On the Limitations of Universally Composable Two-Party Computation Without Set-up Assumptions

Ran Canetti[1], Eyal Kushilevitz[2⋆], and Yehuda Lindell[1]

[1] IBM T.J.Watson Research, 19 Skyline Drive, Hawthorne NY 10532, USA.
canetti@watson.ibm.com, lindell@us.ibm.com

[2] Computer Science Department, Technion, Haifa 32000, Israel.
eyalk@cs.technion.ac.il

Abstract. The recently proposed *universally composable* (UC) *security* framework, for analyzing security of cryptographic protocols, provides very strong security guarantees. In particular, a protocol proven secure in this framework is guaranteed to maintain its security even when deployed in arbitrary multi-party, multi-protocol, multi-execution environments. Protocols for securely carrying out essentially any cryptographic task in a universally composable way exist, both in the case of an honest majority (in the plain model, i.e., without set-up assumptions) and in the case of no honest majority (in the common reference string model). However, in the plain model, little was known for the case of no honest majority and, in particular, for the important special case of two-party protocols.

We study the feasibility of universally composable two-party *function evaluation* in the plain model. Our results show that very few functions can be computed in this model so as to provide the UC security guarantees. Specifically, for the case of *deterministic* functions, we provide a full *characterization* of the functions computable in this model. (Essentially, these are the functions that depend on at most one of the parties' inputs, and furthermore are "efficiently invertible" in a sense defined within.) For the case of *probabilistic* functions, we show that the only functions computable in this model are those where one of the parties can essentially uniquely determine the joint output.

1 Introduction

Traditionally, cryptographic protocol problems were considered in a model where the only involved parties are the actual participants in the protocol, and only a single execution of the protocol takes place. This model allows for relatively concise problem statements, simplifies the design and analysis of protocols, and is a natural choice for the initial study of protocols. However, this model of "stand-alone computation" does not fully capture the security requirements from cryptographic protocols in modern computer networks. In such networks, a protocol

⋆ Part of this work was done while the author was a visitor at IBM T.J.Watson Research Center.

execution may run concurrently with an unknown number of other copies of the protocol and, even worse, with unknown, arbitrary protocols. These arbitrary protocols may be executed by the same parties or other parties, they may have potentially related inputs and the scheduling of message delivery may be adversarially coordinated. Furthermore, the local outputs of a protocol execution may be used by other protocols in an unpredictable way. These concerns, or "attacks" on a protocol are not captured by the stand-alone model. Indeed, over the years definitions of security became more and more sophisticated and restrictive, in an effort to guarantee security in more complex, multi-execution environments. (Examples include [GK88,NY90,B96,DDN00,DNS98,RK99,GM00,CK01] and many more). However, in spite of the growing complexity, none of these notions guarantee security in arbitrary multi-execution, multi-protocol environments.

A recently proposed alternative approach to guaranteeing security in arbitrary protocol environments is to use notions of security that are preserved under general protocol composition. Specifically, a general framework for defining security of protocols has been proposed [C01]. In this framework (called the universally composable (UC) security framework), protocols are designed and analyzed as stand-alone. Yet, once a protocol is proven secure, it is guaranteed that the protocol remains secure even when composed with an unbounded number of copies of either the same protocol or other unknown protocols. (This guarantee is provided by a general *composition theorem.*)

UC notions of security for a given task tend to be considerably more stringent than other notions of security for the same task. Consequently, many known protocols (e.g., the general protocol of [GMW87], to name one) are *not* UC-secure. Thus, the feasibility of realizing cryptographic tasks requires re-investigation within the UC framework. Let us briefly summarize the known results.

In the case of a majority of honest parties, there exist UC secure protocols for computing any functionality [C01] (building on [BGW88,RB89,CFGN96]). Also, in the honest-but-curious case (i.e., when even corrupted parties follow the protocol specification), UC secure protocols exist for essentially any functionality [CLOS02]. However, the situation is different when no honest majority exists and the adversary is malicious (in which case the corrupted parties can arbitrarily deviate from the protocol specification). In this case, UC secure protocols have been demonstrated for a number of specific (but important) functionalities such as key exchange and secure communication, assuming authenticated channels [C01]. However, it has also been shown that in the plain model (i.e., assuming authenticated channels, but without any additional set-up assumptions), there are a number of natural two-party functionalities that *cannot* be securely realized in the UC framework. These include coin-tossing, bit commitment, and zero-knowledge [C01,CF01]. In contrast, in the common reference string model, UC secure protocols exist for essentially any two-party and multi-party functionality, with any number of corrupted parties [CLOS02].

A natural question that remains open is what are the tasks that can be securely realized in the UC framework, with a dishonest majority, a malicious adversary and without set-up assumptions (i.e., in the plain model). We note that previous results left open the possibility that useful relaxations of the coin-tossing, bit commitment and zero-knowledge functionalities can be securely realized. (Indeed, our research began with an attempt to construct UC protocols for such relaxations.)

Our results. We concentrate on the restricted (but still fairly general) case of *two-party function evaluation*, where the parties wish to evaluate some predefined function of their local inputs. We consider both deterministic and probabilistic functions. Our results (which are mostly negative in nature) are quite far-reaching and apply to many tasks of interest. In a nutshell, our results can be summarized as follows. Say that a function g is *efficiently invertible* if there exists an inverting algorithm M that successfully inverts g (i.e., $M(g(x)) \in g^{-1}(g(x))$) for any samplable distribution on the input x. Then, our main results can be informally described as follows:

1. Let $f(\cdot,\cdot)$ be a *deterministic* two-party function. Then, f can be securely evaluated in the UC framework *if and only if* f depends on at most one of its two inputs (e.g., $f(x,y) = g(x)$ for some function $g(\cdot)$) and, in addition, g is efficiently invertible.
2. Let $f(\cdot,\cdot)$ be a *probabilistic* two-party function. Then, f can be securely evaluated in the UC framework only if for any party, and for any input x for that party, there exists an input y for the other party such that $f(x,y)$ is "almost" deterministic (i.e., essentially all the probability mass of $f(x,y)$ is concentrated on a single value).

These results pertain to protocols where *both* parties obtain the same output. Interestingly, the results hold unconditionally, in spite of the fact that they rule out protocols that provide only computational security guarantees. We remark that UC-security allows "early stopping", or protocols where one of the parties aborts after learning the output and before the other party learned the output. Hence, our impossibility results do not (and cannot) rely on an early stopping strategy by the adversary (as used in previous impossibility results like [c86]).

Our results provide an alternative proof to previous impossibility results regarding UC zero-knowledge and UC coin-tossing in the plain model [c01,cf01]. In fact, our results also rule out any interesting relaxation of these functionalities. We stress, however, that these results do *not* rule out the realizability (in the plain model) of interesting functionalities like key-exchange, secure message transmission, digital signatures, and public-key encryption (see [c01,ck02]). Indeed, as noted above, these functionalities are realizable in the plain model.

Techniques. Our impossibility results utilize in an essential way the strong requirements imposed by the UC framework. The UC definition follows the standard paradigm of comparing a real protocol execution to an ideal process involving a trusted third party. It also differs in a very important way. The traditional model considered for secure computation includes the parties running the protocol, plus an adversary $\mathcal{A}$ that controls a set of corrupted parties. In the UC framework, an additional adversarial entity called the environment $\mathcal{Z}$ is introduced. This environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol securely computes a function f in this framework if for any adversary $\mathcal{A}$ that interacts with the parties running the protocol, there exists an ideal process adversary (or "simulator") $\mathcal{S}$ that interacts with the trusted third party, such that no environment can tell whether it is interacting with $\mathcal{A}$ and the parties running the protocol, or with $\mathcal{S}$ in the ideal process.

On a high level, our results are based on the following observation. A central element of the UC definition is that the real and ideal process adversaries $\mathcal{A}$ and $\mathcal{S}$ interact with the environment $\mathcal{Z}$ in an "on-line" manner. This implies that $\mathcal{S}$ must succeed in simulation while interacting with an external adversarial entity that it cannot "rewind". Given the fact that here is no honest majority and that there are no setup assumptions that can be utilized, it turns out that the simulator $\mathcal{S}$ has *no advantage* over a real participant. Thus, a corrupted party can actually run the code of the simulator.

Given the above observation, we demonstrate our results in two steps. First, in Section 3, we prove a general "technical lemma," asserting that a certain adversarial behavior (which is based on running the code of the simulator) is possible in our model. We then use this lemma to prove the characterization mentioned above, in several steps. Let us outline these steps. First, we concentrate on functions where only one of the parties has input, and show that these functions are computable iff they are efficiently invertible. Next, we consider functions where both parties have inputs, and demonstrate that such functions are computable only if they totally ignore at least one of the two inputs. This is done as follows. First, we show that functions that contain an "insecure minor," as defined in [BMM99], cannot be realized. (A series of values $\alpha_1, \alpha_1', \alpha_2, \alpha_2'$ constitute an insecure minor for f if $f(\alpha_1, \alpha_2) = f(\alpha_1', \alpha_2)$ but $f(\alpha_1, \alpha_2') \neq f(\alpha_1', \alpha_2')$; see Table 1 in Section 4.2.) Next, we show that functions that contain an "embedded-XOR" cannot be realized. (Values $\alpha_1, \alpha_1', \alpha_2, \alpha_2'$ form an embedded-XOR if $f(\alpha_1, \alpha_2) \neq f(\alpha_1', \alpha_2) \neq f(\alpha_1', \alpha_2') \neq f(\alpha_1, \alpha_2') \neq f(\alpha_1, \alpha_2)$; see Table 2 in Section 4.3.) We then prove that a function that contains neither an insecure minor nor an embedded-XOR must ignore at least one of its inputs.

Impossibility for relaxed versions of UC. The impossibility results presented in this paper actually also rule out two natural relaxations of the UC definition. First, consider an analogous definition where the environment machine $\mathcal{Z}$ is uniform. (We remark that the UC theorem has been shown to hold under this definition [HMS03].) In this case, some variations of our results hold (for example, an almost identical characterization holds in the case that the functions are defined over a finite domain). Next, consider a relaxed definition of security, where the relaxation relates to the order of quantifiers. The actual UC definition requires that for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish real executions with $\mathcal{A}$ from ideal process executions with $\mathcal{S}$ (i.e., $\forall\mathcal{A}\exists\mathcal{S}\forall\mathcal{Z}$). Thus, a single simulator $\mathcal{S}$ must successfully simulate for all environments $\mathcal{Z}$. A relaxation of this would allow a different simulator for every environment; i.e., $\forall\mathcal{A},\mathcal{Z}\exists\mathcal{S}$. (We note that the UC composition theorem is not known to hold in this case.) As above, the characterization remains almost the same even for this relaxed definition. Due to lack of space, we present our results only for the standard UC definition, however, similar impossibility results do hold for the relaxed definitions above; see [CKL03].

Related work. Characterizations of the functions that are securely computable were made in a number of other models and with respect to different notions of security. E.g., in the case of *honest-but-curious* parties and information-theoretic privacy, characterization of the functions that can be computed were found for the two-party case [CK89,K89], and for boolean functions in the multi-party case [CK89]. In [BMM99], the authors consider a setting of computational security against malicious parties where the output is given to only one of the parties, and provide a characterization of the *complete* functions. (A function is complete if given a black-box for computing it, it is possible to securely compute any other function.) Some generalizations were found in [K00]. Similar completeness results for the information-theoretic honest-but-curious setting are given in [KKMO00]. Interestingly, while the characterizations mentioned above are very different from each other, there is some similarity in the type of structures considered in those works and in ours (e.g., the insecure minor of [BMM99] and the embedded-OR of [KKMO00]).

2 Review of UC security

We present a very brief overview of how security is defined in the UC framework, restricted to our case of two parties, non-adaptive adversaries, and authenticated communication. For further details see [C01,CKL03].

As in other general definitions (e.g., [GL90,MR91,B91]), the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs (in one or more iterations). Informally, a protocol securely carries out a given task if running the protocol with a realistic adversary amounts to "emulating" an ideal process where the parties hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it, without any other interaction. We call the algorithm run by the trusted party an ideal functionality.

To allow proving a universal composition theorem, the notion of emulation in this framework is considerably stronger than in previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary, $\mathcal{A}$, that controls the communication channels and potentially corrupts parties. "Emulating an ideal process" means that for any adversary $\mathcal{A}$ there should exist an "ideal process adversary" (or, simulator) $\mathcal{S}$ that results in a similar distribution on the outputs for the parties. Here an additional entity, called the environment $\mathcal{Z}$, is introduced. The environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ that interacts with the protocol and the environment there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$. A bit more precisely, Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ be the ensemble describing the output of environment $\mathcal{Z}$ after interacting with parties running protocol π and with adversary $\mathcal{A}$. (Without loss of generality, we assume that the environment outputs one bit.) Similarly, let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ be the ensemble describing the output of environment $\mathcal{Z}$ after interacting in the ideal process with adversary $\mathcal{S}$ and parties that have access to ideal functionality $\mathcal{F}$.

Definition 1 *Let $\mathcal{F}$ be an ideal functionality and let π be a two-party protocol. We say that π* securely realizes *$\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$, the ensembles $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ are indistinguishable.*

NON-TRIVIAL PROTOCOLS AND THE REQUIREMENT TO GENERATE OUTPUT. In the UC framework, the ideal process does not require the ideal-process adversary to deliver messages that are sent by the ideal functionality to the dummy parties. Consequently, the definition provides no guarantee that a protocol will ever generate output or "return" to the calling protocol. Indeed, in our setting where message delivery is not guaranteed, it is impossible to ensure that a protocol "terminates" or generates output. Rather, the definition concentrates on the security requirements *in the case that the protocol generates output.*

A corollary of the above fact is that a protocol that "hangs", never sends any messages and never generates output, securely realizes any ideal functionality. However, such a protocol is clearly not interesting. We thus use the notion of a non-trivial protocol. Such a protocol has the property that if the real-life adversary delivers all messages and does not corrupt any parties, then the ideal-process adversary also delivers all messages (and does not corrupt any parties). In the rest of this work we concentrate on non-trivial protocols.

3 The Main Theorem – Deterministic Functions

In this section, we prove a theorem that serves as the basis for our impossibility results. To motivate the theorem, recall the way an ideal-model simulator typically works. Such a simulator interacts with an ideal functionality by sending it an input (in the name of the corrupted party) and receiving back an output. Since the simulated view of the corrupted party is required to be indistinguishable from its view in a real execution, it must hold that the input sent by the simulator to the ideal functionality corresponds to the input that the corrupted party (implicitly) uses. Furthermore, the corrupted party's output from the protocol simulation must correspond to the output received by the simulator from the ideal functionality. That is, such a simulator can *"extract"* the input used by the corrupted party, and can cause the corrupted party to *output* a value that corresponds to the output received by the simulator from the ideal functionality.

The following theorem shows that, essentially, a malicious P_2 can do "whatever" the simulator can do. That is, consider the simulator that exists when P_1 is corrupted. This simulator can extract P_1's input and can cause its output to be consistent with the output from the ideal functionality. Therefore, P_2 (when interacting with an *honest* P_1) can also extract P_1's input and cause its output to be consistent with an ideally generated output. Indeed, P_2 succeeds in doing this by internally running the ideal-process simulator for P_1. In other models of secure computation, this cannot be done because a simulator typically has some additional "power" that a malicious party does not. (This power is usually the ability to rewind a party or to hold its description or code.) Thus, we actually show that in the *plain model* and without an honest majority, the simulator for the UC setting has no power beyond what a real (adversarial) party can do in a real execution. This enables P_2 to run the simulator as required. We now describe the above-mentioned strategy of P_2.

STRATEGY DESCRIPTION FOR P_2: The malicious P_2 internally runs two separate machines (or entities): P_2^a and P_2^b. Entity P_2^a interacts with (the honest) P_1 and runs the simulator that is guaranteed to exist for P_1, as described above. In contrast, entity P_2^b emulates the ideal functionality for the simulator as run by P_2^a. Loosely speaking, when P_2^a obtains P_1's input, it hands it to P_2^b, who then computes the function output and hands it back to P_2^a. Entity P_2^a then continues with the emulation, and causes P_1 to output the value that P_2^a received from P_2^b. We now formally define this strategy of P_2. We begin by defining the structure of this adversarial attack, which we call a "split adversarial strategy", and then proceed to define what it means for such a strategy to be "successful".

Definition 2 (split adversarial strategy): *Let $f : X \times X \to \{0,1\}^*$ be a function and let Π_f be a protocol. Let $X_2 \subseteq X$ be a polynomial-size subset of inputs (i.e., $|X_2| = \mathrm{poly}(k)$, where k is the security parameter), and let $x_2 \in X_2$. Then, a corrupted party P_2 is said to run a* split adversarial strategy *if it proceeds as follows. P_2 internally runs P_2^a and P_2^b. Next:*

1. *Upon input (X_2, x_2), party P_2 internally gives the machine P_2^b the input pair (X_2, x_2). (P_2^a does* not *receive the input value. This fact will become important later.)*
2. *An execution between (an honest) P_1 running Π_f and P_2 works as follows:*
   (a) *P_2^a interacts with P_1 according to some specified strategy (to be fixed).*
   (b) *At some stage of the execution P_2^a hands P_2^b a value x_1'.*
   (c) *When P_2^b receives x_1' from P_2^a, it computes $y = f(x_1', x_2')$ for some $x_2' \in X_2$ of its choice.*[3]
   (d) *P_2^b hands P_2^a the value y, and P_2^a continues interacting with P_1.*

Informally speaking, a split adversarial strategy is said to be *successful* if the value x_1' procured by P_2^a is essentially (the honest) P_1's input. Furthermore, P_2^a

[3] The choice of x_2' can depend on the values of both x_1' and x_2 and can be chosen by any efficient strategy. The fact that x_2' must come from the polynomial-size subset of inputs X_2 is needed later.

should succeed in causing P_1 to output the value y that it received from P_2^b. As mentioned, P_2^a does this by internally running the simulator that is guaranteed to exist for a corrupted P_1. This means that step 2b above corresponds to the ability of the simulator (as run by P_2^a) to *extract* P_1's input, and step 2d corresponds to the simulator's ability to cause the *output* of P_1 to be consistent with the ideally computed output. Formally,

Definition 3 *Let $\mathcal{Z}$ be an environment who hands an input $x_1 \in X$ to P_1 and a pair (X_2, x_2) to P_2, where $X_2 \subseteq X$, $|X_2| = \text{poly}(k)$, and $x_2 \in_{\text{R}} X_2$. Furthermore, $\mathcal{Z}$ continually activates P_1 and P_2 in succession. Then, a split adversarial strategy for P_2 is said to be* successful *if in a real execution with the above $\mathcal{Z}$ and an honest P_1, the following holds:*

1. *The value x_1' output by P_2^a in step 2b of Definition* 2 *is such that for* every *$x_2 \in X_2$, $f(x_1', x_2) = f(x_1, x_2)$.*
2. *P_1 outputs y, where y is the value that P_2^b gives P_2^a in step 2d of Definition* 2.

Loosely speaking, the theorem below states that a successful split adversarial strategy exists for any protocol that securely realizes a two-party function. In Section 4, we will show that the existence of successful split adversarial strategies rules out the possibility of securely realizing large classes of functions. We are now ready to state the theorem:

Theorem 4 *Let f be a two-party function, and let $\mathcal{F}_f$ be the two-party ideal functionality that receives x_1 and x_2 from P_1 and P_2, respectively, and hands both parties $f(x_1, x_2)$. If $\mathcal{F}_f$ can be securely realized by a non-trivial protocol*[4] *Π_f, then there exists a machine P_2^a such that, except with negligible probability, the split adversarial strategy for $P_2 = (P_2^a, P_2^b)$ is successful.*

Proof: The intuition behind the proof is as follows. If $\mathcal{F}_f$ can be securely realized by a protocol $\Pi_{\mathcal{F}}$, then this implies that for any real-life adversary $\mathcal{A}$ (and environment $\mathcal{Z}$), there exists an ideal-process adversary (or "simulator") $\mathcal{S}$. As we have mentioned, this simulator interacts with the ideal process and must hand it the input that is (implicitly) used by the corrupted party. That is, $\mathcal{S}$ must be able to *extract* a corrupted party's input. Now, consider the case that $\mathcal{A}$ controls party P_1 and so $\mathcal{S}$ must extract P_1's input. The key point in the proof is that $\mathcal{S}$ must essentially accomplish this extraction without any rewinding and without access to anything beyond the protocol messages. This is due to the following observations. First, $\mathcal{S}$ interacts with the environment $\mathcal{Z}$ in the same way that parties interact with each other in a real execution. That is, $\mathcal{Z}$ is a party that sits externally to $\mathcal{S}$ (or, otherwise stated, $\mathcal{S}$ has only black-box access to $\mathcal{Z}$) and $\mathcal{S}$ is not able to rewind $\mathcal{Z}$. Now, since $\mathcal{Z}$ and $\mathcal{A}$ can actively cooperate in attacking the protocol, $\mathcal{Z}$ can generate all the adversarial messages, with $\mathcal{A}$

[4] Recall that a non-trivial protocol is such that if the real model adversary corrupts no party and delivers all messages, then so does the ideal model adversary. This rules out the trivial protocol that does not generate output. See Section 2 for details.

just forwarding them to their intended recipients. This implies that $\mathcal{S}$ actually has to extract the input from $\mathcal{Z}$. However, as mentioned, $\mathcal{S}$ interacts with $\mathcal{Z}$ in the same way that real parties interact in a protocol execution. Given that $\mathcal{S}$ can extract in such circumstances, it follows that a malicious P_2 interacting in a real protocol execution with an honest P_1, can also extract P_1's input (by using $\mathcal{S}$). Thus, P_2^a's strategy is to run the code of the simulator $\mathcal{S}$ (and the role of P_2^b is to emulate $\mathcal{S}$'s interface with $\mathcal{F}_f$). The above explains the ability of P_2^a to extract P_1's input. The fact that P_2^a causes the honest P_1 to output $y = f(x_1', x_2')$ also follows from the properties of $\mathcal{S}$. We now formally prove the above, by formalizing a split adversarial strategy, and proving that it is successful.

First, we formulate the simulator $\mathcal{S}$ mentioned above. Assume that $\mathcal{F}_f$ can be securely realized by a protocol Π_f. Then, for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between an execution of the ideal process with $\mathcal{S}$ and $\mathcal{F}_f$ and an execution of the real protocol Π_f with $\mathcal{A}$. We now define a specific adversary $\mathcal{A}$ and environment $\mathcal{Z}$. The adversary $\mathcal{A}$ controls party P_1 and is a dummy adversary who acts as a bridge that passes messages between $\mathcal{Z}$ and P_2. Now, let X_2 be some polynomial-size set of inputs (chosen by $\mathcal{Z}$), and let (x_1, x_2) be P_1 and P_2's respective inputs as decided by $\mathcal{Z}$, where $x_2 \in_{\mathrm{R}} X_2$. Then, $\mathcal{Z}$ writes (X_2, x_2) on P_2's input tape and plays the role of honest P_1 on input x_1 by itself. That is, $\mathcal{Z}$ runs P_1's protocol instructions in Π_f on input x_1 and the incoming messages that it receives from $\mathcal{A}$ (which are in turn received from P_2). The messages that $\mathcal{Z}$ passes to $\mathcal{A}$ are exactly the messages as computed by an honest P_1 according to Π_f. At the conclusion of the execution of Π_f, the environment $\mathcal{Z}$ obtains some output, as defined by the protocol specification for P_1 that $\mathcal{Z}$ runs internally; we call this $\mathcal{Z}$'s local P_1-output. $\mathcal{Z}$ then reads P_2's output tape and outputs 1 if and only if $\mathcal{Z}$'s local P_1-output and P_2's output *both* equal $f(x_1, x_2)$. Observe that in the real-life model $\mathcal{Z}$ outputs 1 with probability negligibly close to 1. This is because such an execution of Π_f, with the above $\mathcal{Z}$ and $\mathcal{A}$, looks exactly like an execution between two honest parties P_1 and P_2 upon inputs x_1 and x_2, respectively. Now, since Π_f is a non-trivial protocol, in an ideal execution with no corrupted parties, both parties output $f(x_1, x_2)$. Therefore, the same result must also hold in a real execution (except with negligible probability).

We now describe the strategy for P_2^a. By the assumption that Π_f securely realizes $\mathcal{F}_f$, there exists an ideal process simulator $\mathcal{S}$ for the *specific* $\mathcal{A}$ (and $\mathcal{Z}$) described above, where P_1 is corrupted. P_2^a invokes this simulator $\mathcal{S}$ and emulates an ideal process execution of $\mathcal{S}$ with the above $\mathcal{A}$ and $\mathcal{Z}$. That is, every message that P_2^a receives from P_1 it forwards to $\mathcal{S}$ as if $\mathcal{S}$ received it from $\mathcal{Z}$. Likewise, every message that $\mathcal{S}$ sends to $\mathcal{Z}$ in the emulation, P_2^a forwards to P_1 in the real execution. When $\mathcal{S}$ outputs a value x_1' that it intends to send to $\mathcal{F}_f$, entity P_2^a hands it to P_2^b. Then, when P_2^a receives a value y back from P_2^b, it passes this to $\mathcal{S}$, as if $\mathcal{S}$ receives it from $\mathcal{F}_f$, and continues with the emulation. (Recall that this value y is computed by P_2^b and equals $f(x_1', x_2')$.)

We now prove that, except with negligible probability, the above P_2^a is such that $P_2 = (P_2^a, P_2^b)$ is a successful split adversarial strategy. That is, we prove that items (1) and (2) from Definition 3 hold with respect to this P_2. We begin by proving that, except with negligible probability, the value x_1' output by P_2^a is such that for every $x_2 \in X_2$, $f(x_1', x_2) = f(x_1, x_2)$. First, we claim that $\mathcal{S}$'s view in the ideal process with the above $\mathcal{A}$ and $\mathcal{Z}$ is identical to its view in the emulation with $P_2 = (P_2^a, P_2^b)$. (Actually, for now it suffices to show that this holds until the point that $\mathcal{S}$ outputs x_1'.) To see this, notice that in the ideal process with $\mathcal{A}$ and $\mathcal{Z}$, the simulator $\mathcal{S}$ holds the code of $\mathcal{A}$. However, all $\mathcal{A}$ does is forwarding messages between $\mathcal{Z}$ and P_2. Thus, the only "information" received by $\mathcal{S}$ is through the messages that it receives from $\mathcal{Z}$. Now, recall that in this ideal process, $\mathcal{Z}$ plays the honest P_1 strategy upon input x_1. Therefore, the messages that $\mathcal{S}$ receives from $\mathcal{Z}$ are distributed exactly like the messages that $\mathcal{S}$ receives from the honest P_1 in the emulation with $P_2 = (P_2^a, P_2^b)$. Since this is the only information received by $\mathcal{S}$ until the point that $\mathcal{S}$ outputs x_1', we have that its views in both cases are identical. It remains to show that in the ideal process with $\mathcal{A}$ and $\mathcal{Z}$, simulator $\mathcal{S}$ must obtain and send $\mathcal{F}_f$ an input x_1' such that for every $x_2 \in X_2$, $f(x_1', x_2) = f(x_1, x_2)$, except with negligible probability. (Item (1) follows from this because if $\mathcal{S}$ obtains such an x_1' in the ideal process, then it also obtains it in the emulation with $P_2 = (P_2^a, P_2^b)$ where its view is identical.) This can be seen as follows. Assume, by contradiction, that with non-negligible probability x_1' is such that for some $\tilde{x}_2 \in X_2$, $f(x_1', \tilde{x}_2) \neq f(x_1, \tilde{x}_2)$. Now, if in an ideal execution, P_2's input equals $\tilde{x}_2$ and $\mathcal{S}$ sends x_1' to $\mathcal{F}_f$, then P_2 outputs $f(x_1', x_2) \neq f(x_1, x_2)$. By the specification of $\mathcal{Z}$, when this occurs $\mathcal{Z}$ outputs 0. Now, recall that X_2 is of polynomial size and that P_2's input is uniformly chosen from X_2. Furthermore, the probability that $\mathcal{S}$ sends x_1' such that $f(x_1', \tilde{x}_2) \neq f(x_1, \tilde{x}_2)$ is independent of the choice of x_2 for P_2. Therefore, the overall probability that $\mathcal{Z}$ outputs 0 in the ideal process is non-negligible. However, we have already argued above that in a real protocol execution, $\mathcal{Z}$ outputs 1 with overwhelming probability. Thus, $\mathcal{Z}$ distinguishes the real and ideal executions, contradicting the security of the protocol. We conclude that except with negligible probability, item (1) of Definition 3 holds.

We proceed to prove item (2) of Definition 3. Assume, by contradiction, that in the emulation with $P_2 = (P_2^a, P_2^b)$, party P_1 outputs $y' \neq y$ with non-negligible probability (recall that $y = f(x_1', x_2'))$. First, consider the following thought experiment: Modify P_2^b so that instead of choosing x_2' as some function of x_1' and x_2, it chooses $x_2' \in_{\mathrm{R}} X_2$ instead; denote this modified party $\tilde{P}_2^b$. It follows that with probability $1/|X_2|$, the value chosen by the modified $\tilde{P}_2^b$ *equals* the value chosen by the unmodified P_2^b. Therefore, the probability that P_1 outputs $y' \neq y$ in an emulation with the modified $\tilde{P}_2 = (P_2^a, \tilde{P}_2^b)$ equals $1/|X_2|$ times the (non-negligible) probability that this occurred with the unmodified P_2^b. Since X_2 is of polynomial size, we conclude that P_1 outputs $y' \neq y$ with non-negligible probability in an emulation with the modified $\tilde{P}_2 = (P_2^a, \tilde{P}_2^b)$. Next, we claim that the view of $\mathcal{S}$ in the ideal process with $\mathcal{Z}$ and $\mathcal{F}_f$ is identical to its view in the emulation by $\tilde{P}_2 = (P_2^a, \tilde{P}_2^b)$. The fact that this holds until $\mathcal{S}$ outputs x_1' was shown above in the proof of item (1). The fact that it holds from that point on follows from the observation that in the emulation by $\tilde{P}_2 = (P_2^a, \tilde{P}_2^b)$, simulator

$\mathcal{S}$ receives $f(x_1', x_2')$ where $x_2' \in_{\mathrm{R}} X_2$. However, this is exactly the same as it receives in an ideal execution (where $\mathcal{Z}$ chooses $x_2 \in_{\mathrm{R}} X_2$ and gives it to the honest P_2). It follows that the distribution of messages received by P_1 in a real execution with $\tilde{P}_2 = (P_2^a, \tilde{P}_2^b)$ is exactly the same as the distribution of messages received by $\mathcal{Z}$ from $\mathcal{S}$ in the ideal process. Thus, $\mathcal{Z}$'s local P_1-output is identically distributed to P_1's output in the emulation with $\tilde{P}_2 = (P_2^a, \tilde{P}_2^b)$. Since in this emulation P_1 outputs $y' \neq y$ with non-negligible probability, we have that $\mathcal{Z}$'s local P_1-output in the ideal process is also not equal to y with non-negligible probability. By the specification of $\mathcal{Z}$, it follows that $\mathcal{Z}$ outputs 0 in the ideal process with non-negligible probability. This is a contradiction because in a real execution $\mathcal{Z}$ outputs 0 with at most negligible probability. Thus, $\mathcal{Z}$ distinguishes the real and ideal processes. This completes the proof. ■

4 A Characterization for Deterministic Functionalities

This section provides a characterization of two-party deterministic functions with a single output that can be securely computed in the UC framework without any set-up assumptions. More specifically, let $f : X \times X \to \{0,1\}^*$ be a deterministic function. Say that f is realizable if there exists a protocol that securely realizes the ideal functionality $\mathcal{F}_f$ that obtains a value $x_1 \in X$ from P_1, a value $x_2 \in X$ from P_2, and hands $f(x_1, x_2)$ to both P_1 and P_2. We provide a characterization of the realizable functions f.

This section is organized as follows: We first present a characterization for the case of functions that depend on only one of the two inputs. Next, we present two different impossibility results for functions that depend on both inputs. Finally, we combine all results to obtain the desired characterization. All the impossibility results here are obtained by applying Theorem 4 to the specific setting.

It is stressed that the functionalities that we consider provide outputs to *both* parties, and that our results do not rely in any way on the fact that a corrupted party can always abort the computation after it has learned the joint output, and before the other party does. Indeed, the UC framework explicitly *permits* such behavior, by stating that even in the ideal process, parties are not guaranteed to obtain output (if the ideal-model adversary chooses to do so).

4.1 Functions of one input

This section considers functions that depend on only one party's input. We show that a function of this type can be securely computed in a universally composable way if and only if it is efficiently invertible. Formally,

Definition 5 *A function $f : X \to \{0,1\}^*$ is* efficiently invertible *if there exists a probabilistic polynomial-time inverting machine M such that for every (non-uniform) polynomial-time samplable distribution $\hat{X}$ over X,*

$$\Pr_{x \leftarrow \hat{X}}[M(1^k, f(x)) \in f^{-1}(f(x))] > 1 - \mu(k)$$

for some negligible function $\mu(\cdot)$.

Discussion. A few remarks regarding Definition 5: First, note that every function f on a finite domain X is efficiently invertible. Second, note that a function that is *not* efficiently invertible is not necessarily even weakly one-way. This is because the definition of invertibility requires the existence of an inverter that works for *all* distributions, rather than only for the uniform distribution (as in the case of one way functions). In fact, a function that is not efficiently invertible can be constructed from any NP-language L that is not in BPP, as follows. Let R_L be the NP-relation for L, i.e., $x \in L$ iff $\exists w$ s.t. $R_L(x, w) = 1$. Then, define $f_L(x, w) = (x, R_L(x, w))$. It is easy to see that f_L is not efficiently invertible unless $L \in BPP$ (this holds only when the distributions $\hat{X}$ are allowed to be non-uniform).

Finally, note that a function f_L as defined above corresponds in fact to the ideal zero-knowledge functionality for the language L. That is, the ideal functionality $\mathcal{F}_{f_L}$ as defined above is exactly the ideal zero-knowledge functionality $\mathcal{F}_{zk}^{R_L}$ for relation R_L, as defined in [c01,clos02]. Consequently, the characterization theorem below (Theorem 6) provides, as a special case, an alternative proof that $\mathcal{F}_{zk}^{R_L}$ cannot be realized unless $L \in BPP$ [c01].

We now show that a function f that depends on only one party's input is realizable if and only if it is efficiently invertible.

Theorem 6 *Let $f : X \to \{0,1\}^*$ be a function and let $\mathcal{F}_f$ be a functionality that receives x from P_1 and sends $f(x)$ to P_2. Then, $\mathcal{F}_f$ can be securely realized in a universally composable way by a non-trivial protocol if and only if f is efficiently invertible.*

Proof: We first show that if f is efficiently invertible then it can be securely realized. This is done by the following simple protocol: Upon input x and security parameter k, party P_1 computes $y = f(x)$ and runs the inverting machine M on $(1^k, y)$. Then, P_1 sends P_2 the value x' output by M. (In order to guarantee security against an external adversary that does not corrupt any party, the value x' will be sent encrypted, say using a shared key that is the result of a universally composable key exchange protocol run by the parties.) Simulation of this protocol is demonstrated by constructing a simulator who receives $y = f(x)$, and simulates P_1 sending P_2 the output of $M(1^k, y)$. Details are omitted.

Let f be a function that is not efficiently invertible. Then, for every non-uniform polynomial-time machine M there exists a polynomial-time samplable distribution $\hat{X}$ over X such that $\Pr_{x \leftarrow \hat{X}}[M(1^k, f(x)) \neq f^{-1}(f(x))]$ is non-negligible. We now show the impossibility of realizing such an f. Assume, by contradiction, that there exists a protocol Π_f that securely realizes f. Consider a real execution of Π_f with an honest P_1 (with input x), and a corrupted P_2 who runs a successful split adversarial strategy. (By Theorem 4, such a successful strategy exists.) The adversary $\mathcal{A}$, who controls P_2, is such that at the conclusion of the execution, it hands $\mathcal{Z}$ the value x' obtained in step 2b of Definition 2. Finally, define the environment $\mathcal{Z}$ for this scenario to be so that it samples a value x from some distribution $\hat{X}$ and hands it to P_1. Then, $\mathcal{Z}$ outputs 1 if and only if $x' \in f^{-1}(f(x))$, where x' is the value that it receives from $\mathcal{A}$. Observe

	α_2	α_2'
α_1	a	b
α_1'	a	c

Table 1. An Insecure Minor (assuming $b \neq c$)

that by item (1) of Definition 3, $f(x') = f(x)$ with overwhelming probability and so in a real execution, $\mathcal{Z}$ outputs 1 with overwhelming probability. (Here, the set X_2 described in Definition 2 contains the empty input.)

Next, consider an ideal execution with the same $\mathcal{Z}$ and with an ideal-process simulator $\mathcal{S}_2$ for the above P_2. Clearly, in such an ideal execution $\mathcal{S}_2$ receives $f(x)$ only (because all it sees is the output of the corrupted party P_2). Nevertheless, $\mathcal{S}_2$ succeeds in handing $\mathcal{Z}$ a value x' such that $f(x') = f(x)$; otherwise, $\mathcal{Z}$ would distinguish a real execution from an ideal one. Thus, $\mathcal{S}_2$ can be used to construct an inverting machine M for f: Given $y = f(x)$, M runs $\mathcal{S}_2$, gives it y in the name of $\mathcal{F}_f$, and outputs whatever value $\mathcal{S}_2$ hands to $\mathcal{Z}$. The fact that M is a valid inverting machine follows from the above argument. That is, if there exists an efficiently samplable distribution $\hat{X}$ for which M does not succeed in inverting f, then when the environment $\mathcal{Z}$ chooses x according to $\hat{X}$, it distinguishes the real and ideal executions with non-negligible probability. This contradicts the fact that f is not efficiently invertible, concluding the proof. ■

4.2 Functions with Insecure Minors

This section presents an impossibility result for realizing two-input functions with a special combinatorial property, namely the existence of an insecure minor. In fact, this property was already used to show non-realizability results in a different context of informational-theoretic security [BMM99].

A function $f : X \times X \to \{0,1\}^*$ is said to contain an insecure minor if there exist inputs $\alpha_1, \alpha_1', \alpha_2$ and α_2' such that $f(\alpha_1, \alpha_2) = f(\alpha_1', \alpha_2)$ and $f(\alpha_1, \alpha_2') \neq f(\alpha_1', \alpha_2')$; see Table 1. (In the case of boolean functions, the notion of an insecure minor boils down to the so called "embedded-OR"; see, e.g., [KKMO00].) Such a function has the property that when P_2 has input α_2, then party P_1's input is "hidden" (i.e., given $y = f(x_1, \alpha_2)$, it is impossible for P_2 to know whether P_1's input, x_1, was α_1 or α_1'). Furthermore, α_1 and α_1' are not "equivalent", in that when P_2 has α_2' for input, then the result of the computation with P_1 having $x_1 = \alpha_1$ differs from the result when P_1 has $x_1 = \alpha_1'$ (because $f(\alpha_1, \alpha_2') \neq f(\alpha_1', \alpha_2')$). We stress that there is no requirement that $f(\alpha_1, \alpha_2) \neq f(\alpha_1, \alpha_2')$ or $f(\alpha_1', \alpha_2) \neq f(\alpha_1', \alpha_2')$ (i.e., in Table 1, a may equal b or c, but clearly not both). We now show that no function containing an insecure minor can be securely computed without set-up assumptions. (In the treatment below the roles of P_1 and P_2 may be switched.)

Theorem 7 *Let f be a two-party function containing an insecure minor, and let $\mathcal{F}_f$ be the two-party ideal functionality that receives x_1 and x_2 from P_1 and P_2 respectively, and hands both parties $f(x_1, x_2)$. Then, $\mathcal{F}_f$ cannot be securely realized by a non-trivial protocol.*

Proof: We prove this theorem using Theorem 4 and item (1) of Definition 3. As we have mentioned above, a function with an insecure minor hides the input of P_1. However, by Theorem 4, P_2 can obtain P_1's input. This results in a contradiction.

Formally, let f be a function and let $\alpha_1, \alpha_1', \alpha_2, \alpha_2'$ form an insecure minor in f. Assume by contradiction that $\mathcal{F}_f$ can be securely realized by a protocol Π_f. Then, consider a real execution of Π_f with an honest P_1 and a corrupted P_2 who runs a successful split adversarial strategy. (By Theorem 4, such a successful strategy exists.) The environment $\mathcal{Z}$ for this execution chooses a pair of inputs (x_1, x_2) where $x_1 \in_{\mathrm{R}} \{\alpha_1, \alpha_1'\}$ and $x_2 \in_{\mathrm{R}} \{\alpha_2, \alpha_2'\}$.[5] (I.e., in this case the set X_2 described in Definition 2 equals $\{\alpha_2, \alpha_2'\}$.) $\mathcal{Z}$ then gives P_1 and P_2 their respective inputs, x_1 and x_2. Now, since P_2 is successful, P_2^a must output x_1' such that for every $x_2 \in X_2$, $f(x_1', x_2) = f(x_1, x_2)$. When $\mathcal{A}$ receives the value x_1' (from P_2), it checks if $f(x_1', x_2) = f(\alpha_1, x_2)$ or if $f(x_1', x_2) = f(\alpha_1', x_2)$ (recall that $\mathcal{A}$ knows x_2). Note that since $\alpha_2' \in X_2$, the value x_1' must match only one of α_1 and α_1'. $\mathcal{A}$ then hands $\mathcal{Z}$ the value α_1 or α_1' appropriately (i.e., according to which input matches x_1'). Finally, $\mathcal{Z}$ outputs 1 if and only if the value that it receives from $\mathcal{A}$ equals x_1. Observe that by item (1) of Definition 3, in the above real execution, $\mathcal{Z}$ outputs 1 with overwhelming probability.

Next, consider an ideal execution with the above $\mathcal{A}$ and $\mathcal{Z}$. By the assumption that $\mathcal{F}_f$ is secure, there exists an appropriate ideal process simulator $\mathcal{S}_2$ for the corrupt P_2. Now, $\mathcal{S}_2$ is given the output $f(x_1, x_2)$ and must provide $\mathcal{Z}$ with the value x_1 with overwhelming probability. (Recall that in the real execution, $\mathcal{A}$ provides $\mathcal{Z}$ with this value with overwhelming probability.) We conclude by analyzing the probability that $\mathcal{S}$ can succeed in such a task. First, with probability 1/2, we have that $x_2 = \alpha_2'$. In this case, $f(\alpha_1, x_2) \neq f(\alpha_1', x_2)$. Therefore, $\mathcal{S}_2$ can always succeed in obtaining the correct x_1. However, with probability 1/2, we have that $x_2 = \alpha_2$. In this case, $f(\alpha_1, x_2) = f(\alpha_1', x_2)$. Therefore, information theoretically, $\mathcal{S}_2$ can obtain the correct x_1 with probability at most 1/2. It follows that $\mathcal{Z}$ outputs 0 in such an ideal execution with probability at least 1/4. Therefore, $\mathcal{Z}$ distinguishes the ideal and real executions with non-negligible probability, in contradiction to the security of Π_f. ■

4.3 Functions with embedded-XOR

This section presents an impossibility result for realizing two-input functions with another combinatorial property, namely the existence of an embedded-XOR. A function f is said to contain an embedded-XOR if there exist inputs α_1,α_1',α_2

[5] Since $\mathcal{Z}$ is non-uniform, we can assume that it is given an insecure minor of f for auxiliary input.

	α_2	α_2'
α_1	a	b
α_1'	c	d

Table 2. An Embedded-XOR – if $\{a, d\} \cap \{b, c\} = \emptyset$.

and α_2' such that the two sets $A_0 \stackrel{\text{def}}{=} \{f(\alpha_1, \alpha_2), f(\alpha_1', \alpha_2')\}$ and $A_1 \stackrel{\text{def}}{=} \{f(\alpha_1, \alpha_2'), f(\alpha_1', \alpha_2)\}$ are disjoint; see Table 2. (In other words, the table describes an embedded-XOR if no two elements in a single row or column are equal. The name "embedded-XOR" originates from the case of boolean functions f, where one can pick $A_0 = \{0\}$ and $A_1 = \{1\}$.) The intuitive idea is that none of the parties, based on its input (among those in the embedded-XOR sub-domain), should be able to bias the output towards one of the sets A_0, A_1 of its choice. In our impossibility proof, we will in fact show a strategy for, say, P_2 to bias the output. We now show that no function containing an embedded-XOR can be securely computed in the plain model.

Theorem 8 *Let f be a two-party function containing an embedded-XOR, and let $\mathcal{F}_f$ be the two-party ideal functionality that receives x_1 and x_2 from P_1 and P_2 respectively, and hands both parties $f(x_1, x_2)$. Then, $\mathcal{F}_f$ cannot be securely realized by a non-trivial protocol.*

Proof: Again, we prove this theorem using Theorem 4. However, the use here is different. That is, instead of relying on the extraction property (step 2b of Definition 2 and item (1) of Definition 3), we rely on the fact that P_2 can influence the output by choosing its input as a function of P_1's input (step 2c), and then cause P_1 to output the value y that corresponds to these inputs (step 2d and item (2) of Definition 3). That is, P_2 is able to bias the output, something which it should not be able to do when a function has an embedded-XOR.

Formally, let f be a function and let $\alpha_1, \alpha_1', \alpha_2, \alpha_2'$ form an embedded-XOR in f with corresponding sets A_0, A_1 (as described above). Assume by contradiction that $\mathcal{F}_f$ can be securely realized by a protocol Π_f. Then, consider a real execution of Π_f with an honest P_1 and a corrupted P_2 who runs a successful split adversarial strategy. (By Theorem 4, such a successful strategy exists.) The environment $\mathcal{Z}$ for this execution chooses a pair of inputs (x_1, x_2) where $x_1 \in_{\mathrm{R}} \{\alpha_1, \alpha_1'\}$ and $x_2 \in_{\mathrm{R}} \{\alpha_2, \alpha_2'\}$. (I.e., in this case, the set X_2 from Definition 2 equals $\{\alpha_2, \alpha_2'\}$.) $\mathcal{Z}$ then gives P_1 and P_2 their respective inputs, x_1 and x_2. Now, by item (1) of Definition 3, except with negligible probability, P_2^a must output x_1' such that for every $x_2 \in X_2$, $f(x_1', x_2) = f(x_1, x_2)$. Given this x_1', machine P_2^b chooses x_2' so that $f(x_1', x_2') \in A_0$. (Such an x_2' exists by the definition of an embedded-XOR. Also, recall that any efficient strategy by which P_2^b chooses x_2' is allowed, see step 2c of Definition 2.) At the conclusion, $\mathcal{Z}$ outputs 1 if and only if the output of P_1 is in the set A_0. Observe that by item (2) of Definition 3, $\mathcal{Z}$ outputs 1 with overwhelming probability in the above real execution.

Next, consider an ideal execution with the above $\mathcal{A}$ and $\mathcal{Z}$. By the assumption that $\mathcal{F}_f$ is secure, there exists an appropriate ideal process simulator $\mathcal{S}_2$ for the corrupted P_2. In particular, the result of an ideal execution with $\mathcal{S}$ must be that P_1 outputs a value in A_0 with overwhelming probability. We now analyze the probability of this event happening. First, notice that in an ideal execution, $\mathcal{S}_2$ must hand the corrupted P_2's input to $\mathcal{F}_f$ *before* receiving anything. Thus, $\mathcal{S}_2$ has *no information* on x_1 when it chooses $x_2 \in \{\alpha_2, \alpha_2'\}$. Since $x_1 \in_{\mathrm{R}} \{\alpha_1, \alpha_1'\}$ and since, by the definition of embedded-XOR, exactly one of $f(x_1, \alpha_2), f(x_1, \alpha_2')$ is in A_0 (and the other is in A_1), it follows that no matter what $\mathcal{S}_2$ chooses as P_2's input, it cannot cause the output to be in A_0 with probability greater than 1/2. Therefore, $\mathcal{Z}$ outputs 1 in an ideal execution with probability at most negligibly greater than 1/2, while in the real execution $\mathcal{Z}$ output 1 with overwhelming probability; i.e., $\mathcal{Z}$ distinguishes the real and ideal executions. This contradicts the security of Π_f. ■

4.4 A Characterization

Let $f : X \times X \to \{0,1\}^*$. Each of Theorems 7 and 8 provides a necessary condition for $\mathcal{F}_f$ to be securely realizable (namely, f should not contain an insecure minor or an embedded-XOR, respectively). Theorem 6 gives a characterization of those functionalities $\mathcal{F}_f$ that are securely realizable, assuming that f depends on the input of one party only. In this section we show that the combination of these three theorems is actually quite powerful. Indeed, we show that this provides a full characterization of the two-party, single output deterministic functions that are realizable (with the output known to both parties). In fact, we show that the realizable functions are very simple.

Theorem 9 *Let $f : X \times X \to \{0,1\}^*$ be a function and let $\mathcal{F}_f$ be a functionality that receives x_1 and x_2 from P_1 and P_2 respectively, and hands both parties $f(x_1, x_2)$. Then, $\mathcal{F}_f$ can be securely realized in a universally composable way by a non-trivial protocol if and only if f is an efficiently invertible function depending on (at most) one of the inputs (x_1 or x_2).*

Proof: First, we prove the theorem for the case that f contains an insecure-minor or an embedded-XOR (with respect to either P_1 or P_2). By Theorems 7 and 8, in this case $\mathcal{F}_f$ cannot be securely realized. Indeed, such functions f do *not* solely depend on the input of a single party; that is, for each party there is some input for which the output depends on the other party's input.

Next, we prove the theorem for the case that f does not contain an insecure-minor (with respect to either P_1 or P_2) or an embedded-XOR. We prove that in this case f depends on the input of (at most) one party and hence, by Theorem 6, the present theorem follows. Pick any $x \in X$ and let $a = f(x, x)$. Let $B_1 = \{x_1 | f(x_1, x) = a\}$ and $B_2 = \{x_2 | f(x, x_2) = a\}$. Since $f(x, x) = a$ then both sets are non-empty. Next, we claim that at least one of $\bar{B}_1$ and $\bar{B}_2$ is empty; otherwise, if there exist $\alpha_1 \in \bar{B}_1$ and $\alpha_2 \in \bar{B}_2$, then setting $\alpha_1' = \alpha_2' = x$ gives us a minor which is either an insecure minor or an embedded-XOR. To see this, denote $b = f(\alpha_1, x)$ and $c = f(x, \alpha_2)$; by the definition of $\bar{B}_1, \bar{B}_2$ both b and c are different than a. Consider the possible values for $d = f(\alpha_1, \alpha_2)$. If $d = b$ or $d = c$, we get an insecure minor; if $d = a$ or $d \notin \{a, b, c\}$, we get an embedded-XOR. Thus, we showed that at least one of $\bar{B}_1, \bar{B}_2$ is empty; assume, without loss of generality, that it is $\bar{B}_2$. There are two cases:

1. $\bar{B}_1$ *is also empty:* In this case, f is constant. This follows because when $\bar{B}_1 = \bar{B}_2 = \phi$, we have that for every x_1, x_2, $f(x_1, x) = f(x, x_2) = a$. Assume, by contradiction, that there exists a point (x_1, x_2) such that $f(x_1, x_2) \neq a$ (and so f is not constant). Then, x, x_1, x, x_2 constitutes an insecure minor, and we are done.
2. $\bar{B}_1$ *is not empty:* In this case, we have that for every $x_1 \in \bar{B}_1$ the function is fixed (i.e., $f(x_1, \cdot)$ is a constant function). This follows from the following. Assume by contradiction that there exists a point x_2 such that $f(x_1, x) \neq f(x_1, x_2)$ (as must be the case if $f(x_1, \cdot)$ is not constant). We claim that x, x_1, x, x_2 constitutes an insecure minor. To see this, notice that $f(x, x) = f(x, x_2) = a$ (because $\bar{B}_2 = \phi$). However, $f(x_1, x) \neq f(x_1, x_2)$. By definition, this is an insecure minor.

We conclude that either f is a constant function, or for every x_1, the function $f(x_1, \cdot)$ is constant. That is, f depends only on the input of P_1, as needed. ■

5 Probabilistic Functionalities

This section concentrates on *probabilistic* two-party functionalities where both parties obtain the same output. We show that the only functionalities that can be realized are those where one of the parties can almost completely determine the (joint) output. This rules out the possibility of realizing any "coin-tossing style" functionality, or any functionality whose outcome remains "unpredictable" even if one of the parties deviates from the protocol. It is stressed, however, that our result does not rule out the realizability of other useful probabilistic functionalities, such as functionalities where, when both parties remain uncorrupted, they can obtain a random value that is unknown to the adversary. An important example of such a functionality, that is indeed *realizable*, is key-exchange.

Let $f = \{f_k\}$ be a family of functions where, for each value of the security parameter k, we have that $f_k : X \times X \to \{0,1\}^*$ is a probabilistic function. We say that f is unpredictable for P_2 if there exists a value $x_1 \in X$ such that for all $x_2 \in X$ and for all $v \in \{0,1\}^*$ it holds that $\Pr(f_k(x_1, x_2) = v) < 1 - \epsilon$, where $\epsilon = \epsilon(k)$ is a non-negligible function. (We term such x_1 a safe value.) Unpredictable functions for P_1 are defined analogously. A function family is unpredictable if it is unpredictable for either P_1 or P_2.

Theorem 10 *Let $f = \{f_k\}$ be an unpredictable function family and let $\mathcal{F}_f$ be the two-party functionality that, given a security parameter k, waits to receive x_1 from P_1 and x_2 from P_2, then samples a value v from the distribution of $f_k(x_1, x_2)$, and hands v to both P_1 and P_2. Then, $\mathcal{F}_f$ cannot be securely realized by any non-trivial protocol.*

The proof can be found in the full version of this paper [CKL03], and follows from a theorem for the probabilistic case that is analogous to Theorem 4.

References

[B91] D. Beaver. Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology,* 4(2):75–122, 1991.

[B96] D. Beaver. Adaptive Zero-Knowledge and Computational Equivocation. In 28*th STOC*, pages 629–638, 1996.

[BMM99] A. Beimel, T. Malkin and S. Micali. The All-or-Nothing Nature of Two-Party Secure Computation. In *CRYPTO'99*, Springer-Verlag (LNCS 1666), pages 80–97, 1999.

[BGW88] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In 20*th STOC,* pages 1–10, 1988.

[C01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42*nd FOCS,* pages 136–145, 2001. Full version available at `http://eprint.iacr.org/2000/067`.

[CFGN96] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Multi-Party Computation. In 28*th STOC*, pages 639–648, 1996.

[CF01] R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO 2001*, Springer-Verlag (LNCS 2139), pages 19–40, 2001.

[CK01] R. Canetti and H. Krawczyk. Analysis of Key Exchange Protocols and Their Use for Building Secure Channels. In *Eurocrypt 2001*, Springer-Verlag (LNCS 2045), pages 453–474, 2001.

[CK02] R. Canetti and H. Krawczyk. Universally composable key exchange and secure channels. In *Eurocrypt 2002*, Springer-Verlag (LNCS 2332), pages 337–351, 2002.

[CKL03] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-up Assumptions (full version). *Cryptology ePrint Archive*, `http://eprint.iacr.org/`, 2003.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In 34*th STOC*, pages 494–503, 2002.

[CK89] B. Chor, and E. Kushilevitz. A Zero-One Law for Boolean Privacy. In 21*st STOC*, pages 62–72, 1989.

[C86] R. Cleve. Limits on the security of coin-flips when half the processors are faulty. In 18*th STOC*, pages 364–369, 1986.

[DDN00] D. Dolev, C. Dwork and M. Naor. Non-malleable cryptography. *SIAM Journal of Computing,* 30(2):391–437, 2000.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In 30*th STOC*, pages 409–418, 1998.

[GM00] J. Garay and P. Mackenzie. Concurrent Oblivious Transfer. In 41*st FOCS*, pages 314–324, 2000.

[GK88] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal of Computing*, 25(1):169–192, 1996.

[GMW87] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC,* pages 218–229, 1987.

[GL90] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90,* Springer-Verlag (LNCS 537), pages 77–93, 1990.

[HMS03] D. Hofheinz, J. Müller-Quade and R. Steinwandt. On Modeling IND-CCA Security in Cryptographic Protocols. *Cryptology ePrint Archive,* Report 2003/024, `http://eprint.iacr.org/`, 2003.

[KKMO00] J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and Completeness in Private Computations. *SICOMP*, 29(4):1189–1208, 2000.

[K00] J. Kilian. More general completeness theorems for secure two-party computation. In 32*nd STOC*, pages 316–324, 2000.

[K89] E. Kushilevitz. Privacy and Communication Complexity. In 30*th FOCS*, pages 416–421, 1989.

[MR91] S. Micali and P. Rogaway. Secure computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[NY90] M. Naor and M. Yung. Public key cryptosystems provably secure against chosen ciphertext attacks. In 22*nd STOC*, 427–437, 1990.

[RB89] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In 21*st STOC*, pages 73–85, 1989.

[RS91] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO'91,* Springer-Verlag (LNCS 576), pages 433–444, 1991.

[RK99] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt'99*, Springer-Verlag (LNCS 1592), pages 415–431, 1999.