New Impossibility Results for Concurrent Composition and a Non-interactive Completeness Theorem for Secure Computation

Shweta Agrawal[1], Vipul Goyal[2], Abhishek Jain[1], Manoj Prabhakaran[3], and Amit Sahai[1]

[1] UCLA
[2] Microsoft Research, India
[3] UIUC

Abstract. We consider the *client-server* setting for the concurrent composition of secure protocols: in this setting, a single server interacts with multiple clients concurrently, executing with each client a specified protocol where only the client should receive any nontrivial output. Such a setting is easily motivated from an application standpoint. There are important special cases for which positive results are known – such as concurrent zero knowledge protocols – and it has been an open question whether other natural functionalities such as Oblivious Transfer (OT) are possible in this setting.

In this work:

- We resolve this open question by showing that unfortunately, even in this very limited concurrency setting, **broad new impossibility results** hold, ruling out not only OT, but in fact all nontrivial finite asymmetric functionalities. Our new negative results hold even if the inputs of all honest parties are fixed in advance, and the adversary receives no auxiliary information.
- Along the way, we establish a new **unconditional completeness result** for asymmetric functionalities, where we characterize functionalities that are *non-interactively complete* secure against active adversaries. When we say that a functionality $\mathcal{F}$ is non-interactively complete, we mean that every other asymmetric functionality can be realized by parallel invocations of several copies of $\mathcal{F}$, with no other communication in any direction. Our result subsumes a completeness result of Kilian [STOC'00] that uses protocols which require additional interaction in both directions.

1 Introduction

Consider the following scenario: Goldman Sachs, which has collected in-depth market research and analysis on various companies, wishes to offer a paid service to high-profile investors who are interested in using this market research. For this purpose, it has set up a large server to which potential clients can connect in order to obtain answers to queries of an authorized kind (for which they have

R. Safavi-Naini and R. Canetti (Eds.): CRYPTO 2012, LNCS 7417, pp. 443–460, 2012.

paid). Potential investors, however, are wary of Goldman Sachs learning about their business plans, and would want to hide the queries from Goldman Sachs – indeed, Goldman Sachs should learn nothing at all except that the client performed an authorized query. On the other hand, Goldman Sachs would like to ensure that in one session, a client can only learn the answer to a single query of a kind that it is authorized to ask. In particular, it would like to ensure that multiple clients who may connect to the server simultaneously, cannot perform a *coordinated attack* to learn more information than what each of them paid for (or even carry out an unauthorized query).

Can we satisfy these requirements? While well-known two-party computation protocols (e.g., [35,14]) offer remarkably powerful simulation-based security guarantees, these guarantees hold only in the stand-alone model, where only one protocol execution takes place. Our scenario is slightly more complex, as it has some mild concurrency in that multiple clients may interact with the server concurrently. At the same time, we are making the plausible assumption that the server is programmed only to interact with clients using the prescribed protocol, and we do not seek to guarantee security of any other protocols that the clients may be engaged in while they are communicating with the server[1]. Arguably, such a *client-server* setting (formally, asymmetric functionalities in a "fixed-role concurrent self composition" setting) is of great practical relevance. But apart from the practical significance, from a theoretical point of view, it is an important question as to whether restricting to such a model of concurrent executions of a single protocol, allows us to recover strong security guarantees for two-party computation (at least for some functionalities).

In this work, we consider secure computation in the client-server setting, and show that, even in this highly restricted concurrency setting, broad impossibility results for secure computation hold.

- We establish **new and broad impossibility results** for achieving security under fixed-roles concurrent self composition, ruling out *all finite functionalities* (except "trivial" ones which have universally composable protocols), including many natural functionalities such as oblivious transfer. Our results hold even if the inputs to the parties in all sessions are fixed before the protocols commence.
- Along the way, we establish a new **unconditional completeness result** for asymmetric functionalities, where we characterize functionalities that are *non-interactively complete*[2] for secure computation against active adversaries. This subsumes a result of Kilian [22] that used a protocol which had additional interaction and, for its security, relied on the properties of a Nash equilibrium of a zero-sum game.

[1] If we did consider the security of other protocols that the clients were engaged in, then known sweeping impossibility results would apply. See further details below.

[2] We say that a functionality $\mathcal{F}$ is non-interactively complete if every other asymmetric functionality can be realized by parallel invocations of several copies of $\mathcal{F}$, with no other communication in any direction.

Background: Security under Concurrent Composition. With the proliferation of the network setting, in particular the *Internet*, the last decade has seen a push towards constructing protocols that have strong concurrent composability guarantees. For example, we could require security under *concurrent self-composition* (which is the focus of this work): a protocol should remain secure even when there are multiple copies executing concurrently. The framework of universal composability (UC) [4] was introduced to capture the more general setting of *concurrent general composition*, where a protocol may be executed concurrently with not only several copies of itself but also with other arbitrary protocols.

General positive results for UC secure computation have been obtained based on various *trusted* setup assumptions such as a common random string [6,9,1,10,20,25]. Whether a given set of players is actually willing to trust an external entity, however, is debatable. Indeed a driving goal in cryptographic research is to eliminate the need to trust other parties. Ideally, we would like to achieve security under concurrent composition in the *plain model* (which is the main focus of this work).

The Dark Side of Concurrency. Unfortunately, in the plain model, by and large, most of the results have been negative.[3] UC secure protocols for most functionalities of interest were ruled out in [6,7,32]. (Recently, these were extended to various public key models in [21].) These impossibility results were extended to the setting of general composition by Lindell [26] who proved that security under concurrent general composition implies UC security. Later, Lindell [27] established broad negative results even for the setting of concurrent self-composition by showing equivalence of concurrent self-composition and general composition for functionalities where each party can "communicate" to the other party via its output (referred to as *bit transmitting functionalities*). Barak et al. [2] (and more recently, [16]) obtained negative results for very specific functions in the "static-input" setting (i.e., where the inputs of honest parties are fixed in advance for all the protocol sessions).

On the positive side, there has been much success in obtaining construction for zero-knowledge and related functionalities, with security in similar models (e.g., [11,34,23,31,2,28]). Very recently, Goyal [16] has been able to obtain positive results for a large class of functionalities in this setting with the restriction that an honest party uses the *same*, static input in all of the sessions. However these results do not translate to the more general setting where the server may choose different inputs in different sessions. Indeed, in a scenario such as the one discussed earlier, if the server is required to use the same static input in all sessions, it will have to allow all clients the same level of access, and further use its *entire* database in every computation (which may be impractical).

[3] We remark that several works have obtained positive results for "non-standard" simulation-based security, in which the simulator, for example, has access to super-polynomial computational power [30,33,3,29,18,8,12]. In this work, we will focus on security w.r.t. standard (polynomial time) simulation.

Our Question: Static-Input, Fixed-Roles Concurrent Self-composition? In this work, we study the (im)possibility of achieving secure two-party computation with respect to a very weak notion of concurrency as occurs in the *client-server* setting: one server interacts with many clients using the same protocol, always playing the same role, while each (honest) client interacts only with the server. Further, the functionalities being computed are *asymmetric*, in that only the client gets any output.[4] The adversarial model is that either the clients or the server may be adversarial – this is referred to as the *fixed roles* setting in the literature [27].[5,6] We note that this is a very natural setting and captures several important functionalities such as **oblivious transfer**. However, despite extensive prior work on concurrent security, this setting has remained largely unstudied for general secure two-party computation (with the exception of [2,16] as discussed above). Indeed, concurrent security of oblivious transfer under (unbounded) concurrent composition was left was an explicit open problem by Lindell (see [28, pg. 6]).

The importance of the above question stems from the fact that if the answer is positive, then it would enable security of many application scenarios, such as the one discussed at the beginning. On the other hand, if the answer is negative, then the situation would indeed be quite stark, and reaffirm the need for relaxing the standard security definitions. The recent positive results of Goyal [16] may give hope that the answer is positive. Unfortunately, in this work, we show that the latter case holds. We now proceed to describe our results.

1.1 Our Results

We obtain negative results regarding two-party computation with security under concurrent self-composition in the *fixed-roles* setting, along with positive results on UC-secure reductions that were used to get the negative results. (These are formally stated in Section 5.)

- We give a **full-characterization** of security under concurrent self-composition for deterministic *asymmetric* finite functionalities (in which only one party receives output). (Theorem 4.) Specifically, we prove that no *non-trivial* deterministic asymmetric finite functionality can be computed securely under concurrent self composition, while *trivial* ones can be computed UC securely against active adversaries. (A deterministic asymmetric

[4] Functionalities in which both parties receive output (which is not entirely a function of their local input) are more "complex" and already ruled out by Lindell [27], when the inputs could be adaptively chosen.

[5] One could consider the corruption model where one or more clients and the server are corrupted. We note that if communication pattern is "bipartite" (i.e., honest clients do not talk to each other), then this is also covered in the fixed roles setting.

[6] Note that in the setting of *inter-changeable roles* (i.e., where parties may assume different roles in different protocol executions), essentially all non-trivial functionalities allow bit-transmission, and hence the negative results of Lindell [27] are applicable.

functionality is said to be non-trivial if there does not exist a single input for the receiver that "dominates" all other inputs.) Our results are *unconditional* and hold in the *static-input* setting, and thus also rule out the more general case where a party may choose its input in a session *adaptively*, based on the outcomes of the previous sessions. Further, our results are "robust" in the sense that the impossibility is not because honest parties could choose their inputs by reacting to signals sent by corrupt parties over subliminal channels.

In particular, our results rule out concurrent security of 1-out-of-2 oblivious transfer and thus settle the open question of Lindell [28]. Furthermore, to the best of our knowledge, these are the first broad impossibility results in the *static-input* setting. (In contrast, prior works which considered static inputs [2,16] only ruled out very specific functionalities.)

- To prove the above, we first construct a new UC-secure *asynchronous non-interactive* protocol for 1-out-of-2 oblivious transfer ($\mathcal{F}_{\mathsf{OT}}$) using any given non-trivial deterministic asymmetric functionality $\mathcal{F}$, thereby subsuming a result of Kilian [22]. By a non-interactive protocol we mean that the only step in the protocol is to invoke, in parallel, several copies of the given functionality $\mathcal{F}$; we say that the protocol is asynchronous if it remains secure even if the adversary can adaptively schedule the different invocations of $\mathcal{F}$. (Theorem 1.)
  - Combining the above protocol with a UC-secure non-interactive protocol from [19, Full version] for any asymmetric functionality $\mathcal{F}$ given $\mathcal{F}_{\mathsf{OT}}$, we obtain a full characterization of completeness among deterministic asymmetric functionalities with respect to non-interactive reductions. (Theorem 6.)
- We further devise a *composition theorem* for static-input fixed-role concurrent security. (Theorem 3.) This theorem holds only for asynchronous non-interactive protocols. Given this theorem and our asynchronous non-interactive OT protocol, we complete the proof of our main result by establishing the impossibility of static-input, fixed-role concurrently secure protocols for $\mathcal{F}_{\mathsf{OT}}$. (Theorem 2.)

Independent Work. Independent of our work, exciting results concerning the impossibility of concurrently secure computation have been obtained recently by Garg *et al.* We refer the reader to [13], in these proceedings, for more details.

1.2 Our Techniques

Asynchronous Non-interactive Protocol for $\mathcal{F}_{\mathsf{OT}}$. As mentioned above, the first ingredient in our main result, and a contribution of independent interest, is an asynchronous non-interactive protocol for $\mathcal{F}_{\mathsf{OT}}$ given any non-trivial deterministic asymmetric functionality $\mathcal{F}$. To obtain the impossibility result in the static-input setting it is vital that this protocol is non-interactive *and asynchronous*.

Let $\mathcal{F}$ be a *non-trivial* asymmetric functionality that takes inputs x and y from Alice and Bob respectively, and outputs $f(x, y)$ to Bob. The intuitive reason why

such a functionality $\mathcal{F}$ can yield $\mathcal{F}_{\mathsf{OT}}$ is that Bob has no input which will let it learn Alice's input to $\mathcal{F}$ exactly (up to equivalent inputs), where as Alice does not learn anything about Bob's input to $\mathcal{F}$. In particular, one can find two inputs $\hat{y}^0$ and $\hat{y}^1$ for Bob, such that if Alice chooses her input x at random, the only way Bob can learn $f(x, \hat{y}^0)$ with full certainty is if he chooses $\hat{y}^0$ as his input; similarly, unless he deterministically chooses $\hat{y}^1$ as his input, Bob will be left with some entropy regarding $f(x, \hat{y}^1)$. Thus Bob can choose to learn at most one of $f(x, \hat{y}^0)$ and $f(x, \hat{y}^1)$ exactly. There are two main challenges in turning this idea into an implementation of $\mathcal{F}_{\mathsf{OT}}$:

- Bob learns some information about $f(x, \hat{y}^0)$ when he uses $\hat{y}^1$ as an input, and vice versa, whereas in $\mathcal{F}_{\mathsf{OT}}$, he should not learn any information about one of Alice's two inputs (and learn the other one completely).
- Alice and Bob may deviate from the prescribed distributions when choosing their inputs to $\mathcal{F}$. In particular, any solution to the above issue should work even if Bob uses an input other than $\hat{y}^0$ and $\hat{y}^1$.

Kilian [22] handles the issue of active adversaries using properties of a Nash equilibrium of an appropriate zero-sum game. However, this approach (apart from being not asynchronous) appears suitable only for constructing an erasure channel, and does not permit Bob to use an input. (Converting the erasure channel to $\mathcal{F}_{\mathsf{OT}}$ requires interaction.) The way [24] handles active corruption using cut-and-choose checks is highly interactive and again inadequate for our purposes.

One natural approach one could consider is to use an extractor to amplify the uncertainty Bob has about $f(x, \hat{y}^0)$ or $f(x, \hat{y}^1)$ from many invocations of $\mathcal{F}$ (with x independently randomly chosen each time). That is, if Bob has some uncertainty about at least one of the strings, R^0 and R^1 where R^0 (respectively, R^1) is defined as the string of outputs Bob would have received if he chose $\hat{y}^0$ (respectively, $\hat{y}^1$) as his input in all invocations of $\mathcal{F}$, then Alice can choose two seeds for a strong extractor and obtain two masks r_0 and r_1 as the strings extracted from R^0 and R^1 respectively, using these seeds. Unfortunately, *this is not secure in an asynchronous protocol.* Alice must transmit the extractor seeds she picked to Bob (for which she can use instances of $\mathcal{F}$). However, in an asynchronous non-interactive protocol, a corrupt Bob can *first receive the extractor seeds* before picking its inputs for the other instances of $\mathcal{F}$; hence the seeds are not independent of the information Bob obtains about R^0 and R^1, and the guarantees of extraction no more hold.

We get around this by avoiding using *the full power of extractors*, and in fact using a deterministic function in its place. For instance, when f is a boolean function, consider defining r_0 as simply the XOR of all the bits in R^0 (and similarly r_1). Since Alice picks her input x to $\mathcal{F}$ independently for each invocation, this still guarantees that if Bob has uncertainty about sufficiently many bits in R^0, then he has almost no information about r_0.

But using a linear function in place of the extractor will not suffice when Bob can use inputs other than $\hat{y}^0$ and $\hat{y}^1$. In particular, for boolean functions again, if there is an input $\hat{y}^2$ such that $f(x, \hat{y}^2) = f(x, \hat{y}^0) \oplus f(x, \hat{y}^1)$, then using this input in all invocations, Bob can learn $R^0 \oplus R^1$, and $r_0 \oplus r_1$. Our solution to this is to use a simple "quadratic" function, after appropriately mapping Bobs outputs to a field. This lets us ensure that one of the two "extracted" strings r_0 and r_1 remains almost uniformly random, even given the other string.

Impossibility for $\mathcal{F}_{\mathsf{OT}}$. As the next step towards our final impossibility result, we rule out a protocol for concurrent OT even for the case where both parties have fixed roles. The basic idea behind this impossibility result builds on the techniques from [2,16]. The proof proceeds in the following high-level steps; we refer the reader to Section Section 4 for details. Suppose, towards contradiction, we are given a protocol Π_{OT} that securely realizes OT in our setting.

1. First, we will construct an instance of the *chosen protocol attack* for this protocol. More precisely, we will construct a protocol $\widehat{\Pi_{\mathsf{OT}}}$ such that the protocols Π_{OT} and $\widehat{\Pi_{\mathsf{OT}}}$ are *insecure* when executed concurrently. We will have three parties in the system: Alice and Eve running Π_{OT} (as sender and receiver respectively); Eve and David running $\widehat{\Pi_{\mathsf{OT}}}$ (as sender and receiver respectively). In our chosen protocol attack, Eve will be the corrupted party acting as man-in-the-middle between Alice and David and violating security of Π_{OT} (see Section 4 for more details of this step).
2. Next, we will use *one time programs* (OTPs) [15] to eliminate David. In more detail, Eve simply gets a set of one-time programs implementing the next message function of David.
3. To execute these one-time programs, the (possibly adversarial) Eve is required to carry out a number of oblivious transfer invocations. In these invocations, Alice can be given the required key and can act as the sender. Fortunately, oblivious transfer is exactly the functionality that Π_{OT} provides! So these OT invocations can be executed using the protocol Π_{OT}.
4. Thus, now there are only Alice and Eve running a number of concurrent executions of Π_{OT}. Hence, the chosen protocol attack we started with can now be carried out in our setting of concurrent self-composition with static-inputs and fixed-roles.

2 Preliminaries

Below we present some of the important definitions we need. Through out this paper, we denote the security parameter by κ.

Secure Computation under Concurrent Self-composition. In this section, we present the definition for concurrently secure two-party computation.

The definition we give below is an adaptation of the definition of security under concurrent self-composition from [27], but with two further restrictions to the model — *fixed-roles* and *static-inputs* — i.e., there are only two parties engaged in multiple executions of a given protocol, with each playing the same "role" in

all the sessions, and the inputs of the honest parties for each session is fixed in advance. (Recall that since the focus in this work is on obtaining impossibility results, our results are stronger by adding these restrictions.) Some parts of the definition below have been taken almost verbatim from [27].

A two-party functionality[7] $\mathcal{F}$ with input domains X and Y for the two parties is defined by two functions $f_1 : X \times Y \to Z_A$ and $f_2 : X \times Y \to Z_B$, where Z_A, Z_B are the output domains. For most part we shall consider the input and output domains to be finite sets. Such functionalities are called *finite functionalities*. (Again, since our focus is on impossibility results, it is more interesting to show impossibility of finite functionalities than of infinite functionalities.)

We will denote the two parties that wish to jointly instantiate $\mathcal{F}$ as Alice and Bob (or sometimes P_1 and P_2). If Alice's input is $x \in X$ and Bob's input is $y \in Y$, then the functionality would output $f_1(x, y)$ to Alice and $f_2(x, y)$ to Bob (unless aborted by a corrupt party). A functionality is called *asymmetric* if only Bob receives any output; more precisely, in an asymmetric functionality f_1 is the constant function (Alice does receive this fixed output to indicate the termination of execution). An asymmetric functionality will be defined using a single function $f : X \times Y \to Z$.

In this work, we consider a malicious, static adversary. The scheduling of the messages across the concurrent executions is controlled by the adversary. The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario, where a trusted party computes the function output on the inputs of the parties. We do not require fairness (i.e., our impossibility is not a consequence of requiring fairness) and hence in the ideal model, we allow a corrupt party to receive its output in a session and then optionally block the output from being delivered to the honest party, in that session. Unlike in the case of stand-alone computation, in the setting of concurrent executions, the trusted party computes the functionality many times, each time upon different inputs. We refer the reader to the full version of the paper, or [27] for further details of the real and ideal model executions.

The output pair of the ideal-model adversary $\mathcal{S}$ and the honest party (or both parties, when neither corrupt) in an ideal-model execution of a functionality $\mathcal{F}$ with security parameter κ, input vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and auxiliary input z to $\mathcal{S}$, will be denoted as $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\kappa, \boldsymbol{x}, \boldsymbol{y}, z)$. Similarly, the output pair of the real-model adversary $\mathcal{A}$ and the honest party (or parties) in a real-model concurrent execution of a protocol Π with security parameter κ, input vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and auxiliary input z to $\mathcal{A}$ will be denoted by $\text{REAL}_{\Pi,\mathcal{A}}(\kappa, \boldsymbol{x}, \boldsymbol{y}, z)$.

Definition 1 (Security under Concurrent Self-Composition). *A protocol Π is said to* securely realize *a functionality $\mathcal{F}$* under concurrent composition *if for every real model non-uniform probabilistic polynomial-time adversary $\mathcal{A}$, there exists an ideal-model non-uniform probabilistic expected polynomial-time adversary $\mathcal{S}$, such that for all polynomials $m = m(\kappa)$, every $z \in \{0,1\}^*$,*

[7] All functionalities considered in this paper are (unfair) secure function evaluation (SFE) functionalities. For simplicity we shall not explicitly qualify them as SFE or non-reactive.

every pair of input vectors $\boldsymbol{x} \in X^m$, $\boldsymbol{y} \in Y^m$, $\{\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\kappa, \boldsymbol{x}, \boldsymbol{y}, z)\}_{\kappa\in\mathbb{N}}$ *and* $\{\text{REAL}_{\Pi,\mathcal{A}}(\kappa, \boldsymbol{x}, \boldsymbol{y}, z)\}_{\kappa\in\mathbb{N}}$ *are computationally indistinguishable.*

We shall also consider a few *stronger* security definitions, that we achieve in our positive results (used in proving the main negative result). Firstly, we can require the above to hold even with probabilistic *expected* polynomial time adversaries in the real-model. Secondly, we could in fact allow the real-model adversary to be computationally unbounded, and allow the ideal-model adversary to be unbounded too.[8] Also, we shall consider Universally Composable (UC) security [4,5]. We shall not present the details of the UC security definition (which has several slight variants, with essentially the same properties), but remark that it implies concurrent self-composition and more.

Finally, sometimes we permit the real model protocols to invoke ideal functionalities as sub-protocols (denoted like $\Pi^{\mathcal{F}}$, where $\mathcal{F}$ is the ideal functionality invoked by the parties in Π). In all the instances we do this, we can actually consider UC-security of the protocol; however this definition can be easily generalized to security under concurrent self-composition too, and would be useful in stating a composition theorem (without involving UC-security).

Non-interactive Protocols and Asynchronous Non-interactive Protocols. A two-party protocol $\Pi^{\mathcal{F}}$ (i.e., in the $\mathcal{F}$-hybrid model) is called a *non-interactive protocol* if the protocol has the following structure: the two parties carry out local computation; then together they invoke one or more *parallel, synchronized* sessions of $\mathcal{F}$; then they carry out more local computation and produce outputs. By synchronized sessions we mean that even corrupt players can invoke these sessions only in parallel.[9] We shall also require that all copies of $\mathcal{F}$ are invoked with the same fixed roles for the two parties.

A two-party protocol $\Pi^{\mathcal{F}}$ is called an *asynchronous non-interactive protocol* if the protocol has the above structure, but the parallel sessions of $\mathcal{F}$ are parallel but asynchronous. By this we mean that a corrupt player can invoke these sessions in arbitrary order, choosing the inputs for each session based on the outputs from prior sessions; the honest players have to choose their inputs *a priori* and remain oblivious to the order in which the sessions are invoked.

One Time Programs. A one-time program (OTP) [15] for a function f allows a party to evaluate f on a single input x chosen by the party dynamically. As introduced in [15], an OTP is implemented as a package consisting of some software and hardware tokens (specifically *one time memory* tokens), that essentially provided the party with *asynchronous* access to several oblivious transfer invocations. We shall treat OTPs as an *asynchronous non-interactive protocol* in the OT-hybrid model (as was done in [17]), that securely realizes an

[8] This is not a strengthening of the security definition, as the ideal-model is more relaxed now; however, the protocols we shall consider will satisfy this definition in addition to the other definitions.

[9] A more accurate notation for such a protocol that invokes at most t sessions of $\mathcal{F}$, would be $\Pi^{\mathcal{F}^t}$, where $\mathcal{F}^t$ stands for a non-reactive functionality implementing t independent copies of f. For simplicity we do not use this notation.

asymmetric "one-time program functionality" $\mathcal{F}_{\mathsf{OTP}}$ against corruption of the receiver alone. $\mathcal{F}_{\mathsf{OTP}}$ accepts a circuit f from the party P_1 and an input x from the party P_2, and returns $f(x)$ to P_2 (and notifies P_1). We refer the reader to [17] or the full version for details.

Definition 2 (One-Time Program). *(Adapted from [15,17]) A* one-time program (OTP) scheme *is an asynchronous two-party non-interactive protocol $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ (in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model) that UC-securely realizes $\mathcal{F}_{\mathsf{OTP}}$ when the adversary is allowed to corrupt only P_2.*

In the text, we shall refer to the collection of inputs P_1 provides to the $\mathcal{F}_{\mathsf{OT}}$ instances in an execution of $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ when its input is a circuit f, as an *OTP for f*. We note that OTPs exist if a (standalone secure) OT protocol exists [17], which in turn exists if a concurrent secure OT protocol exists. Thus, for proving the impossibility of concurrent secure OT protocols, we can assume the existence of OTPs "for free."

Our use of OTP parallels the use of garbled circuits in the impossibility result in [2]. Following [16], we use OTPs instead of garbled circuits, since they have stronger security properties (namely, security against an actively corrupt receiver), and allows one to simplify the construction in [2]. We refer the reader to the full version for further discussion.

3 A Non-interactive Protocol for OT from Any Non-trivial Asymmetric SFE

The goal of this section is to obtain an asynchronous, non-interactive protocol for $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}$-hybrid model, for *any* finite deterministic "non-trivial" asymmetric functionality $\mathcal{F}$. For our purposes, we define a *trivial* asymmetric functionality as follows.

Definition 3 (Dominating input.). *In an asymmetric functionality defined by a function $f : X \times Y \to Z$, an input $y \in Y$ for the receiver is said to* dominate *another input $y' \in Y$ if $\forall x_1, x_2 \in X$, $f(x_1, y) = f(x_2, y) \implies f(x_1, y') = f(x_2, y')$.*

Definition 4 (Trivial functionality.). *An asymmetric functionality is called* trivial *if there exists an input for Bob that dominates all of its other inputs.*

We now state the main result in this section.

Theorem 1. *For any non-trivial asymmetric functionality $\mathcal{F}$, there exists an asynchronous, non-interactive protocol $\Pi^{\mathcal{F}}$ that UC-securely realizes $\mathcal{F}_{\mathsf{OT}}$, even against computationally unbounded adversaries.*

Background. In [22] Kilian presented an elegant protocol to show that any non-trivial asymmetric functionality is complete for security against active adversaries. The protocol, which constructs an erasure channel from a non-trivial asymmetric functionality $\mathcal{F}$, achieves security against active adversaries

by using an input distribution to $\mathcal{F}$ derived from the Nash equilibrium of a zero-sum game defined using $\mathcal{F}$. Kilian shows that an adversary deviating from the prescribed distribution cannot change the erasure probability in the protocol. This rather enigmatic protocol does invoke $\mathcal{F}$ with fixed roles, but is not useful for showing impossibility of concurrent secure protocol for $\mathcal{F}$, because it is modestly interactive (two steps which should occur one after the other) and more importantly, because it yields only an erasure channel and not a $\binom{2}{1}$-OT.[10] The only known substitute for this protocol, by [24], is much more interactive, involving several rounds of communication in both directions, apart from the invocation of $\mathcal{F}$. Our challenge is to devise an *asynchronous non-interactive* protocol which uses $\mathcal{F}$ with fixed-roles and directly yields $\binom{2}{1}$-OT. Being non-interactive and asynchronous requires that all the sessions are invoked together by an honest party, but the adversary is allowed to schedule them adaptively, and base its inputs for later sessions based on the outputs from earlier sessions (rushing adversary). As mentioned in Section 1.2, this rules out some standard techniques like privacy amplification (using extractors), since the adversary can learn the seed used for extraction *before* it extracts partial information about a string to which the extraction will be applied.

We present a new and simple non-interactive OT protocol which uses a simple non-linear "extraction" strategy that does not require a seed, but is sufficient to amplify the uncertainty about certain values into almost zero information about at least one of two extracted values. Our protocol is in fact UC-secure (in the PPT as well as information theoretic settings) and is asynchronous.

3.1 The New Protocol

Our asynchronous non-interactive protocol UC-securely realizes the $\binom{2}{1}$-OT functionality $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F}$ is a finite[11] asymmetric functionality defined by a function $f : X \times Y \to Z$, and $\mathcal{F}$ is *not trivial*. Note that (since domination is transitive) this means that there are at least *two inputs* in Y — denoted by $\hat{y}^0$ and $\hat{y}^1$ — which are not dominated by any other input $y \in Y$.

To define the protocol, first we pick a prime number $p \geq \min\{|X|, |Z|\}$. Then we can define two maps to relabel the columns corresponding to $\hat{y}^0$ and $\hat{y}^1$ in the function table of f using elements in $\mathbb{Z}_p$ – i.e., two injective functions $N_0 : Z_0 \to \mathbb{Z}_p$, $N_1 : Z_1 \to \mathbb{Z}_p$, where $Z_b = \{f(x, \hat{y}^b) | x \in X\}$ – such that there exist $\hat{x}^0, \hat{x}^1 \in X$ satisfying the following.

[10] Our impossibility result in Section 4 holds only for $\binom{2}{1}$-OT, and not for erasure channels. Indeed, using techniques in [16,18], it would be possible to construct concurrent secure protocols for an asymmteric functionality like erasure channels, in which Bob does not have any input.

[11] For simplicity, following [22], we require $|X|, |Y|$ to be constant. But the security of the protocol presented here only requires $|X|$ to be $\mathsf{poly}(\kappa)$ where κ is the security parameter. Alternatively, if $|Y|$ is $\mathsf{poly}(\kappa)$, we can have the protocol use a uniform distribution over a subset of X of size at most $2|Y|$, restricted to which the functionality is still non-trivial.

$$\begin{bmatrix} N_0\big(f(\hat{x}^0,\hat{y}^0)\big) & N_1\big(f(\hat{x}^0,\hat{y}^1)\big) \\ N_0\big(f(\hat{x}^1,\hat{y}^0)\big) & N_1\big(f(\hat{x}^1,\hat{y}^1)\big) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

To illustrate this for the case of boolean functions ($p = |Z| = 2$), we note that for a non-trivial boolean functionality, the function table of f, restricted to the two columns corresponding to $\hat{y}^0$ and $\hat{y}^1$ must have one of the following minors (possibly with the columns reordered, in the last case): $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{smallmatrix}\right]$ $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$ $\left[\begin{smallmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{smallmatrix}\right]$. In the first two cases N_0, N_1 can be the identity map and in the last case exactly one of N_0, N_1 would be the identity. The formal proof is given in the full version.

The protocol is now defined as follows, in terms of the inputs $\hat{y}^0, \hat{y}^1 \in Y$, $\hat{x}^0, \hat{x}^1 \in X$ and the functions N_0, N_1 as identified above.

Alice's Program: Alice's input is two bits s_0, s_1.

1. Alice carries out the following computations:
   - For $i = 1$ to 2κ, pick $x_i \leftarrow X$.
   - For each i, let $R_i^0 = N_0(f(x_i, \hat{y}^0))$ and $R_i^1 = N_1(f(x_i, \hat{y}^1))$.
   - Let $r_0 = \sum_{i=1}^{\kappa} R_i^0 R_{\kappa+i}^0$, and $r_1 = \sum_{i=1}^{\kappa} R_i^1 R_{\kappa+i}^1$.
   - Let $m_0 = s_0 + r_0$ and $m_1 = s_1 + r_1$ (interpreting bits s_0, s_1 as elements in $\{0,1\} \subseteq \mathbb{Z}_p$).
2. Alice invokes, in parallel, several copies of $\mathcal{F}$ with the following inputs:
   - Sessions $i = 1$ to 2κ with inputs x_i.
   - $2\lceil \log p \rceil$ more sessions to "communicate" the bits of (m_0, m_1): in a session to send a bit 0, use input $\hat{x}^0$, and in a session to send a bit 1, use input $\hat{x}^1$.
3. If all $\mathcal{F}$ sessions are completed, then Alice completes the protocol (i.e., outputs an acknowledgment).

Bob's Program: Bob's input is a choice bit b.

1. Bob invokes the same copies of $\mathcal{F}$ as Alice with the following inputs:
   - In each of the 2κ sessions numbered $i = 1$ to 2κ, use input $\hat{y}^b$, and obtain R_i^b.
   - In each of the sessions used for communication, use input $\hat{y}^0$; obtain all bits of (m_0, m_1).
2. If all sessions of $\mathcal{F}$ are completed, compute $r_b = \sum_{i=1}^{\kappa} R_i^b R_{\kappa+i}^b$, and $s_b = m_b - r_b$. Then, if $s_b = 0$ output 0, otherwise output 1.

Below we sketch the intuition behind the proof of security. The formal simulation and proof is presented in the full version.

The protocol is easily seen to be correct. Also, security when Alice is corrupt is easy to argue as $\mathcal{F}$ does not give any output to Alice. (The simulator can extract Alice's inputs by considering what Bob would output when his input is $b = 0$ and $b = 1$.) The interesting case is when Bob is corrupt.

Note that in the protocol, all the instances of $\mathcal{F}$ are invoked in parallel by the honest parties, but a rushing adversary that corrupts Bob can dynamically schedule the sessions and also choose its inputs for these sessions adaptively,

based on the outputs received thus far. The main idea behind showing security against Bob is that one of r_0 and r_1 appears completely random to Bob (even given the other one), no matter how he chooses his inputs y_i. For this we define a pair of $\mathcal{F}$ sessions $(i, \kappa + i)$ to be a "0-undetermined pair" if $R_i^0 R_{\kappa+i}^0$ is not completely determined by the view of the adversary in those sessions, combined with $R_i^1 R_{\kappa+i}^1$; simiarly we define the pair to be a "1-undetermined pair" if $R_i^1 R_{\kappa+i}^1$ is not completely determined by the view of the adversary in those sessions, combined with $R_i^0 R_{\kappa+i}^0$. Then, it can be shown that there will be a constant probability that any pair will be either 0-undetermined or 1-undetermined.

Note that for any input y that the adversary chooses in the first session out of a pair, it does not dominate at least one of $\hat{y}^0$ and $\hat{y}^1$. With constant probability the adversary will be left with some uncertainty about either $f(x, \hat{y}^0)$ or $f(x, \hat{y}^1)$, where x stands for Alice's input in this session. Suppose $f(x, \hat{y}^0)$ is not fully determined. Now, with a further constant probability Alice's input in the other session in the pair would be $x' = \hat{x}^1$. Then, even if Bob learns x' exactly, he remains uncertain of $f(x, \hat{y}^0).f(x', \hat{y}^0) = f(x, \hat{y}^0) \cdot 1$. This uncertainty remains even if Bob learns $f(x, \hat{y}^1).f(x', \hat{y}^1) = f(x, \hat{y}^1) \cdot 0$, as it is independent of x.

This slight uncertainty about a term in r_0 or r_1 gets amplified by addition in $\mathbb{Z}_p$, as summarized in the following lemma.

Lemma 1. *Let p be a fixed prime number. Let D be a distribution over $\mathbb{Z}_p$, such that for some constant $\epsilon > 0$, for all $z \in \mathbb{Z}_p$, $\Pr_{a \leftarrow D}[a = z] < 1 - \epsilon$. For any positive integer N, let $a_1, \cdots, a_N$ be i.i.d random variables sampled according to D. Then the statistical distance between the distribution of $\sum_{i=1}^{N} a_i$ (summation in $\mathbb{Z}_p$) and the uniform distribution over $\mathbb{Z}_p$ is negligible in N.*

This follows from an elementary argument. The proof is given in the full version.

To complete the intuitive argument of security, we need to also consider how the simulator for Bob can extract his input bit b. The simulator would let Bob schedule several sessions, until a constant fraction of the pairs $(i, \kappa + i)$ have had both sessions completed. At this point Bob would have already accumulated sufficient uncertainty about one of r_0 and r_1, say $r_{\overline{b}}$, that will remain no matter what he learns from the remaining sessions. Further, not having invoked the remaining sessions will ensure that at this point he still has no information about the other element r_b. So, at this point, the simulator will send $b = 1 - \overline{b}$ to the ideal $\mathcal{F}_{\mathsf{OT}}$ and learn what r_b should be, and can henceforth "pretend" that it was always using that value of r_b. Pretending thus (i.e., sampling Alice's inputs for the remaining sessions according to the right conditional distribution) can be efficiently done by rejection sampling (since p is a constant).

4 Impossibility of Concurrent Oblivious Transfer

Theorem 2. *There does not exist a protocol that securely realizes $\mathcal{F}_{\mathsf{OT}}$ under concurrent self-composition even in the static-input, fixed-role setting.*

Suppose towards contradiction, we are given a protocol Π_{OT} that securely realizes the OT functionality under static-input, fixed-roles concurrent self-composition. We will exhibit a set of inputs (for sender and receiver) and a real-world adversarial receiver that is able to perform a concurrent attack and manages to learn a secret value, called secret with probability 1. We will then prove that if there exists an adversarial receiver in the ideal world that is able to learn secret with high enough probability, then we can break the stand-alone security of Π_{OT} against a cheating sender, thus arriving at a contradiction. Our formal proof follows the high-level structure as discussed in Section 1.2. We now proceed to give details of each of the steps.

Chosen Protocol Attack. Let Π_{OT} be a protocol that securely realizes the OT functionality. To fix notation, let us consider two parties Alice and Bob that are executing an instance of Π_{OT}. Say that Alice's input bits are denoted by s_0 and s_1 and Bob's input bit is b. Upon successful completion of the protocol, Bob obtains s_b. Next, let $\widehat{\Pi_{\mathsf{OT}}}$ be a slightly modified version of Π_{OT} where the receiver Bob also has both of Alice's inputs, s_0 and s_1 in addition to his bit b. In $\widehat{\Pi_{\mathsf{OT}}}$, Bob and Alice run an execution of Π_{OT} with inputs b and s_0, s_1 respectively. Upon receiving an output s^*, Bob checks whether $s^* = s_b$. If so, he sends $\mathsf{secret} = s_{\bar{b}}$ to Alice.

Now, consider the following scenario involving three parties Alice, Eve and David. Alice holds input bits s_0, s_1, while David holds s_0, s_1, as well as a random input bit b. Alice plays the sender with receiver Eve in an execution of Π_{OT}, and Eve plays sender with receiver David in an execution of $\widehat{\Pi_{\mathsf{OT}}}$. It is clear that a malicious Eve can launch "man-in-the-middle" attack, where she simply forwards Alice's message to David and David's back to Alice, in order to learn the value $\mathsf{secret} = s_{\bar{b}}$. However, note that if the execution of Π_{OT} is replaced with an ideal call to the OT functionality, then the attack does not carry through.

Converting $\widehat{\Pi_{\mathsf{OT}}}$ to Π_{OT}. Note that the above attack is valid in the setting of concurrent general composition. However, we are interested in the setting of concurrent self composition, where only Π_{OT} is executed concurrently. Towards this end, in this section, we will convert the protocol $\widehat{\Pi_{\mathsf{OT}}}$ into a series of OT calls which can be implemented by Π_{OT}. Since Π_{OT} executed concurrently with $\widehat{\Pi_{\mathsf{OT}}}$ is insecure, this will allow us to show that Π_{OT} is insecure under concurrent self composition.

To begin, we transform the protocol $\widehat{\Pi_{\mathsf{OT}}}$ run by Eve (as sender) and David into a sequence of calls made by Eve to an ideal reactive functionality (with inputs fixed in advance). As in [2], it is natural to instantiate this ideal functionality by the *next message function* $\mathcal{F}^{\mathsf{David}}$ of David's strategy in protocol $\widehat{\Pi_{\mathsf{OT}}}$. Then Eve can simulate the interaction with David by invoking $\mathcal{F}^{\mathsf{David}}$ each time she expects a message from David. More precisely, the inputs to $\mathcal{F}^{\mathsf{David}}$ will be the bits s_0 and s_1, a random bit b, a message from Eve denoted by e_i, and a state state_{i-1} (since $\mathcal{F}^{\mathsf{David}}$ is a reactive functionality, it needs to know state_{i-1} in order to compute state_i), and the output of $\mathcal{F}^{\mathsf{David}}$ will be David's i^{th} message

in $\widehat{\Pi_{\mathsf{OT}}}$ and state_i. Thus, Eve can, as before, play the receiver in an execution of Π_{OT} with Alice as the sender and carry out the man in the middle attack by invoking $\mathcal{F}^{\mathsf{David}}$. We will denote the real world attacker played by Eve as $\hat{\mathcal{E}}^{\mathsf{real}}$.

As the next step, we will to replace the ideal calls to $\mathcal{F}^{\mathsf{David}}$ made by $\hat{\mathcal{E}}^{\mathsf{real}}$ by a series of OTs executed between Alice and $\hat{\mathcal{E}}^{\mathsf{real}}$. Let n denote the number of messages that David sends in protocol $\widehat{\Pi_{\mathsf{OT}}}$. Then, consider the one time programs $\mathsf{OTP\text{-}msg}_1, \ldots, \mathsf{OTP\text{-}msg}_n$ where $\mathsf{OTP\text{-}msg}_i$ computes the i^{th} next message function of David. We will provide Alice with all the keys $\{\mathsf{keys}_i\}_{i=1}^n$ to the OTPs. Then, to evaluate the i^{th} OTP, $\hat{\mathcal{E}}^{\mathsf{real}}$ executes (multiple sessions of) Π_{OT} with Alice to obtain the keys corresponding to her input.

Also note that OTPs are *stateless* by definition, but David's next message functionality is *stateful*. We handle this by allowing each $\mathsf{OTP\text{-}msg}_i$ to pass its private state to $\mathsf{OTP\text{-}msg}_{i+1}$ using standard techniques. Specifically, we will add to the (fixed) inputs of $\mathsf{OTP\text{-}msg}_i$ a key K for an authenticated encryption scheme. Now, $\mathsf{OTP\text{-}msg}_i$ outputs not only David's next message d_i, but also an authenticated encryption of its resultant state, denoted by $\tau_i = \mathsf{Enc}_K(\mathsf{state}_i)$. As input, $\mathsf{OTP\text{-}msg}_i$ requests not only message e_i, but also a valid authenticated encryption τ_{i-1} such that $\mathsf{Dec}_K(\tau_{i-1}) = \mathsf{state}_{i-1}$. This forces the functionality $\mathcal{F}^{\mathsf{David}}$ implemented by OTPs, to be invoked in proper order. For the rest of the article, we will assume that any OTPs we use are made "stateful" in this way.

Note that there are two kinds of executions of Π_{OT} being carried out between Alice and $\hat{\mathcal{E}}^{\mathsf{real}}$: the "main" OT execution where Alice uses inputs s_0, s_1, and the additional OT executions that allow Eve to obtain keys for the one time programs. For clarity of exposition, we will refer to the "main" execution as $\Pi_{\mathsf{OT}}^{\mathsf{main}}$ and each of the remaining ones as $\Pi_{\mathsf{OT}}^{\mathsf{David}}$.

Attack in the Real World. Now, we will describe the explicit execution of OTs between Alice and $\hat{\mathcal{E}}^{\mathsf{real}}$, where $\hat{\mathcal{E}}^{\mathsf{real}}$ recovers $\mathsf{secret} = s_{\bar{b}}$. The protocol is as follows:

Alice's program: Alice is given input bits s_0, s_1 for $\Pi_{\mathsf{OT}}^{\mathsf{main}}$ and all the one time program keys $\{\mathsf{keys}_i\}_{i=1}^n$ for $\mathsf{OTP\text{-}msg}_1, \ldots, \mathsf{OTP\text{-}msg}_n$. She behaves honestly according to the protocol Π_{OT} and responds honestly to all OT invocations made by $\hat{\mathcal{E}}^{\mathsf{real}}$.

$\hat{\mathcal{E}}^{\mathsf{real}}$'s Program: $\hat{\mathcal{E}}^{\mathsf{real}}$ is given input bit $\hat{b}$ for $\Pi_{\mathsf{OT}}^{\mathsf{main}}$ and the one time programs $\{\mathsf{OTP\text{-}msg}_i\}_{i=1}^n$ where $\mathsf{OTP\text{-}msg}_i$ computes the i^{th} next message function of David. Let s_0, s_1 and b denote the fixed inputs (hardwired) in the OTPs such that $\mathsf{secret} = s_{\bar{b}}$. For $i = 1, \ldots, n$, do:

1. Upon receiving i^{th} message from Alice in $\Pi_{\mathsf{OT}}^{\mathsf{main}}$, say a_i, suspend (temporarily) the ongoing $\Pi_{\mathsf{OT}}^{\mathsf{main}}$ session and start a new $\Pi_{\mathsf{OT}}^{\mathsf{David}}$ session with Alice to compute the i^{th} message that David would have sent in response had he received a_i from Eve. Depending on a_i, retrieve the corresponding keys keys_i from Alice to input to $\mathsf{OTP\text{-}msg}_i$. End the $\Pi_{\mathsf{OT}}^{\mathsf{David}}$ protocol.
2. Run $\mathsf{OTP\text{-}msg}_i$ with keys keys_i and obtain output d_i.
3. If $i \leq n-1$, resume the suspended $\Pi_{\mathsf{OT}}^{\mathsf{main}}$ protocol and send d_i back to Alice as response. Otherwise, output the value secret received from $\mathsf{OTP\text{-}msg}_n$.

Thus, using a sequence of OT executions, a dishonest $\hat{\mathcal{E}}^{\mathsf{real}}$ is able to recover $\mathsf{secret} = s_{\bar{b}}$ with probability 1.

Infeasibility of Ideal World Attacker. Suppose for contradiction that there exists an ideal world attacker $\hat{\mathcal{E}}^{\mathsf{ideal}}$ that succeeds in outputting secret with probability $1 - \mathrm{negl}$. Then, we can construct a stand-alone cheating sender that executes Π_{OT} with an honest receiver $\mathcal{R}$ and uses $\hat{\mathcal{E}}^{\mathsf{ideal}}$ to learn $\mathcal{R}$'s secret input bit b with non-negligible probability. Please see the full version for details.

5 Putting It All Together

By combining the results from the previous sections, we now state our final results. All functionalities referred to below are 2-party finite deterministic non-reactive functionalities. For brevity and clarity we drop these qualifiers.

In the full version, we prove the following composition theorem for security under concurrent self-composition in the static-input, fixed-role setting.

Theorem 3. *Suppose $\Pi^{\mathcal{F}}$ is an asynchronous, non-interactive protocol that securely realizes $\mathcal{G}$ under concurrent self-composition in the static-input, fixed-role setting, and is secure against expected PPT adversaries. Also, suppose ρ is a protocol (in the plain model) that securely realizes $\mathcal{F}$ under concurrent self-composition in the static-input, fixed-role setting. Then Π^{ρ} securely realizes $\mathcal{G}$ under concurrent self-composition in the static-input, fixed-role setting.*

Since UC-security implies security under concurrent self-composition in the static-input, fixed-role setting, by composing the OT protocol in Theorem 1 with a hypothetical protocol for any non-trivial asymmetric functionality $\mathcal{F}$ (using Theorem 3), we will obtain a protocol for $\mathcal{F}_{\mathsf{OT}}$, contradicting Theorem 2. This gives our main impossibility result:

Theorem 4. *For any non-trivial asymmetric functionality $\mathcal{F}$, there does not exist a protocol (in the plain model) that securely realizes $\mathcal{F}$ under self-composition even in the static-input, fixed-role setting. (On the other hand, every trivial asymmetric functionality has a UC-secure protocol.)*

Another consequence of the protocol in Theorem 1 is to give a characterization of functionalities that are *non-interactively complete* against active adversaries. This is because $\mathcal{F}_{\mathsf{OT}}$ itself has this property, as was shown by the following non-interactive (but not asynchronous) protocol from [19].

Theorem 5. *[19, Full version] For any asymmetric functionality $\mathcal{G}$, there exists a non-interactive protocol $\Phi^{\mathcal{F}_{\mathsf{OT}}}$ that UC-securely realizes $\mathcal{G}$, even against computationally unbounded adversaries.*

Since the protocols in our positive results above are UC-secure, by the UC theorem their composition is secure. Further, composing a non-interactive protocol in $\mathcal{F}_{\mathsf{OT}}$-hybrid with a non-interactive protocol for $\mathcal{F}_{\mathsf{OT}}$ in $\mathcal{F}$-hybrid gives a non-interactive protocol in $\mathcal{F}$-hybrid. This gives us the following characterization:

Theorem 6. *In the class of asymmetric functionalities, every non-trivial functionality is non-interactively complete with respect to UC security (against active adversaries). That is, for any two asymmetric functionalities $\mathcal{F}$, $\mathcal{G}$, if $\mathcal{F}$ is non-trivial, then there exists a non-interactive protocol $\Psi^{\mathcal{F}}$ that UC-securely realizes $\mathcal{G}$, even against computationally unbounded adversaries.*

Acknowledgements. Research supported in part from a DARPA/ONR PROCEED award, NSF grants 1136174, 1118096, 1065276, 0916574, 0830803 and 0747027, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, NSF or the U.S. Government.

References

1. Barak, B., Canetti, R., Nielsen, J.B., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: FOCS (2004)
2. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: FOCS (2006)
3. Barak, B., Sahai, A.: How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In: FOCS (2005)
4. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS (2001)
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols (2005), `http://eprint.iacr.org/2000/067`
6. Canetti, R., Fischlin, M.: Universally Composable Commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)
7. Canetti, R., Kushilevitz, E., Lindell, Y.: On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 68–86. Springer, Heidelberg (2003)
8. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: FOCS (2010)
9. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC (2002)
10. Canetti, R., Pass, R., Shelat, A.: Cryptography from sunspots: How to use an imperfect reference string. In: FOCS (2007)
11. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC (1998)
12. Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently Secure Computation in Constant Rounds. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 99–116. Springer, Heidelberg (2012)
13. Garg, S., Kumarasubramanian, A., Ostrovsky, R., Visconti, I.: Impossibility Results for Static Input Secure Computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 419–436. Springer, Heidelberg (2012)
14. Goldreich, O., Micali, S., Wigderson, A.: How to play ANY mental game. In: STOC (1987)

15. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-Time Programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
16. Goyal, V.: Positive results for concurrently secure computation in the plain model. IACR Cryptology ePrint Archive 2011, 602 (2011)
17. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding Cryptography on Tamper-Proof Hardware Tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg (2010)
18. Goyal, V., Jain, A., Ostrovsky, R.: Password-Authenticated Session-Key Generation on the Internet in the Plain Model. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 277–294. Springer, Heidelberg (2010)
19. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding Cryptography on Oblivious Transfer – Efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008), full version on `http://www.cs.uiuc.edu/~mmp/`
20. Katz, J.: Universally Composable Multi-party Computation Using Tamper-Proof Hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
21. Kidron, D., Lindell, Y.: Impossibility results for universal composability in public-key models and with fixed inputs. J. Cryptology 24(3) (2011)
22. Kilian, J.: More general completeness theorems for secure two-party computation. In: STOC (2000)
23. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-logarithm rounds. In: STOC (2001)
24. Kraschewski, D., Müller-Quade, J.: Completeness Theorems with Constructive Proofs for Finite Deterministic 2-Party Functions. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 364–381. Springer, Heidelberg (2011)
25. Lin, H., Pass, R., Venkitasubramaniam, M.: A unified framework for concurrent security: universal composability from stand-alone non-malleability. In: STOC (2009)
26. Lindell, Y.: General composition and universal composability in secure multi-party computation. In: FOCS (2003)
27. Lindell, Y.: Lower Bounds for Concurrent Self Composition. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 203–222. Springer, Heidelberg (2004)
28. Lindell, Y.: Lower bounds and impossibility results for concurrent self composition. J. Cryptology 21(2) (2008)
29. Micali, S., Pass, R., Rosen, A.: Input-indistinguishable computation. In: FOCS (2006)
30. Pass, R.: Simulation in Quasi-Polynomial Time, and Its Application to Protocol Composition. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 160–176. Springer, Heidelberg (2003)
31. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS (2002)
32. Prabhakaran, M., Rosulek, M.: Cryptographic Complexity of Multi-Party Computation Problems: Classifications and Separations. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 262–279. Springer, Heidelberg (2008)
33. Prabhakaran, M., Sahai, A.: New notions of security: achieving universal composability without trusted setup. In: STOC (2004)
34. Richardson, R., Kilian, J.: On the Concurrent Composition of Zero-Knowledge Proofs. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 415–431. Springer, Heidelberg (1999)
35. Yao, A.C.: How to generate and exchange secrets. In: FOCS (1986)