J. Cryptol. (2011) 24: 517–544
DOI: 10.1007/s00145-010-9069-7

Journal of
CRYPTOLOGY

Impossibility Results for Universal Composability in Public-Key Models and with Fixed Inputs*

Dafna Kidron and Yehuda Lindell

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
dafna.kidron@gmail.com; lindell@cs.biu.ac.il

Communicated by Ivan Damgård

Received 11 October 2007 and revised 10 May 2010
Online publication 8 June 2010

Abstract. Universal composability and concurrent general composition consider a setting where secure protocols are run concurrently with each other and with arbitrary other possibly insecure protocols. Protocols that meet the definition of universal composability are guaranteed to remain secure even when run in this strongly adversarial setting. In the case of an honest majority, or where there is a trusted setup phase of some kind (like a common reference string or the key-registration public-key infrastructure of Barak et al. in FOCS 2004), it has been shown that any functionality can be securely computed in a universally composable way. On the negative side, it has also been shown that in the *plain model* where there is no trusted setup at all, there are large classes of functionalities which cannot be securely computed in a universally composable way without an honest majority.

In this paper, we extend these impossibility results for universal composability. We study a number of public-key models and show for which models the impossibility results of universal composability hold and for which they do not. We also consider a setting where the inputs to the protocols running in the network are fixed before any execution begins. The majority of our results are negative and we show that the known impossibility results for universal composability in the case of no honest majority extend to many other settings.

Key words. Universal composability, Impossibility results, Concurrent general composition, Public-key models.

1. Introduction

In the setting of secure multiparty computation, a set of parties with private inputs wish to jointly compute some functionality of their inputs. Loosely speaking, the security requirements of such a computation are that nothing is learned from the protocol other than the output (privacy), and that the output is distributed according to the prescribed functionality (correctness). More exactly, the result of an execution of a secure protocol

* This research was partially supported by the Israel Science Foundation (grant No. 781/07).

must be like the result of an "ideal execution" with an incorruptible trusted party who honestly computes the function for the parties (cf. [7] or [19, Sect. 7.1]). These security requirements must hold in the face of a malicious adversary who controls some subset of the parties and can arbitrarily deviate from the protocol instructions. Powerful feasibility results have been shown for this problem in both the information-theoretic and computational settings [6,15,20,32]. In the computational setting, it has been shown that *any* multiparty probabilistic polynomial-time functionality can be securely computed for any number of corrupted parties, assuming the existence of enhanced trapdoor permutations [19,20,32].

Security Under Concurrent Composition The above-described feasibility results relate only to the stand-alone setting, where a single protocol is run in isolation. However, in modern network settings, protocols must remain secure even when many protocol executions take place concurrently and are being attacked in a coordinated manner. Informally speaking, a protocol is said to be secure under *concurrent general composition* if it is secure when run many times concurrently, alongside other secure and insecure protocols. The question of what can and cannot be securely computed in this strongly adversarial setting has been the topic of much research over recent years. In this paper, we focus on the framework of *universal composability* [8]; this framework presents a definition of security with the important property that any protocol meeting the definition is guaranteed to remain secure under concurrent general composition. Such protocols are called *UC-secure* for short.

It has been shown that UC-secure protocols exist for essentially any functionality in the case of an honest majority [8], or where there is a common reference string [13] or an active key-registration functionality [2]. Thus, in these cases, the same broad feasibility results of the stand-alone model hold (except that in the stand-alone model, neither an honest majority nor a trusted setup phase is needed). When considering the case of *no honest majority* and *no trusted setup* in the setting of universal composability, the situation is completely different. Specifically, it has been shown that in such a setting, large classes of functionalities cannot be UC realized [8,10,14,17]. Due to this, a search has been initiated to find alternative models and definitions of security for this setting; see, for example, [1,22,30,31].

Our Results—Public-Key Models In this paper, we extend the broad impossibility results of [14]. The UC impossibility results proven in [14] hold for the plain model (where there is no trusted setup whatsoever) and for the case of no honest majority. However, they do not consider the case that some basic public-key infrastructure may be in place. This is especially serious because standard secure computation can only really be carried out when there are *authenticated channels* (see [3] for a study of this issue), and in practice such authenticated channels are implemented using a *public-key infrastructure*. Thus, the actual impossibility results of [14] do not cover the most interesting setting where a basic public-key infrastructure is used for obtaining authenticated channels. The public-key infrastructure needed for obtaining authenticated channels was shown in [9] to be a basic bulletin-board functionality (i.e., this functionality has the property that any party can register any key and no checks are carried out on the registered key; the only guarantee is that a key that is retrieved for a certain party is

indeed the same as what was registered). We call the ideal functionality that implements this bulletin board the *bulletin-board CA*, denoted $\mathcal{F}_{\text{bbca}}$. We use $\mathcal{F}_{\text{bbca}}$ as our starting point and show that all of the impossibility results of [14] actually hold in the $\mathcal{F}_{\text{bbca}}$-hybrid model as well (i.e., impossibility carries over even when all parties have access to the ideal functionality that implements $\mathcal{F}_{\text{bbca}}$). That is, we prove the following:

Impossibility with only a bulletin-board CA. *There exist large classes of deterministic two-party functionalities that cannot be UC realized by any protocol, even in the $\mathcal{F}_{\text{bbca}}$-hybrid model.*

Our proof works by showing that the main lemma of [14] can be adapted to hold even when the parties are given access to the ideal functionality $\mathcal{F}_{\text{bbca}}$. This lemma shows that there exists a *successful split adversarial strategy* for every UC-secure two-party protocol. Loosely speaking, this strategy means that it is possible for an adversary to extract the honest party's input and also bias the output that the honest party receives. This clearly implies that many (if not most) functionalities cannot be UC realized. The importance of this result is in showing that UC security is also impossible to achieve in the realistic model where a public-key infrastructure is in place for obtaining authenticated channels. We remark that impossibility holds even if $\mathcal{F}_{\text{bbca}}$ enforces all keys to be *unique*.

Before proceeding to describe our other results, one remark is in order. In reality, the $\mathcal{F}_{\text{bbca}}$ ideal functionality would typically be implemented using digital certificates and a *public key* of one or more certificate authorities. Could we not just use the public key of the CA as a common reference string and thereby construct UC-secure protocols for any functionality using the result of [13]? We argue that this is not the case. This is mainly due to the fact that the level of trust needed from an authority choosing a common reference string is far higher than that needed from a CA. In order to see this, observe that the party choosing the common reference string may be able to learn all of the parties' inputs over all executions by simply passively eavesdropping on the communication. Indeed, a close look at protocols in this model demonstrates that this is usually the case. In contrast, a CA who posts false keys for honest parties must carry out an active attack in every protocol execution. Furthermore, even if it does so, it can learn whatever a man-in-the-middle attacker can learn in an unauthenticated channels setting, which is rather limited; see [3] for more discussion on this. We conclude that although implementing $\mathcal{F}_{\text{bbca}}$ requires some trust, and this trust in reality also boils down to some "string", there is a fundamental difference between trusting a public key that can be used for achieving authenticated channels and trusting a common reference string. Thus, it would be highly desirable to have UC-secure protocols in the $\mathcal{F}_{\text{bbca}}$-hybrid model. Unfortunately, this is ruled out by the aforementioned result.

Having considered this basic CA functionality, we study stronger versions with the aim of drawing the line between feasibility and impossibility, and of clarifying what is needed to bypass the impossibility. In addition to the impossibility result described above, we obtain the following informally stated results:

1. *Feasibility for bulletin-board CA with independent keys:* We consider a further strengthening of the CA to one that prevents any party from retrieving a public key

from the bulletin board before all parties have registered their keys. This assumes that the registered keys are kept secret during registration. Note that this forces the parties' keys to be independent of each other because no party can see any other party's registered key before it registers its own. We show that UC secure protocols *can* be constructed in this model. (In fact, this is very easy to achieve by simply observing that it is possible to securely toss coins in this model.)

2. *Impossibility for the bare public-key model:* A popular public-key model that has been used in a number of settings in cryptography is the "bare" public-key model. In this model, the CA is the same as the bulletin-board CA except that all keys must be registered before any execution of the secure protocol begins. We note that there is no limitation on the arbitrary other protocols that may run during the registration phase.[1] In this case, we show that once again, the UC impossibility results carry over. This is of special interest because this model has recently be used to achieve stronger notions of non-malleable concurrent zero-knowledge [18, 29]. It seems that the aim of this direction is to eventually achieve UC security (or equivalently, security under concurrent general composition) in this model. We show that the security achieved in this model *must* fall short of UC security.
3. *Feasibility for a strong bare public-key model:* We observe that if the bare public-key model is strengthened so that no protocols whatsoever are run during the registration phase, then it is possible to run a coin-tossing protocol that is secure in the stand-alone model in order to construct a common reference string (and thereby achieve UC security [13]). The reason that it suffices to consider the stand-alone model for the coin-tossing is due to the fact that no protocols whatsoever are run during this period. Thus this strengthening trivially enables UC security; unfortunately, we view it as highly unrealistic.
4. *Feasibility for active key registration by the CA:* We analyze the key-registration functionality used by [2] to achieve UC security and show where the proof of impossibility fails with respect to the functionality. This highlights what properties of the functionality are used to bypass the impossibility results.

In addition to the above, we study what happens when the CA may be partially corrupted (if it is fully corrupted, then the UC impossibility results clearly hold, irrespective of what the functionality does). We show the following:

1. *Passive CA corruptions:* we observe that if the CA functionality behaves in a semi-honest way and reveals its internal state to the adversary (but otherwise acts honestly), then UC security can be achieved. This follows immediately from the fact that the CA can generate a uniformly distributed common reference string (in which case there is no hidden internal state), and thus the protocol of [13] can be used.
2. *Indistinguishable malicious CA corruptions:* we study what happens if the CA behaves maliciously, yet generates messages that are indistinguishable from those

[1] Typically, the bare public-key model was considered for self composition where the only protocol running is the secure one. Thus the issue of arbitrary other protocols was not raised. We interpret the bare public-key model in the context of concurrent general composition in this way because we view it as a far more realistic model. The interpretation whereby no protocol whatsoever is being run during the registration phase is considered next (and we call it the "strong" bare public-key model).

generated by an honest CA. We show that in this case the UC impossibility results also hold, irrespective of how the CA is defined.

We believe that our results provide a comprehensive study of UC feasibility in public-key models. It is our hope that they make sense out of the confusing myriad of public-key models that are in the literature. (Of course, we do not claim to have covered all possible public-key models, nor all that appear in the literature. Nevertheless, we hope that given the results here, an analysis of other models is relatively simple.) Our results are summarized in the following table:

The Model	The Result	Notes
Bulletin-board CA	Impossibility holds	Used by [9]
Bulletin-board with independent keys	UC security achievable	Generate a CRS and use [13]
Bare public-key model	Impossibility holds	A popular model
Strong bare public-key model	UC security achievable	Not a realistic model
Active key registration	UC security achievable	Result shown in [2]
Passive CA corruptions	UC security achievable	Generate a CRS and use [13]
Indistinguishable malicious CA corruptions	Impossibility holds	–

Concurrent General Composition with Fixed Inputs In addition to the above study on UC security in public-key models we ask another question that relates to the possibility of obtaining security under concurrent general composition. This question is concerned with how the parties' inputs are chosen. It has been shown that when the honest parties chooses their inputs adaptively,[2] then for a large class of functionalities (in fact, most functionalities), security under concurrent self composition[3] is equivalent to security under concurrent general composition [27]. Therefore, all of the impossibility results that hold for concurrent general composition also hold for concurrent self composition with adaptively chosen inputs. This equivalence does not hold when the inputs are all *fixed* ahead of time (i.e., where the honest parties receive a vector specifying the input for each execution). The fact that there exist protocols that are secure under concurrent self composition with fixed inputs but not with adaptively chosen inputs was demonstrated in [23]. Later, this separation was shown to hold even for the zero-knowledge functionality. That is, it has been shown that it is possible to construct zero-knowledge protocols that are secure under concurrent self composition with arbitrary roles (meaning that players can simultaneously be provers and verifiers), as long as the inputs are all fixed before any execution begins [4]. We stress that such a construction is impossible to achieve when inputs are adaptively chosen. Thus, it is strictly easier to achieve concurrent self composition with fixed inputs than it is to achieve with adaptively chosen inputs. We ask the following question:

For what functionalities is it possible to construct protocols that remain secure when run once concurrently together with an arbitrary other protocol, and the inputs are fixed before any execution begins.

[2] This adaptive choice of inputs means that the inputs used by honest parties may be determined as a function of the outputs that they have already received in previous executions that have concluded. We stress that it is always assumed that the adversary can chooses its inputs adaptively. The question of interest here is with respect to the honest parties.

[3] In the setting of concurrent self composition, a single protocol is run many times concurrently.

We call this setting *minimal concurrent general composition with fixed inputs* (it is minimal in the sense that there are only two protocol executions). We remark that it has already been shown that when inputs may be adaptively chosen, broad impossibility holds even if there are only two executions as above [26]. The novelty in the question here is therefore the fact that the inputs are a priori fixed. We show the following:

Impossibility for fixed inputs. *There exist large classes of deterministic two-party functionalities that cannot be securely realized by any protocol under minimal concurrent general composition with fixed inputs.*

We prove this theorem by defining a variant of the UC model where the environment first writes the inputs to all parties, and only then does the execution begin. We then show that security under minimal concurrent general composition with fixed inputs implies this UC variant, and finally that all of the impossibility results of [14] hold for this variant.

Subsequent Work The question of finding a "minimal" setup assumption for UC security has been studied recently in [16]. They show that it suffices to have a public-key infrastructure where each party has "some knowledge" of a secret associated with their public key. Their model differs from our bulletin-board and bare public-key models in this addition of guaranteed knowledge of a secret. Thus, they manage to avoid the possibility results with this assumption. Another work of relevance is that of [24] who provide a general framework for modeling different setup assumptions for achieving UC security.

2. Preliminaries

2.1. *Brief Overview of Universal Composability*

We present a very brief overview of how security is defined in the UC framework; see [8] for further details. As in other general definitions (e.g., [5,7,21,28]), 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. Informally, a protocol securely carries out a given task if running the protocol with a real adversary amounts to "emulating" an ideal process in which the parties hand their inputs to a trusted party who computes the appropriate functionality and hands their outputs back, without any other interaction. We call the algorithm run by the trusted party the ideal functionality, and describe the interaction in the ideal model to be between the parties and the ideal functionality (with the understanding that what we really mean is the trusted party running this functionality).

In order to prove the 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 every adversary $\mathcal{A}$ there should exist an "ideal process adversary", or simulator, $\mathcal{S}$ such that the distribution over all parties' inputs and outputs is essentially the same in the ideal and real processes. In the UC framework, 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 UC 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 $\mathcal{A}$. 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 the ideal functionality $\mathcal{F}$. We note that all entities run in time that is polynomial in the security parameter, denoted by k. In addition, the environment receives an initial input z, and security is required to hold for all such inputs (this makes the environment a non-uniform machine). Security in the UC framework is formalized in the following definition.

Definition 2.1. Let $\mathcal{F}$ be an ideal functionality and let π be a two-party protocol. We say that π UC realizes $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that for every environment $\mathcal{Z}$, the ensembles $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ are indistinguishable.

Variants of the UC Definition As with the results of [14], our results hold for all known variants of the UC definition and are resilient to changes in definition of polynomial-time, the order of activations and so on. Nevertheless, in order to write our proofs we need to specify a model. We take the model where all messages are sent via the adversary, including the messages that are sent between parties and the ideal functionalities. In order to model private values that may be sent between the parties and functionalities, we specify that these messages are composed of a *public header* and possibly *private contents* (although in this work there will only be one functionality that has private contents; all others are completely public). This convention was used in [13] (see [25, page 97]) and can be modeled in the regular UC framework (where messages are sent directly and privately between honest parties and the ideal functionalities), by defining a canonical form for the ideal functionality that always asks the adversary when to receive a message and when to send it (and the query is based on sending the public header). We remark that the adversary cannot modify messages sent between parties and the ideal functionality. The question of whether it can or cannot modify messages sent between honest parties is of no relevance here because we always consider the scenario where there are two parties, one of which is corrupted. For a full detailed description of the exact UC definition that we use here, see [25].

Non-trivial Protocols and the Requirement to Generate Output As we have mentioned above, in the variant of UC that we consider here, the ideal-process adversary can choose when (if ever) to deliver messages that are sent between the parties and the ideal functionality. Consequently, the definition provides no guarantee that a protocol will ever generate output or "return" to the calling protocol. 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, UC realizes any ideal functionality. However, such a protocol is clearly not interesting. We therefore use the notion of a non-trivial protocol [13]. 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). Thus, non-trivial protocols have the *minimal* property that when all participants are honest (and the adversary does not prevent any messages from being delivered), then all parties receive output. Again, as with [14], our impossibility results are for non-trivial protocols only.

The UC Composition Theorem As mentioned, a universally composable protocol remains secure under a very general composition operation. In particular, it maintains its security even when run concurrently with other arbitrary protocols that are being run by arbitrary sets of possibly different sets of parties, with possibly related inputs. Thus, universally composable protocols can be used in modern networks, and security is guaranteed. It is therefore of great importance to understand what functions can and cannot be UC realized under this definition; see [8] for more details.

2.2. *The Impossibility Results of [14]*

The impossibility results of [14] are obtained by proving a lemma that describes an "attack" that is possible against any two-party UC-secure protocol that securely realizes a deterministic function f. The lemma is then used to derive a series of impossibility results for different classes of functions. The lemma refers to deterministic, polynomial-time computable functions $f : X \times X \to \{0,1\}^* \times \{0,1\}^*$, where $X \subseteq \{0,1\}^*$ is an arbitrary, possibly infinite, domain (for simplicity it is assumed that both parties' inputs are from the same domain, but this makes no difference). The functions considered have two outputs, one for each party and are denoted $f = (f_1, f_2)$ where f_1 denotes the first party's output and f_2 denotes the second party's output.

Motivation The idea behind the main lemma of [14] is as follows. An ideal-model simulator in the UC model works by interacting with an ideal functionality; namely, it sends the functionality an input (in the name of the corrupted party) and receives 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 must be able to *"extract"* the input used by the corrupted party, in addition to causing the corrupted party to *output* a value that corresponds to the output received by the simulator from the ideal functionality.

The main point behind the lemma of [14] is the observation that in the plain model, a malicious adversary in the real model can do "whatever" the simulator can do. Thus, since the simulator can extract the adversary's input, a real adversary can extract the honest party's input in a real execution (something that should not be possible in a secure protocol). In other models of secure computation and when some trusted setup assumptions are used, this attack cannot be carried out because the simulator typically has some additional "power" that a malicious party does not. (In stand-alone secure computation, this power is usually the ability to rewind the adversary, something that cannot be done to a real party. In the UC model with setup assumptions like a common reference string, this power is the ability to choose the string and make it not necessarily uniform.)

Split Adversarial Strategies We describe the notion of a *split adversarial strategy* for a corrupted P_2 as used in [14]. In our results on impossibility results for public-key models, we will prove exactly the same lemma except that we will show that it holds in some public-key models (rather than in the plain model).

Intuitively, the adversarial strategy that is constructed for a malicious P_2 is one that consists of two separate machines: P_2^a and P_2^b. Entity P_2^a interacts with (the honest) P_1 and its aim is to "extract" the input used by the honest P_1 (it actually does this by running the ideal simulator for the protocol who, as we have mentioned, is able to extract such inputs). In contrast, entity P_2^b emulates the ideal functionality for the simulator that is run by P_2^a. Loosely speaking, P_2^a first "extracts" the input used by P_1. Entity P_2^a then hands this input to P_2^b, who 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 a value that is consistent with the input that is chosen by P_2^b (this last step must also be carried it in any ideal simulation, and so once again can also be achieved by an attack in a real execution). The formal definition of this strategy appears below. We first present the "structure" of the attack and then what it means to be "successful".

Definition 2.2 (Split Adversarial Strategy). Let $f : X \times X \to \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time function where f_1 and f_2 denote the first and second outputs of f, respectively, and let π_f be a protocol. Let $X_2 \subseteq X$ be a polynomial-size subset of inputs (i.e., $|X_2| = \text{poly}(k)$, 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 consists of machines P_2^a and P_2^b such that:

1. Upon input (X_2, x_2), party P_2 internally gives the machine P_2^b the input pair (X_2, x_2).
2. An execution between (an honest) P_1 running Π_f and $P_2 = (P_2^a, P_2^b)$ works as follows:
   (a) P_2^a interacts with P_1 according to some specified strategy.
   (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 $y_1' = f_1(x_1', x_2')$ for some $x_2' \in X_2$ of its choice (chosen according to any efficient strategy).
   (d) P_2^b hands P_2^a the value y_1', 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 "equivalent to" (the honest) P_1's input x_1 with respect to f_2. That is, the output of P_2, when computed according to f_2 and when P_2 has input $x_2' \in X_2$, is the same whether x_1 or x_1' is used. (Note that x_1' may differ from x_1 with respect to P_1's output, but only the effect on P_2's output is considered.) Furthermore, P_2^a should succeed in causing P_1 to output the value $y_1 = f_1(x_1, x_2')$. That is, the output of P_1 should be consistent with the value x_2' chosen by P_2^b.

Definition 2.3 (Successful Strategies). Let f be a polynomial-time function and π_f a protocol, as in Definition 2.2. Furthermore, let k be the security parameter and 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 $x_2 \in_R X_2$. Then, a split adversarial strategy for a malicious P_2 is said to be successful if for every $\mathcal{Z}$ as above and every input z to $\mathcal{Z}$, the following two conditions hold in a real execution of P_2 with $\mathcal{Z}$ and an honest P_1:

1. The value x_1' output by P_2^a in step (b) of Definition 2.2 is such that for *every* $x_2 \in X_2$, $f_2(x_1', x_2) = f_2(x_1, x_2)$.
2. P_1 outputs $f_1(x_1, x_2')$, where x_2' is the value chosen by P_2^b in step (c) of Definition 2.2.

It is proven in [14] that a successful split adversarial strategy exists for any protocol that UC realizes a two-party function *in the plain model*.

3. UC-Security in Public-Key Models

In this section, we investigate the question of whether or not it is possible to achieve UC-security when there is some type of public-key infrastructure (but there is no honest majority). In its basic form, a public-key infrastructure is a type of "bulletin board" where public keys are published (together with the identity of their owner) and can be retrieved by all parties in a trusted way. (By trust here, we mean that we assume that an adversary cannot modify published keys or tamper with a public-key while it is being retrieved by an honest party.) One of the central questions that arises when considering such an infrastructure is the role of the Certificate Authority (CA) who accepts keys and posts them on the bulletin board. For example, we may consider a very basic public-key model where the CA receives keys without any conditions and posts them, and we may consider a public-key model where the CA requires the users to send the secret-key that is associated with the public-key to be published. As we have discussed in the introduction, we study a number of different models, most of which have appeared in the literature in the past. For each model, we investigate the possibility of obtaining UC-secure protocols in a hybrid model in which the CA is modeled by an ideal functionality. Our proofs of impossibility are achieved by reproving the main lemma of [14] in the different models.

3.1. *Bulletin-Board Certificate Authority (CA)*

We prove broad impossibility results for achieving UC security, even when the protocol may use an ideal functionality that implements a basic bulletin-board CA. The functionality that we define for this CA carries out no checks on the keys that are registered.

Functionality $\mathcal{F}_{\text{bbca}}$

$\mathcal{F}_{\text{bbca}}$ proceeds as follows, running with parties $P_1, P_2, \ldots$ and an adversary $\mathcal{S}$:

- **Register commands:** Upon receiving a message (register, sid, v) from some party P_i, the functionality checks that no pair (P_i, v') is already recorded. If this is the case, it records the pair (P_i, v). Otherwise, it ignores the new message.
- **Retrieve commands:** Upon receiving a message (retrieve, sid, P_i) from some party P_j or the adversary $\mathcal{S}$, the functionality checks if some pair (P_i, v) is recorded. If yes, and v is the recorded value, it sends (sid, P_i, v) to P_j (or $\mathcal{S}$). Otherwise, it returns $(sid, P_i, \perp)$.

Fig. 1. The Bulletin-Board CA functionality.

The only limitation is that each party can register at most one key (this actually makes no difference and our proofs carry through even if parties can register different keys under different session identifiers or sub-session identifiers); see Fig. 1 for a formal description of the functionality.

Note that $\mathcal{F}_{\text{bbca}}$ carries out no checks on the format of the key, and also does not prevent parties from copying keys from other parties. Thus, this is arguably the most basic and simple type of CA that one could imagine. Nevertheless, this does not mean that it is useless. In fact, this is far from true as it suffices for obtaining *authenticated channels* [9]. Recall that authenticated channels are almost always assumed for secure protocols and thus an assumption like the existence of $\mathcal{F}_{\text{bbca}}$ is actually needed anytime that secure protocols are to be used.

We prove that the split adversarial strategy lemma of [14] holds in the $\mathcal{F}_{\text{bbca}}$-hybrid model, thereby implying that all of the impossibility results of [14] for deterministic functionalities hold also in this model; see Sect. 2.2 for the definition of a successful split adversarial strategy and for the intuition behind the lemma.

Lemma 3.1. *Let f be a polynomial-time two-party function, and let $\mathcal{F}_f$ be the two-party ideal functionality that receives x_1 from P_1 and x_2 from P_2, and hands them back their respective outputs $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$. If $\mathcal{F}_f$ can be UC-realized in the $\mathcal{F}_{\text{bbca}}$-hybrid model by a non-trivial protocol π_f,[4] then there exists a machine P_2^a such that for every machine P_2^b of the form described in Definition 2.2, the split adversarial strategy for $P_2 = (P_2^a, P_2^b)$ is* successful, *except with negligible probability.*

Proof. The proof is similar to the proof of the analogous lemma in [14] with the appropriate changes made due to the fact that now we are working in the $\mathcal{F}_{\text{bbca}}$-hybrid model rather than the plain model. As we have mentioned in Sect. 2.2 the basic idea is that if $\mathcal{F}_f$ can be UC realized by a protocol $\pi_{\mathcal{F}}$, then this implies the existence of an ideal-process adversary (or simulator) $\mathcal{S}$ that can *extract* the input used by $\mathcal{A}$. (It must be able to extract this input in order to send it to the ideal functionality.) The key point in the proof is that $\mathcal{S}$ must essentially accomplish this extraction while running a "straight-line black-box" simulation, meaning that it cannot rewind $\mathcal{A}$ and also has no

[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 Sect. 2.1 for details.

access to its code. Stated differently, $\mathcal{S}$ interacts with $\mathcal{A}$ just like real parties interact in a protocol execution. We remark that technically, $\mathcal{S}$ is able to rewind $\mathcal{A}$ and inspect its code. However, we will construct a specific $\mathcal{A}$ and $\mathcal{Z}$ for which these capabilities are rendered useless. Now, if $\mathcal{S}$ can extract $\mathcal{A}$'s input by interacting with it like in a real execution, and if $\mathcal{A}$ behaves like an honest party (which will be the case, as will be shown), then this means that $\mathcal{S}$ can extract an honest party's input in an honest interaction. The next step from this observation is that $\mathcal{S}$ can actually be used by an adversary to extract an honest party's input (thus $\mathcal{S}$ can be used to implement the role of P_2^a in a split adversarial strategy). We also must relate to the fact that in the $\mathcal{F}_{\text{bbca}}$-hybrid model, $\mathcal{S}$ has full control over the functionality and so can modify values that are registered and so on. This is in contrast to $\mathcal{A}$ in a real execution that has no control over the functionality. Nevertheless, we will show that the operations of the functionality are so basic that this does not provide any real advantage to $\mathcal{S}$ (essentially, $\mathcal{Z}$ can verify that it is running in a world with a proper $\mathcal{F}_{\text{bbca}}$ functionality and so can prevent $\mathcal{S}$ from gaining any advantage). The proof works by considering three scenarios:

1. *Scenario* 1*—an $\mathcal{F}_{\text{bbca}}$-hybrid scenario with a corrupted P_1:* In this scenario, we consider a specific real-world adversary $\mathcal{A}$ who has corrupted P_1, a specific environment $\mathcal{Z}$, and an execution of the protocol π_f in the $\mathcal{F}_{\text{bbca}}$-hybrid model with an honest P_2.
2. *Scenario* 2*—an ideal-world scenario with a corrupted P_1:* In this scenario, we consider an ideal execution with the same $\mathcal{Z}$ and with the ideal-world adversary/simulator $\mathcal{S}$ that is guaranteed to exist for $\mathcal{A}$ and π_f. (Again here, P_1 is corrupted and P_2 is honest.)
3. *Scenario* 3*—an $\mathcal{F}_{\text{bbca}}$-hybrid scenario with a corrupted P_2:* In this scenario, we consider a new real adversary $\mathcal{A}'$ who has corrupted P_2 and carries out a split adversarial strategy against an honest P_1.

The proof works by showing that certain properties hold in each of the scenarios. We proceed to the formal proof. Assume that the functionality $\mathcal{F}_f$ can be securely realized by a non-trivial protocol π_f. This implies that for every $\mathcal{F}_{\text{bbca}}$-hybrid adversary $\mathcal{A}$ there exist an ideal-process adversary/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 $\mathcal{F}_{\text{bbca}}$-hybrid protocol π_f with $\mathcal{A}$. We begin by defining a specific $\mathcal{F}_{\text{bbca}}$-hybrid world scenario with a specific adversary $\mathcal{A}$ and environment $\mathcal{Z}$.

Scenario 1*—the First $\mathcal{F}_{\text{bbca}}$-Hybrid Scenario* The scenario consists of an environment $\mathcal{Z}$ and adversary $\mathcal{A}$, and honest parties P_1 and P_2. In this scenario, P_1 is corrupted (and thus controlled by $\mathcal{A}$) and P_2 is honest. Note that the protocol π_f is run in the $\mathcal{F}_{\text{bbca}}$-hybrid model. This means that each party communicates with the ideal bulletin-board CA functionality during the execution of the protocol. We describe each entity's strategy separately:

- *$\mathcal{A}$'s strategy:* $\mathcal{A}$ controls party P_1 who does nothing except to respond to specific requests of $\mathcal{A}$, as follows. When P_1 receives a message (register, *sid*, v) from $\mathcal{A}$, it sends the message (register, *sid*, v) to $\mathcal{F}_{\text{bbca}}$. (Note that $\mathcal{A}$ cannot send this message for P_1 and so P_1 must do it itself.) In addition, $\mathcal{A}$ mediates between $\mathcal{Z}$ and the $\mathcal{F}_{\text{bbca}}$ functionality. Specifically, when $\mathcal{Z}$ sends a message (register, *sid*, v) to $\mathcal{A}$,

adversary $\mathcal{A}$ forwards this message to P_1 to forward to $\mathcal{F}_{\text{bbca}}$. Likewise, when $\mathcal{Z}$ sends a message (retrieve, *sid*, P) to $\mathcal{A}$ for some party P, adversary $\mathcal{A}$ forwards this message to the $\mathcal{F}_{\text{bbca}}$ functionality. When $\mathcal{A}$ receives the response (*sid*, P, v) from $\mathcal{F}_{\text{bbca}}$ it forwards this response to $\mathcal{Z}$. Finally, $\mathcal{A}$ acts as a bridge between $\mathcal{Z}$ and P_2, delivering all messages from $\mathcal{Z}$ to P_2 (except for register and retrieve messages), and delivering all message from P_2 to $\mathcal{Z}$ (that P_2 sent to P_1).

- *$\mathcal{Z}$'s strategy:* $\mathcal{Z}$ chooses inputs for the parties: x_1 for P_1 and x_2 for P_2, where $x_2 \in_R X_2$ is randomly chosen, and X_2 is a polynomial-size set of inputs chosen by $\mathcal{Z}$. The environment $\mathcal{Z}$ writes x_2 on P_2's input tape and keeps x_1 to itself.

  Next, $\mathcal{Z}$ plays the role of the honest P_1 on input x_1. That is, $\mathcal{Z}$ runs the honest P_1's protocol instructions on input x_1 and the incoming messages that it receives from $\mathcal{A}$. When instructed by the protocol π_f to perform a key registration of some key v, $\mathcal{Z}$ sends (register, *sid*, v) to $\mathcal{A}$. When instructed by the protocol to retrieve a party P's key, $\mathcal{Z}$ sends (retrieve, *sid*, P) to $\mathcal{A}$. Upon receiving the answer (*sid*, P, v) from $\mathcal{A}$, it continues with the protocol and relates to the answer as if it was received directly from $\mathcal{F}_{\text{bbca}}$. (We stress that $\mathcal{Z}$ cannot interact directly with $\mathcal{F}_{\text{bbca}}$ in the UC model. Therefore, it must work through an intermediary.)

  $\mathcal{Z}$ determines its output at the end of the protocol execution in the following way. First, it carries out consistency checks with respect to registered and retrieved keys. That is, $\mathcal{Z}$ outputs 0 if one of the following events occur during the execution:

  1. $\mathcal{Z}$ asked to retrieve P_1's key but received a key that does not equal the one that $\mathcal{Z}$ registered previously (or a key is retrieved but $\mathcal{Z}$ never registered one).
  2. $\mathcal{Z}$ asked to retrieve the key of some party P more than once and received different keys in the different requests.

  If the above checks pass, then $\mathcal{Z}$ looks at the local output that it received for P_1 and reads P_2's output tape. (Recall that $\mathcal{Z}$ plays P_1 so it receives some output from the protocol. We call this output the local-P_1 output.) $\mathcal{Z}$ outputs 1 if and only if the local-P_1 output equals $f_1(x_1, x_2)$ and P_2's output equals $f_2(x_1, x_2)$.

We have the following claim:

Claim 3.2. *In the $\mathcal{F}_{\text{bbca}}$-hybrid world described in scenario number* 1, *the environment $\mathcal{Z}$ outputs* 1 *with probability that is negligibly close to* 1.

Proof. Observe that 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. This is due to the fact that $\mathcal{Z}$ plays the honest P_1 and the adversary $\mathcal{A}$ honestly relays all messages between $\mathcal{Z}$ and P_2 and $\mathcal{Z}$ and $\mathcal{F}_{\text{bbca}}$ (where these latter messages are sent via P_1 because only it can register keys for its identity). It follows that since π_f is a non-trivial protocol, both parties receive output. In the case of honest P_1 and P_2, these outputs are $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$, respectively, except with negligible probability (otherwise, an environment would be able to distinguish the real and ideal models when no party is corrupted). Since all consistency checks by $\mathcal{Z}$ pass in this scenario (because $\mathcal{A}$ always "behaves well"), it follows that $\mathcal{Z}$ outputs 1 except with negligible probability. □

Scenario 2*—the Ideal World with* $\mathcal{F}_f$ In this scenario, we have the same environment $\mathcal{Z}$ as above, dummy parties P_1 and P_2 (where P_1 is corrupted), and the ideal adversary/simulator that is guaranteed to exist by the security of the protocol π_f; denote this adversary by $\mathcal{S}$. The strategy of $\mathcal{S}$ is not determined by us. Nevertheless, we prove that the following properties must hold:

1. *Property* 1*:* If $\mathcal{Z}$ sends a message (retrieve, *sid*, P_1) to $\mathcal{S}$ and it previously sent a (register, *sid*, v) message, then the value v_s in $\mathcal{S}$'s response equals v. Otherwise, if no register message was sent, it holds that $v_s = \perp$.
2. *Property* 2*:* If $\mathcal{Z}$ sends a message (retrieve, *sid*, P_2) to $\mathcal{S}$ and it has previously received a response (*sid*, P_2, v) for $v \neq \perp$, then the value v_s in $\mathcal{S}$'s response equals v.

The fact that these properties hold follow from the inspection of $\mathcal{Z}$'s consistency checks. Specifically, if with non-negligible probability one of the properties does not hold, then $\mathcal{Z}$ will output 0 with non-negligible probability in the ideal execution. Since $\mathcal{Z}$ outputs 1 except with negligible probability in the $\mathcal{F}_{\text{bbca}}$-hybrid execution, this implies that $\mathcal{Z}$ distinguishes the $\mathcal{F}_{\text{bbca}}$-hybrid and ideal executions, in contradiction to the security of the protocol. (The above essentially proves that although $\mathcal{S}$ plays $\mathcal{F}_{\text{bbca}}$ in this execution and so can theoretically do anything it wishes, it is actually very limited in what it can do.) We are now ready to prove the following claim.

Claim 3.3. *In the ideal execution,* $\mathcal{S}$ *must send* $\mathcal{F}_f$ *an input* x_1' *for which it holds that for every* $x_2 \in X_2$, $f_2(x_1, x_2) = f_2(x_1', x_2)$*, except with negligible probability. Furthermore,* $\mathcal{Z}$*'s local-*P_1 *output equals* $f_1(x_1, x_2)$ *except with negligible probability.*

Proof. Assume by contradiction that with non-negligible probability $\mathcal{S}$ sends an input x_1' to $\mathcal{F}_f$ such that for some $\tilde{x}_2 \in X_2$ it holds that $f_2(x_1', \tilde{x}_2) \neq f_2(x_1, \tilde{x}_2)$. This implies that if P_2 has input $\tilde{x}_2$, it will output $f_2(x_1', \tilde{x}_2) \neq f_2(x_1, \tilde{x}_2)$. By the specification of $\mathcal{Z}$, when this occurs $\mathcal{Z}$ outputs 0. In order to analyze the probability that P_2's input equals $\tilde{x}_2$, recall that X_2 is of polynomial size and P_2's input is uniformly chosen from X_2. Furthermore, the probability that $\mathcal{S}$ sends this "bad" x_1' is independent of the choice of x_2 for P_2 (because $\mathcal{S}$ has no information about x_2 when it sends x_1'). Therefore, the probability that $\mathcal{Z}$ outputs 0 is at least $1/|X_2|$ times the probability that $\mathcal{S}$ outputs the bad x_1' (which we have already said is non-negligible). Thus, $\mathcal{Z}$ outputs 0 in the ideal process with non-negligible probability. This contradicts the security of π_f because as we have already seen, $\mathcal{Z}$ outputs 0 in an $\mathcal{F}_{\text{bbca}}$-hybrid execution with at most negligible probability.

Regarding the "furthermore" part of the claim, this follows directly from the fact that if $\mathcal{Z}$'s local-P_1 output does not equal $f_1(x_1, x_2)$ then it outputs 0. □

We remark that the above proof does not use the properties that we proved regarding $\mathcal{S}$ and the register/retrieve messages. These properties will be used below.

Scenario 3*—the Second* $\mathcal{F}_{\text{bbca}}$*-Hybrid Scenario* As with the first scenario, here we also consider an $\mathcal{F}_{\text{bbca}}$-hybrid execution of π_f. However, here P_1 is honest while P_2 is corrupted (in contrast to scenario 1). We describe a split adversarial strategy for P_2 that

uses $\mathcal{S}$ in this scenario, and then show that it is successful. We begin by describing P_2^a (we will not refer to an adversary $\mathcal{A}'$ since (P_2^a, P_2^b) is the adversary here). Machine P_2^a internally invokes the simulator $\mathcal{S}$ and emulates an ideal process execution of $\mathcal{S}$ with $\mathcal{F}_f$ and the above $\mathcal{Z}$, while actually running an $\mathcal{F}_{\text{bbca}}$-hybrid execution of π_f with P_1. It works as follows:

1. When P_1 sends (register, sid, v) to $\mathcal{F}_{\text{bbca}}$, machine P_2^a delivers the message to $\mathcal{F}_{\text{bbca}}$ and internally hands the message to $\mathcal{S}$ as if it was received from $\mathcal{Z}$. (Recall that the ideal messages sent between the honest parties and ideal functionalities are sent via the adversary, and so P_2^a receives these messages.) When P_1 sends a (retrieve, sid, P_1) request to $\mathcal{F}_{\text{bbca}}$, P_2^a hands it to $\mathcal{S}$ and delivers the response when $\mathcal{S}$ sends the message (sid, P_1, v) that is intended as a reply to $\mathcal{Z}$.
2. When P_1 sends a (retrieve, sid, P_2) to $\mathcal{F}_{\text{bbca}}$, P_2^a internally hands this message to $\mathcal{S}$, as if $\mathcal{S}$ received it from $\mathcal{Z}$. When $\mathcal{S}$ outputs the response (sid, P_2, v_s), P_2^a checks if there is a recorded tuple (sid, P_2, v) where $v = v_s$ (i.e., P_2^a checks if v_s was already registered for this sid). If no such tuple exists P_2^a sends the message (register, sid, v_s) to $\mathcal{F}_{\text{bbca}}$, and only then delivers the retrieve request of P_1. (In addition, P_2^a records the tuple (sid, P_2, v_s) internally.)
3. Every message that P_2^a receives from the honest P_1 in the execution of π_f, it forwards to $\mathcal{S}$ as if $\mathcal{S}$ received it from $\mathcal{Z}$.
4. Every message (other than responses to retrieve requests) that $\mathcal{S}$ sends to $\mathcal{Z}$ in the emulation, P_2^a forwards to P_1 in the real execution.
5. 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^b hands it back a value y_1' it passes this to $\mathcal{S}$ as if it was received from $\mathcal{F}_f$ and continues the emulation as above. (Recall that y_1' is computed by P_2^b choosing some $x_2' \in X_2$ of its choice and then computing $y_1' = f_1(x_1', x_2')$; see Definition 2.2.)

We now show that the distribution of messages received and sent by $\mathcal{S}$ and P_1 in this $\mathcal{F}_{\text{bbca}}$-hybrid execution with $P_2 = (P_2^a, P_2^b)$ is statistically close to the distribution of messages received and sent by $\mathcal{S}$ and $\mathcal{Z}$ in scenario 2. We focus first on the messages that relate to the $\mathcal{F}_{\text{bbca}}$ functionality:

1. When P_1 sends a (register, sid, v) message to $\mathcal{F}_{\text{bbca}}$, machine P_2^a gives the message to $\mathcal{S}$ before actually forwarding it to the functionality. Likewise, in scenario 2, $\mathcal{S}$ receives the same message from $\mathcal{Z}$.
2. When P_1 sends a (retrieve, sid, P_2) message to $\mathcal{F}_{\text{bbca}}$, machine P_2^a first gives the message to $\mathcal{S}$, just as in scenario 2. Furthermore, in scenario 3, P_1 receives (sid, P_2, v_s) in response from $\mathcal{F}_{\text{bbca}}$, where v_s is chosen by $\mathcal{S}$. This is exactly the same as what happens in scenario 2.
3. When P_1 sends duplicate retrieve messages with the same sid in scenario 3, P_2^a does not perform another registration of v_s since the value is already registered inside $\mathcal{F}_{\text{bbca}}$. As a result, P_1 receives the same v_s from $\mathcal{F}_{\text{bbca}}$ in both requests. Nevertheless, by property 2 described above in scenario 2, $\mathcal{S}$ returns the same value in scenario 2 (except with negligible probability).
4. When P_1 sends (retrieve, sid, P_1) to $\mathcal{F}_{\text{bbca}}$ in scenario 3, $\mathcal{S}$ receives this message from P_2^a. In scenario 2, $\mathcal{S}$ receives the same message from $\mathcal{Z}$. Now, in scenario

3, P_1 receives the retrieved value from $\mathcal{F}_{\text{bbca}}$ and $\mathcal{S}$ cannot modify its value (this value was previously registered by P_1). Nevertheless, by property 1 described above in scenario 2, $\mathcal{S}$ returns the same value in scenario 2 (except with negligible probability).

Finally, we observe that since $\mathcal{Z}$ plays the honest P_1 strategy with input x_1, the distribution over the messages that it interchanges with $\mathcal{S}$ in scenario 2 is statistically close to the distribution over the messages that P_1 interchanges with $\mathcal{S}$ via P_2^a in scenario 3 (the only difference is due to the negligible probability that properties 1 and 2 may not hold). We conclude that up until the point that P_2^a hands x_1' to P_2^b, the distributions in the two scenarios are statistically close. Therefore, the distribution over the value x_1' that P_2^a hands to P_2^b in scenario 3 is statistically close to the distribution over the value x_1' that $\mathcal{S}$ sends to $\mathcal{F}_f$ in scenario 2. This implies that $P_2 = (P_2^a, P_2^b)$ is a successful split adversarial strategy with respect to requirement (1) of Definition 2.3 (see Sect. 2.2).

It remains to show that P_2 is also a successful split adversarial strategy with respect to requirement (2) of Definition 2.3. This can be shown in exactly the same way as in [14]. Namely, by Claim 3.3, we have that $\mathcal{Z}$'s local-P_1 output must equal $f_1(x_1, x_2)$ except with negligible probability. Now, assume by contradiction that with non-negligible probability P_1 outputs a value $\tilde{y}_1$ that does not equal $f_1(x_1, x_2')$ in scenario 3. Recall that P_2^b hands P_2^a the value y_1' that is computed by first choosing x_2' following an arbitrary (polynomial-time) strategy of its choice, and then computing $y_1' = f_1(x_1', x_2')$. Now, modify P_2^b to a machine $\tilde{P}_2^b$ who chooses x_2' uniformly from X_2. Since X_2 is of polynomial-size, it follows that with probability $1/\text{poly}(n)$, the value x_2' chosen by $\tilde{P}_2^b$ equals that chosen by P_2^b. Thus, if P_1 outputs $\tilde{y}_1 \neq f_1(x_1, x_2')$ with non-negligible probability with P_2^b then it will also output $\tilde{y}_1 \neq f_1(x_1, x_2')$ with non-negligible probability in the modified scenario with $\tilde{P}_2^b$. However, now notice that in this modified scenario, the value x_2' is chosen in exactly the same way as $\mathcal{Z}$ chooses it in scenario 2 (namely, it is chosen uniformly from X_2). Furthermore, the value y_1' handed to P_2^a by P_2^b in scenario 3 is distributed exactly like the value that $\mathcal{S}$ receives from $\mathcal{F}_f$ in scenario 2 (because the ideal functionality received x_1' from $\mathcal{S}$ and x_2 from the honest P_2 and x_2 is distributed exactly like x_2'). Now, as we have mentioned, $\mathcal{Z}$ outputs 0 if its local-P_1 output is different from $f_1(x_1, x_2)$ (which has exactly the same distribution as $f_1(x_1, x_2')$). We therefore conclude that P_1 must also output $f_1(x_1, x_2')$ in scenario 3, except with negligible probability (here we again apply the fact that the distribution over all messages seen in the two scenarios are statistically close). Thus, $P_2 = (P_2^a, P_2^b)$ is also a successful split adversarial strategy with respect to requirement (2) of Definition 2.3. This completes the proof of the Lemma 3.1. □

Extension to Unique Keys The above proof can be extended in a straightforward way to the case that the bulletin-board functionality $\mathcal{F}_{\text{bbca}}$ ensures that all public keys are *unique*. This is due to the fact that after $\mathcal{Z}$ registers its own key it can retrieve P_2's key and check that it is different from its own. Thus, such a strengthening of $\mathcal{F}_{\text{bbca}}$ is not helpful for constructing UC secure protocols.

3.2. *Bulletin-Board CA with Independent Keys*

We now further strengthen the public-key model and require that all the public keys be independent of one another. This is achieved by requiring that all parties complete registration before any retrieval requests are made. Of course, we also require that the adversary not see the public-keys before all registration has finished. Technically, this is achieved by defining the public-key to be part of the "private contents" of the messages sent between the honest parties and ideal functionality. The functionality is denoted $\mathcal{F}_{\text{indca}}$ (for independent-key CA) and is defined in Fig. 2.

We have the following theorem:

Theorem 3.4. *Assume that enhanced trapdoor permutations and dense cryptosystems exist. Then, for any multi-party ideal functionality $\mathcal{F}$, there exists a non-trivial protocol π that UC realizes $\mathcal{F}$ in the $\mathcal{F}_{\text{indca}}$-hybrid model in the presence of malicious, static adversaries, and for any number of corruptions.*

The theorem is proven by showing how the common reference string functionality $\mathcal{F}_{\text{crs}}$ with a uniformly distributed string can be UC realized in the $\mathcal{F}_{\text{indca}}$-hybrid model. We then apply the results of [13] that show that any functionality can be UC realized in the $\mathcal{F}_{\text{crs}}$-hybrid model. (We remark that the above theorem can also be stated for adaptive adversaries. We state the static version for the sake of simplicity.)

For the sake of completeness, we present the $\mathcal{F}_{\text{crs}}$ functionality in Fig. 3. We note that the functionality is fixed with a given polynomial, mandating the length of the reference string. We prove the following:

Claim 3.5. *The $\mathcal{F}_{\text{crs}}$ functionality can be UC realized in the $\mathcal{F}_{\text{indca}}$-hybrid model, for any number of corrupted parties.*

Proof. The proof of this claim is straightforward and follows from the simple observation that in the $\mathcal{F}_{\text{indca}}$-hybrid model it is possible to carry out perfect coin-tossing.

Functionality $\mathcal{F}_{\text{indca}}$

$\mathcal{F}_{\text{indca}}$ initializes a variable allow to 1 and proceeds as follows, running with parties $P_1, P_2, \ldots$ and an adversary $\mathcal{S}$:

- **Register commands:** Upon receiving a message (register, *sid*, v) from some party P_i, the functionality checks that allow $= 1$ and that P_i has not already registered a key v'. If the checks pass, then it records the pair (P_i, v). Otherwise, it ignores the new message. (The public-header of the register command consists of (register, *sid*) and the private contents consists of v.)
- **Retrieve commands:** Upon receiving a message (retrieve, *sid*, P_i) from some party P_j or the adversary $\mathcal{S}$, the functionality checks if some pair (P_i, v) is recorded. If yes, and v is the recorded value, it sets allow $= 0$ and sends (*sid*, P_i, v) to P_j (or $\mathcal{S}$). Otherwise, it returns (*sid*, P_i, $\perp$). (The entire retrieve command can be placed in the public header.)

Fig. 2. The Independent-Key CA functionality.

Functionality $\mathcal{F}_{\text{crs}}$

$\mathcal{F}_{\text{crs}}$ proceeds as follows, running with parties $P_1, P_2, \ldots$ and an adversary $\mathcal{S}$:

- Upon receiving a message (crs, sid) from some party P_i, the functionality checks if a pair (sid, r) has been recorded. If yes, it returns (crs, sid, r) to P_i. Otherwise, it chooses a uniformly distributed r (of a given length), records (sid, r) and returns (crs, sid, r) to P_i.

Fig. 3. The uniformly distributed common reference string functionality.

Namely, all parties register uniformly distributed strings (of a given length). Once a party has registered its string, it sends a message saying that it has done so to all others. When a party receives messages from *all* others that they have registered, it retrieves all the strings and defines the common reference string to be the exclusive-or of all the strings. (We stress that the party checks that all parties have indeed registered keys. If not, it aborts.) Clearly, the result is uniformly distributed because the adversary does not know any of the honest parties' registered strings when the corrupted parties register their strings. □

We conclude by commenting on where the proof of Lemma 3.1 fails in this setting. This can be seen by noting that in scenario 2, $\mathcal{Z}$ must register its public-key via $\mathcal{S}$. Thus, $\mathcal{S}$ essentially learns P_1's key before it needs to register its own. This is in contrast to scenario 3 where the key registered by P_1 is secret until P_2 registers its own key.

3.3. *The Bare Public-Key Model*

The bare public-key model, as introduced by [12], has the property that parties can only register public keys *before* the protocol executions begin. Typically, this model has been considered for concurrent self composition where the secure protocol is the only protocol running. There are two ways of interpreting this in the setting of concurrent general composition. An interpretation leading to a more realistic model is one that states that the only limitation is that the public keys are registered before the *secure protocol* begins. In particular, other protocols may be running during the key registration phase. (Stated differently, the requirement that the public keys all be registered before the secure protocol begins is one that is *local* to the secure protocol, and is not a global network requirement.) We remark that this model is very realistic because it is possible to set a date where the secure protocol begins running and to close key registration before this time. A second interpretation is that no protocols may run during the key registration phase. We view this as highly unrealistic, but nevertheless study it in Sect. 3.4.

We remark that the bare public-key model has been used in a number of papers in order to bypass lower bounds and impossibility results. For example, it was used by [12] (and many later works) in order to construct constant-round resettable zero-knowledge protocols (something that is impossible in the plain model). Recently, it has been used to achieve stronger notions of concurrent non-malleable zero-knowledge [18,29].[5] Our

[5] We remark that [18] call their model the "authenticated public-key model". Nevertheless, this is the same as the bare public-key model that was considered previously and is considered here.

Functionality $\mathcal{F}_{\text{bpk}}$

$\mathcal{F}_{\text{bpk}}$ initializes a variable allow to 1 and proceeds as follows, running with parties $P_1, P_2, \ldots$ and an adversary $\mathcal{S}$:

- **Register commands:** Upon receiving a message (register, *sid*, v) from some party P_i, the functionality checks that allow $= 1$ and that P_i has not already registered a key v'. If the checks pass, then it records the pair (P_i, v). Otherwise, it ignores the new message.
- **Retrieve commands:** Upon receiving a message (retrieve, *sid*, P_i) from some party P_j or the adversary $\mathcal{S}$, the functionality checks if some pair (P_i, v) is recorded. If yes, and v is the recorded value, it sends (*sid*, P_i, v) to P_j (or $\mathcal{S}$). Otherwise, it returns (*sid*, P_i, $\perp$).
- **Halt registration commands:** Upon receiving a message (HaltRegistration, *sid*) from any party P_i or the adversary $\mathcal{S}$, the functionality sets allow $= 0$.

Fig. 4. The bare public-key model functionality.

results here show that it is impossible to further strengthen these results to achieve protocols that are UC-secure (or equivalently, secure under concurrent general composition).

We define the bare-public key functionality in Fig. 4. The functionality has an internal Boolean flag, called allow, which is set to 0 once one of the parties sends the message HaltRegistration. This step represents the end of the registration phase and from this point on, the functionality rejects any registration request from any party. We allow any party to call HaltRegistration to ensure that no honest party begins running the secure protocol before the registration phase is halted. We stress that as with all the CA functionalities we have seen so far, with the exception of $\mathcal{F}_{\text{indca}}$, the public keys are not secret during the registration phase. (If this were the case, then $\mathcal{F}_{\text{bpk}}$ would just be a special case of $\mathcal{F}_{\text{indca}}$.)

We have the following lemma:

Lemma 3.6. *Let f be a polynomial-time two-party function, and let $\mathcal{F}_f$ be the two-party ideal functionality that receives x_1 from P_1 and x_2 from P_2, and hands them back their respective outputs $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$. If $\mathcal{F}_f$ can be UC-realized in the $\mathcal{F}_{\text{bpk}}$-hybrid model by a non-trivial protocol π_f, then there exists a machine P_2^a such that for every machine P_2^b of the form described in Definition 2.2, the split adversarial strategy for $P_2 = (P_2^a, P_2^b)$ is* successful, *except with negligible probability.*

Proof. Without loss of generality, we focus on *canonical* secure protocols that begin with the following steps (the instructions are stated for party P_i and are the same for all parties):

1. Party P_i chooses a public-key v_i (according to the protocol instructions) and sends (register, *sid*, v_i) to $\mathcal{F}_{\text{bpk}}$.
2. For each $j \neq i$, party P_i sends (retrieve, *sid*, P_j) to $\mathcal{F}_{\text{bpk}}$ and saves the retrieved key. If it receives back a tuple (*sid*, P_j, $\perp$), as would be the case if P_j has not yet registered a key, then it continues trying to retrieve the key.
3. After retrieving all the parties' keys, P_i sends (HaltRegistration, *sid*) and begins the secure protocol execution.

4. Whenever P_i is instructed to retrieve a key within the secure protocol, it takes the appropriate key that was previously stored.

Our focus on canonical protocols is without loss of generality because under the assumption that all key registration takes place before the protocol executions begin, nothing is lost by having the parties retrieve all keys before the protocol begins.

Once we are given that the secure protocol is as above, the proof of the lemma here is almost identical to that Lemma 3.1. The only important observation relates to scenario 3. Recall that the corrupted P_2 does not run the canonical protocol described above, but rather works as in Lemma 3.1. In particular, the strategy of machine P_2^a within P_2 is such that when P_1 first asks to retrieve P_2's key, P_2^a sends this request to $\mathcal{S}$ who returns a key v_s. Party P_2 registers this key v_s with the CA functionality and only then delivers the retrieve request from P_1 to the functionality. In principle, this is a problem in the bare public-key model because the key v_s is chosen later (and may depend on the protocol messages). Nevertheless, observe that this registration of v_s only takes place in the *first* retrieve request. In contrast, in all later retrieve requests P_2 does nothing but deliver the retrieve requests and responses between P_1 and the functionality. Now, in the canonical protocol form, P_1 retrieves P_2's key before sending a HaltRegistration message. This ensures that the first retrieval of P_2's key is carried out during the registration phase, and thus P_2^a can still register the key v_s that $\mathcal{S}$ chooses. □

3.4. *A Strong Bare Public-Key Model*

As we have mentioned, it is possible to consider an even stronger bare public-key model in which it is guaranteed that *no protocols whatsoever are executing* during the key registration phase. This can be formally modeled in the UC framework by having a period where $\mathcal{Z}$ cannot interact with $\mathcal{A}$. In this case, it is easy to see that UC security can be achieved by just having all parties run a single coin-tossing protocol that is secure in the stand-alone model (since no other protocols are running, the protocol runs "stand-alone"). The result of the coin-tossing is then taken as a common reference string, thereby allowing the use of [13] to UC-realize any functionality. We personally do not find this setting very interesting because it seems highly unrealistic to expect that there be a sterile period where no protocols whatsoever are executed.

Of course, the proof of Lemma 3.1 in this case is due to the fact that $\mathcal{Z}$ cannot register the public key in scenarios 1 and 2. Thus, the key must be chosen by $\mathcal{A}$ (or $\mathcal{S}$), giving $\mathcal{S}$ an advantage.

3.5. *Active Key Registration by the CA*

All of the public-key models that we have seen above have the property that the CA is passive in that it merely accepts keys (albeit while invoking certain checks). In this section, we consider the key registration functionality introduced by [2] that plays a more active role.

In [2], it has been shown that essentially any functionality can be UC-realized in the $\mathcal{F}_{\text{kr}}^f$-hybrid model, for any number of corrupted parties. Thus, clearly an analogue to Lemma 3.1 cannot be proven for this model. Our aim in this section is therefore to explain *where* the proof of Lemma 3.1 fails when considering the $\mathcal{F}_{\text{kr}}^f$ functionality. This

Functionality $\mathcal{F}_{\text{kr}}^{f}$

$\mathcal{F}_{\text{kr}}^{f}$ is parameterized by a function f and a security parameter n. It initializes a set R of strings to be empty at the first activation and then proceeds as follows, running with parties $P_1, P_2, \ldots$ and an adversary $\mathcal{S}$:

- **Register commands:** Upon receiving a message (register, *sid*) from some party P_i (that is either corrupted or uncorrupted), the functionality sends (register, *sid*, P_i) to $\mathcal{S}$ and receives a value p' back. Then, if $p' \in R$, it sets $p = p'$. Otherwise, it chooses a random string $r \in_R \{0,1\}^n$, sets $p = f(r)$ and adds p to R. Finally, it records the pair $(P_i, f(r))$ and sends (registered, *sid*, p) to P_i and $\mathcal{S}$.
- **Register by a corrupted party:** Upon receiving a message (register, *sid*, r) from a corrupted party P_i, the functionality records $(P_i, f(r))$. In this case, $f(r)$ is not added to R.
- **Retrieve commands:** Upon receiving a message (retrieve, *sid*, P_i) from some party P_j, the functionality sends a message (retrieve, *sid*, P_i, P_j) to $\mathcal{S}$ and obtains a value p back from $\mathcal{S}$. If (P_i, p) is recorded, then the functionality returns (*sid*, P_i, p) to P_j. Else, it returns (*sid*, P_i, $\perp$) to P_j.

Fig. 5. The key registration functionality of [2].

highlights the crucial property of the $\mathcal{F}_{\text{kr}}^{f}$ functionality that enables the construction of protocols and thus can be useful when attempting to design other CA functionalities that can be used for constructing UC-secure protocols.

Failure of Lemma 3.1 in the $\mathcal{F}_{\text{kr}}^{f}$-Hybrid Model In the second scenario of Lemma 3.1, the simulator $\mathcal{S}$ plays the role of the CA functionality $\mathcal{F}_{\text{kr}}^{f}$. In particular, this means that it determines the value r to be used in computing a new key $f(r)$. The main issue that arises here is that, depending on the choice of the function f, it may not be possible for $\mathcal{Z}$ to distinguish the case that a key p is correctly formed (i.e., by choosing r and computing $f(r)$) from the case that is not correctly formed. Indeed, in the construction by [2], they utilize the fact that keys can be generated in alternate ways that all look indistinguishable from the defined generation $f(r)$. Of course, this is crucial when moving to scenario 3 because party P_2 cannot choose the keys as it wishes and so cannot emulate scenario 2 while in scenario 3. The conclusion is that in order to overcome these impossibility results, a CA functionality is needed that breaks the symmetry between the capabilities of the ideal simulator $\mathcal{S}$ when "playing the role" of the functionality and a true corrupted party who interacts with the actual functionality.

3.6. *CA Corruptions and UC Security*

In this section, we investigate the possibility of obtaining UC secure protocols even when the CA may be corrupted. Of course, this is of interest only for public-key models for which it is possible to achieve UC security (without corruptions). Thus, for example, this is of interest in the $\mathcal{F}_{\text{crs}}$, $\mathcal{F}_{\text{kr}}^{f}$ and $\mathcal{F}_{\text{indca}}$ models. We also note that if full malicious corruptions are allowed, then there is no difference between the public-key model and the plain model (the functionality can be viewed as a regular party). Thus, UC security

can clearly not be realized in such a case. We therefore consider weaker corruptions. We will state out results for general CA functionalities and not specific ones; see below.

3.6.1. *Passive (Semi-Honest) CA Corruptions*

We say that a CA functionality is passively corrupted if the adversary has access to the internal state of the functionality, but the functionality continues to act according to its specification. (Thus, the adversary has malicious corruptions of regular parties but only semi-honest corruptions of the CA functionality.) We show that there *exists* a CA functionality that enables the construction of UC secure protocols even under passive corruptions. The functionality is the common-reference string functionality $\mathcal{F}_{\text{crs}}$ defined with a uniformly distributed string, as in Fig. 3. We have the following theorem:

Theorem 3.7. *Assume that enhanced trapdoor permutations and dense cryptosystems exist. Then, for any multi-party ideal functionality $\mathcal{F}$, there exists a non-trivial protocol π that UC realizes $\mathcal{F}$ in the $\mathcal{F}_{\text{crs}}$-hybrid model in the presence of malicious, static adversaries, and for any number of corruptions. This holds even when the $\mathcal{F}_{\text{crs}}$ functionality is run by a party that is corrupted in a semi-honest fashion.*

The proof of the theorem follows from the fact that UC secure protocols can be achieved in the uniformly distributed $\mathcal{F}_{\text{crs}}$, as shown in [13]. This is due to the fact that when the common reference string is uniformly distributed, the functionality has *no* additional internal state. Thus, the semi-honest corruption provides no advantage to the adversary.

There is one important point to note regarding the above. Namely, in order to prove security in this model (where the adversary has access to the internal state of $\mathcal{F}_{\text{crs}}$) it must be the case that the common reference string is generated truly randomly and not pseudorandomly. Formally, this is achieved by having $\mathcal{F}_{\text{crs}}$ simply output random bits taken directly from its random tape. This is needed because the UC simulator must be able to choose the random tape as it wishes (typically so that it includes some type of trapdoor), and if $\mathcal{F}_{\text{crs}}$ takes a smaller part of its random tape and applies a pseudorandom generator, the simulator will not be able to do this. Note that in practice, this means that a (semi) trusted party who provides the $\mathcal{F}_{\text{crs}}$ service must generate the common reference string using a pure hardware method. In particular, it cannot take a small random seed and apply a pseudorandom generator.

3.6.2. *Indistinguishable Malicious CA Corruptions*

An indistinguishable malicious corruption of the CA entity means that the adversary has full control over the CA, with the limitation that the distribution over the messages produced by the corrupted CA is indistinguishable from the distribution over the messages produced by an honest CA. Stated differently, the CA behaves in an *honest-looking manner*; see [11].[6] We show that Lemma 3.1 holds also in this scenario.

[6] We note that [11] differentiates between global honest-looking behavior and local honest-looking behavior. In the setting of two-party computation where one party is corrupted, this makes no difference.

Lemma 3.8. *Let f be a polynomial-time two-party function, and let $\mathcal{F}_f$ be the two-party ideal functionality that receives x_1 from P_1 and x_2 from P_2, and hands them back their respective outputs $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$. If $\mathcal{F}_f$ can be UC-realized in the $\mathcal{F}_{\text{bbca}}$-hybrid model by a non-trivial protocol π_f, then there exists a machine P_2^a such that for every machine P_2^b of the form described in Definition 2.2, the split adversarial strategy for $P_2 = (P_2^a, P_2^b)$ is* successful, *except with negligible probability. This holds even if $\mathcal{F}_{\text{bbca}}$ is only honest-looking, as defined in* [11].

Proof. Once again, the proof is very similar to the proof of Lemma 3.1. Recall that in scenario 3, party P_2 is corrupted and internally runs the code of the simulator $\mathcal{S}$ that is guaranteed to exist in scenario 2. In scenario 3, P_2 emulates the ideal process with $\mathcal{Z}$ for $\mathcal{S}$, while interacting with a real honest party P_2. The central point here is that since the CA is also corrupted, the adversarial P_2 can determine the messages that the CA sends to P_1 in scenario 3, and make them be exactly the CA functionality messages that $\mathcal{S}$ sends to $\mathcal{Z}$ in scenario 2 (recall that in scenario 2, $\mathcal{S}$ plays the role of the CA functionality).

The only additional point to prove is that in scenario 2, the messages generated by the ideal adversary $\mathcal{S}$ when it plays the role of the CA functionality are indistinguishable from the messages that the real honest CA functionality generates (even when viewed together with the entire transcript). However, this is derived from the basic UC definition that guarantees that the environment outputs 1 with at most negligible difference between the real and ideal models. More specifically, if it were true that these messages of $\mathcal{S}$ are distinguishable, then there exists an appropriate polynomial-time distinguisher D that distinguishes them. Now, all that needs to be done is to modify $\mathcal{Z}$ so that at the end of the execution it runs D on the transcript and outputs whatever D outputs. This implies that $\mathcal{Z}$ also distinguishes, in contradiction to the assumed security of the protocol. Given this, $\mathcal{Z}$ can be modified back to the way it was and the proof continues as before. □

4. Universal Composability with Fixed Inputs

An interesting question that arises in the setting of concurrent composition is how inputs are chosen. As we have mentioned in the introduction, when honest parties choose their inputs adaptively (as a function of previous outputs), concurrent self composition is equivalent to concurrent general composition [27]. Therefore, all of the impossibility results that hold for concurrent general composition also hold for concurrent self composition with adaptively chosen inputs. In contrast, when the honest parties' inputs are all fixed ahead of time, there exist functionalities that can be securely computed under concurrent self composition but not under concurrent general composition [4,23,26].

In this section, we ask whether or not an analogous situation holds for concurrent general composition. That is, is it possible to achieve concurrent general composition with fixed inputs in cases that it is impossible where the inputs may be chosen adaptively? To be more exact, we investigate for which functions f it is possible to construct a secure protocol π_f for the functionality $\mathcal{F}_f$ (as defined above) such that π remains secure when run concurrently with a *single other protocol* π', and when the inputs to π_f and π' are fixed before either execution begins.

Defining Concurrent General Composition with Fixed Inputs We define the real and hybrid models exactly as in [26] with the exception that we denote the vector of inputs for π_f by $\overline{x}_{\pi_f}$ and for π' by $\overline{x}_{\pi'}$, and we fix these inputs at the onset.[7] We denote by $\text{REAL}_{\pi_f,\pi',\mathcal{A}}(n, \overline{x}_{\pi_f}, \overline{x}_{\pi'}, z)$ a real concurrent execution of protocols π_f and π' with adversary $\mathcal{A}$, where n is the security parameter, $\overline{x}_{\pi_f}$ is the vector of inputs for the parties in π_f, $\overline{x}_{\pi'}$ is the vector of inputs for the parties in π', and z is the auxiliary input for the adversary. Likewise, we denote by $\text{HYBRID}^{\mathcal{F}_f}_{\pi',\mathcal{S}}(n, \overline{x}_{\pi_f}, \overline{x}_{\pi'}, z)$ an execution of the protocol π' together with an ideal call to $\mathcal{F}_f$ that takes the place of the real execution of π_f; the inputs are the same as above. Intuitively, π_f is secure in this setting if the output distributions of the REAL and HYBRID executions are indistinguishable. We call this *minimal* concurrent general composition because only two protocol executions take place. We have the following formal definition:

Definition 4.1. Let π_f be a polynomial-time protocol and let $\mathcal{F}_f$ be an ideal functionality. Then, π_f *securely realizes* $\mathcal{F}_f$ *under minimal concurrent general composition with fixed inputs* if for every polynomial-time protocol π' and every probabilistic non-uniform polynomial-time real-model adversary $\mathcal{A}$ for the concurrent executions of π_f and π', there exists a probabilistic non-uniform polynomial-time hybrid-model adversary $\mathcal{S}$ such that for all $\overline{x}_{\pi_f}, \overline{x}_{\pi'} \in (\{0,1\}^*)^m$ and $z \in \{0,1\}^*$:

$$\left\{\text{HYBRID}^{\mathcal{F}_f}_{\pi',\mathcal{S}}(n, \overline{x}_{\pi_f}, \overline{x}_{\pi'}, z)\right\}_{n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi_f,\pi',\mathcal{A}}(n, \overline{x}_{\pi_f}, \overline{x}_{\pi'}, z)\right\}_{n\in\mathbb{N}}$$

where $\stackrel{\text{c}}{\equiv}$ denotes computational indistinguishability.

In order to prove our results, we introduce a restricted UC model where the environment first writes the inputs to all parties' input tapes and only once it has finished do the parties begin executing the protocol. Everything else remains the same and so it is possible to define this modification as a restriction on the environments considered if one wishes. A protocol π that UC realizes a functionality $\mathcal{F}$ in this model is said to *securely realize* $\mathcal{F}$ *in the UC model with fixed inputs*. We first prove the following lemma:

Lemma 4.2. *Let π_f be a polynomial-time protocol and $\mathcal{F}_f$ a functionality. If π_f securely realizes $\mathcal{F}_f$ under minimal concurrent general composition with fixed inputs, then π_f securely realizes $\mathcal{F}_f$ in the UC model with fixed inputs.*

Proof Sketch. This is proven in almost exactly the same way as the main theorem in [26]. Recall that π_f is the secure protocol and π' an arbitrary other protocol. Then, in [26] it is shown that the second protocol π' can essentially be used to emulate the behavior of the environment $\mathcal{Z}$ in the UC model. This is achieved by π' defining designated parties $P_{\mathcal{Z}}$ and $P_{\mathcal{A}}$, where $P_{\mathcal{A}}$ is corrupted and $P_{\mathcal{Z}}$ is not. The party $P_{\mathcal{Z}}$ runs the internal code of an environment $\mathcal{Z}$ from the UC setting and $P_{\mathcal{A}}$ does the same for a UC adversary $\mathcal{A}$. Party $P_{\mathcal{Z}}$ sends inputs to the parties (instead of writing them directly

[7] In [26], there is a distinction between arbitrary and fixed sets of parties. Here we focus on arbitrary sets of parties only, although the analogous results there can be carried through here.

on their input tapes as $\mathcal{Z}$ would), receives back their outputs (instead of reading them directly from their tapes like $\mathcal{Z}$ would), and interacts with $P_{\mathcal{A}}$ in the same way that $\mathcal{Z}$ interacts with $\mathcal{A}$. Party $P_{\mathcal{A}}$ behaves similarly for the UC-adversary $\mathcal{A}$. The other honest parties in π' are simply instructed to receive values from $P_{\mathcal{Z}}$ and use them as inputs for the ideal functionality $\mathcal{F}_f$ (in the ideal model) or in the protocol π_f (in the real model). They then send their outputs back to $P_{\mathcal{Z}}$ when they finish. Finally, $P_{\mathcal{Z}}$ outputs the same bit that $\mathcal{Z}$ would output given its view. The crucial point is that it is possible to set up the protocol π' so that when it is run together with $\mathcal{F}_f$ the output of $P_{\mathcal{Z}}$ is distributed identically to the output of $\mathcal{Z}$ in an ideal execution of $\mathcal{F}_f$ in the UC model. Furthermore, if π' is run together with π_f, then the output of $P_{\mathcal{Z}}$ is distributed identically to the output of $\mathcal{Z}$ in a real execution of π_f in the UC model. The conclusion is therefore that if π_f is not UC secure, then it is also not secure under concurrent general composition. (This is because when run with π' the outputs in the REAL and HYBRID distributions will be distinguishable. In particular, $P_{\mathcal{Z}}$ outputs 1 with probability that is non-negligibly different in both.)

The proof here is almost the same with the exception being how the parties' inputs to π_f are chosen. In [26], these are sent by the party $P_{\mathcal{Z}}$ playing $\mathcal{Z}$ in π' (and $P_{\mathcal{Z}}$ chooses them by internally running $\mathcal{Z}$ on its auxiliary input z). However, here the inputs must be a priori fixed. In order to do this, first observe that since we are in the UC model with fixed inputs, the environment writes the inputs to the honest parties before any interaction takes place. Thus, by using an averaging argument it is easy to see that if the environment distinguishes the REAL and IDEAL executions with non-negligible probability in the UC model with fixed inputs, then there exists a vector of inputs $\overline{x}_{\pi_f}$ for which the environment distinguishes the REAL and IDEAL with non-negligible probability. (Note that the environment may choose the inputs as a function of its auxiliary input z and its random coins. Thus, the averaging argument is needed to fix a specific input vector. Note also that this works only because $\mathcal{Z}$ determines the parties' inputs based on its auxiliary input and own random tape only; in particular, the inputs are determined independently of the honest parties' random tapes.) Next, party $P_{\mathcal{Z}}$'s input in π' is set to the value z and random-tape for $\mathcal{Z}$ that results in the environment writing $\overline{x}_{\pi_f}$. Furthermore, all the parties' inputs in π_f are set to $\overline{x}_{\pi_f}$. When the inputs are set in this way, the execution in the setting of concurrent general composition perfectly emulates the setting of the UC model with fixed inputs (when $\mathcal{Z}$ has the appropriate z and random tape). More specifically, the output of $P_{\mathcal{Z}}$ in a REAL execution of π_f with π' will be distributed exactly like the output of the environment $\mathcal{Z}$ in a real execution in the UC model with fixed inputs (when $\mathcal{Z}$ has the appropriate z and random tape). Likewise, the output in a HYBRID execution of π' with $\mathcal{F}_f$ will be exactly like an ideal execution in the UC model with fixed inputs. Thus, $P_{\mathcal{Z}}$ will distinguish the REAL and HYBRID models with the same probability that $\mathcal{Z}$ distinguishes the REAL and IDEAL executions in the UC model with fixed inputs. We conclude that if π_f is not secure in the UC model with fixed inputs then it is not secure under minimal concurrent general composition with fixed inputs. This implies the lemma. □

We note that the above lemma holds as long as the order of quantifiers between $\mathcal{S}$ and $\mathcal{Z}$ can be switched (as is the case with the definition of universal composability in the latest version of [8]). Otherwise, minimal concurrent general composition only implies

a weaker form of universal composability. In any case, the known impossibility results all hold for this weaker form and so for our purposes here there is no difference; see [26] for more discussion on this (note that the issues that arise here and in [26] in this respect are exactly the same).

We are now ready to prove our impossibility results, and we begin once again with an analogue to Lemma 3.1.

Lemma 4.3. *Let f be a polynomial-time two-party function, and let $\mathcal{F}_f$ be the two-party ideal functionality that receives x_1 from P_1 and x_2 from P_2, and hands them back their respective outputs $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$. If $\mathcal{F}_f$ can be securely realized in the UC model with fixed inputs by a non-trivial protocol π_f (in the plain model or in the $\mathcal{F}_{\text{bbca}}$, $\mathcal{F}_{\text{ukca}}$ or $\mathcal{F}_{\text{bpk}}$-hybrid models), then there exists a machine P_2^a such that for every machine P_2^b of the form described in Definition 2.2, the split adversarial strategy for $P_2 = (P_2^a, P_2^b)$ is* successful, *except with negligible probability.*

Proof. Note that in the proof of Lemma 3.1, the environment $\mathcal{Z}$ chooses the inputs for P_1 and P_2 at the onset. Thus, in the UC model with fixed inputs the entire proof goes through without change. (One small technicality is that the actual P_1 should not receive x_1 since this is the input used by $\mathcal{Z}$ who plays P_1. Thus, x_2 can be written to P_2's input tape at the onset, and $\mathcal{Z}$ can write the all-zero string to P_1's input tape. In this way, x_1 is kept secret by $\mathcal{Z}$ who uses it in its computation.) □

Next, we use the following observation:

> *All of the impossibility results of* [14] *that use successful split adversarial strategies hold even for the UC model with fixed inputs.*

Note that unlike the impossibility results from Sect. 3, we need the above observation because the impossibility results of [14] were proven for the standard UC model, and not for the variant with fixed inputs. Combining Lemmas 4.2 and 4.3 with this observation, we conclude that there exist large classes of functionalities for which it is impossible to achieve minimal concurrent general composition with fixed inputs.

We stress that this impossibility result is extremely strong. It is not possible to achieve concurrent general composition even when only two protocol executions take place and even when the inputs to these executions are a priori fixed.

Acknowledgements

We would like to thank the anonymous referees for their helpful comments.

References

[1] B. Barak, A. Sahai, How to play almost any mental game over the net—concurrent composition via super-polynomial simulation, in *46th FOCS* (2005), pp. 543–552

[2] B. Barak, R. Canetti, J. Nielsen, R. Pass, Universally composable protocols with relaxed set-up assumptions, in *45th FOCS* (2004), pp. 186–195

[3] B. Barak, R. Canetti, Y. Lindell, R. Pass, T. Rabin, Secure computation without authentication, in *CRYPTO 2005*. LNCS, vol. 3621 (Springer, Berlin, 2005), pp. 361–377
[4] B. Barak, M. Prabhakaran, A. Sahai, Concurrent non-malleable zero-knowledge, in *47th FOCS* (2006), pp. 345–354
[5] D. Beaver, Foundations of secure interactive computing, in *CRYPTO'91*. LNCS, vol. 576 (Springer, Berlin, 1991), pp. 377–391
[6] M. Ben-Or, S. Goldwasser, A. Wigderson, Completeness theorems for non-cryptographic fault-tolerant distributed computation, in *20th STOC* (1988), pp. 1–10
[7] R. Canetti, Security and composition of multiparty cryptographic protocols. *J. Cryptol.* **13**(1), 143–202 (2000)
[8] R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in *42nd FOCS* (2001), pp. 136–145
[9] R. Canetti, Universally composable signature, certification, and authentication, in *17th Computer Security Foundations Workshop* (2004), pp. 219–235
[10] R. Canetti, M. Fischlin, Universally composable commitments, in *CRYPTO 2001*. LNCS, vol. 2139 (Springer, Berlin, 2001), pp. 19–40
[11] R. Canetti, R. Ostrovsky, Secure computation with honest-looking parties: What if nobody is truly honest? in *31st STOC* (1999), pp. 255–264
[12] R. Canetti, O. Goldreich, S. Goldwasser, S. Micali, Resettable zero-knowledge, in *32nd STOC* (2000), pp. 235–244
[13] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai, Universally composable two-party and multi-party computation, in *34th STOC* (2002), pp. 494–503
[14] R. Canetti, E. Kushilevitz, Y. Lindell, On the limitations of universal composable two-party computation without set-up assumptions. *J. Cryptol.* **19**(2), 135–167 (2006)
[15] D. Chaum, C. Crépeau, I. Damgård, Multi-party unconditionally secure protocols, in *20th STOC* (1988), pp. 11–19
[16] I. Damgård, J.B. Nielsen, C. Orlandi, On the necessary and sufficient assumptions for UC computation, in *7th TCC*. LNCS, vol. 5978 (Springer, Berlin, 2010), pp. 109–127
[17] A. Datta, A. Derek, J.C. Mitchell, A. Ramanathan, A. Scedrov, Games and the impossibility of realizable ideal functionality, in *3rd TCC*. LNCS, vol. 3876 (Springer, Berlin, 2006), pp. 360–379
[18] Y. Deng, G.D. Crescenzo, D. Lin, Concurrently non-malleable zero knowledge in the authenticated public-key model. *Cryptology ePrint Archive*, Report #2006/314, 2006
[19] O. Goldreich, *Foundations of Cryptography: Volume 2—Basic Applications* (Cambridge University Press, Cambridge, 2004)
[20] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game—A completeness theorem for protocols with honest majority, in *19th STOC* (1987), pp. 218–229
[21] S. Goldwasser, L. Levin, Fair computation of general functions in presence of immoral majority, in *CRYPTO'90*. LNCS, vol. 537 (Springer, Berlin, 1990), pp. 77–93
[22] Y. Kalai, Y. Lindell, M. Prabhakaran, Concurrent general composition of secure protocols in the timing model, in *37th STOC* (2005), pp. 644–653
[23] E. Kushilevitz, Y. Lindell, T. Rabin, Information-theoretically secure protocols and security under composition, in *38th STOC* (2006), pp. 109–118
[24] H. Lin, R. Pass, M. Venkitasubramaniam, A unified framework for concurrent security: Universal composability from stand-alone non-malleability, in *41st STOC* (2009), pp. 179–188
[25] Y. Lindell, *Composition of Secure Multi-Party Protocols—A Comprehensive Study*, LNCS, vol. 2815 (Springer, Berlin, 2003)
[26] Y. Lindell, General composition and universal composability in secure multi-party computation, in *44th FOCS* (2003), pp. 394–403
[27] Y. Lindell, Lower bounds for concurrent self composition, in *1st Theory of Cryptography Conference (TCC)*. LNCS, vol. 2951 (Springer, Berlin, 2004), pp. 203–222
[28] S. Micali, P. Rogaway, Secure computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, LNCS, vol. 576 (Springer, Berlin, 1991), pp. 392–404
[29] R. Ostrovsky, G. Persiano, I. Visconti, Concurrent non-malleable witness indistinguishability and its applications. *Cryptology ePrint Archive*, Report #2006/256, 2006

[30] R. Pass, Simulation in quasi-polynomial time, and its application to protocol composition, in *Eurocrypt 2003*. LNCS, vol. 2656 (Springer, Berlin, 2003), pp. 160–176
[31] M. Prabhakaran, A. Sahai, New notions of security: Universal composability without trusted setup, in *36th STOC* (2004), pp. 242–251
[32] A. Yao, How to generate and exchange secrets, in *27th FOCS* (1986), pp. 162–167