Universally Composable Multiparty Computation with Partially Isolated Parties

Ivan Damgård[1], Jesper Buus Nielsen[1] and Daniel Wichs[2]

[1] University of Aarhus, Denmark
[2] New York University, USA

Abstract. It is well known that universally composable multiparty computation cannot, in general, be achieved in the standard model without setup assumptions when the adversary can corrupt an arbitrary number of players. One way to get around this problem is by having a *trusted third party* generate some global setup such as a *common reference string (CRS)* or a *public key infrastructure (PKI)*. The recent work of Katz shows that we may instead rely on physical assumptions, and in particular *tamper-proof hardware tokens*. In this paper, we consider a similar but *strictly weaker* physical assumption. We assume that a player (Alice) can *partially isolate* another player (Bob) for a brief portion of the computation and prevent Bob from communicating more than some limited number of bits with the environment. For example, isolation might be achieved by asking Bob to put his functionality on a tamper-proof hardware token and assuming that Alice can prevent this token from communicating to the outside world. Alternatively, Alice may interact with Bob directly but in a special office which she administers and where there are no high-bandwidth communication channels to the outside world. We show that, under *standard* cryptographic assumptions, such physical setup can be used to UC-realize any two party and multiparty computation in the presence of an active and *adaptive* adversary corrupting any number of players. We also consider an alternative scenario, in which there are some trusted third parties but no single such party is trusted by all of the players. This compromise allows us to significantly limit the use of the physical set-up and hence might be preferred in practice.
Key words: universally composable security, multiparty computation, public-key infrastructure

1 Introduction

Traditionally, the security of cryptographic protocols was considered in the stand-alone setting where a single run of the protocol executes in isolation. In the real world, when many copies of a single protocol and related protocols may be executing concurrently, security in the stand-alone setting becomes insufficient.

The universal composability (UC) framework was introduced by Canetti in [Can01] to fix this problem and allow us to prove the security of protocols in the real-world setting without resorting to intractably complicated proofs. The initial work of Canetti gave hope that UC security is achievable by showing that any multiparty computation (MPC) can be realized in the UC framework, assuming a strict majority of the players are honest. Unfortunately, this work was followed by results showing that many natural functionalities cannot be UC realized without an honest majority, including essentially all non-trivial two party computations such as commitments and zero knowledge proofs [CKL03].

To get around these negative results, one can require the existence of additional setup infrastructure available to the parties. For example, such setup can consist of a common reference string (CRS) which is honestly sampled from some pre-defined distribution and given to all the players [CLOS02] or a public key infrastructure (PKI) where a trusted certificate authority (CA) verifies that each player knows the secret key corresponding to his registered public key [BCNP04]. Both of the above setup assumptions require a trusted party to initialize the infrastructure and the protocols become completely insecure if this party is corrupted.

In this paper we rely on a physical assumption instead of a trusted third party. Namely, we assume that a player (Alice) can ensure that another player (Bob) is *partially isolated* for a short portion of the computation. During this time, Bob can only exchange a limited number of bits with the environment but Alice's communication is unrestricted. We show that, under *standard cryptographic assumptions*, the above physical setup allows us to UC realize any two-party and multiparty computation in the presence of an *active and adaptive* adversary corrupting any number of parties. We do *not* assume erasures.

1.1 Related Work

The idea of relying on physical assumptions to achieve universal composability was first proposed by Katz in [K07]. In particular, the work of Katz assumes the existence of *tamper-proof hardware tokens*. A player, Bob, puts some arbitrary functionality inside such a token and sends it to another player, Alice. Alice can then only interact with the token through the intended interface. In addition, it is assumed that Alice can *isolate* the token during this interaction, ensuring that it has no way of communicating with the outside world. In general, there seem to be two ways to take advantage of the fact that Bob's functionality is placed on tamper-proof hardware (rather than having Bob run it remotely):

(1) The tamper-proof hardware token is isolated and cannot communicate with the environment.
(2) The tamper-proof hardware token is a new and *separate* entity from Bob. Bob never sees the content of Alice's interaction with the token.

In [K07], Katz shows how to use tamper-proof tokens to UC realize any multiparty computation in the presence of an active but *static* adversary, under the Diffie Hellman (DH) assumption. We note that this solution only makes use of advantage (1), though this distinction is not explicit and the formal model allows for both advantages.

The work of Chandran et al. [CGS08] extends the result of Katz by considering an adversary who might not necessarily *know* the code of the token he creates. In addition, the adversary may perform *reset attacks* on received tokens, effectively getting the power to rewind tokens at will. The work of Moran and Segev [MS08], on the other hand, presents a protocol for two asymmetrically powerful parties: a powerful Goliath and a limited David. Only Goliath has the ability to create tamper-proof hardware tokens. Moreover, Goliath is not assumed to be computationally bounded (but David is). Both of these works crucially rely on advantage (2) above.

The work of Damgård et al. [DNW08] introduces a new and slightly different physical assumption – namely, that parties can be *partially* isolated so that their communication with the environment is limited. This setting was studied only with regard to *zero knowledge proofs of knowledge* (ZK PoK). Damgård et al. present a witness indistinguishable (WI) PoK protocol for the case where only the prover is partially isolated (while the verifier's communication is unrestricted) and a ZK PoK protocol for the case where both parties are partially isolated.

1.2 Our Contribution

In this paper we consider the partial isolation physical assumption for multiparty computation in general. First, we notice that there is a relationship between the partial isolation model of [DNW08] and the tamper-proof hardware model of [K07]. Namely a party can be (fully) isolated by placing its functionality on a tamper-proof hardware token. However, isolation can also be implemented in many other ways. For example, we may imagine a setting where Bob simply brings his laptop into an office administered by Alice who ensures that there is no wireless or wired internet access available to Bob. Bob then connects his laptop to Alice's machine and they run an interactive protocol between them. During this time, Alice can communicate with the environment as much as she wants, but Bob cannot. In the above example, we see a crucial difference between the isolated parties model and the tamper-proof hardware model: we cannot (in general) assume that some isolated entity is *separate* from Bob – it might be Bob himself who is isolated! Of course, since Bob sees Alice's interaction with himself while he is isolated we do not get advantage (2). Therefore, even the full isolation model is strictly weaker than the tamper proof hardware model and the protocols of [CGS08,MS08] cannot be used in the isolation setting.

Moreover, as a further weakening of our physical assumptions, we only assume that parties can be *partially* isolated from the environment. Specifically, as in [DNW08], we assume the existence of some threshold ℓ, such that Alice can prevent Bob from exchanging more than ℓ bits with the environment. In practice, this might significantly easier to achieve than full isolation. In our example, where Bob meets Alice in her office, it might be significantly easier for Alice to only block *high-bandwidth* communication channels to the outside world than to block *all* such channels. Even if the parties do choose to use tamper-proof hardware tokens, it might not be trivial to fully isolate a token as is required in Katz's model. Partial isolation might be much simpler to achieve. For example, if the tamper-proof token is a smart-card that is too small to have its own power supply, Alice can then observe (and limit) the card's power consumption to limit communication. A study by [BA03] shows that one bit of wireless communication by a smart-card has the same power consumption as 1000 32-bit elementary operations and hence this could be a practical solution in limiting the amount of communication that is possible.

The work of [DNW08] defines partial isolation for the case of zero knowledge proofs of knowledge directly. In this work, we define partial isolation in general as an ideal functionality (similarly to Katz's functionality for tamper-proof hardware tokens). We then construct a protocol for arbitrary two-party and multiparty computation using this functionality. In our protocol, the use of physical assumptions is limited to a short setup phase during which parties register keys with one another while the registrant is partially isolated. In practice, the use of physical setup might be expensive and difficult for individuals. We also propose a hybrid model in which there are some trusted Certificate Authorities (CAs) but no such authority needs to be trusted by all the players. Each player either trusts an external CA (and many players may trust the same one) or can act as his own CA and trust nobody else. This model might be natural in many scenarios where large organizations (countries, companies...) do not trust each other but individuals trust the organization they belong to.

1.3 Overview of Construction

Our basic approach is to set up a public key infrastructure (PKI) between the parties so that each player must know the secret key corresponding to his registered public key. The result of [BCNP04] shows that such a PKI, when created by a trusted third party, can be used to UC-realize the ideal commitment functionality, which in turn allow one to UC-realize arbitrary multiparty computation.

Consider the following naive approach of setting up such a PKI. Each player chooses a public key and registers it with every other player. When a player, Bob, wants to register his key with another player, Alice, he simply sends her his public key and runs a zero knowledge (ZK) proof of knowledge (PoK) to convince her that he knows the corresponding secret key.

Unfortunately, using a standard ZK PoK (secure in the stand-alone setting) does not give us security in the UC-framework. However, if Alice can ensure that Bob is isolated for the duration of the proof, a standard PoK protocol does guarantee that Bob knows his secret key. In fact, the result of [DNW08] shows that for any threshold ℓ there is an ℓ-Isolated Proof of Knowledge (ℓ-IPoK) protocol, which ensures that the prover knows a witness *even* if he can exchange up to ℓ bits of information with the environment during the proof. By using an ℓ-IPoK protocol, Alice will be assured that Bob knows his secret key even if she can only partially isolate Bob and keep him from communicating more than ℓ bits.

We *cannot*, however, guarantee that Alice (who is not isolated during the proof) does not learn anything from such a proof. As is shown in [DNW08], no witness hiding protocol can be zero knowledge simulatable with respect to a verifier that communicates arbitrarily with the environment. Instead, we will only rely on the witness indistinguishability (WI) property of an ℓ-IPoK protocol. This means that our PKI is not perfect and verifying parties might get some limited information about the registered private keys. Nevertheless, we show that an imperfect PKI of this type can be used to implement the ideal commitment functionality. To do so, we modify the commitment scheme of [BCNP04] (which is based on the prior scheme of [CLOS02]) so that it remains secure even if the adversary sees a witness indistinguishable proof of knowledge of the commitment private key. As is shown in [CLOS02], the commitment functionality allows us to implement all other two-party and multiparty computation.

2 The Formal Model of Our Setting

2.1 The $\mathcal{F}_{\text{isolate}}$ ideal functionality.

We model partial isolation using an ideal functionality $\mathcal{F}_{\text{isolate}}$ described in Fig. 1. It describes a situation where P is partially isolated from the environment during an interaction with P'. This is similar to the ideal functionality $\mathcal{F}_{wrap}$ defined in [K07] to model tamper-proof hardware, but there are several important differences.

The $\mathcal{F}_{\text{isolate}}$ ideal functionality is parametrized by an isolation parameter ℓ, a security parameter κ and a polynomial p.

Isolation of P: Wait until receiving messages $(\texttt{isolate}, sid, P, P')$ from P' and $(\texttt{isolate}, sid, P, P', M)$ from P. If there is already a stored tuple of the form $(P, P', \cdot, \cdot, \cdot, \cdot)$ then ignore the command. Otherwise:

1. Parse the string M as the description of an ITM with four communication tapes; two tapes ("in" and "out") for regular protocol communication with P' and two tapes for secret communication with P. Let the value `state` encode the initial state of M (including the value of a work tape and an initialized random tape). Define new values $\texttt{inCom} := 0$, $\texttt{outCom} := 0$ and store the tuple $(P, P', M, \texttt{state}, \texttt{inCom}, \texttt{outCom})$.
2. Send $(\texttt{isolate}, sid, P)$ to P'.

Interaction with P': On input $(\texttt{run}, sid, P, P', \texttt{msg})$ from P', retrieve the tuple $(P, P', M, \texttt{state}, \texttt{inCom}, \texttt{outCom})$. If there is no such tuple then ignore the command.

1. Place the string `msg` on the "in" tape designated for P and run M for $p(\kappa)$ steps.
2. If there is any value $\texttt{msg}'$ on the output tape for P' then send the message $(\texttt{reply}, sid, P, \texttt{msg}')$ to P'.
3. If there is any value $\texttt{msg}'$ on the output tape for P and $\texttt{outCom} + |\texttt{msg}'| < \ell$ then send the message $(\texttt{secretCom}, sid, P', P, \texttt{msg}')$ to P and update $\texttt{outCom} := \texttt{outCom} + |\texttt{msg}'|$.
4. Update the value of `state` in the stored tuple to encode the updated state of M and the values of its tapes.

Communication: On input $(\texttt{secretCom}, sid, P, P', \texttt{msg})$ from P, if there is no tuple of the form $(P, P', M, \texttt{state}, \texttt{inCom}, \texttt{outCom})$ then ignore. Also if the tuple has $\texttt{inCom}+|\texttt{msg}| > \ell$ then ignore the command. Otherwise

1. Update $\texttt{inCom} := \texttt{inCom} + |\texttt{msg}|$, place `msg` on the "in" tape for P and run M for $p(\kappa)$ steps.
2. Proceed with steps 2,3,4 of the above command.

Release of P: On input $(\texttt{release}, sid, P, P')$ from P', retrieve the tuple $(P, P', M, \texttt{state}, \texttt{inCom}, \texttt{outCom})$ and send $(\texttt{release}, sid, P, P', \texttt{state})$ to P.

Fig. 1. The $\mathcal{F}_{\text{isolate}}$ Ideal Functionality

When Alice wants to isolate Bob, both of them call the `isolate` command and Bob sends a description of his functionality (modeled as an ITM M) and current state to $\mathcal{F}_{\text{isolate}}$. Alice can then interact with Bob's functionality by issuing `run` commands to $\mathcal{F}_{\text{isolate}}$ which internally runs Bob's code to produce replies for Alice. At the conclusion of the interaction, Alice sends a `release` command.

The main differences between Katz's $\mathcal{F}_{wrap}$ functionality and our $\mathcal{F}_{\text{isolate}}$ functionality are as follows. Firstly, we want to capture the fact that it might be Bob himself who is isolated and not some separate token. Therefore, we make a restriction on how honest parties can use this functionality in legitimate protocols. We require that, if Bob is honest, he will be inactive between the time that he issues the `isolate` command and the time that the `release` command is issued by Alice. In addition, when the `release` command is issued, $\mathcal{F}_{\text{isolate}}$ sends Bob the current updated state of his functionality M, which might contain information about the interaction that took place with Alice. Secondly, we want to capture the fact that our isolation is only partial and that there might be some limited secret communication between a partially isolated party and the environment. We parameterize $\mathcal{F}_{\text{isolate}}$ with a communication threshold ℓ. Bob's functionality M can send up to ℓ bits of communication to its creator (and hence the environment) and can receive up to ℓ bits of communication from its creator using `secretCom` commands. We require that only corrupted parties takes advantage of this secret communication — i.e., it describes an allowed flaw of the isolation rather than a useful feature.

2.2 PKI and Certificate Authorities

We use the ideal functionality $\mathcal{F}_{\text{isolate}}$ to setup a public key infrastructure. In the general multiparty computation setting, there are many parties which will register keys and try to implement ideal functionalities among them. We denote these parties by $P_1, \ldots, P_n$. In addition we have parties $CA_1, \ldots, CA_m$ acting as certificate authorities. We allow the case where a player P_i acts as his own certificate authority ($P_i = CA_k$). In general, however, we only require that each party P_i trusts some certificate authority CA_k and many parties may trust the same certificate authority. Any player P_j who wishes to interact with P_i needs to register a key with an authority CA_k trusted by P_i.

We model the certificate authorities as additional players in the game. In the ideal world, the certificate authorities have no inputs and receive no outputs. We define a *certificate authority trust structure* as the mapping of players to the certificate authority they trust, and we assume that each player trusts at least one CA. The group of players who trust a single CA is called the certificate authority's *trust group*. To model the notion of trust, we assume that when an adversary actively corrupts a certificate authority he also actively corrupts all of the players in the authority's trust group. The adversary may actively corrupt an arbitrary number of real players P_i and an arbitrary number of certificate authorities subject to the above restriction. We call any such adversarial corruption strategy a *legal corruption strategy*. An adversary can also passively corrupt any CAs at will. For simplicity, we will just require that an honest CA makes all of its interactions public so such corruptions are unnecessary.

2.3 Statement of Result

We are now ready to state the main theorem of our paper.

Theorem 1. *Assume the existence of one-way permutations and dense public key, IND-CPA secure encryption schemes with pseudorandom ciphertexts. Then any polynomial time ideal functionality can be UC realized in the $\mathcal{F}_{\text{isolate}}$-hybrid model under any certificate authority trust structure. We assume that an active and adaptive adversary can corrupt any number of players and certificate authorities using a legal corruption strategy. We do not assume erasures.*

We can instantiate the theorem with the trust structure in which each player acts as his own certificate authority and trusts nobody else. This shows that, as a special case of Theorem 1, any polynomial time ideal functionality can be UC realized in the $\mathcal{F}_{\text{isolate}}$-hybrid model without any additional certificate authority parties and with an adaptive and active adversary corrupting any number of players. The proof of the above theorem spans the remainder of the paper. As in [K07], we will only show how to UC-realize the ideal functionality for multiple commitments and the rest follows from the work of [CLOS02].

3 Proofs of Knowledge and Isolated Proofs of Knowledge

Our construction relies heavily on proofs of knowledge (PoK). Here we review some terminology and results. Recall that an *NP relation* $\mathcal{R}$ is a set of pairs (x, w) where $(x, w) \in^? \mathcal{R}$ can be checked in poly-time in the length of x. For such a relation we define the *witnesses for an instance* $W_{\mathcal{R}}(x) = \{w | (x, w) \in \mathcal{R}\}$ and the *language* $L(\mathcal{R}) = \{x | W_R(x) \neq \emptyset\}$. Given an NP relation $\mathcal{R}$, a PoK is an interactive protocol between two parties called a *Prover* P and a *Verifier* V. The protocol is specified by the PPT ITMs (P, V) where P is given an input $(x, w) \in \mathcal{R}$ and V is given the instance x. The parties run the protocol and, at the end, the verifier outputs a *judgment* $J = \texttt{accept}$ or $J = \texttt{reject}$. We require *completeness*: when P and V are honest then V outputs the judgment $J = \texttt{accept}$ with all but negligible probability.

In our setting, the prover may communicate with an external adversarial environment during the proof, but this communication is limited to some pre-defined bound of ℓ bits. The verifier, on the other hand, has unbounded communication with the environment. This setting is considered in [DNW08], which defines the notion of an *ℓ-Isolated Proof of Knowledge* (ℓ-IPoK) protocol. Such a protocol ensures that a successful prover knows a witness, even in the above environment.

Setup: First the environment $\mathcal{E}$ is run to produce x which it sends to P^* and V. At this stage P^* and $\mathcal{E}$ can communicate arbitrarily.

Execution: Then for $r = 1, \ldots, \rho$ the verifier V is activated to produce a message $v^{(r)}$ that is input to P^* which is activated to produce a message $p^{(r)}$ that is input to V. In addition, P^* can at any point send a message y to $\mathcal{E}$ and receive a response z from $\mathcal{E}$. However, the total number of bits sent and the total number of bits received during the execution stage are both bounded by ℓ. At the conclusion of the ρ rounds, the verifier V produces a judgment $J \in \{\texttt{accept}, \texttt{reject}\}$.

Extraction: If $J = \texttt{reject}$ then the extractor $\mathcal{X}$ wins the extraction game. Otherwise, we construct the view σ to be the description of P^*, its initial random tape, the messages $v^{(r)}, p^{(r)}$ exchanged between P^* and V, and the transcript of the communication between P^* and $\mathcal{E}$. We let $w = \mathcal{X}(\kappa, \sigma)$. If $w \in W_R(x)$, then $\mathcal{X}$ wins the game; otherwise it looses.

Fig. 2. Knowledge soundness extraction game.

Formally, knowledge soundness of an ℓ-IPoK protocol is defined by requiring that for any adversarial prover given by a PPT ITM P^*, there exists a *strict* PPT extractor $\mathcal{X}$ which wins the knowledge soundness extraction game outlined in Fig. 2 with all but negligible probability. This should hold for any environment given by a PPT ITM $\mathcal{E}$.

It was shown in [DNW08] that there exists an Isolated Proof of Knowledge compiler (called an IPoK) which, for any NP relation $\mathcal{R}$ and any ℓ polynomial in the security parameter, produces a protocol that is an ℓ-IPoK for R. In addition, the protocol is *witness indistinguishable* (*WI*). This means that for any malicious verifier V^*, and any two pairs $(x, w_1) \in \mathcal{R}$, $(x, w_2) \in \mathcal{R}$ the verifier cannot distinguish between a prover that uses the witness w_1 and a prover that uses the witness w_2, even when given w_1 and w_2. Formally, letting $\textsc{Exec}(P(x, w), V(x))$ denote the transcript of the execution between P and V where P uses the witness w for the instance x, we require that for any PPT cheating verifier V^*

$$(\textsc{Exec}(P(x, w_1), V^*(x)), w_1, w_2) \approx (\textsc{Exec}(P(x, w_2), V^*(x)), w_1, w_2)$$

This notion is significantly weaker than zero knowledge (ZK) but [DNW08] shows that one cannot achieve ZK without isolating the verifier as well and hence we will have to rely on witness indistinguishability only.

4 Construction

We use the results of [CLOS02] which show that one can UC-realize arbitrary MPC given the ideal functionality for multiple commitments $\mathcal{F}_{\mathrm{MCOM}}$ which we review in Fig. 3.

Commit Phase: On input $(\texttt{commit}, sid, ssid, P_j, m)$ from P_i, if there is already a stored tuple of the form $(sid, ssid, P_i, P_j, \cdot)$ then ignore the command. Otherwise store the tuple $(sid, ssid, P_i, P_j, m)$ and send a receipt $(\texttt{receipt}, sid, ssid, P_i)$ to P_j.
Reveal Phase: On input $(\texttt{reveal}, sid, ssid, P_j)$ from P_i, if a tuple $(sid, ssidP_i, P_j, m)$ is stored then send a message $(\texttt{reveal}, sid, ssid, P_i, m)$ to P_j. Otherwise, ignore the command.

Fig. 3. The $\mathcal{F}_{\mathrm{MCOM}}$ Ideal Functionality

There are several challenges in UC realizing the $\mathcal{F}_{\mathrm{MCOM}}$ functionality. Obviously, we need a commitment scheme which is hiding and binding. In addition, the simulator needs to be able to generate commitments for honest parties before knowing the message being committed to and later be able to decommit to any specified message. A scheme with this property is called *equivocal*. For adaptive security, the simulator needs to be able to simulate the corruption of an honest party and thus reveal all of the randomness used to generate such simulated commitments as though they were generated honestly. We call this *strong equivocality*. The simulator also needs to extract the message contained in any valid commitment even if it was adversarially generated. This is called *extractability*.

Luckily, the result of [CLOS02] contains just such a scheme. It relies on two public keys, an extraction key pk_X and an equivocation key pk_E, that are generated randomly and placed in a CRS. The corresponding secret keys, which are known by the simulator but not the players in the real world, give it the power of strong equivocality and extractability. It was already noticed in [BCNP04] that the players can choose these keys themselves. A sender uses his extraction key and the receiver's equivocation key to generate commitment.

We use the basic idea of [BCNP04] but modify it to fit our setting. Firstly, if the honest sender knows his own extraction secret key (and cannot erase it) then the adversary learns this key when the sender is corrupted. This allows the adversary to distinguish if previous commitments sent by the sender were generated honestly (as is done in the real world) or if they were equivocated (as is done by the simulator in the ideal world). To get around this issue, we have the sender and receiver do a coin-flip to generate the extraction public key so that neither party knows the corresponding secret key. To simulate the coin-flip, it is enough to have a strongly equivocal commitment scheme (i.e., no extraction is needed) and so players only register their equivocation public keys. The second problem arises from the fact that the sender's commitments can only be extracted (and in general are only binding) when the sender has no information about the receiver's equivocation key. However, in our setting the adversary gets to run as a verifier in a WI ℓ-IPoK of the equivocation secret key, which might potentially leak useful information. We show how to modify the original scheme so that it remains extractable even with respect to an adversary that sees such proofs.

We begin by formalizing an abstraction which captures the properties achieved by the scheme of [CLOS02]. Then we show how to turn any scheme which has those properties into one that is secure even if the adversary has access to a prover running a WI PoK protocol and using the equivocation secret key as a witness.

4.1 The Commitment Scheme

Extractability. We define an *extraction game* between a challenger and an adversary as follows:

1. The challenger generates random $(pk_E, sk_E) \leftarrow \mathsf{gen}_E(1^\kappa)$ and the adversary is given pk_E.
2. The adversary chooses a pair $(pk_X, sk_X) \in \mathcal{R}_X$, a commitment C and a pair (m', r') and sends these to the challenger.
3. Let $m = \mathsf{extract}_{pk_E}^{(pk_X, sk_X)}(C)$. If $m' \neq m$ and $C = \mathsf{commit}_{pk_E}^{pk_X}(m'; r')$ then the adversary wins the extraction game.

We say that a commitment scheme is extractable if there is a PPT algorithm $\mathsf{extract}$ such that, for any PPT adversary $\mathcal{A}$, the success probability of $\mathcal{A}$ winning the extraction game is negligible in κ.

Binding. We define a *binding game* between a challenger and an adversary:

1. The challenger generates a random $(pk_E, sk_E) \leftarrow \mathsf{gen}_E(1^\kappa)$ and the adversary is given pk_E.
2. The adversary generates some public key $pk_X \in \{0,1\}^t$. In addition, the adversary specifies a commitment C and two pairs (m, r), (m', r) and sends these to the challenger.
3. The adversary wins the binding game if $m \neq m'$, $C = \mathsf{commit}_{pk_E}^{pk_X}(m; r)$ and $C = \mathsf{commit}_{pk_E}^{pk_X}(m'; r')$.

We say that a commitment scheme is binding if, for any PPT adversary $\mathcal{A}$, the success probability of $\mathcal{A}$ winning the binding game is negligible in κ.

Strong Equivocality/Hiding. We define equivocality by insisting that there is no adversary that can distinguish between the *commitment game* and the *equivocation game* defined below:

The *commitment game* between a challenger and adversary proceeds as follows:

1. The challenger generates a random $(pk_X, sk_X) \leftarrow \mathsf{gen}_X(1^\kappa)$ and gives pk_X to the adversary.
2. The adversary specifies $(pk_E, w_E) \in \mathcal{R}_E$, and a message m.
3. The challenger computes $C = \mathsf{commit}_{pk_E}^{pk_X}(m; r)$ where r is chosen randomly and gives (C, r) to the adversary.

The *equivocation game* between a challenger and adversary as follows:

1. The challenger generates a random $(pk_X, sk_X) \leftarrow \mathsf{gen}_X(1^\kappa)$ and gives pk_X to the adversary.
2. The adversary specifies $(pk_E, w_E) \in \mathcal{R}_E$, and a message m.
3. The challenger computes $(C, \mathsf{aux}) = \mathsf{ecommit}_{pk_E, w_E}^{pk_X}()$, $r \leftarrow \mathsf{equivocate}_{pk_E, w_E}^{pk_X}(C, \mathsf{aux}, m)$ and gives (C, r) to the adversary.

We say that a commitment scheme is *strongly equivocal* if there exists PPT algorithm $\mathsf{ecommit}$ and PPT algorithm $\mathsf{equivocate}$ such that no PPT adversary can distinguish between the commitment game and the equivocation game with more than negligible probability.

Fig. 4. Security of a Two Key Extractable and Equivocal Commitment Scheme

A *Two-Key Extractable and Strongly Equivocal Commitment Scheme* has two key generation algorithms $(pk_E, sk_E) \leftarrow \mathsf{gen}_E(1^k)$ and $(pk_X, sk_X) \leftarrow \mathsf{gen}_X(1^\kappa)$ for the equivocation and extraction keys respectively. The commitment algorithm takes as input the two public keys and a message m. It produces a commitment $C = \mathsf{commit}_{pk_E}^{pk_X}(m; r)$ using the randomness r. To decommit, the sender simply sends (m, r) and the receiver verifies $C \stackrel{?}{=} \mathsf{commit}_{pk_E}^{pk_X}(m; r)$.[3] In addition, we need the ability

[3] Because we consider adaptive security where the environment can always corrupt the sender to learn all the randomness r used to commit, there is no reason to consider commitment schemes where the decommitment does not consist of sending all this randomness: If the simulator can produce it to simulate a corruption of the sender, it can also produce it to simulate a decommitment.

to easily recognize well-formed public key/secret key pairs. For that, we assume that there is an NP relation $\mathcal{R}_E$ which defines well formed equivocation key pairs (pk_E, sk_E), and a relation $\mathcal{R}_X$ that defines well formed extraction key pairs (pk_X, sk_X). We assume that every key pair generated by gen_E (resp. gen_X) is contained in $\mathcal{R}_E$ (resp. $\mathcal{R}_X$) but allow the set of well-formed key pairs to contain other elements.

Lastly, we require that the extraction keys are dense. More precisely, for $(pk_X, sk_X) \leftarrow \mathsf{gen}_X(1^\kappa)$, the element pk_X is statistically close to a uniformly random element from some $\mathcal{G} = \{0,1\}^t$. We use $\oplus$ to denote bit-wise xor of elements from $\mathcal{G}$. The security properties of the scheme are outlined in Fig. 4. The commitment scheme of [CLOS02] meets our definition.

The observation that the scheme meets the given security requirements was essentially already made in [BCNP04]. For completeness, we include a short description of the scheme in Appendix A.

4.2 Security After WI Proofs

In the security definitions for extractability and binding of the commitment scheme, it is crucial that the adversary has no information about the equivocation secret key sk_E. However, in our protocols the adversary will get to see a witness indistinguishable (WI) proof of knowledge of such a secret key. For this reason, we augment the security definitions for extractability and binding to give the adversary unlimited protocol access to a prover running a WI proof of knowledge for the relation $\mathcal{R}_E$ using the public key pk_E as the instance and the secret key sk_E as a witness. We show how to turn any two-key extractable and strongly equivocal commitment scheme into a scheme that has *security after WI proofs* - i.e. is secure in the above setting.

Assume we have a two-key extractable and strongly equivocal commitment scheme defined by $(\mathsf{gen}_E, \mathsf{gen}_X, \mathsf{commit}, \mathsf{extract}, \mathsf{ecommit}, \mathsf{equivocate})$ and the equivocation key relation $\mathcal{R}_E$. We define a new commitment scheme $(\mathsf{gen}'_E, \mathsf{gen}'_X, \mathsf{commit}', \mathsf{extract}', \mathsf{ecommit}', \mathsf{equivocate}')$ with equivocation relation $\mathcal{R}'_E$ as follows:

Let gen'_E generate two equivocation keys $(pk_E^{(0)}, sk_E^{(0)}) \leftarrow \mathsf{gen}_E(1^\kappa), (pk_E^{(1)}, sk_E^{(1)}) \leftarrow \mathsf{gen}_E(1^\kappa)$ and let $pk'_E = (pk_E^{(0)}, pk_E^{(1)})$, $sk'_E = sk_E^{(0)}$. We define the relation

$$\mathcal{R}'_E := \left\{ \left(pk_E^{(0)}, pk_E^{(1)}, w_E\right) \;\middle|\; \left(pk_E^{(0)}, w_E\right) \in \mathcal{R}_E \text{ or } \left(pk_E^{(1)}, w_E\right) \in \mathcal{R}_E \right\} .$$

It is clear that this is an NP relation and that $(pk'_E, sk'_E) = ((pk_E^{(0)}, pk_E^{(1)}), sk_E^{(0)}) \in \mathcal{R}'_E$. We let gen'_X be the same as gen_X so the extraction keys are generated as in the original scheme.

Now assume that the message space is some $\{0,1\}^s$. We use $\oplus$ to denote bitwise xor in $\{0,1\}^s$. To commit to m the sender chooses a uniformly random $m^{(1)}$ and computes $m^{(0)} = m \oplus m^{(1)}$. The sender then computes

$$C^{(0)} = \mathsf{commit}^{pk_X}_{pk_E^{(0)}}(m^{(0)}; r^{(0)}) \;, \quad C^{(1)} = \mathsf{commit}^{pk_X}_{pk_E^{(1)}}(m^{(1)}; r^{(1)}) \tag{0}$$

and sends the commitment $C = (C^{(0)}, C^{(1)})$.

To open the commitment, the sender sends $(m, r) = (m, (m^{(1)}, r^{(0)}, r^{(1)}))$. The receiver checks that $C^{(0)}, C^{(1)}$ were correctly computed using equation (4.2).

Equivocality/Hiding. We use a series-of-games argument to show that the above scheme is strongly equivocal. Let us define Game 1 to be the commitment game used in the definition of strong equivocality in Fig. 4.

In Step 2 of the game, the adversary specifies $(pk'_E, w_E) = ((pk_E^{(0)}, pk_E^{(1)}), w_E) \in \mathcal{R}'_E$ such that $(pk_E^{(b)}, w_E) \in \mathcal{R}_E$ for $b = 0$ or $b = 1$. In addition, since the relation $\mathcal{R}_E$ is in NP, it is easy to check which is the case (if both, we let $b = 0$). Let $\bar{b} = 1 - b$. We define Game 2 which proceeds as Game 1 except that the challenger computes the commitment by choosing $m^{(\bar{b})}$ randomly and setting $m^{(b)} = m - m^{(\bar{b})}$. Games 1 and 2 have identical distributions and so are indistinguishable.

We define Game 3 which proceeds as Game 2, but the challenger computes $C^{(b)}$ and $r^{(b)}$ using $(C^{(b)}, \mathsf{aux}) = \mathsf{ecommit}^{pk_X}_{pk_E^{(b)}, w_E}()$ and $r^{(b)} = \mathsf{equivocate}^{pk_X}_{pk_E^{(b)}, w_E}(C^{(b)}, m^{(b)}, \mathsf{aux})$. The strong equivocality of the original scheme ensures that Game 2 and 3 are indistinguishable via a simple reduction.

In Game 3, the commitments $C^{(0)}, C^{(1)}$ are computed independently of the message m and hence Game 3 implicitly defines the functions $\mathsf{ecommit}'$ and $\mathsf{equivocate}'$. Since Games 1 and 3 are indistinguishable the equivocality/hiding property holds for the new scheme.

Extractability and Binding After WI Proofs. We show that the extractability property for the new scheme holds even when the adversary has unlimited protocol access to a prover P running a witness indistinguishable proof for the relation $\mathcal{R}_E$. The argument that binding holds as well proceeds in almost exactly the same way and hence we skip it.

Let us assume that there is an adversary $\mathcal{A}'$ which wins the extraction game for the above scheme with non-negligible probability. This time, the adversary is also given protocol access to a prover P running a WI proof for the relation $\mathcal{R}'_E$ using the instance $pk'_E = \left(pk_E^{(0)}, pk_E^{(1)}\right)$ and the witness $sk'_E = sk_E^{(0)}$. We construct an adversary $\mathcal{A}$ which wins the extraction game for the original scheme.

The adversary $\mathcal{A}$ gets a challenge pk_E generated randomly by its challenger. It will pick a bit b at random and choose $(pk_E^{(b)}, sk_E^{(b)}) \leftarrow \mathsf{gen}_E(1^\kappa)$ and set $pk_E^{(1-b)} = pk_E$. Then it sends $pk'_E = (pk_E^{(0)}, pk_E^{(1)})$ as a challenge to $\mathcal{A}'$ and gets back $(pk_X, sk_X) \in \mathcal{R}_X$. Then $\mathcal{A}$ outputs (pk_X, sk_X) to its challenger. (Recall that our construction did not change $\mathcal{R}_X$.) In addition, it will act as a prover for the instance $(pk_E^{(0)}, pk_E^{(1)})$ using the witness $sk_E^{(b)}$. This is different from the original game where the witness $sk_E^{(0)}$ is always used. However, since the proof is WI, the success probability of $\mathcal{A}'$ can be affected at most negligibly.

Next $\mathcal{A}'$ generates some commitment $C = (C^{(0)}, C^{(1)})$ and some decommitment $(m', r') = (m', (m'^{(1)}, r^{(0)}, r^{(1)}))$. Define $m'^{(0)} = m' - m'^{(1)}$. The adversary $\mathcal{A}$ sends $(m'^{(1-b)}, r^{(1-b)})$ to its challenger.

Let

$$m^{(0)} = \mathsf{extract}^{pk_X, sk_X}_{pk_E^{(0)}}(C^{(0)}) \ , \quad m^{(1)} = \mathsf{extract}^{pk_X, sk_X}_{pk_E^{(1)}}(C^{(1)})$$

and $m = m^{(0)} \oplus m^{(1)}$. Then, if $\mathcal{A}'$ wins the extraction game, $m \neq m'$ and so $m^{\hat{b}} \neq m'^{\hat{b}}$ for some $\hat{b} \in \{0, 1\}$. Since b was only used in choosing which witness to use in the WI proof, with probability negligibly close to $1/2$ we have $\hat{b} = 1 - b$. If this is the case, then $\mathcal{A}$ wins the original extraction game. Hence the success probability of $\mathcal{A}$ is negligibly close to half the success probability of $\mathcal{A}'$ which is non-negligible.

4.3 The Protocol

Let ($\mathsf{gen}_S, \mathsf{gen}_R, \mathsf{commit}, \mathsf{extract}, \mathsf{ecommit}, \mathsf{equivocate}$) be a two-key extractable and strongly equivocal commitment scheme secure after WI proofs. Assume the scheme has an equivocation key relation $\mathcal{R}_E$ and that random extraction public keys are statistically close to uniformly random elements from some $\mathcal{G} = \{0,1\}^t$. We use such a scheme to UC realize the $\mathcal{F}_{\mathrm{MCOM}}$ functionality in the $\mathcal{F}_{\mathrm{isolate}}$-hybrid model with isolation parameter ℓ. We label the players involved $P_1, \ldots, P_n$. We also have some certificate authorities $CA_1, \ldots, CA_m$ and some certificate authority trust structure. We specify the protocol in Fig. 5.

Key Registration: The first step in the protocol is for each party to register a key with every other party. Each party P_i that wants to talk to another party P_j registers a public key with a certificate authority CA_k that is trusted by P_j. Formally, this step happens when P_i gets an input ($\mathtt{register}, sid, P_i, CA_k$). The registration is done as follows:

1. Party P_i chooses an equivocation public/secret key pair $(pk_{(E,i)}, sk_{(E,i)}) \leftarrow \mathsf{gen}_E(1^k)$. In addition, P_i generates the PPT ITM M implementing the prover functionality of a WI ℓ-IPoK protocol for the relation $\mathcal{R}_E$ using the instance $pk_{(E,i)}$ and the witness $sk_{(E,i)}$. The random tape of M is initialized with fresh random coins (enough to run one proof). The machine M is set to run a single proof and, at its conclusion, goes into an inactive state in which it produces no further output.
2. The player P_i sends ($\mathtt{isolate}, sid, P_i, CA_k, M$) to the ideal functionality $\mathcal{F}_{\mathrm{isolate}}$ and a key registration request ($\mathtt{register}, sid, P_i, CA_k, pk_{(E,i)}$) to CA_k.
3. The authority CA_k, upon receiving ($\mathtt{register}, sid, P_i, CA_k, pk_{(E,i)}$) from P_i sends ($\mathtt{isolate}, sid, P_i, CA_k$) to $\mathcal{F}_{\mathrm{isolate}}$. It then runs as a verifier in the ℓ-IPoK protocol by sending challenge messages through the interface provided by $\mathcal{F}_{\mathrm{isolate}}$. At the conclusion of this protocol, CA_k sends ($\mathtt{release}, sid, P_i, CA_k$) to $\mathcal{F}_{\mathrm{isolate}}$.
4. If the conversation is accepting, CA_k sends the message ($\mathtt{certify}, sid, CA_k, P_j, P_i, pk_{(E,i)}$) to every player P_j in its trust group.
5. When P_j receives ($\mathtt{certify}, sid, CA_k, P_j, P_i, pk_{(E,i)}$) from CA_k it sends ($\mathtt{registered}, sid, P_j, P_i, CA_k$) to P_i.
6. The party P_i ignores all commands instructing it to commit to P_j until it receives a message ($\mathtt{registered}, sid, P_j, P_i, \cdot, pk_{(E,i)}$) from P_j and a message ($\mathtt{certify}, sid, CA_d, P_i, P_j, pk_{(E,j)}$) from some trusted authority CA_d. Until then, it also ignores all coin-flip requests or commit messages from P_j.

Commitment Setup: The first time that P_i wants to send a commitment to P_j they run a coin-flipping protocol to decide on the extraction key $pk_{(X,i,j)}$. This protocol proceeds as follows:

1. P_i sends a "coin flip request" to P_j.
2. P_j picks a random $g_1 \leftarrow \mathcal{G}$ and a random extraction key $pk_X \leftarrow \mathcal{G}$. It sends pk_X and $C = \mathsf{commit}^{pk_X}_{pk_{(E,i)}}(g_1; r)$ to P_i.
3. P_i sends a random $g_2 \leftarrow \mathcal{G}$ to P_j.
4. P_j sends the opening (g_1, r) to P_i and P_i verifies that C was generated correctly as a commitment to g_1. Both parties compute $pk_{(X,i,j)} = g_1 \oplus g_2$.

Commit: Whenever P_i gets input ($\mathtt{commit}, sid, ssid, P_j, m$), it retrieves the key $pk_{(E,j)}$. Then it computes $C = \mathsf{commit}^{pk_{(X,i,j)}}_{pk_{(E,j)}}(m; r)$ and sends ($\mathtt{commit}, sid, ssid, P_j, C$) to P_j, which outputs ($\mathtt{receipt}, sid, ssid, P_i$).

Open: If P_i later gets input ($\mathtt{reveal}, sid, ssid, P_j$), then it sends ($\mathtt{commit}, sid, ssid, P_j, (m, r)$) to P_j. If $C = \mathsf{commit}^{pk_{(X,i,j)}}_{pk_{(E,j)}}(m; r)$, then P_j outputs ($\mathtt{reveal}, sid, ssid, P_i, m$).

Fig. 5. The Commitment Protocol

In the ideal world, the additional certificate authorities are not involved in the protocol at all. They get no inputs from the environment and receive no outputs. However, in the real world, parties cannot use the commitment functionality without registering their keys first. We model this discrepancy by adding a dummy registration phase to the ideal world functionality. When $\mathcal{F}_{\mathrm{MCOM}}$

gets the input $(\texttt{register}, sid, P_i, CA_k)$ from P_i then, for every P_j in the trust group of CA_k, it sends $(\texttt{certify}, sid, CA_k, P_i)$ to P_j and $(\texttt{registered}, sid, CA_k, P_j)$ to P_i. The adversary decides when these messages are delivered.

The ideal functionality $\mathcal{F}_{\text{MCOM}}$ ignores all request from P_i to commit to P_j until P_i receives the messages $(\texttt{certify}, sid, CA_d, P_j)$ and $(\texttt{registered}, sid, CA_k, P_j)$ for some CA_k, CA_d. This corresponds to the real world where a sender cannot initiate the commitment protocol until he registers a key with some CA_k trusted by the receiver and the receiver registers a key with some CA_d trusted by the sender.

4.4 Outline of Proof of Theorem 1

We now proceed to go over the intuition for how the simulation is performed and why it is indistinguishable. A detailed description/proof of the simulation is included in Appendix B.

We show that for any certificate authority trust structure, any environment $\mathcal{E}$, and any real-world adversary $\mathcal{A}$ attacking the above protocol using a valid corruption strategy, there exists an ideal-world simulator $\mathcal{S}$ such that $\mathcal{E}$ cannot distinguish between interacting with $\mathcal{A}$ in the real-world versus interacting with $\mathcal{S}$ and dummy parties using the ideal functionality $\mathcal{F}_{\text{MCOM}}$. The simulator internally runs a copy of the protocol. The simulator also internally runs a copy of $\mathcal{A}$ and lets $\mathcal{A}$ attack the internal copy of the protocol. It passes messages from $\mathcal{E}$ to its internal copy of $\mathcal{A}$ and outputs from $\mathcal{A}$ to $\mathcal{E}$.

The simulator runs all key registrations honestly, by following the code of **Key Registration** above. In particular, for an honest party P_i, the simulator will pick a key pair $(pk_{(E,i)}, sk_{(E,i)})$ honestly and remember the secret key. For a corrupt P_i, which successfully registers a public keys $pk_{(E,i)}$ with an honest CA_k, the simulator will see the PPT ITM M given by the adversary to the $\mathcal{F}_{\text{isolate}}$ functionality. In addition, M is able to run an ℓ-IPoK for the relation $\mathcal{R}_E$ and the instance $pk_{(E,i)}$ and, by the specification of $\mathcal{F}_{\text{isolate}}$, M communicated at most ℓ bits with its environment during this proof. By the definition of an ℓ-IPoK, this allows $\mathcal{S}$ to extract a witness $w_{(E,i)}$ from M, such that $(pk_{(E,i)}, w_{(E,i)}) \in \mathcal{R}_E$. For public keys $pk_{(E,i)}$ registered by a corrupted P_i at a corrupted CA_k no witness can be computed.

The coin-flipping protocols are simulated in two different ways depending on whether P_j is honest or not. If P_j is honest and accepts the coin-flipping, then it received some message $(\texttt{registered}, sid, CA_k, P_j, P_i, pk_{(E,i)})$ from an authority CA_k trusted by P_j. In addition CA_k was honest since P_j is in the trust group of CA_k and P_j is honest. Therefore $\mathcal{S}$ knows $w_{(E,i)}$ such that $(pk_{(E,i)}, w_{(E,i)}) \in \mathcal{R}_E$. The simulator uses $w_{(E,i)}$ to equivocate the commitment sent by P_j. In particular, the simulator uses $\mathsf{ecommit}$ in step 2 of the coin-flip protocol. When it receives g_2 from P_i in step 3, it then samples a random key-pair $(pk_{(X,i,j)}, sk_{(X,i,j)})$, lets $g_1 = pk_{(X,i,j)} \oplus g_2$. It then uses the $\mathsf{equivocate}$ algorithm in step 4 to open the commitment C to g_2. This results in a key $pk_{(X,i,j)} = g_1 \oplus g_2$ for which $\mathcal{S}$ knows $sk_{(X,i,j)}$ such that $(pk_{(X,i,j)}, sk_{(X,i,j)}) \in \mathcal{R}_X$. If P_j is corrupted for the coin-flip, then $\mathcal{S}$ simulates P_i by sending a random g_2 as in the protocol. Note that when P_i is honest, then $pk_{(E,i)}$ was picked at random by the simulator and the adversary did not see $sk_{(E,i)}$, except that it saw a WI proof for $sk_{(E,i)}$. Therefore the commitment C sent in step 2 of the coin-flip is computationally binding for P_j, and $pk_{(X,i,j)}$ will have been produced by a Blum coin-flip using a computationally binding commitment scheme. Intuitively it follows that, even if P_j is corrupted, the public key $pk_{(X,i,j)}$ is a random key. Formally, we rely on the "coin tossing lemma" from [CDPW] to argue that strong equivocality/hiding still hold even when the extraction public key $pk_{(X,i,j)}$ is generated using a Blum coin flip protocol as above rather than randomly as in the definition of the commitment and equivocation games in Fig. 4.

The commitments are simulated in two different ways, depending on whether P_j is honest or not. If P_j is honest, then P_j was also honest when $pk_{(X,i,j)}$ was generated. Therefore $\mathcal{S}$ knows a secret key $sk_{(X,i,j)}$ for the key $pk_{(X,i,j)}$ used by P_i. The simulator uses $sk_{(X,i,j)}$ to extract a message m from all commitments C sent by P_i to P_j. By the **Extractability** (Fig. 4) the probability that P_i later opens C to $C = \mathsf{commit}_{pk_{(E,j)}}^{pk_{(X,i,j)}}(m'; r')$ with $m' \neq m$ is negligible. If P_j is corrupted, then $\mathcal{S}$ simulates P_i without knowing m. As above, since P_i is honest and uses the key $pk_{(E,j)}$ to commit to P_j, the simulator knows a witness $w_{(E,j)}$ for the instance $pk_{(E,j)}$ in $\mathcal{R}_E$. The simulator uses $w_{(E,j)}$ to equivocate the commitment. In particular, it computes a commitment without knowing m using $\mathsf{ecommit}$. Later to simulate the opening of the commitment or the corruption of P_i, $\mathcal{S}$ receives m and uses the $\mathsf{equivocate}$ command to compute r which serves as both, an opening of m and an explanation of the randomness used to generate C. By **Strong Equivocality/Hiding** (Fig. 4), it follows that computing r using $\mathsf{equivocate}$ is indistinguishable from the way it is done in the protocol.

References

[BA03] K. Barr and K. Asanovic. Energy aware lossless data compression. In *The International Conference on Mobile Systems - MobiSys* pages 231-244,San Francisco, CA, USA, 5–8 May 2003. ACM Press, 2003.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science*, pages 186–195, Rome, Italy, 17–19 October 2004. IEEE.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser and Yehuda Lindell Resettably-Sound Zero-Knowledge and its Applications In *42nd Annual Symposium on Foundations of Computer Science*, Las Vegas, Nevada, 14–17 October 2001. IEEE.

[Can] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive 2000/067.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, 14–17 October 2001. IEEE. Full version in [Can].

[CDPW] Ran Canetti and Yevgeniy Dodis and Rafael Pass and Shabsi Walfish. Universally Composable Security with Global Setup. Cryptology ePrint Archive 2006/042. Published Version in [CDPW07].

[CDPW07] Ran Canetti and Yevgeniy Dodis and Rafael Pass and Shabsi Walfish. Universally Composable Security with Global Setup. In *Theory of Cryptography Conference - TCC 2007, Proceedings*, pages 61–85, Amsterdam 2007. Springer-Verlag. Lecture Notes in Computer Science Volume 4392/2007.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *Advances in Cryptology - EuroCrypt 2003*, pages 68–86, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science Volume 2656.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 494–503, Montreal, Quebec, Canada, 2002.

[CGS08] Nishanth Chandran, Vipul Goyal and Amit Sahai. New Constructions for UC Secure Computation using Tamper-proof Hardware In *Proceedings of Eurocrypt 2008*. Pages 545-562, Springer-Verlag LNCS 4965.

[DNW08] Ivan Damgård, Jesper Buus Nielsen, Daniel Wich. Isolated Proofs of Knowledge and Isolated Zero Knowledge. In *Proceedings of Eurocrypt 2008*. Pages 509-526, Springer-Verlag LNCS 4965. Full version in Cryptology ePrint Archive 2007/331.

[K07] Jonathan Katz. Universally Composable Multi-party Computation Using Tamper-Proof Hardware. In *Proceedings of EuroCrypt 2007*, pages 115-128, Springer Verlag LNCS 4515.

[MS08] Tal Moran and Gil Segev. David and Goliath Commitments: UC Computation for Asymmetric Parties Using Tamper-Proof Hardware. In *Proceedings of Eurocrypt 2008*. Pages 527-544, Springer-Verlag LNCS 4965.

A A Two Key Extractable and Strongly Equivocal Commitment Scheme

In this section we briefly describe the construction of a two-key extractable and strongly equivocal commitment scheme defined in [CLOS02]. Most of the observations here were already made in [BCNP04]. For our purposes we only need to make a slight modification and use a dense public key encryption scheme.

We start with a strongly equivocal, perfectly hiding commitment scheme which is not extractable. For example, we can use the Pedersen commitment scheme which is an efficient scheme based on the DL assumption. Alternatively we can use the Feige-Shamir commitment scheme which is based on the existence of one-way permutations (OWP) alone. This is the approach taken by [CLOS02] where it is shown that a small modification to the Feige-Shamir scheme makes it *strongly* equivocal as well. In the Feige-Shamir scheme, the secret key sk_E is a random string w and the public key pk_E is $f(w)$ where f is some one-way-function. We can define the relation $\mathcal{R}$ as the set of elements $(f(w), w)$ for some one way function f. For *any* such pair, the equivocated commitments and honestly produced commitments have equivalent distributions. The message space of the Feige-Shamir scheme is only 1-bit. The scheme has the property that knowledge of w allows one to create equivocated commitments and openings which are indistinguishable from real commitments and openings even if the adversary knows w as well. However, for an adversary that only sees $f(w)$, the scheme is binding.

To get extractability, we take a strongly equivocal, perfectly hiding commitment scheme and restrict the message space to only 1-bit. Then we use a dense public key CPA secure encryption scheme $(\mathsf{gen}, \mathsf{Enc}, \mathsf{Dec})$ where the ciphertexts are pseudorandom elements in some easily sampleable range $\mathcal{C}$ and where each ciphertext has only one valid decryption for any public key. To commit to a bit b, the sender computes $C_{com} = \mathsf{commit}_{pk_E}(b; r_{com})$, $C_b = \mathsf{Enc}_{pk_X}(r_{com}; r_{enc})$ and $C_{1-b} \leftarrow \mathcal{C}$ and send the commitment $C = (C_{com}, C_0, C_1)$.

To equivocate using the secret key sk_E we simply compute $(C_{com}, \mathsf{aux}) \leftarrow \mathsf{ecommit}_{pk_E, sk_E}()$, $r_{com}^{(0)} \leftarrow \mathsf{equivocate}_{pk_E, sk_E}(C_{com}, \mathsf{aux}, 0)$, $r_{com}^{(1)} \leftarrow \mathsf{equivocate}_{pk_E, sk_E}(C_{com}, \mathsf{aux}, 1)$ and $C_0 = \mathsf{Enc}(r_{com}^{(0)}; r_{enc}^{(0)})$, $C_1 = \mathsf{Enc}(r_{com}^{(1)}; r_{enc}^{(1)})$. To equivocate to a bit b send $(b, r_{com}^{(b)}, r_{enc}^{(b)})$. It is easy to see that equivocality is preserved because of CPA-security of the encryption scheme and the pseudorandomness of the ciphertexts.

The extractability property holds because the values C_0, C_1 define the encrypted messages $r_{com}^{(0)}$, $r_{com}^{(1)}$ which can be decrypted using the encryption secret key. If both equations $C_{com} = \mathsf{commit}_{pk_E}(0; r_{com}^{(0)})$ and $C_{com} = \mathsf{commit}_{pk_E}(1; r_{com}^{(1)})$ hold, then the adversary breaks the computational binding property of the original equivocal commitment scheme. If only one such equation holds, say for the bit b, then b is the extracted message and the committer cannot produce a decommitment for $1 - b$. This is true even if the adversary knows the decryption key sk_X. Similarly, binding holds because of the computational binding property of the original strongly equivocal commitment scheme.

B Simulation

In this section we give the full simulation proof for Theorem 1.

B.1 Simulating Key Registration

The environment decides when a player should perform the registration phase. In the real world, the party P_i receives a registration command $(\texttt{register}, P_i, CA_k)$ from the environment. The first time it gets such a command, it chooses the equivocation keys $(pk_{(E,i)}, sk_{(E,i)}) \leftarrow \mathsf{gen}_E(1^\kappa)$. It uses its secret key $sk_{(E,i)}$ to construct the PPT ITM M implementing the prover functionality. Subsequently, it reuses the same public/secret keys when registering with other CAs. It sends $(\texttt{isolate}, sid, P_i, CA_k, M)$ to the functionality $\mathcal{F}_{\text{isolate}}$ and $(\texttt{register}, P_i, CA_k, pk_{(E,i)})$ to CA_k.[4]

We have four cases to consider: CA_k can be corrupted or honest and P_i can be corrupted or honest. If CA_k is corrupted then so are all of the players in the trust group of CA_k. Hence if P_i and CA_k are both corrupted then the entire process of key registration is just internal communication of $\mathcal{A}$ and can be simulated trivially. We need only consider the following cases:

Honest P_i, corrupted CA_k: In the ideal world, when the simulator sees the message $(\texttt{register}, sid, P_i, CA_k)$ sent to the ideal functionality $\mathcal{F}_{\text{MCOM}}$ on behalf of an honest party P_i, it acts just like an honest party would in the real world. Namely it chooses an equivocation public/secret key pair $(pk_{(E,i)}, sk_{(E,i)}) \leftarrow \mathsf{gen}_E(1^\kappa)$ and constructs the prover M (and sets its random tape) which it sends to $\mathcal{F}_{\text{isolate}}$. Since registration is independent of any inputs chosen by the environment, it is easy to simulate honest parties by just honestly following the protocol. The simulator keeps a record of the public/secret key pairs it generates for each party during this stage of the simulation. The adversary $\mathcal{A}$ acting on behalf of the corrupted CA_k is then free to interact with the encapsulated prover functionality M by interacting with the $\mathcal{F}_{\text{isolate}}$. This gives the corrupt CA_k nothing more than protocol access to the prover functionality. The simulation here is identical to the real world interaction.

Corrupted P_i, honest CA_k: When the adversary $\mathcal{A}$ attempts key registration on behalf of a corrupted party P_i, it sends $(\texttt{register}, P_i, CA_k, pk_{(E,i)})$ to CA_k and $(\texttt{isolate}, P_i, CA_k, M)$ to $\mathcal{F}_{\text{isolate}}$. We note that the equivocation public key might be different from other such public keys registered by P_i with other certificate authorities. This is not a problem. The simulator $\mathcal{S}$ intercepts these messages and recovers the machine M and $pk_{(E,i)}$. It runs the verifier protocol with M. If the protocol is rejecting then the simulator ignores the registration request. If the protocol is accepting then it uses the ℓ-IPoK extractor $\mathcal{X}$ (Fig. 2) to extract a witness $w_{(E,i)}$ for the instance $pk_{(E,i)}$ in the relation $\mathcal{R}_E$. If the extractor is successful then the simulator sends $(\texttt{register}, P_i, CA_k)$ to $\mathcal{F}_{\text{MCOM}}$. If the extractor fails then then simulation fails. The simulation is equivalent to the real world protocol up to the negligible probability of the extractor failing.

In both of the cases, the simulator possesses a witness $w_{(E,i)}$ for the equivocation public key $pk_{(E,i)}$ registered by any party P_i with any certificate authority CA_k.

The notion of ℓ-IPoK requires that for all cheating provers P^* there exists an extractor which works for that cheating prover. Since the machines M submitted by corrupted P_i can be different and are determined adaptively, we cannot assume that the simulator $\mathcal{S}$ has a build-in extractor for

[4] As P_i only uses $sk_{(E,i)}$ during registration, it could delete $sk_{(E,i)}$ after registration (along with the randomness used in giving the proofs of knowledge of $sk_{(E,i)}$). This would give the additional security that if P_i is later transiently and passively corrupted, then the commitment functionality remains secure for P_i when it becomes uncorrupted. This is easy to see, as the internal state of P_i after deleting $sk_{(E,i)}$ contains only values which are publicly known. As transient faults are not in the focus here, we do not elaborate further on this issue.

each such M. We therefore need a little care. In the extraction game in Fig. 2, we can let the prover P^* be a universal machine. In the setup phase where P^* and $\mathcal{E}$ can communicate arbitrarily it expects an input M from $\mathcal{E}$ which it parses as a machine M. Then it runs M towards V, as $\mathcal{F}_{\text{isolate}}$ runs M towards P' in Fig. 1. Since this involves running M at most a polynomial number of steps ($p(\kappa)$ steps per move in the proof, see Fig. 1) this P^* is a PPT ITM. Therefore there exists a PPT extractor $\mathcal{X}$ which extracts all successful proofs given by P^*. Note that by definition of ℓ-IPoK this $\mathcal{X}$ is *strict* PPT and therefore can be run by a simulator in the UC framework. The simulator $\mathcal{S}$ can use this $\mathcal{X}$ to extract witnesses: Whenever some P' accepts a proof from some P, $\mathcal{S}$ parses the transcript as an interaction between the above P^* and V, as if they has run in Fig. 2. Then it gives this transcript to $\mathcal{X}$. Except with negligible probability $\mathcal{X}$ will produce a witness.

B.2 Simulating the Coin-Flipping Protocol

In the real world, the first time that a party P_i wants to send a commitment to a party P_j, it sends an initiation request to P_j to run a coin-flipping protocol. In the ideal world, the first time that the environment sends $(\texttt{commit}, \mathit{sid}, \mathit{ssid}, P_i, P_j, m)$ to $\mathcal{F}_{\text{MCOM}}$ on behalf of an honest party P_i, the simulator needs to simulate the coin-flipping. Alternatively, the adversary $\mathcal{A}$ can send a request on behalf of a corrupted party P_i to initiate a coin-flip with another party P_j.

Honest P_j: If P_j is honest, then the coin-flip is simulated as follows:

1. In the first step, P_i sends an initiation request $pk_{(E,i)}$ to P_j to run a coin-flipping protocol. If P_j accepts the request, then clearly P_i registered $pk_{(E,i)}$ at some CA_k which has P_j in its trust group. Since P_j was honest at the time the registration was made (as it is honest now), and P_j is in the trust group of CA_k, it follows that CA_k was honest at the time of registration. Therefore the simulator extracted a witness $w_{(E,i)}$ for $pk_{(E,i)}$.
2. In the second step, in the real world, P_j picks a random $g_1 \leftarrow \{0,1\}^t$ and a random extraction key $pk_{(X,j)} \leftarrow \mathcal{G}$ and sends $pk_{(X,j)}, C = \mathsf{commit}^{pk_{(X,j)}}_{pk_{(E,i)}}(g_1; r)$ to P_i. In the simulation, $\mathcal{S}$ picks $pk_{(X,j)}$ in the same manner, but computes C as $(C, \mathtt{aux}) = \mathsf{ecommit}^{pk_{(X,j)}}_{pk_{(E,i)}, w_{(E,i)}}()$. It sends $(pk_{(X,j)}, C)$ to P_i.
3. In the third step, P_i sends a random $g_2 \leftarrow \mathcal{G}$ to P_j.
4. In the fourth step of the protocol P_j sends the opening (g_1, r) to P_i. In the simulation, $\mathcal{S}$ generates $(pk_{(X,i,j)}, sk_{(X,i,j)}) \leftarrow \mathsf{gen}(1^\kappa)$, computes $g_1 := pk_{(X,i,j)} \oplus g_2$, $r = \mathsf{equivocate}^{pk_{(X,j)}}_{pk_{(E,i)}, w_{(E,i)}}(C, \mathtt{aux}, g_1)$, and sends (g_1, r) to P_i. In addition, the simulator records $(pk_{(X,i,j)}, sk_{(X,i,j)})$.

The simulation has the following properties:

– The only places where the simulator veers from the real-execution is in Steps 2 and 4 where it uses $\mathsf{ecommit}$ and $\mathsf{equivocate}$ instead of sending an honest commitment C and opening (g_1, r). Also, it chooses g_1 as $g_1 = pk_{(X,i,j)} \oplus g_2$ for $pk_{(X,i,j)}$ generated using the key generation algorithm, rather than picking g_1 uniformly at random from $\mathcal{G}$. By the fact that the the key generation algorithm produces public keys which are statistically close to uniformly random elements from $\mathcal{G}$, we can ignore the last modification. By the equivocality, the first modification is indistinguishable too.

If the adversary corrupts P_j after it sends the commitment in Step 2 but before it sends the opening in Step 4, then the simulator simply chooses a random g_1 and computes $r \leftarrow \mathsf{equivocate}^{pk_{(X,j)}}_{pk_{(E,i)},w_{(E,i)}}(C, \mathsf{aux}, g_1)$. It sets the internal state of P_j to look like the commitment C in Step 2 was computed using (g_1, r). If the adversary corrupts P_j after the end of Step 4, then the simulator sets the internal state of P_j to look like C was generated using the pair (g_1, r) sent in Step 4. Hence the simulation is indistinguishable, even if the adversary corrupts P_j during the coin-flip or later on.

– When P_j is honest through the end of the protocol, the simulator knows an extraction secret key $sk_{(X,i,j)}$ for $pk_{(X,i,j)}$.

Corrupt P_j: If P_j is corrupt, then the coin-flip is simulated as follows:

1. In the first step, $\mathcal{S}$, on the behalf of P_i, sends an initiation request $pk_{(E,i)}$ to P_j.
2. In the second step, the adversary $\mathcal{A}$ sends $(pk_{(X,j)}, C)$ on behalf of P_j.
3. In the third step, the simulator, on behalf of P_i, sends a random $g_2 \leftarrow \mathcal{G}$ to P_j.
4. In the fourth step the adversary, on behalf of P_j, sends the opening (g_1, r) to P_i. If $C = \mathsf{commit}^{pk_{(X,j)}}_{pk_{(E,i)}}(g_1; r)$ the simulator, on behalf of P_i, accepts the key $pk_{(X,i,j)} = g_1 \oplus g_2$ for future commitments to P_j.

The simulation has the following properties:

– If P_i is honest through the end of the protocol, then the adversary does not know the equivocation secret key of $pk_{(E,i)}$ — it only saw a WI proof for this key. Since $\mathsf{commit}^{pk_{(X,j)}}_{pk_{(E,i)}}$ is computationally binding against an adversary which saw WI proof for the secret key of $pk_{(E,i)}$, this means that $\mathsf{commit}^{pk_{(X,j)}}_{pk_{(E,i)}}$ is computationally binding for $\mathcal{A}$. Therefore $pk_{(X,i,j)}$ is the result of a Blum coin-flip using a computationally binding commitment scheme when P_i is honest.

Intuitively, because $pk_{(X,i,j)}$ is the result of a Blum coin-flip, it is a random key. Unfortunately, this is not necessarily true. For example P_j cam quit in step 3 depending on what the key is (e.g. if the first bit is 0) and hence, given that P_j did not quit, the key is no longer random (e.g. the first bit is always 1). This makes it difficult to argue that the *Strong Equivocality/Hiding* (Fig. 4) properties still hold even when the extraction public key is chosen through a Blum coin-flip rather than being chosen randomly by a challenger. We can use the "coin tossing lemma" from [CDPW] to deal with this. The lemma says that any indistinguishability task that is hard for a random public key, is also hard for a public key derived through a Blum coin-flip protocol using a computationally binding commitment scheme. This means that the strong equivocality and hiding properties are maintained for a corrupt P_j even if the extraction key $pk_{(X,i,j)}$ is generated by the above coin-flip with P_j being corrupted.

B.3 Simulating "Commit" and "Reveal"

Simulating "Commit" and "Reveal" where Sender is Honest: In the ideal world, the environment sends $(\texttt{commit}, sid, ssid, P_j, m)$ to $\mathcal{F}_{\text{MCOM}}$ on behalf of an honest P_i. The simulator sees the public part of the message, namely $(\texttt{commit}, sid, ssid, P_j)$. Upon seeing this, the simulator computes $(C, \mathsf{aux}) = \mathsf{ecommit}^{pk_{(X,i,j)}}_{pk_{(E,j)},w_{(E,j)}}()$ and sends $(\texttt{commit}, sid, ssid, P_i, P_j, C)$ to P_j on behalf of P_i. It remembers $(sid, ssid, P_i, P_j, C, \mathtt{aux})$. It is possible to simulate like this, as no matter whether P_j is honest or corrupted, the simulator knows $w_{(E,j)}$: If P_j is honest, then $\mathcal{S}$ generated $(pk_{(E,j)}, w_{(E,j)})$. If P_j is corrupted, then $\mathcal{S}$ extracted $w_{(E,j)}$ — since P_i is honest, it was honest when it $(\mathtt{certify}, sid, CA_k, P_i, P_j, pk_{(E,j)})$ from a CA_k it trusts. Since P_i trusts CA_k, also CA_k was honest at this point. So, P_j registered at $pk_{(E,j)}$ at an honest CA_k. As described in Section B.1, this means that the simulator extracted $w_{(E,j)}$.

Later, if the environment sends $(\mathtt{reveal}, sid, ssid, P_i, P_j)$ to $\mathcal{F}_{\mathrm{MCOM}}$ on behalf of a (still) honest player P_i, the simulator receives the message m from $\mathcal{F}_{\mathrm{MCOM}}$. The simulator recalls the tuple $(sid, ssid, P_i, P_j, C, \mathtt{aux})$ used for that commitment and computes $r \leftarrow \mathsf{equivocate}^{pk_{(X,i,j)}}_{pk_{(E,j)}, w_{(E,j)}}(C, \mathtt{aux}, m)$. The simulator sends $(\mathtt{reveal}, sid, ssid, P_i, m, r)$ to P_j.

If the adversary $\mathcal{A}$ corrupts the player P_i, then the simulator receives the committed messages m from $\mathcal{F}_{\mathrm{MCOM}}$. It recalls the corresponding stored tuples $(sid, ssid, P_i, P_j, C, \mathtt{aux})$. It computes r using the $\mathsf{equivocate}$ functionality as above and sets the internal state of P_i as though the commitment C was generated using (m, r).

The simulated commit/reveal actions are indistinguishable from the real world by the indistinguishability of the commitment game and the equivocation game. Here we also rely on the previously mentioned fact that these games remain indistinguishable even if the public key $pk_{(X,i,j)}$ is chosen using a Blum coin-flip.

Simulating "Commit" Where Sender is Corrupted: When the internal copy of $\mathcal{A}$ generates the commitment C on behalf of a corrupted player P_i by sending $(\mathtt{commit}, sid, ssid, P_j, C)$ and P_j is corrupted then this is internal communication of $\mathcal{A}$ and can be simulated trivially. In the protocol an honest P_j would now output $(\mathtt{receipt}, sid, ssid, P_i)$. The simulator must make $\mathcal{F}_{\mathrm{MCOM}}$ do the same, on behalf of P_j. Since P_j is honest, it was honest through the end of the protocol flipping the key $pk_{(X,i,j)}$ used to produce C. As discussed in Section B.2, then means that the simulator knows the extraction secret key $sk_{(X,i,j)}$ of $pk_{(X,i,j)}$. The simulator computes $m = \mathsf{extract}^{pk_{(X,i,j)}, sk_{(X,i,j)}}_{pk_{(E,j)}}(C)$ and input $(\mathtt{commit}, sid, ssid, m)$ to the functionality $\mathcal{F}_{\mathrm{MCOM}}$ on behalf of the corrupted P_i.

Simulating "Reveal" Where Sender is Corrupted: When the internal copy of $\mathcal{A}$ generates the decommitment $(\mathtt{reveal}, sid, ssid, P_j, m', r')$ on behalf of a corrupted P_i and $C = \mathsf{commit}^{pk_{(X,i,j)}}_{pk_{(E,j)}}(m', r')$, then and honest P_j in the real protocol would output $(\mathtt{reveal}, sid, ssid, P_i, m')$. In the ideal world the simulator simply input $(\mathtt{reveal}, sid, ssid, P_j)$ to $\mathcal{F}_{\mathrm{MCOM}}$ on behalf of the corrupted P_j. The functionality $\mathcal{F}_{\mathrm{MCOM}}$ will decommit to some message m.

There are two possibilities. The commitment might have been made when the sender was honest, and then the sender got corrupted later. In this case some message m was revealed to the simulator when the sender got corrupted. Secondly, the sender might have been corrupted at the time the commitment was made in which case the simulator used $\mathsf{extract}$ to recover a message m and input $(\mathtt{commit}, sid, ssid, m)$ to the functionality $\mathcal{F}_{\mathrm{MCOM}}$ on behalf of P_j (this is the case we just described above). In the second case, it follows from the definition of the extraction game and the assumption that the scheme is extractable, that $m = m'$, except with negligible probability.

The first case, is more involved as the following might have happened: 1) First C was computed by the simulator using $\mathsf{ecommit}$ (while the sender was honest). 2) Then the sender was corrupted, and the simulator used `equivocate` to open C to (m, r), where m is the message it received from the ideal functionality. 3) The corruption of the sender was done *before* the decommitment to C was computed by the simulator and delivered to the receiver. 4) The adversary instead sent the decommitment $(\texttt{reveal}, \mathit{sid}, \mathit{ssid}, P_j, m', r')$ to C. However, Steps 1 and 2 can be replaced by an honest commitment by the indistinguishability of the commitment game and the equivocation game. Then, by extractability, $m = m'$ except with negligible probability.

Indistinguishability of Simulation and Real World Interaction For each part of the simulation, we gave an argument for why it is "legitimate". We rely on the hybrid argument which can be easily formalized from the arguments outlined in each of the above sections. Namely, we start with the real world interaction. Then, in each of the above sections, we argued that the change introduced by that portion of the simulation is indistinguishable from the simulation without that change. Hence the hybrid argument ensures that the full simulation is indistinguishable from the real world interaction.