym}$ be a functionality that is complete in the stand-alone setting and π be any protocol which implements $\mathcal{F}_{sym}$. Let π 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) π is at least $\frac{m}{k^c}$-bits.*

Proof of Theorem 3 Since $\mathcal{F}_{sym}$ is complete, therefore $\exists a_0, a_1, b_0, b_1$ such that $f(a_0, b_0) = f(a_0, b_1) = f(a_1, b_0) \neq f(a_1, b_1)$. For the purposes of simplifying the rest of the proof, without loss of generality, we denote a_0 as 0, a_1 as 1, b_0 as 0, b_1 as 1, $f(a_0, b_0) = f(a_0, b_1) = f(a_1, b_0)$ as 0 and $f(a_1, b_1)$ as 1.

Theorem 3 follows by an application of Theorem 2. Observe that Theorem 2 requires us to prove two properties, namely, Key Transmission and Chosen Protocol Attack. Next we prove both of them.

Key Transmission. 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 $\mathcal{A}_1$. We start by giving the distribution $\mathcal{D}_1$ that on input X_0, X_1 generates (u_1, u_2, u_3, u_4) (as input for the honest party S for 4 sessions) and auxiliary input z for the adversary $\mathcal{A}_1$. In this case z is going to be the empty string. Sample u_1 and u_3 randomly in $\{0, 1\}$. Let $u_2 = X_0 \oplus u_1$ and $u_4 = X_1 \oplus u_3$.

Our adversary $\mathcal{A}_1$ on input b proceeds by executing π, 4 times with inputs $(1-b, 1-b, b, b)$. Let the outputs be o_1, o_2, o_3, o_4. Finally it outputs $o_1 \oplus o_2$ if $b = 0$ and $o_3 \oplus o_4$ if $b = 1$. Observe that $\mathcal{A}_1$ on input b (playing the role of the receiver in four executions of π) will always output X_b.

Now we need to argue that every ideal-world adversary $\mathcal{S}_1$ given b, z can be simulated by querying (for only one of the values) an oracle $\mathcal{O}$ that holds X_0 and X_1 except with negligible probability given that

$$\text{EXEC}_{\mathcal{F},\mathcal{S}_1}(k, \bar{x}, z) \overset{c}{\equiv} \text{EXEC}_{\pi,\mathcal{A}_1}(k, \bar{x}, z). \tag{1}$$

Clearly the view of ideal-world adversary $\mathcal{S}_1$ can always be simulated by just one query to the oracle $\mathcal{O}$ unless $\mathcal{S}_1$ queries with the input 1 for all four sessions. In fact, even if $\mathcal{S}_1$ decides to use any other input apart from b_0, b_1, he would learn only as much information about a_0, a_1 as using b_1 (i.e., 1). However we next argue that if this is the case, then the ideal-world view of $\mathcal{S}_1$ and the real-world view of $\mathcal{A}_1$ can clearly be distinguished.

Consider an ideal-world adversary that queries with inputs 1 in all sessions with a non-negligible probability. Then the honest sender A obtains the output 1 both in session 1 and in session 3 with a probability of $\frac{1}{4}$ (over the random choices of u_1, u_3). In particular this happens when both u_1 and u_3 are chosen to be 1. On the other hand in the interaction with the real adversary $\mathcal{A}_1$ the output of the honest party is always 0 either in session 1 or in session 3.

Chosen Protocol Attack. Let a sender A with input bit x and receiver B with input bit y be two parties interacting via π to evaluate the functionality $\mathcal{F}_{sym}$. For the sake of contradiction let us assume that π securely realizes $\mathcal{F}_{sym}$ in the concurrent setting. Let π' be the following protocol between C and D with inputs x' and $(y', \tilde{m}, w)$ respectively. First C, D execute the same protocol as π with C as sender and D as receiver with inputs x' and y' respectively. Upon the completion of the protocol π, D checks if the output m he received is equal to the input $\tilde{m}$ and sends w to C if this is the case. He sends $\perp$ otherwise.

The chosen protocol attack in the symmetric case proceeds as follows. Consider an adversary $\mathcal{A}_2$ who corrupts the two players B, C. We will now describe a concurrent scheduling of executions of π and π' and *static* input distribution $\mathcal{D}_2$ for A and D such that an ideal-execution of π, π' is distinguishable from a real execution of π, π'. Let the distribution $\mathcal{D}_2$ be such that it generates a random bit x for an honest sender A of π and a random bit y for an honest receiver D of π'. Also $\tilde{m} = f(x, y)$ and $w \leftarrow \{0, 1\}^k$ is given to D. Note that C and D execute π' using the identities of A, B respectively.

Message Scheduling by $\mathcal{A}_2$. The adversary $\mathcal{A}_2$ schedules messages as follows. It forwards messages that it receives from A to D (note that D acts the receiver of π in π') and the messages it receives from D to A. Observe that $\mathcal{A}_2$ will be successful in completing the protocol π between A and D. Finally, since the output that D obtains will be $f(x, y)$, $\mathcal{A}_2$ will obtain w from D. Recall also that the honest party A, also receives an output and outputs that value. Consider any simulator $\mathcal{S}_2$ that interacts in a hybrid execution with A via $\mathcal{F}_{sym}$ instead of π and also executes π' with D. Let y' be the input that $\mathcal{S}$ sends to $\mathcal{F}_{sym}$. Let x and $y, \tilde{m}, w$ be the inputs of the parties A and D respectively as described above. Now observe that the real-world adversary $\mathcal{A}_2$ always outputs w. There are two cases:

1. If $y \neq y'$ with a non-negligible probability then the output of the honest party A in the real world will be non-negligibly far from the output in the ideal world (over the choices of x).
2. On the other hand if y is always (except with negligible probability) equal to y' then this means that the adversary extracts the input y and also outputs w. Recall that w is generated only when D obtains an output $f(x, y)$ that is equal to $\tilde{m}$. Now note that this output is 0 with a constant probability (over choice of x, y). Finally conditioned on the fact that D outputs 0, $\mathcal{S}_2$'s ability to extract D's input contradicts the stand-alone security of π for the honest receiver D.

Now by invoking Theorem 2, we see that no concurrent secure protocol exists for the function $\mathcal{F}_{sym}$.

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 at the same time S should not learn anything. More formally, let $f : X \times Y \to Z$ be any finite function[14] then an asymmetric functionality $\mathcal{F}_{asym}$ is defined as, $\mathcal{F}_{asym}(x, y) = (\perp, f(x, y))$ where S gets no output and R gets $f(x, y)$. We show that there does not exist a protocol π that concurrently securely realizes any *complete* $\mathcal{F}_{asym}$ functionality as defined below.

We know that $\mathcal{F}_{asym}$ is *complete* [Kil00][15] 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[16]) Let π be any protocol which implements any $\mathcal{F}_{asym}$ functionality that is complete in the stand-alone setting. 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 definition (Definition 1) of concurrent security,* can not *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) can not change. However, complete asymmetric functionalities possess a larger structure than what is available in complete symmetric functionalities. We crucially use this additional structure in our proof. We also note that a direct analogue of Corollary 2 holds in the asymmetric setting as well.

Proof of Theorem 4 Since $\mathcal{F}_{asym}$ is complete, $\forall b_0, \exists b_1, a_0, a_1$ such that $f(a_0, b_0) = f(a_1, b_0)$ & $f(a_0, b_1) \neq f(a_1, b_1)$. This in particular implies [17] that there exists a_0, a_1, a_2 and b_0, b_1 (b_0, b_1 are chosen to be such that that $\forall i \in \{0, 1\}, \forall y \in Y, y \neq b_i, \exists y^{x_0}, y^{x_1}$ such that $f(y^{x_0}, y) = f(y^{x_1}, y) \& f(y^{x_0}, b_i) \neq f(y^{x_1}, b_i)$)[18] such that the truth table of $\mathcal{F}_{asym}$ look as follows:

	b_0	b_1
a_0	A	C
a_1	D	B
a_2	A	B

where $A \neq D$ and $C \neq B$.

Theorem 4 follows by an application of Theorem 2. Observe that Theorem 2 requires us to prove two properties, namely, Key Transmission and Chosen Protocol Attack. Next we prove both of them:

[14] Recall that a function is said to be finite if both the domain and the range are of finite size.

[15] Recall that a functionality is said to be complete if it can be used to securely realize any other functionality.

[16] We thank Vipul Goyal and Amit Sahai for pointing us to an issue in the proof of this theorem (in an earlier draft of the paper) and a fix to it.

[17] We refer the reader to Figure 5 of [KMQ11] for more details.

[18] It can be observed that such b_0, b_1 can always be found if $\mathcal{F}_{asym}$ is complete.

Key Transmission. 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 $\mathcal{A}_1$. We start by giving the distribution $\mathcal{D}_1$ that on input $X_0, X_1 \in \{0,1\}$ generates $(u_1, u_2 \dots u_{2k})$ (a sequence of inputs for the honest party S) and auxiliary input (z_0, z_1) for the adversary $\mathcal{A}_1$.

Description of $\mathcal{D}_1$*.* For each $i \in [2k]$ sample u_i randomly in X (recall that X is the domain of f). Then $(u_1, u_2 \dots u_{2k})$ will serve as the sequence of inputs for the honest party S for $2k$ session.
Next we describe how the auxiliary input for the adversary $\mathcal{A}_1$ is defined but in order to do that we need to define some notation. Fix any prime $p \geq |Z|$. Define two injective functions $N_0, N_1 : Z \to \mathbb{Z}_p$ as follows. Set $N_0(A) = 0, N_0(D) = 1, N_1(C) = 1, N_1(B) = 0$ and extend them to be any arbitrary injective functions: $Z \to \mathbb{Z}_p$. Observe that this can always be done since $p \geq |Z|$. Additionally observe that by design N_0 is such that $N_0(f(a_0, b_0))$ and $N_0(f(a_1, b_0))$ are 0 and 1 respectively. On the other hand N_1 is such that $N_1(f(a_0, b_1))$ and $N_1(f(a_1, b_1))$ are 1 and 0 respectively. Now define *masks* m_i^0 and m_i^1 as follows:

$$m_i^0 = N_0(f(u_i, b_0)) \cdot N_0(f(u_{i+k}, b_0))$$

and

$$m_i^1 = N_1(f(u_i, b_1)) \cdot N_1(f(u_{i+k}, b_1)).$$

Finally set the auxiliary inputs for the adversary as (z_0, z_1) where $z_0 = X_0 + \sum m_i^0 \mod p$ and $z_1 = X_1 + \sum m_i^1 \mod p$.

Description of $\mathcal{A}_1$*.* Our adversary $\mathcal{A}_1$, on input c and auxiliary input (z_0, z_1) proceeds as follows. It proceeds by executing π, $\ell(k) = 2k$ times with the input b_c in all the sessions and obtains $f(u_i, b_c)$ for every $i \in [2k]$. It then computes m_i^c for each $i \in [k]$ as defined above. Finally it outputs $z_c - \sum m_i^c \mod p$. Observe that $\mathcal{A}_1$ playing the role of R on input c will always be able to output X_c.

Simulating the idea world simulator $\mathcal{S}_1$ *for* $\mathcal{A}_1$*.* Now we need to argue that every ideal-world adversary $\mathcal{S}_1$ (simulating $\mathcal{A}_1$), given c and (z_0, z_1) as inputs learns only one of the two values X_0, X_1 regardless of its inputs in the $2k$ sessions for which it queries the ideal functionality. That is, $\mathcal{S}_1$ can be simulated by a machine M[19] that queries (for only one of the values) an oracle $\mathcal{O}$ that holds X_0 and X_1 except with negligible probability given that

$$\text{EXEC}_{\mathcal{F},\mathcal{S}_1}(k, \bar{x}, z) \overset{c}{\equiv} \text{EXEC}_{\pi,\mathcal{A}_1}(k, \bar{x}, z)$$

Notation used in description of M*.* We start by setting up some conventions and defining some notation. Note that $\mathcal{S}_1$ can make arbitrary queries to the ideal functionality for input values $(v_1^*, v_2^*, \dots v_{2k}^*)$ of its choice in any arbitrary order and M must simulate that. Without loss of generality we restrict ourselves to simulators $\mathcal{S}_1$ such that $\forall i \in [k]$, $\mathcal{S}_1$ queries the ideal functionality with input v_i^* before it queries the ideal functionality with v_{i+k}^*. Secondly we require that when $\mathcal{S}_1$ makes the query v_i^* (for $i \in [k]$) it should have already made the queries v_j^* for all $j \in [i-1]$. Note that any general simulator can be reduced to a simulator that satisfies the above constraints by renaming the indices of the queries made.
We will refer to the point where the simulator makes the query $v_{\mu k}^*$ as the *threshold point* where $0 < \mu < 1$ is a specific constant (determined by $|Z|$ – recall that $|Z|$ is finite – and to be determined later). Looking ahead M behaves in one way before the threshold point is reached and in another way after that. Additionally, note that when the threshold point is reached there are at least $2k - 2\mu k$ sessions for which $\mathcal{S}_1$ has not queried yet. Intuitively speaking this will give M the flexibility in simulation.

Hidden Bit and its security properties. We will use the notion of hidden bit in our description of M. Now we describe this notion. We will then argue some properties about this notion which will be used in the proof of indistinguishability that follows. Corresponding to each pair of masks m_i^0 and m_i^1 where $i \in [k]$, we define a bit $c(i)$ called the *hidden bit* as follows. Very roughly, $c(i)$ defines the mask among m_i^0 and m_i^1 that is hidden from the adversary. More specifically, m_i^0 is hidden if $c(i) = 0$ and on the other hand m_i^1

[19] Note that here our machine M is simulating the ideal-world adversary (or the simulator) itself. In particular it simulates the queries the simulator makes to the ideal functionality with the auxiliary input of $\mathcal{S}_1$ being (z_0, z_1).

is hidden if $c(i) = 1$. Now we make this notion formal. Recall that by the completeness of f, for $v = b_1$ there exists two inputs v^{x_0}, v^{x_1} such that $f(v^{x_0}, b_0) \neq f(v^{x_1}, b_0)$ but $f(v^{x_0}, v) = f(v^{x_1}, v)$. Secondly $\forall v \in Y \backslash \{b_1\}$ there exists two inputs v^{x_0}, v^{x_1} such that $f(v^{x_0}, b_1) \neq f(v^{x_1}, b_1)$ but $f(v^{x_0}, v) = f(v^{x_1}, v)$. Then we let,

$$c(i) = \begin{cases} 0 & \text{if } v_i^* = b_1 \text{ and } u_i \in \{v_i^{*x_0}, v_i^{*x_1}\} \text{ and } u_{k+i} = a_1 \\ 1 & \text{if } v_i^* \neq b_1 \text{ and } u_i \in \{v_i^{*x_0}, v_i^{*x_1}\} \text{ and } u_{k+i} = a_0 \\ \perp & \text{otherwise.} \end{cases} \tag{2}$$

The reader should keep in mind that the hidden bit (as defined above) does not depend on the choice of the value v_{k+i}^* that the adversary makes. Now we give some intuition behind this function. We give intuition for the case when $v_i^* = b_1$. The other case is analogous. Observe that in this case whenever u_{k+i} is such that $u_{k+i} = a_1$, we have that $N_1(f(u_{k+i}, b_1)) = 0$ and $N_0(f(u_{k+i}, b_0)) \neq 0$. This ensures that $m_i^1 = 0$. Furthermore whenever $u_i \in \{{b_1}^{x_0}, {b_1}^{x_1}\}$, we can expect to have some "uncertainty" (defined precisely in the sequel) in m_i^0. Next we state and prove properties about the hidden bit that we use in our proof more formally.

Lemma 2. *$\forall i \in [k], v_i^* \in Y$, the hidden bit $c(i) \in \{0, 1\}$ (over the random choices of u_i, u_{k+i}) with probability at least $\frac{2}{|X|^2}$.*

Proof. u_i is chosen randomly in X, therefore $u_i \in \{v_i^{*x_0}, v_i^{*x_1}\}$ with probability $\frac{2}{|X|}$. Secondly, in the case when $v_i^* = b_1$ we have that $u_{k+i} = a_1$ with probability $\frac{1}{|X|}$. Similarly in the case when $v_i^* \neq b_1$, we have that $u_{k+i} = a_0$ with probability $\frac{1}{|X|}$. Since, u_i and u_{k+i} are sampled independently we obtain the desired probability as $\frac{1}{|X|} \cdot \frac{2}{|X|}$ which evaluates to $\frac{2}{|X|^2}$.

Lemma 3. *$\forall i \in [k], v_i^* \in Y$, $\exists$ a constant $\tau > 0$, such that $\forall z \in \mathbb{Z}_p, \Pr[m_i^{c(i)} = z | c(i) \in \{0, 1\}] < 1 - \tau$, where the probability is taken over the random choices of u_i, u_{k+i}.*

Proof. Consider the case when v_i^* is such that that $v_i^* = b_1$ (the other case when $v_i^* \neq b_1$ is symmetric). Since we are given that $c(i) \in \{0, 1\}$ and because we are in the case that $v_i^* = b_1$, by observation, it follows that $c(i) = 0$. This in particular implies that $u_{k+i} = a_1$. This implies that $N_1(f(u_{k+i}, b_1)) = 0$, $m_i^1 = 0$ and $N_0(f(u_{k+i}, b_0)) \neq 0$. Then we have that $g = N_0(f(v_i^{*x_0}, b_0))$ and $h = N_0(f(v_i^{*x_1}, b_0))$ are such that $g \neq h$. Additionally we have that $\Pr[m_i^{c(i)} = g | (c(i) \in \{0, 1\}) \wedge (v_i^* = b_1)] = \frac{1}{2}$ and similarly $\Pr[m_i^{c(i)} = h | (c(i) \in \{0, 1\}) \wedge (v_i^* = b_1)] = \frac{1}{2}$. Since $m_i^{c(i)}$ could take two different values in $\mathbb{Z}_p$, we have that with τ set to $\frac{1}{4}$ the lemma holds. Note that the proof in the other case in which $v_i^* \neq b_1$ follows in an analogous manner.

Description of M. M proceeds as follows:

1. M starts by setting (z_0, z_1) to random values in $\mathbb{Z}_p$ and executing $\mathcal{S}_1$ on this input.
2. *Before Threshold Point is reached*: M responds to every query v_i^* for $i \in [2k]$ of $\mathcal{S}_1$ (for the ideal functionality $\mathcal{F}_{asym}$) by first sampling u_i randomly in $|X|$ and responding with $f(u_i, v_i^*)$.
3. *When Threshold Point is reached*: Now M proceeds by computing $c = \mathsf{majority}\{c(1), c(2), \ldots c(\mu k)\}$ (breaking ties arbitrarily and ignoring the $\perp$ values). M obtains $X_{\bar{c}}$ from $\mathcal{O}$ and randomly samples all the values of u_i, that have not been sampled so far, subject to the condition $\sum_{i=1}^k m_i^{\bar{c}} = z_{\bar{c}} - X_{\bar{c}}$. This sampling is performed via rejection sampling. More specifically, M simply chooses the remaining values of u_i at random as long as they are suitable and if the constraint is not satisfied it rejects[20] and re-samples.
4. M simply responds to any remaining queries from $\mathcal{S}_1$ by using the values of u_i sampled in the previous step.

[20] Observe that the M runs in expected polynomial time as a randomly sampled values of u_i will be suitable with probability at least $\frac{1}{|X|}$.

Indistinguishability. Next we will argue by a sequence of hybrids that the view of $\mathcal{S}_1$ when interacting with the ideal functionality in the real experiment is indistinguishable from the view of $\mathcal{S}_1$ when interacting with M that has access to $\mathcal{O}$. Informally, our main tool is going to be the presence of sufficient entropy in the masks $m_i^b, b \in \{0,1\}, i \in [k]$. We start by stating a lemma about independent random variables in $\mathbb{Z}_p$. Roughly speaking, suppose we have k independent random variables over $\mathbb{Z}_p$ such that each variable has some uncertainty, i.e., each variable does not take any fixed value. Then, the sum of sufficiently many such random variables is close to uniform. Formally,

Lemma 4. *Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be k independent random variables such that $\exists \tau > 0$ such that, for all i, for all $z \in \mathbb{Z}_p$, $\Pr[\alpha_i = z] < 1 - \tau$. Then $\alpha = \sum_{i=1}^{k} \alpha_i \mod p$ is a uniformly distributed in $\mathbb{Z}_p$ except with negligible probability (in k).*

Proof. The proof of the lemma, a generalization of the XOR lemma (see Lemma 1.1 from Goldreich [Gol11]) may be obtained by an application of Lemma 4.2 in Page 10 of [Rao07]. A full proof may be found in Appendix D.

Now we give our hybrids.

- `Hybrid`$_0$: This hybrid corresponds to the honest simulation of $\mathcal{S}_1$. This involves generating honest sender inputs values $u_1, \ldots u_{2k}$ and the auxiliary information for $\mathcal{S}_1$, z_0, z_1 according to the distribution $\mathcal{D}_1(X_0, X_1)$. At this point $\mathcal{S}_1$ can make arbitrary queries to the ideal functionality for input values $(v_1^*, v_2^*, \ldots v_{2k}^*)$ of its choice in any arbitrary order and we answer these queries using honest sender inputs values previously generated. Recall that without loss of generality we restricted ourselves to $\mathcal{S}_1$ such that $\forall i \in [k]$, $\mathcal{S}_1$ queries the ideal functionality with input v_i^* before it queries the ideal functionality with v_{i+k}^*. Secondly we require that when $\mathcal{S}_1$ makes the query v_i^* (for $i \in [k]$) it should have already made the queries v_j^* for all $j \in [i-1]$.
- `Hybrid`$_1$: In this hybrid we change how the sampling operations are performed.
  1. M starts by setting (z_0, z_1) to random values in $\mathbb{Z}_p$ and executing $\mathcal{S}_1$ on this input.
  2. *Before Threshold Point is reached*: We respond to every query v_i^* for $i \in [2k]$ of $\mathcal{S}_1$ (for the ideal functionality $\mathcal{F}_{asym}$) by first sampling u_i randomly in $|X|$ and responding with $f(u_i, v_i^*)$.
  3. *When Threshold Point is reached*: Randomly sample u_is, that have not been sampled so far, subject to the condition $\sum_{i=1}^{k} m_i^0 = z_0 - X_0$ and $\sum_{i=1}^{k} m_i^1 = z_1 - X_1$. This sampling is performed via rejection sampling. More specifically, M simply chooses the remaining u_is at random as long as they are suitable and if the constraints are not satisfied it rejects and re-samples.
  4. M simply responds to any remaining queries from $\mathcal{S}_1$ by using the values of u_i sampled in the previous step.

  Indistinguishability of `Hybrid`$_0$ *and* `Hybrid`$_1$. It suffices to prove that joint distribution of $\sum_{i=\mu k+1}^{k} m_i^0$ and $\sum_{i=\mu k+1}^{k} m_i^1$ is close to uniform.

  We start by observing that over the random choice of u_i, u_{k+i} for $i \in \{\mu k+1, \ldots, k\}$, $\sum_{i=\mu k+1}^{k} m_i^1$ is distributed uniformly. Now we need to argue given $\sum_{i=\mu k+1}^{k} m_i^1$, $\sum_{i=\mu k+1}^{k} m_i^0$ is still uniform.

  For each $i \in \{\mu k+1, \ldots, k\}$, invoking Lemma 2 for $v_i^* = b_1$ we have that $\Pr[c(i) = 0] \geq \frac{2}{|X|^2}$. Therefore we have that sufficiently many (i.e. close to $\frac{2k}{|X|^2}$) choices of i such that $c(i) = 0$. Let the set of these values of i be I.

  Now for each $i \in I$, invoking Lemma 3 for $v_i^* = b_1$ it can be argued that, $\exists$ a constant $\tau > 0$ such that $\forall z \in \mathbb{Z}_p \Pr[m_i^0 = z] < 1 - \tau$). Hence, we have that m_i^0 is an uncertain random variable in $\mathbb{Z}_p$ (i.e. a random variable which satisfies the constrains of Lemma 4). The sum of such independent random variables is close to uniform by Lemma 4.
- `Hybrid`$_2$: This hybrid is same as `Hybrid`$_1$ except in the way sampling is done when the threshold point is reached. When the threshold point is reached we evaluate $c = \mathsf{majority}\{c(1), c(2), \ldots c(\mu k)\}$ (breaking ties arbitrarily and ignoring the $\perp$ values). We then obtain X_{1-c} from $\mathcal{O}$ and randomly sample all the values of u_i, that have not been sampled so far, subject to the condition $\sum_{i=1}^{k} m_i^{1-c} = z_{1-c} - X_{1-c}$. In other words we drop the additional constraint that $\sum_{i=1}^{k} m_i^c = z_c - X_c$. Just like in the previous hybrid, this sampling is also performed via rejection sampling.

Indistinguishability of $\texttt{Hybrid}_1$ *and* $\texttt{Hybrid}_2$*.* It suffices to prove that $\sum_{i=1}^{\mu k} m_i^c$ is close to uniform. For each $i \in [\mu k]$, invoking Lemma 2 for the adversary's query v_i^* we have that $\Pr[c(i) \in \{0,1\}] \geq \frac{2}{|X|^2}$. Therefore we have that sufficiently many (i.e. close to $\frac{2k}{|X|^2}$) choices of i we have that $c(i) \in \{0,1\}$. Additionally we know that c is the majority of $c(i)$ for $i \in [\mu k]$. We consider the case when $c = 0$. The other case when $c = 1$ follows in an analogous manner. Let the set of these values of i for which $c(i) = 0$ be I.

Now for each $i \in I$, invoking Lemma 3 (not that $v_i^* = b_1$ for $i \in I$) it can be argued that, $\exists$ a constant $\tau > 0$ such that $\forall z \in \mathbb{Z}_p \Pr[m_i^0 = z] < 1 - \tau$). Hence, we have that m_i^0 is an uncertain random variable in $\mathbb{Z}_p$ (i.e. a random variable which satisfies the constrains of Lemma 4). The sum of such independent random variables is close to uniform by Lemma 4.

Chosen Protocol Attack. Let a sender A with input bit x and receiver B with input bit y be two parties interacting via π to evaluate the functionality $\mathcal{F}_{asym}$. For the sake of contradiction let us assume that π securely realizes $\mathcal{F}_{asym}$ in the concurrent setting. Let π' be the following protocol between C and D with inputs x' and $(y', \tilde{m}, w)$ respectively. First C, D execute the same protocol as π with C as sender and D as receiver with inputs x' and y' respectively. Upon the completion of the protocol π, D checks if the output m he received is equal to the input $\tilde{m}$ and sends w to C in this case. He sends $\perp$ otherwise. The chosen protocol attack in the asymmetric case proceeds as follows. Consider an adversary $\mathcal{A}_2$ who corrupts the two players B, C. We will now describe a concurrent scheduling of executions of π and π' and *static* input distribution $\mathcal{D}_2$ for A and D such that an ideal-execution of π, π' is distinguishable from a real execution of π, π'. Let the distribution $\mathcal{D}_2$ be such that it generates a random value x in $\{0,1,2\}$ for an honest sender A of π and a random bit y for an honest receiver D of π'. Also $\tilde{m} = f(x,y)$ and $w \leftarrow \{0,1\}^k$ is given to D. Note that C and D execute π' using the identities of A, B respectively.

Message Scheduling by $\mathcal{A}_2$*.* The adversary $\mathcal{A}_2$ schedules messages as follows. It forwards messages that it receives from A to D (note that D acts the receiver of π in π') and the messages it receives from D to A. Observe that $\mathcal{A}_2$ will be successful in completing the protocol π between A and D. Finally, since the output that D obtains will be $f(x,y)$, $\mathcal{A}_2$ will obtain w from D. Recall that the honest party A does not get any output. Consider any simulator $\mathcal{S}_2$ that interacts in a hybrid execution with A via $\mathcal{F}_{asym}$ instead of π and also executes π' with D. Let y' be the input that $\mathcal{S}$ sends to $\mathcal{F}_{asym}$. Let x and $y, \tilde{m}, w$ be the inputs of the parties A and D respectively as described above. Now observe that the real-world adversary $\mathcal{A}_2$ always outputs w. There are two cases:

1. If $y \neq y'$ with a non-negligible probability then with a probability of at least $\frac{1}{3}$ $\mathcal{S}_2$ will not be able to provide the correct output to $\mathcal{A}_2$ and hence will not be able to output w.
2. On the other hand if y is always (except with negligible probability) equal to y' then this means that the adversary extracts the input y and also outputs w. Recall that w is generated only when D obtains an output $f(x,y)$ equal to $\tilde{m}$. Now note that this output is 0 with a constant probability (over the choice of x, y). Finally conditioned on the fact that D outputs 0, $\mathcal{S}_2$'s ability to extract D's input contradicts the stand-alone security of π for the honest receiver D.

Now by invoking Theorem 2, we see that no concurrent secure protocol exists for the function $\mathcal{F}_{asym}$.

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.

We refer the reader to Appendix A.3 for the definition of stateless two-party computation that we use in our paper. This is a simplified version of the definition of Goyal and Maji [GM11b]. We note that this definition is slightly stronger than the definition used in the full version [GM11a]. We stress that our impossibility result is only for the original variant [GM11b]. In particular the original notion considers computation among stateless parties in a concurrent setting. Finally we note that our impossibility result crucially relies on the fact that we are in the concurrent setting.

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 $\mathcal{F}_{universal} : (\{0,1\}^{p_1(k)} \times \{0,1\}^{q_1(k)}) \times (\{0,1\}^{p_2(k)} \times \{0,1\}^{q_2(k)}) \to (\{0,1\}^{r(k)} \times \{0,1\}^{r(k)})$ 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 (not necessarily fixed) polynomials. We stress that we consider a functionality which outputs the same value to both parties. This is consistent with the definition of [GM11a]. Let π 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_π be the next message function of P_2 in the protocol π. In other words, C_π 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 π in which it sees this message history. More formally, C_π 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 (as per Definition 3) for the $\mathcal{F}_{universal}$ functionality. (unconditionally)*

For the sake of contradiction, assume that there exists a protocol π, 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 π. 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 π. 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.

Construction of the ideal-world adversary in the first scenario: First, we consider a setting of two parties P_1 and P_2. In this setting we will consider two incarnations of each P_1 and P_2. Since we are in the setting of *static* inputs we need to specify the inputs for all incarnations of both parties. Let the input of P_1 in the first incarnation be (C_{eq},x) and the input in the second incarnation be $(C_\pi,((C_{eq},x),r))$. On the other hand, the input of P_2 in the first incarnation be (C_{eq},y) and the input in the second incarnation be $(C_\pi,\perp)$. For notational convenience, we denote the first incarnation of P_1 with input (C_{eq},x) by A and the second incarnation of P_1 with input $(C_\pi,((C_{eq},x),r))$ by B. Also note that the inputs of A and B are correlated in the fact that they use the same x which is a k bit random string.

In this setting, we start by describing a specific *deterministic* real-world adversary M that corrupts P_2. In both executions of π, with A and B, M ignores its inputs and plays the role of P_2. Note that the execution with A implements the "equality checking" circuit while the execution with B implements the "next-message function for P_2" circuit. In this setting our adversary M interacts honestly with A (without ever resetting it). However, it does not generate the honest responses on its own. In fact it obtains these responses from B. Note that M needs to execute a complete execution of π (with B) for each message that it needs to send to A. Our adversary M can achieve this by resetting B. Next we describe this more formally.

Recall that π is an r round protocol. For every $i \in \{1, 2, \ldots, r\}$, on receiving h_i from A, M proceeds as follows:

- Let $h_1, h_2, \ldots, h_i$ be the messages received from A, so far. M resets B and starts a new execution of π with inputs $(C_\pi, (h_1, h_2, \ldots, h_i))$. Recall the input of B is $(C_\pi, (C_{eq}, (x, r)))$. The output M obtains in the execution corresponds to the message $C_\pi((h_1, h_2, \ldots, h_i), (C_{eq}, x), r)$. M sends this message to A.

From the above description it is clear that for each $i \in [r]$ the message sent by M to A is same as the message an honest P_2 with input (C_{eq}, x) and random coins r would have sent. Observe that at the end of this interaction in the real world A outputs 1 (by correctness of the protocol) which would be included in the output of the real-world experiment.

By assumption, the protocol π is resettably secure, and therefore, there exists an ideal-world adversary $\mathcal{S}_M$[21] such that the output distributions of the real-world experiment and the ideal-world experiment are computationally indistinguishable. Recall that the ideal-world adversary $\mathcal{S}_M$ interacts with ideal functionalities implementing $\mathcal{F}_{universal}$. We will denote the functionality corresponding to the interaction with A by $\mathcal{F}_1$ and the functionality corresponding to the interaction with B by $\mathcal{F}_2$.[22]

As already pointed out, the output of A in the real world is always 1. Additionally, for A to output the value 1 in the ideal world, the simulator $\mathcal{S}_M$ must present the "correct value" x to the $\mathcal{F}_1$. Furthermore $\mathcal{F}_1$ responds by sending 1 to $\mathcal{S}_M$, when the "correct value" of x is sent to $\mathcal{F}_1$. It is easy to see we can modify $\mathcal{S}_M$ to construct another ideal-world adversary $\mathcal{S}'_M$ (that is also non-black box in M) that interacts in the same setting and always (except with negligible probability) outputs the value x.

Construction of the real-world adversary in the second scenario: Now we consider a new setting. In this setting we consider a single execution of π between two parties P_1 and P_2. Recall that we are in the setting of static inputs and we need to specify the inputs for P_1 and P_2. We will only consider a single incarnation for both P_1 and P_2. Let the inputs for P_1 and P_2 be (C_{eq}, y) and (C_{eq}, x), respectively. In this setting we consider a real-world adversary $\mathcal{A}$ that corrupts P_1 and internally uses the ideal-world adversary $\mathcal{S}'_M$ constructed in the first scenario and outputs x (with a noticeable probability). Let the expected running time of $\mathcal{S}'_M$ be τ. Recall that $\mathcal{S}'_M$ expects to interact with two ideal functionalities - $\mathcal{F}_1$ and $\mathcal{F}_2$. Furthermore, $\mathcal{S}'_M$ can query these functionalities multiple times (with different query inputs). Our adversary $\mathcal{A}$ deals with communication in the following manner.

Communication with $\mathcal{F}_2$: On receiving a query of the form $(h_1, h_2, \ldots h_t)$ sent by $\mathcal{S}'_M$ to $\mathcal{F}_2$, $\mathcal{A}$ resets the honest party P_2 (to the initial state) and feeds it with the inputs $h_1, h_2, \ldots h_t$. Let α be the response of P_2 when these inputs are sent to P_2. $\mathcal{A}$ forwards α to $\mathcal{S}'_M$. We stress that we crucially use the fact that a real-world adversary can reset the honest party P_2 in simulating the view of $\mathcal{S}'_M$.

Communication with $\mathcal{F}_1$: Our adversary $\mathcal{A}$ samples a random number $i \in [2 \cdot \tau]$. Let the i-th query sent by the adversary to the ideal functionality $\mathcal{F}_1$ be q_i. On receiving a query q_j (such that $j < i$) sent by $\mathcal{S}'_M$ to $\mathcal{F}_1$, $\mathcal{A}$ responds with 0. Finally, on receiving i-th query, q_i, $\mathcal{A}$ aborts everything and outputs q_i.

Observe that $\mathcal{S}'_M$ make less than 2τ queries with a probability of at least $\frac{1}{2}$. Furthermore, it always (except with negligible probability) queries $\mathcal{F}_1$ for the "correct" x such that all the queries made before this query do not match x. And the ideal functionality will continue to answer with 0 in this case. So, we can conclude that $\mathcal{A}$ outputs the "correct" x with probability at least $\frac{1}{4\tau}$. Finally it is not hard to see that no ideal-world adversary can output x (for large enough x) in the ideal world. This concludes the proof.

[21] We use the sub-script M to stress that the simulator $\mathcal{S}$ could potentially make non-black box use of the adversary M.

[22] Note that the two functionalities represent the same ideal functionality. This separation is done just for the sake of simplicity of exposition. A and B represent the two incarnation of P_1. Furthermore, $\mathcal{F}_1$ and $\mathcal{F}_2$ represent the separations of $\mathcal{F}_{universal}$ allowing for $\mathcal{S}_M$ to interact with A to B.

6 Acknowledgments

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

References

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

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[BCS96] Gilles Brassard, Claude Crépeau, and Miklos Santha. Oblivious transfers and intersecting codes. *IEEE Transactions on Information Theory*, 42(6):1769–1780, 1996.

[BMM99] Amos Beimel, Tal Malkin, and Silvio Micali. The all-or-nothing nature of two-party secure computation. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 1999.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS* [DBL06], pages 345–354.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552. IEEE Computer Society, 2005.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, Lecture Notes in Computer Science, pages 19–40. Springer, 2001.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for uc secure computation using tamper-proof hardware. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, pages 545–562, Berlin, Heidelberg, 2008. Springer-Verlag.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, pages 541–550, 2010.

[CPS07] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259, 2007.

[DBL06] *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings.* IEEE Computer Society, 2006.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *STOC*, pages 409–418, 1998.

[GGJS11] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Bringing people of different beliefs together to do uc. In *TCC*, pages 311–328, 2011.

[GGJS12] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Concurrently secure computation in constant rounds. In *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 99–116, Berlin, Heidelberg, 2012. Springer-Verlag.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *i*-hop homomorphic encryption and rerandomizable yao circuits. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2010.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In *CRYPTO*, pages 277–294, 2010.

[GK08] Vipul Goyal and Jonathan Katz. Universally composable multi-party computation with an unreliable common reference string. In *TCC*, pages 142–154, 2008.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2008.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In Dahlia Malkhi, editor, *DISC*, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2002.

[GM00] Juan A. Garay and Philip D. MacKenzie. Concurrent oblivious transfer. In *FOCS*, pages 314–324, 2000.

[GM11a] Vipul Goyal and Hemanta K. Maji. Stateless cryptographic protocols. In *FOCS*, 2011. `http://research.microsoft.com/en-us/people/vipul/gm11.pdf`.

[GM11b] Vipul Goyal and Hemanta K. Maji. Stateless cryptographic protocols. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 678–687, Washington, DC, USA, 2011. IEEE Computer Society.

[GM12] Vipul Goyal and Hemanta K. Maji. Personal communication., 2012.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In *CRYPTO*, pages 323–341, 2007.

[Gol11] Oded Goldreich. *Randomized Methods in Computation.* http://www.wisdom.weizmann.ac.il/õded/PDF/rnd.pdf, 2011.

[Goy11] Vipul Goyal. Positive results for concurrently secure computation in the plain model. Cryptology ePrint Archive, Report 2011/602, FOCS 2012 (to appear), 2011. `http://eprint.iacr.org/`.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 54–71, Berlin, Heidelberg, 2009. Springer-Verlag.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2007.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *In EUROCRYPT, Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

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

[Kil91] Joe Kilian. A general completeness theorem for two-party games. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 553–560. ACM, 1991.

[Kil00] Joe Kilian. More general completeness theorems for secure two-party computation. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 316–324, New York, NY, USA, 2000. ACM.

[KMQ11] Daniel Kraschewski and Jörn Müller-Quade. Completeness theorems with constructive proofs for finite deterministic 2-party functions. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 364–381. Springer, 2011.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 335–354. Springer, 2004.

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

[KSW97] John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1997.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403. IEEE Computer Society, 2003.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.

[Lin08] Yehuda Lindell. Lower bounds and impossibility results for concurrent self composition. *J. Cryptology*, 21(2):200–249, 2008.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC*, pages 179–188, 2009.

[MPR06] Silvio Micali, Rafael Pass, and Alon Rosen. Input-indistinguishable computation. In *FOCS* [DBL06], pages 367–378.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques*, EUROCRYPT'03, pages 160–176, Berlin, Heidelberg, 2003. Springer-Verlag.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proc. 36th STOC*, pages 232–241. ACM, 2004.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *FOCS*, pages 404–413, 2003.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.

[Rao07] Anup Rao. An exposition of bourgain's 2-source extractor. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(034), 2007.

[RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT*, pages 415–431, 1999.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.

A Preliminaries

In this section we define two-party concurrent computation (both in the unbounded and bounded setting) and resettable two-party computation more formally.

A.1 Concurrent Self Composition

Towards the goal of defining concurrent composition, we first describe the ideal world model process; next we describe the real process; and finally present the definition.

Concurrent execution in the ideal world. We now consider an ideal execution with an adversary $\mathcal{S}$ who has auxiliary input z and controls one of the parties. We will describe the execution in the setting in which the adversary controls P_2. The execution in the setting in which the adversary controls P_1 proceeds in a similar manner. Execution in the setting in which P_2 is corrupted proceeds as follows:

Inputs: Party P_1 receives an input vector $\bar{x}$ and party P_2 receives an input vector $\bar{y}$. Let $|x| = |y| = \ell$ denote the number of concurrent sessions of the protocol.

Session initiation: The adversary initiates a new session by sending a start-session message to the trusted party. The trusted party then sends (start-session, i) to the honest party, where i) is the index of the session (i.e., this is the i-th session to be started).

Honest party sends input to trusted party: Upon receiving (start-session, i) from the trusted party, the honest party P_1 sends its i-th input x_i to the trusted party as a message (i, x_i).

Adversary sends input to the trusted party and receives output: Whenever the adversary wishes, it may send a message (i, y_i) to the trusted party, for any y_i of its choice. Upon sending this pair, it receives back $(i, f_2(x_i, y_i))$. (If the i start-session message has not yet been sent to the trusted party, then (i, y_i) message from the adversary is ignored. In addition, once an output indexed by i has already been sent to the adversary, all further input messages for i are ignored by the trusted party.)

Adversary instructs trusted party to answer honest party: When the adversary sends a message of the type (send-output, i) to the trusted party, the trusted party sends $(i, f_1(x_i, y_i))$ to the honest party P_1. (If (i, x_i) and (i, y_i) have not yet been received by the trusted party, then this (send-output, i) message is ignored.)

Outputs: The honest party P_1 always outputs the vector $(f_1(x_{i_1}, y_{i_1}), f_1(x_{i_2}, y_{i_2}), \cdots)$ of outputs that it received from the trusted party. Formally, whenever it receives an output, it writes it to its output-tape. Thus, the outputs do not appear in ascending order according to the session numbers, but rather in the order that they are received. The adversary may output an arbitrary (probabilistic polynomial-time computable) function of its auxiliary input z, the initial input y, and the outputs obtained from the trusted party.

Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a functionality, and let $\mathcal{S}$ be a non-uniform probabilistic polynomial-time machine (representing the ideal-world adversary). Then the ideal execution of f (with security parameter k, initial inputs $(\bar{x}, \bar{y})$ and auxiliary input z to $\mathcal{S}$), denoted by $\mathrm{EXEC}_{f,\mathcal{S}}(k, \bar{x}, \bar{y}, z)$ is denoted as the output pair of the honest party and the adversary $\mathcal{S}$ from the above execution.

Execution in the real world. We next consider the real world in which a real two-party protocol is executed (and there exists no trusted third party). Formally, a two-party protocol $\rho = (\rho_1, \rho_2)$ is defined by two sets of instructions ρ_1 and ρ_2 (overloading notation) for parties P_1 and P_2, respectively. A protocol is said to be polynomial-time if the running-time of both ρ_1 and ρ_2 in a single execution is bounded by a fixed polynomial in the security parameter k, irrespective of the total number of sessions executed. Let f be as above and let ρ be a probabilistic polynomial-time two-party protocol for computing f. In addition, let $\mathcal{A}$ be a nonuniform probabilistic polynomial-time adversary (with non-uniform input z) that controls either P_1 or P_2. We describe the case in which the party P_2 is corrupted. The setting in which party P_1 is corrupted proceeds in a similar manner. The adversary $\mathcal{A}$ starts i-th session by sending a $\mathsf{start\text{-}session}$ message to the honest party, who uses his input x_i and follows the protocol instructions ρ_1 while the adversary $\mathcal{A}$ follows any arbitrary polynomial time strategy. Upon the conclusion of this execution the honest party writes its output from the execution on its output-tape while the adversary $\mathcal{A}$ may output any arbitrary polynomial time function of all the values it receives in the execution and its random coins. The parties run concurrent executions of the protocol and the scheduling of all messages throughout the executions is controlled by the adversary. That is, the execution proceeds as follows. The adversary sends a message of the form (i, α) to the honest party. The honest party then adds the message α to the view of its i-th execution of ρ and replies according to the instructions of ρ and this view. The adversary continues by sending another message (j, β), and so on.

Then, the real-world concurrent execution of ρ (with security parameter k, initial inputs $(\bar{x}, \bar{y})$, and auxiliary input z to the adversary $\mathcal{A}$), denoted $\mathrm{EXEC}_{\rho,\mathcal{A}}(k, \bar{x}, \bar{y}, z)$, is defined as the output pair of the honest party and $\mathcal{A}$, resulting from the above process.

Security as emulation of a real-world execution in the ideal world. Having defined the ideal and real worlds, we can now define security of protocols. Loosely speaking, a protocol is secure if for every real-world adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$ such that for all initial inputs $\bar{x}, \bar{y}$, the outcome of an ideal execution with $\mathcal{S}$ is computationally indistinguishable from the outcome of a real protocol execution with $\mathcal{A}$. Notice that $\mathcal{S}$ does not know the initial input of the honest party. We now present a formal definition.

Definition 1 *(security under concurrent self composition): Let f and ρ be as above. Protocol ρ is said to securely compute f under* $\mathsf{concurrent\ self\ composition}$ *if for every real-world non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ controlling party P_i $(i \in \{1, 2\})$ there exists an ideal-world non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ controlling P_i such that*

$$\left\{\mathrm{EXEC}_{f,\mathcal{S}}(k, \bar{x}, \bar{y}, z)\right\}_{k\in\mathbb{N};z\in\{0,1\}^*;\bar{x},\bar{y}\in(\{0,1\}^*)^k} \overset{c}{\equiv} \left\{\mathrm{EXEC}_{\rho,\mathcal{A}}(k, \bar{x}, \bar{y}, z)\right\}_{k\in\mathbb{N};z\in\{0,1\}^*;\bar{x},\bar{y}\in(\{0,1\}^*)^k}$$

A.2 Bounded Concurrency

Just like unbounded concurrent self composition, m-bounded concurrent self composition can be 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. We will now elaborate on the differences.

We note that the definitions of the ideal world is the same for unbounded and m-bounded concurrency. However, in the real world in the setting of m-bounded concurrency, we restrict how the the adversary can schedule its messages. More specifically, the adversary must fulfill the following condition: for every execution i, from the time that the i-th execution begins until the time that it ends, messages from at most m other executions can be sent. (Formally, view the schedule as the ordered series of messages of the form (`index`, `message`) that are sent by the adversary. Then, in the interval between the beginning and termination of any given execution, the number of different indices can be at most m.) We note that this definition of concurrency covers the case that m executions are run simultaneously. However, it also includes a more general case where many more than m executions take place, but each execution overlaps with at most m other executions. In this setting, the value of m is fixed ahead of time, and the protocol design may depend on the choice of m. We denote the output of the adversary and honest party in the setting of m-bounded concurrency by $\mathsf{REAL}^m_{\rho,\mathcal{A}}(k, \bar{x}, \bar{y}, z)$.

Definition 2 *(security under m-bounded concurrent self composition): Let f and ρ be as above. Also, let $m = m(k)$ be a fixed polynomial. Protocol ρ is said to securely compute f under m-bounded concurrent self composition if for every real-world non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ controlling party P_i ($i \in \{1, 2\}$) there exists an ideal-world non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ controlling P_i such that*

$$\left\{\mathrm{EXEC}_{f,\mathcal{S}}(k, \bar{x}, \bar{y}, z)\right\}_{k\in\mathbb{N};z\in\{0,1\}^*;\bar{x},\bar{y}\in(\{0,1\}^*)^k} \overset{c}{\equiv} \left\{\mathrm{EXEC}^m_{\rho,\mathcal{A}}(k, \bar{x}, \bar{y}, z)\right\}_{k\in\mathbb{N};z\in\{0,1\}^*;\bar{x},\bar{y}\in(\{0,1\}^*)^k}$$

A.3 Stateless Two-Party Computation

Just like unbounded concurrent self composition, stateless two-party computation can be 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 except that the capabilities of the adversary in the two cases vary significantly. We will now elaborate on this. We consider without loss of generality a simplified version of the definition of Goyal and Maji [GM11b]. Since in this model the ideal adversary can reset the functionality, we will often refer to the model as resettably secure computation.

We consider 2-parties P_1 and P_2 trying to compute a function of heir local inputs using an interactive protocol. The l-th *incarnation* of a party P_1 is defined by its input x_l and random tape ψ_l. Similarly the l-th *incarnation* of a party P_2 is defined by its input y_l and random tape ω_l. Let $\bar{X}$ be the vector of input and random tape pairs for party P_1; for all incarnations. Similarly, $\bar{Y}$ be the vector of input and random tape pairs for party P_2. We assume the setting of *static* inputs. More specifically the inputs for all the incarnations used in the protocol are fixed before the protocol begins. Every incarnation of a party has a random tape independently chosen but when an adversary resets an incarnation, it reuses the same random tape. An adversary controls either P_1 or P_2, which is statically corrupted.

Execution in the ideal world. We now consider an ideal execution with an adversary $\mathcal{S}$ who has auxiliary input z and controls one of the parties. We will describe the execution in the setting in which the adversary controls P_2. The execution in the setting in which the adversary controls P_1 proceeds in a similar manner. Execution in the setting in which P_2 is corrupted proceeds as follows.

Inputs: Party P_1 receives an input a random tape and vector $\bar{X}$ and party P_2 receives an input vector $\bar{Y}$.

Select Incarnation: The adversary $\mathcal{S}$ initiates a new session by selecting the incarnation[23] for P_1 and sending it to the ideal functionality. The trusted party then sends (start-session, l) to the honest party, where l is the index of the incarnation (i.e., this is the l-th incarnation to be started).

Computation: Upon receiving (start-session, l) from the trusted party, the honest party P_1 sends the input corresponding to its l-th incarnation x_l to the trusted party as a message (l, x_l). The adversary may send any input (l, y_l) to the trusted party for any y_l of its choice. The trusted party computes the function f on the inputs of the two parties and sends $f_2(x_l, y_l)$ to P_2.

Adversary instructs trusted party to answer honest party: When the adversary sends a message of the type (send-output, l) to the trusted party, the trusted party sends $(l, f_1(x_l, y_l))$ to the honest party P_1. (If (l, x_l) and (l, y_l) have not yet been received by the trusted party, then this (send-output, l) message is ignored.)

Reset: The adversary can reset the ideal party P_1 at point of time. When the adversary sends the (reset, P_1) messages to the ideal functionality in the ideal world, then the trusted party forwards this to P_1. At this point P_1 returns to its first stage where the adversary can select any incarnation for it.

Outputs: The honest party P_1 always outputs the output that it received from the trusted party. The adversary may output an arbitrary (probabilistic polynomial-time computable) function of its auxiliary input z, the initial input y, and the outputs obtained from the trusted party.

Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a functionality, and let $\mathcal{S}$ be a non-uniform probabilistic polynomial time machine (representing the ideal-world adversary). Then the ideal execution of f (with security parameter k, initial inputs $(\bar{X}, \bar{Y})$ and auxiliary input z to $\mathcal{S}$), denoted by $\textsc{Reset}\text{-IDEAL}_{f,\mathcal{S}}(k, \bar{X}, \bar{Y}, z)$ is denoted as the output pair of the honest party and the adversary $\mathcal{S}$ from the above execution.

[23] An incarnation is a stateless device implementing P_1.

Execution in the real world. We next consider the real world in which a real two-party protocol is executed (and there exists no trusted third party). Let $\mathcal{A}$ be a nonuniform probabilistic polynomial-time adversary $\mathcal{A}$ (with non-uniform input z) that controls either P_1 or P_2. We describe the case in which the party P_2 is corrupted. The setting in which party P_1 is corrupted proceeds in a similar manner. The adversary $\mathcal{A}$ interacts with the the honest party, who uses his input x_l and randomness r_l in the l-th incarnation. The adversary can choose the incarnation of the P_1 it interacts with. In this interaction P_1 follows the protocol specifications honestly while the adversary $\mathcal{A}$ follows any arbitrary polynomial time strategy. Furthermore, the adversary can reset P_1 and execute the protocol again with a (possibly new) incarnation of its choice. Upon the conclusion of this execution the honest party writes its output from the execution on its output-tape while the adversary $\mathcal{A}$ may output any arbitrary polynomial time function of all the values it receives in the execution and its random coins.

Then, the real-world execution of ρ (with security parameter k, initial inputs $(\bar{x}, \bar{y})$, and auxiliary input z to the adversary $\mathcal{A}$), denoted $\text{Reset-REAL}_{\rho,\mathcal{A}}(k, \bar{x}, \bar{y}, z)$, is defined as the output pair of the honest party and $\mathcal{A}$, resulting from the above process.

Security as emulation of a real-world execution in the ideal world. Having defined the ideal and real worlds, we can now define security of protocols. Loosely speaking, a protocol is secure if for every real-world adversary $\mathcal{A}$, there exists an ideal model adversary $\mathcal{S}$ such that for all initial inputs $\bar{X}, \bar{Y}$, the outcome of an ideal execution with $\mathcal{S}$ is computationally indistinguishable from the outcome of a real protocol execution with $\mathcal{A}$. Notice that $\mathcal{S}$ does not know the initial input of the honest party. We now present the definition.

Definition 3 *(security under reset attacks): Let f and ρ be as above. Protocol ρ is said to* resettably *securely compute f if for every real-world non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ controlling party P_i ($i \in \{1, 2\}$) there exists an ideal-world non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ controlling P_i such that*

$$\left\{\text{Reset-IDEAL}_{f,\mathcal{S}}(k, \bar{x}, \bar{y}, z)\right\}_{k\in\mathbb{N};z\in\{0,1\}^*;\bar{x},\bar{y}\in(\{0,1\}^*)^k} \overset{c}{\equiv} \left\{\text{Reset-REAL}_{\rho,\mathcal{A}}(k, \bar{x}, \bar{y}, z)\right\}_{k\in\mathbb{N};z\in\{0,1\}^*;\bar{x},\bar{y}\in(\{0,1\}^*)^k}$$

B Garbled Circuits

In this section we briefly recall the notion and constructions for garbled circuits that we need in this paper. Garbled circuits were first constructed by Yao[Yao86] and have since then found numerous uses in secure multi-party computation and elsewhere. A vast body of literature explaining this notion exists. However since our construction relies heavily on garbled circuits in order to clarify the notation that we use in this paper we present an informal overview. We assume that the reader is familiar with the notion and constructions. We first start by considering the semi-honest (or, honest-but-curious) case and then extend the construction to the malicious case.

B.1 Garbled Circuits for Semi-Honest Adversaries

A formal simulation based security definition for garbled circuit construction in the case of an honest but curious adversary is presented in [HK07,KO04]. Some parts of the following text have been taken verbatim from [HK07,KO04].

Let $F_k : \{0,1\}^k \to \{0,1\}^k$ denote any polynomial-time computable function[24]. Note that a garbled circuit only hide the nature of a gate used in a circuit and does not hide the number of gates used etc. However this can be achieved by using some form of canonicalization. This is fairly standard when using garbled circuits and we refer the reader to Section 4 of [GHV10] for more discussion. Formally,

Definition 4 *(garbled circuits) Let $F_k : \{0,1\}^k \to \{0,1\}^k$ be the description of any function as above. Yao [Yao86]'s* garbled circuit *technique is a pair of algorithms* $(\texttt{Yao}_1, \texttt{Yao}_2)$ *such that,*

[24] The garbled circuit technique also extends to functions whose input and output are polynomial in k. However for simplicity of exposition we limit ourselves to functions with input and output lengths exactly k.

- $\mathtt{Yao}_1$ *is a randomized algorithm which takes as input F_k and outputs a tuple $(GC, \mathbf{Z})$ where GC is a* garbled circuit *and $\mathbf{Z} = \{Z_{i,\sigma}\}_{i\in\{1,\ldots,k\},\sigma\in\{0,1\}}$ are* keys *corresponding to input wires.*
- $\mathtt{Yao}_2$ *is a deterministic algorithm which takes as input a garbled circuit, GC and keys corresponding to an input x. More specifically it takes as input a set $\{Z_{i,x_i}\}_{i\in\{1,\ldots,k\}}$ (denoted by $\mathbf{Z}_x$) of keys where x_i is the i^{th} bit of x. It outputs an invalid symbol $\perp$ or a value $v \in \{0,1\}^k$.*
- **Correctness:** *For any function F_k, let $(GC, \mathbf{Z}) \leftarrow \mathtt{Yao}_1(F_k)$, and for any input x, let $v = \mathtt{Yao}_2(GC, \mathbf{Z}_x)$. Then we require that $v = F_k(x)$.*
- **Privacy (honest-but-curious case):** *$\exists$ a PPT simulator* $\mathtt{YaoSim}$ *such that for all PPT adversaries $\mathcal{A}$ with auxiliary input z,*

$$\left\{(GC, \mathbf{Z}) \leftarrow \mathtt{Yao}_1(F_k) : \mathcal{A}(k, z, x, (GC, \mathbf{Z}_x))\right\}_{k\in\mathbb{N},z\in\{0,1\}^*,x\in\{0,1\}^k}$$

$$\stackrel{c}{\equiv} \left\{v = F_k(x) : \mathcal{A}(k, z, x, \mathtt{YaoSim}(k, x, v))\right\}_{k\in\mathbb{N},z\in\{0,1\}^*,x\in\{0,1\}^k}$$

B.2 Garbled Circuit, Malicious Adversaries

Although garbled circuits were designed for the honest-but-curious case, since [GMR89], they have been used in the presence of malicious adversaries in the interactive setting via compilations with zero-knowledge proofs. [GKR08,BPS06] consider garbled circuit constructions secure against a malicious adversary. In the following for the sake of completeness we provide a formal definition of security of garbled circuits in the malicious, adaptive setting and provide a construction for the same. The construction follows from the work of [GKR08]. In our construction we sill assume a garbled circuit constructions secure in the semi-honest setting (according to Definition 4). Some of the following text has been taken verbatim from [GKR08].

Before we get into details we highlight the main concern. A malicious adversary evaluating the garbled circuits can be *adaptive* in its choice of the input x. In particular, a malicious adversary may choose the input (and thus the keys it obtains) adaptively based on the actual circuit (and also the keys so far). Therefore our simulator needs to "hold off" on embedding the output at the time of the generation of the garbled circuit. Goldwasser et. al. [GKR08] consider such a setting. They deal with this problem by providing the adversary with a garbled circuit that computes masked functions. The unmasking values are provided along with the keys. More specifically, a simulator can be constructed which "waits" until all the input keys have been queried (or specified). Finally when all the input bits have been specified the simulator can manipulate the output masks based on the output that it needs to world.

Next we formalize the notion of garbled circuits in the setting of malicious adversaries below by using the simulation based definition for the functionality $\mathcal{F}$ (that can be invoked n times) to evaluate functions $F_1, F_2 \ldots F_n$.

Reactive functionality $\mathcal{F}$. A reactive functionality $\mathcal{F}$ consists of a sequence of functions $F_1, F_2, \ldots, F_n : \{0,1\}^k \rightarrow \{0,1\}^k$.[25] On the i^{th} invocation of the functionality $\mathcal{F}$ function F_i is evaluated and the output of F_i can potentially depend on the inputs provided to functions F_i's for all $1 \leq j < i$. We remark that in our setting we will use the next message function of a party (in an execution of an n round protocol) and model it as a functionality $\mathcal{F} = (F_1, F_2, \ldots, F_n)$.

Towards the goal of defining security, we first describe the ideal world process; next we describe the real process; and finally present the definition.

Ideal execution of a reactive functionality $\mathcal{F}$. Consider an ideal execution of an adversary $\mathcal{S}$ on input $\bar{x}$ and auxiliary input z interacting with the ideal functionality $\mathcal{F}$. Let $\mathcal{F} = (F_1, F_2, \ldots, F_n)$. Execution proceeds as follows,

- **Input:** $\mathcal{S}$ receives an input $\bar{x} = (x_1, \ldots, x_n)$. It is also provided with an auxiliary input z.
- **Evaluation:** For each $i \in [n]$, $\mathcal{S}$ sends as input x'_i and obtains a response $F_i(x'_i)$, where as mentioned before, F_i can additionally depend on all of the inputs $x'_j, \forall 1 \leq j < i$.

[25] As in the case of semi-honest setting technique extends to functions whose input and output are polynomial in k.

- **Output:** $\mathcal{S}$ outputs any arbitrary probabilistic polynomial-time function of its view. Let $\mathrm{EXEC}_{\mathcal{F},\mathcal{S}}(k, \bar{x}, z)$ denote the output distribution of the ideal-world adversary $\mathcal{S}$ in interaction with the ideal functionality $\mathcal{F}$.

Real-world execution of garbled circuits by an adversary. Let $\mathcal{F} = (F_1, \ldots, F_n)$ be a reactive functionality as above and let $(\mathtt{Yao}_1, \mathtt{Yao}_2)$ be as above. A real-world evaluation of $\mathcal{F}$ using garbled circuits uses a pair algorithms $(\mathtt{Yao}_1, \mathtt{Yao}_2)$ such that,

- $\mathtt{Yao}_1$**:** $\mathtt{Yao}_1$ on input a description of a reactive functionality $\mathcal{F} = (F_1, \ldots, F_n)$ outputs a tuple $(GC, \mathbf{Z})$ denoting the garbled circuit and the keys for input wires.
- $\mathtt{Yao}_2$**:** $\mathtt{Yao}_2$ is a deterministic algorithm which on input a garbled circuit GC and keys $(Z_{1,b_1}, Z_{2,b_2}, \ldots, Z_{t,b_t})$ where $t = i \cdot k$ (where b (of length t) is the concatenation of the inputs $x_1, \ldots, x_i$) outputs an evaluation of F_i.

Let $\mathcal{A}$ be any polynomial-time algorithm. Then the real-world execution proceeds as follows:

- **Input:** Let $\bar{x} = (x_1, \ldots, x_n)$ be the input of the adversary $\mathcal{A}$. It is also provided with an auxiliary input z. Furthermore, it is provided GC as an input from the following setup algorithm.
- **Setup:** Let $(GC, \mathbf{Z}) \leftarrow \mathtt{Yao}_1(\mathcal{F})$. Let g ($n \cdot k$ in this case) be the number of keys.
- **Key Queries:** Let $\mathcal{K}$ be a key oracle. At any point $\mathcal{A}$ sends a message of the form $(i, b), i \in [g], b \in \{0, 1\}$ to $\mathcal{K}$ to which $\mathcal{K}$ responds by $Z_{i,b}$.
- **Output:** $\mathcal{A}$ outputs any arbitrary probabilistic polynomial-time function of its view. Let $\mathrm{EXEC}_{\mathcal{K},\mathcal{A}}(k, \bar{x}, z)$ denote the output distribution of $\mathcal{A}$.

Security: The definition of security, following the ideal-real model is as follows. [26]

Definition 5 *(garbled circuits - malicious case) Let $\mathcal{F}$ be any reactive functionality as above. Then $\exists$ a PPT ideal-world adversary (i.e., a black-box simulator)* $\mathtt{YaoSim}$ *such that for every PPT real-world adversary $\mathcal{A}$,*

$$\left\{\mathrm{EXEC}_{\mathcal{F},\mathtt{YaoSim}^{\mathcal{A}}}(k, \bar{x}, z)\right\}_{k\in\mathbb{N}, z\in\{0,1\}^*, \bar{x}\in(\{0,1\}^k)^n} \overset{c}{\equiv} \left\{\mathrm{EXEC}_{\mathcal{K},\mathcal{A}}(k, \bar{x}, z)\right\}_{k\in\mathbb{N}, z\in\{0,1\}^*, \bar{x}\in(\{0,1\}^k)^n}$$

Next using a construction for garbled circuits $(\widehat{\mathtt{Yao}_1}, \widehat{\mathtt{Yao}_2})$ secure in the semi-honest setting (Definition 4) we give a construction $(\mathtt{Yao}_1, \mathtt{Yao}_2)$ that offers security against malicious adversaries (Definition 5). A formal construction is provided in Construction 1.

Lemma 5. *(garbled circuits secure against malicious adversaries) Assuming garbled circuit secure against honest-but-curious adversaries exist, there exists a garbled circuit construction that is secure against malicious adversaries (Definition 5).*

Proof (Informal). In order to argue the above lemma we need to argue that for every adversary $\mathcal{A}$ (that obtains its keys *adaptively* from the oracle $\mathcal{K}$) there exists a simulator $\mathtt{YaoSim}$ that can simulate its view interacting directly with the functionality $\mathcal{F}$. Note that the adversary $\mathcal{A}$ expects to receive as input a garbled circuit. Our simulator $\mathtt{YaoSim}$ starts by generating a garbled circuit that just outputs random values $m_1, m_2 \ldots m_n$. Now our simulator $\mathtt{YaoSim}$ needs to simulate the key oracle $\mathcal{K}$ for the adversary $\mathcal{A}$ in a way such that $\mathcal{A}$ obtains the correct output values from the evaluation when unmasked with the sub-masks. We will describe our strategy in ensuring that the correct output F_i is obtained. Every $i \in [n]$ can be handled in an analogous manner. For the adversary $\mathcal{A}$ to be able to evaluate F_i, the adversary $\mathcal{A}$ needs to obtain all the keys $\{Z_{j,b_j}\}_{j\in[ik]}$ where b is the concatenation of $x_1, x_2 \ldots x_i$. Furthermore each key Z_{j,b_j} contains a sub-mask $s_{i,j}$ that is needed to evaluate F_i. All these masks are set randomly until the last key is queried. Let Z_{t,b_t} be the last key among $\{Z_{j,b_j}\}_{j\in[ik]}$ that the adversary $\mathcal{A}$ queries to $\mathcal{K}$. Note that the entire input (i.e., $x_1, \ldots x_i$) of the adversary gets specified once it has made all these queries. Our simulator $\mathtt{YaoSim}$, when queried with the last key Z_{t,b_t} obtains the appropriate output for F_i from the ideal functionality $\mathcal{F}$. It sets sets the sub-mask $s_{i,t} = \oplus_{j=1,j\neq t}^{ik} s_{i,j} \oplus m_i$. This allows our simulator to force the output obtained from the ideal functionality $\mathcal{F}$. ■

[26] We provide a stronger black-box definition of security. Known constructions satisfy this definition of security.

Let $\mathcal{F} = (F_1, \dots F_n)$ where each F_i is such that $F_i : \{0,1\}^k \to \{0,1\}^k$. Let $(\widehat{\texttt{Yao}_1}, \widehat{\texttt{Yao}_2})$ be a pair of algorithms that satisfy Definition 4. Then let algorithms $(\texttt{Yao}_1, \texttt{Yao}_2)$ be as follows.

$\texttt{Yao}_1$:

1. For each $i \in \{1, \dots n\}$, choose masks $m_i \leftarrow \{0,1\}^k$.
2. Let $\mathcal{F}' = (F_1(x_1) \oplus m_1, F_2(x_2) \oplus m_2, \dots F_n(x_n) \oplus m_n)$. Let $(\widehat{GC}, \widehat{\mathbf{Z}}) = \widehat{\texttt{Yao}_1}(\mathcal{F}')$. Note that $\widehat{\mathbf{Z}}$ consists of key pairs for kn input wires and one key from the first ki pairs of keys is needed to evaluate F_i. The specific keys needed depend on the inputs $x_1, x_2 \dots x_i$.
3. For each $i \in [n]$, for each $j \in [ik]$, choose a sub-mask $s_{i,j} \leftarrow \{0,1\}^k$ subject to $\oplus_{j=1}^{ik} s_{i,j} = m_i$. Append sub-mask $s_{i,j}$ to the keys $\widehat{Z}_{j,0}$ and $\widehat{Z}_{j,1}$
4. **Output:** $(GC, \mathbf{Z}) = (\widehat{GC}, \widehat{\mathbf{Z}})$ after the transformations above.

$\texttt{Yao}_2$:

On input GC, and x_i (where b is the concatenation of x_i with the previous inputs $x_1, x_2 \dots x_{i-1}$) proceed as follows:

1. Observe that the keys Z_{j,b_j} for every $j \in [(i-1)k]$ have already been obtained for evaluations of $F_1, \dots F_{i-1}$. Obtain the keys Z_{j,b_j} for every $j \in (i-1)k + 1, \dots ik$.
2. For each $j \in [ik]$, extract the values $\widehat{Z}_{j,b_j}$ and the sub-mask $s_{i,j}$ from Z_{j,b_j}.
3. Compute the mask $m_i = \oplus_{j=1}^{ik} s_{i,j}$ and let $O_i = \widehat{\texttt{Yao}_2}(GC, \{\widehat{\mathbf{Z}}_{j,b_j}\}_{j \in \{1 \dots ik\}})$.
4. **Output:** Output $m_i \oplus O_i$.

Construction 1: Garbled circuits: malicious case

C Garbled Circuits with Bit OT

In this work we use the techniques introduced by [BCS96] for constructing a string OT protocol denoted, OT_2^k using a bit OT protocol, denoted OT_2^1. This is crucial for us in proving our Theorem 2 proved in Section 3.2. We do not present all the details from their construction. However for the sake of completeness we recall some basic terminology and give some intuition.

String OT was used in the construction of garbled circuits in order to construct a mechanism that allows a sender S to send a key (among a pair of keys per input wire) to the receiver in a way that ensures that the receiver learns only one key (and learns nothing else about the other key). However in order to argue impossibility for more general functionalities we would like to achieve the same effect given access to bit OT only. One natural solution to this problem is to allow R to obtain a sequence of bits in bit OT calls where each time R can choose a bit (on a specific location) from the two keys. In this case R can choose to obtain one key completely. This would allow R to obtain a key of its choice. However, the problem is that a malicious receiver could potentially obtain bits from both the keys. This would render the garbled circuit completely insecure. We deal with this problem using an idea introduced by [BCS96]. The idea is that S instead of using the keys (which are actually strings) for bit OT directly (as input) uses an "encoding" of the keys (for each input wire) which ensures that regardless of the choices of bits that malicious R receives, R can only obtain one key for each input wire. An important feature of the [BCS96] technique is that it allows us to specify the inputs of the honest sender *statically*, i.e., before any bit OT protocol is actually executed.

OT_2^k. S has input (w_0, w_1) and R has input c.

1. S picks $x_0, x_1 \leftarrow \{0,1\}^n$ such that $f(x_0) = w_0$ and $f((x_1) = w_1$.
2. Perform n executions of OT_2^1 where in the i invocation of OT_2^1, S uses the input (x_0^i, x_1^i) and R always uses the input c and S obtains z^i.
3. R recovers w_c by computing $f(z)$.

Construction 2: Protocol 1.1 in [BCS96] for OT_2^k from OT_2^1

Let $I = \{i_1, i_2, \dots i_m\}$ be a set such that $1 \le i_1 < i_2 \dots i_m \le n$. We define x^I to be the concatenation of $x^{i_1} x^{i_2} \dots x^{i_m}$. [BCS96] give explicit constructions of functions $f : \{0,1\}^n \to \{0,1\}^k$ with the property that for any two subsets $I, J \subseteq \{1, 2, \ldots n\}$, seeing the bits of x_0^I and x_1^J releases information on at most one of $f(x_0)$ or $f(x_1)$. Observe that this immediately gives us what we are looking for. Plugging this in our garbled circuits construction (for malicious adversaries) we get a garbled circuit where keys are only one bit long.

Construction 2 is information theoretically secure when the function f is instantiated appropriately as defined in [BCS96]. Now we will describe the simulation strategy of this OT_2^k protocol at a very intuitive level. Let $\tilde{R}$ be a malicious receiver. Now we need to simulate its view when it can obtain bits of x_0 and x_1 given access to an oracle from which we can only obtain either w_0 or w_1. Let T_0, T_1 be two sets indicating the indices for which $\tilde{R}$ queries with input 0 and 1 respectively. Observe that T_0 and T_1 will always be disjoint. For every query made by $\tilde{R}$ we need to provide it with a response. Our strategy would be to provide it with a random bit as long as neither T_0 nor T_1 "biases" f (Definition 2.1 of [BCS96], i.e., no information about either w_0 or w_1 has been revealed by the values of x_0, x_1 provided to $\tilde{R}$ so far). At the first point when T_b (for some $b \in \{0, 1\}$) biases f, we obtain the corresponding value w_b. We then sample x_b consistently with w_b and the values provided to $\tilde{R}$ so far and continue the simulation. By Definition 2.2 of [BCS96], for the encoding function f only one among T_0 and T_1 can bias f. This fact allows us to continue choosing bits in x_{1-b} at random without querying for w_{1-b}. This allows us to conclude that we will be able to simulate $\tilde{R}$ by making at most one query for either w_0 or w_1.

D Proof of the XOR Lemma

Recall that we want to prove the following lemma.

Lemma 4. *Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be k independent random variables such that $\exists \tau > 0$ such that, for all i, for all $z \in \mathbb{Z}_p$, $\Pr[\alpha_i = z] < 1 - \tau$. Then $\alpha = \sum_{i=1}^k \alpha_i \mod p$ is a uniformly distributed in $\mathbb{Z}_p$ except with negligible probability (in k).*

Proof. To prove this lemma we first recall the following lemma from [Rao07].

Lemma 6. *(Lemma 4.2 in page 10 of [Rao07]) X be a distribution on a finite abelian group G such that $|\mathbb{E}(\psi(X))| \leq \epsilon$ for every non-trivial character ψ. Then X is $\epsilon\sqrt{|G|}$ close to the uniform distribution.*

We will obtain our proof by setting $G = \mathbb{Z}_p, X = \sum_{i=1}^k \alpha_i \mod p$ in the above lemma and by showing that ϵ is negligible for each non-trivial character ψ of X. Then using the above lemma we conclude that the statistical distance between $\sum_{i=1}^k \alpha_i \mod p$ and the uniform distribution is negligible in k.

In order to show that for every non-trivial character ψ, $|\mathbb{E}(\psi(X))| \leq \epsilon$, we first recall that up to isomorphisms we may restrict ourselves to considering characters which map G to the unit circle in the complex plane. Furthermore, since $|G| = p$, we may restrict ourselves to the multiplicative group of p-th roots of unity. There are only $p - 1$ such non-trivial characters and we now proceed by considering the character $\psi : G \to \mathbb{C}$ such that $\psi(x) = e^{\frac{-2\pi jx}{p}}$. The argument is identical for other characters as well.

By independence of α_i and $X = \sum \alpha_i$ we have that $|\mathbb{E}(\psi(X))| = |\mathbb{E}(e^{\frac{-2\pi jX}{p}})| = \prod_{i=1}^k |\mathbb{E}(e^{\frac{-2\pi j\alpha_i}{p}})|$. Next we show that each individual expectation is bounded away from 1. Based on this we can conclude that the product of the expectations is negligible.

Denote by $t_0, t_1, \ldots, t_{p-1}$ the probabilities that α_i takes over G. So, $\sum_s t_s = 1$. Observe that

$$|\mathbb{E}(e^{\frac{-2\pi j\alpha_i}{p}})| = \left|\sum_{s=0}^{p-1} t_s e^{\frac{-2\pi js}{p}}\right| \leq \sum_{s=0}^{p-1} \left|t_s e^{\frac{-2\pi js}{p}}\right|$$

Note that for all $z \in \mathbb{Z}_p \Pr[\alpha = z] < 1 - \tau$, there exist at least two non-zero values among $t_0, \ldots, t_{p-1}$. Also observe that for all $s, s' \in \{0, 1, \ldots, p - 1\}$ such that $s \neq s'$, $e^{\frac{-2\pi js}{p}}$ is not a real multiple of $e^{\frac{-2\pi js'}{p}}$. Thus we can conclude that $|\mathbb{E}(e^{\frac{-2\pi j\alpha_i}{p}})| = q < 1$ and hence we have our result.