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). Recently, an alternative solution was proposed by Katz in [K07], suggesting that one may rely on physical assumptions rather than trusted third parties. Concretely, the solution assumed it physically possible to construct tamper proof hardware tokens which can be run in complete isolation from the surrounding environment. Here we improve upon the work of [K07] by constructing a scheme in which the tokens only need to be partially isolated and may have some *limited communication with the environment*. In addition we improve on Katz's work by presenting a scheme which is secure against *adaptive adversaries* and is based on *general cryptographic assumptions*. 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 limit the use of the physical set-up and hence might be preferred in practice.
Key words: universally composable security, 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 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, but which the parties are unable to create themselves from scratch in a UC secure manner. For example, a common reference string (CRS) which is honestly sampled from some pre-defined distribution allows us to UC realize any two-party and multiparty functionality with any number of (adaptively) corrupted parties [CLOS02]. Alternatively, we can rely on a public key infrastructure (PKI) where a trusted certificate authority (CA) is given a public key /secret key pair and verifies that the registrant knows the secret key corresponding to the public key being registered [BCNP04]. Both of the above setup assumptions require a trusted party to

initialize the infrastructure. For example, no security guarantees are given when the CRS is generated maliciously. Similarly, security is lost if the CA leaks the secret key of the key pair being registered.

A recent novel approach, proposed by Katz in [K07], suggests relying on physical assumptions instead of trusting a third party. Concretely, Katz uses the physical assumption of tamper proof hardware tokens that give a user only protocol access to their implemented functionality. These tokens are used to isolate a party from the environment during a portion of the computation. The party P places some arbitrary functionality (which might be malicious if P is malicious) inside a tamper proof hardware token and sends the token to a receiving party P'. The receiving party runs the hardware token in an isolated setting, ensuring that the token has no available channel of communication to the outside world. The notion of isolation is crucial in simulating the interaction as it allows us to rewind the functionality of the hardware token without needing to rewind the environment as well. The tamper-resistance property, on the other hand, ensures that access to a hardware token does not give P' more power than having protocol access to a functionality implemented remotely by P. Using these two physical assumptions, [K07] shows that one can UC realize the multiple commitment ideal functionality $\mathcal{F}_{\text{MCOM}}$ (see Fig. 3) in the presence of an active static adversary under the DDH assumption. Using the result of [CLOS02], this in turn allows us to UC realize any multiparty functionality under the same conditions.

This work improves upon that of [K07] in several respects. Most notably, we weaken the isolation requirement and allow the hardware token to communicate with the outside world as long as the number of communicated bits is below some pre-defined threshold. We argue that this relaxation might make it significantly simpler to securely implement the required physical setup. In addition, the work of [K07] leaves two open problems: constructing a protocol which is secure against adaptive adversaries and constructing a protocol which is based on general cryptographic assumptions, rather than relying on DDH. We present a scheme which meets both of these requirements.

From a technical standpoint, our construction has little in common with that of Katz. In our construction, the hardware token implements the prover functionality in a proof of knowledge (PoK) protocol. We use a protocol which ensures that one can extract a witness as long as the prover's communication with the outside world is limited. This problem is studied in a companion paper [DNW07] where we show how to construct an Isolated Proof of Knowledge (IPoK) compiler that takes as input a communication threshold ℓ and compiles a protocol in which a witness can be extracted from any cheating prover who communicates at most ℓ bits with its environment. Although such a protocol cannot be zero knowledge (ZK) when the receiving party is not isolated, we will show that the weaker notion of witness indistinguishability (WI) suffices. The parties in our protocol set up a PKI between themselves. Each player chooses a public key that he sends to every other player along with a hardware token which gives a WI proof of knowledge of the corresponding secret key. Our main technical contribution is showing how to implement the multiple commitment functionality $\mathcal{F}_{\text{MCOM}}$ using such a PKI.

We also propose an alternative approach to setting up such a PKI, by relying on some trusted Certificate Authorities but not requiring that any such authority is trusted by all of the players. Each player can safely register keys with arbitrary untrusted CAs, but trusts his own CA to correctly verify and distribute other players' public keys. The setting considered by Katz, in which there are no trusted third parties, becomes a special case of this scenario in which each player acts as his own CA and trusts nobody else. However, every player has to register a public key with every other player (which requires physical assumptions that may be expensive to realize) so there is a benefit

to having fewer partially-trusted CAs. An additional benefit, even if players choose to have only one trusted CA between them, is that the CA receives no sensitive information. This means that the protocol is secure against attacks by passive adversaries on the CA. It also gives us a form of forward security in the sense that it is useless for an adversary to corrupt the CA after all keys have been registered and retrieved by all players in the protocol.

In Section 2 we present a formal model of partially isolated tamper proof hardware tokens and multiple semi-trusted certificate authorities. In Section 3 we give some background on proofs of knowledge (PoK) and isolated proofs of knowledge (IPoK). In Section 4 we outline and prove our construction.

2 The Formal Model of Our Setting

2.1 Isolation and Black-Box Access

In this section we formalize the physical assumptions we need for our construction. Namely, we assume that a player P can encapsulate some arbitrary functionality and deliver it to a player P' such that:

1. The player P' only has black-box protocol access to the functionality.
2. There is some threshold ℓ such that the functionality cannot send/receive more than ℓ bits of communication to/from any entity other then P'.

We assume familiarity with the UC framework. We model the features of partial isolation using an ideal functionality $\mathcal{F}_{\text{isolate}}$ described in Fig. 1.

The ideal functionality $\mathcal{F}_{\text{isolate}}$ captures the two requirements of black-box access and isolation. When the party P wants to give another party P' black-box access to some functionality described by an ITM M, it uses the `create` command of $\mathcal{F}_{\text{isolate}}$ to encapsulate M. The party P' can only interact with M as it would with any remote party, by sending protocol messages to M through $\mathcal{F}_{\text{isolate}}$. The encapsulated functionality also has access to secret communication with its creator P, but $\mathcal{F}_{\text{isolate}}$ ensures that communication in either direction does not exceed ℓ bits. In our protocols there will not be any communication between the honest parties and the functionalities they encapsulate. This is because we consider such secret communication a flaw we are forced to allow rather than a feature for legitimate use. Lastly, we do not assume that the encapsulated functionality has a source of randomness. This is without loss of generality since the machine M can be picked randomly by P and have the randomness "hardcoded" into it. In general, we *do* assume that the encapsulated functionality is able to keep local state (i.e., is not resettable). Alternatively, we could have allowed resets but insisted that the functionality has a source of randomness and can generate new randomness upon being reset. We chose the former option as it is simpler and seems easier to achieve in practice. We will see that in theory we can also get security with resettable (or even stateless) hardware tokens by using resettable-ZK proofs of knowledge [BGGL01]. In practice such protocols would be highly inefficient.

There are many possible ways one could attempt to instantiate the model in the real world using physical assumptions. The work of Katz, which introduced the idea of using physical assumptions for UC security, proposed the assumption that tamper-resistant hardware tokens can be created to encapsulate any functionality. However, the paper did not mention how one would guarantee isolation. In reality it might be difficult to ensure that such a hardware token does not contain an

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

Creation: On input $(\texttt{create}, sid, P, P', M)$ 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. In addition, there is a random tape which we assume is initialized by the sender and is part of the description of M and, lastly, there is a blank work tape.
2. Let the value $\texttt{state}$ encode the initial state of M (including the values of all the tapes). Set the values $\texttt{inCom} := 0$, $\texttt{outCom} := 0$ and store the tuple $(P, P', M, \texttt{state}, \texttt{inCom}, \texttt{outCom})$.
3. Send $(\texttt{create}, sid, P)$ to P'.

Execution: 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 $\texttt{msg}$ 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 $\texttt{state}$ 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 $(P, P', M, \texttt{inCom}, \texttt{outCom})$ 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 $\texttt{msg}$ on the "in" tape for P and run M for $p(\kappa)$ steps.
2. Proceed with steps 2,3,4 of the execution command.

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

antenna or have access to some hidden communication channel. As an extreme solution, we could require that the receiving party interacts with the token inside of a Faraday cage to prevent communication with the outside world. However, by allowing some limited amount of communication, we might be able to use significantly simpler solutions. For example, if the hardware token is a smart-card which is too small to have its own power supply, then the receiving party can observe (and limit) its 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. Alternatively, the physical distance between the token and the party that executes it might give the token a significantly higher communication capability with the legitimate user than with any other party. Possibly, the fixed speed of light alone could be enough to partially isolate the token.

We may even imagine a setting where P simply brings a laptop into an office administered by P' who ensures that there is no wireless or wired internet access available at the location. The player P can then connect the laptop to a machine administered by P' with a cable. It should be reasonable to assume that the laptop has significantly less bandwidth in communicating with the outside world via any hidden channels compared with the bandwidth between the two connected machines. In this setting it is easy to ensure black-box access without relying on tamper proof hardware tokens. More generally, by requiring only incomplete isolation, we might make it significantly easier to simultaneously achieve black-box access.

For the rest of the paper we will assume the existence of a physical setup which allows us to implement the ideal functionality $\mathcal{F}_{\text{isolate}}$ for some isolation parameter ℓ without worrying about the specifics of the setup.

2.2 PKI and Certificate Authorities

We use the ideal functionality $\mathcal{F}_{\text{isolate}}$ to setup a public key infrastructure without requiring fully trusted certificate authorities. 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 $P_1, \ldots, P_n$. Each party P_i trusts some certificate authority CA_i and many parties may trust the same certificate authority. We also allow the case where a player acts as his own certificate authority (i.e. $P_i = CA_i$) and does not trust any other party. Any player P_j who wishes to interact with P_i needs to register a key with CA_i.

Since key registration uses a physical realization of the $\mathcal{F}_{\text{isolate}}$ functionality, which might be an expensive step, we make sure that our construction has a single key registration stage and afterward the parties can perform any number of arbitrary multiparty computations. An honest player will register the same public key with each authority for efficiency. However, this is not required and corrupted players may register a different public key with each CA.

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 an active and adaptive adversary who can corrupt any number of players and certificate authorities using a legal corruption strategy.*

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 R is a set of pairs (x, w) where $(x, w) \stackrel{?}{\in} R$ can be checked in poly-time in the length of x. For such a relation we define the witnesses for an instance $W_R(x) = \{w|(x, w) \in R\}$ and the language $L(R) = \{x|W_R(x) \neq \emptyset\}$. Given an NP relation 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 $(x, w) \in R$ 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 two properties of the PoK protocol:

Completeness: When the prover and verifier are honest then the interaction between P and V results in V outputting the judgment $J = \texttt{accept}$ with all but negligible probability.

Knowledge Soundness: Informally, whenever the prover succeeds in convincing the verifier there exists an efficient extractor procedure which is given the internal state of the prover and recovers a witness.

In our setting, the prover may also communicate with the environment 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 a companion paper [DNW07] which defines the notion of an ℓ-Isolated Proof of Knowledge (ℓ-IPoK) protocol where soundness holds even if the prover and the environment can exchange up to ℓ bits of communication. Formally,

Setup: First $\mathcal{E}$ 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 $\mathcal{X}$ 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.

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

It was shown in [DNW07] that there exists an Isolated Proof of Knowledge compiler (called an IPoK) which, for any NP relation 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 $(x, w_1) \in R$, $(x, w_2) \in R$ 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 $\text{EXEC}(P(x,w),V(x))$ 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^*

$$(\text{EXEC}(P(x,w_1),V^*(x)),w_1,w_2) \approx (\text{EXEC}(P(x,w_2),V^*(x)),w_1,w_2)$$

This notion is significantly weaker than zero knowledge (ZK) but [DNW07] shows that one cannot achieve ZK without isolating the verifier as well and hence we will have to rely on witness indistinguishability only.

Although the IPoK compiler generates an ℓ-IPoK protocol for any pre-defined threshold ℓ, the cost of a large ℓ is a large communication complexity of the generated protocol. The efficiency measured as the ratio of the communication complexity C to the secret communication threshold ℓ of the compiled protocol can be as low as $C/\ell = \mathcal{O}(1)$. This means that as long as the prover's secret communication capability with the environment is bounded by a constant fraction of its communication capability with the verifier, the setting admits a WI protocol where an extractor can extract a witness from the prover.

When the prover is completely isolated (i.e., $\ell = 0$), the notion of a 0-IPoK and standard PoK are essentially equivalent. In particular, we can use the resettable-ZK PoK protocol of [BGGL01] which remains witness indistinguishable even when the verifying party can reset the prover and force it to run multiple instances of a proof with the same random coins. This means that we can securely use hardware tokens which do not have the ability to keep long-term state and can be reset at will by the receiving party. The techniques of [BGGL01] can also be extended to the general case of $\ell > 0$ and [DNW07] gives an informal overview of how to construct an IPoK compiler so that the compiled protocols remain witness indistinguishable even when implemented by a resettable prover.

4 Construction

We use the results of [CLOS02] which show that one can UC-realize arbitrary MPC given the ideal functionality for multiple commitments $\mathcal{F}_{\text{MCOM}}$ 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 $\mathcal{F}_{\text{MCOM}}$ Ideal Functionality

There are several challenges in UC realizing the $\mathcal{F}_{\text{MCOM}}$ 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. Worse yet, 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. This means that the simulator needs to produce a fake commitment c, then receive a message m, and come up with a valid decommitment to m *and also* all of the randomness that explains how c was honestly produced as a commitment for m. We call this strong equivocality. Lastly, the simulator 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 and an equivocation key, 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.

We use the basic idea of [CLOS02] but modify it to fit our setting where players choose the keys themselves. As a first attempt, we could have each party choose and register an extraction key and an equivocation key. To commit, the sender would use his extraction public key and the receiver's equivocation public key. The simulator would have the corresponding secret keys and thus be able to equivocate commitments made to a corrupt receiver by an honest sender and extract commitments made by a corrupt sender to an honest receiver. There are two problems with the above idea. 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 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 after an adversary sees (possibly many) runs of WI proofs of knowledge of the equivocation secret keys.

4.1 The Commitment Scheme

A Two Key Extractable and Strongly Equivocal Commitment Scheme has two key generation algorithms $(pk_E, sk_E) \leftarrow \mathsf{gen}_E(1^k)$, $(pk_X, sk_X) \leftarrow \mathsf{gen}_X(1^\kappa)$ for the equivocation and extraction keys respectively. The commitment algorithm takes as input the two public keys and a message m and 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 $C \stackrel{?}{=} \mathsf{commit}_{pk_S, pk_R}(m; r)$.[3] In addition, we require that

[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 scheme 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.

there is an NP relation $\mathcal{R}$ such that, given any $(pk_E, sk_E) \leftarrow \mathsf{gen}_E(1^\kappa)$, we have $(pk_E, sk_E) \in \mathcal{R}$. We allow the relation to have more 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 a random bitstring in $\{0,1\}^n$ for some n. The security of the scheme is defined by the following three properties:

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

1. The challenger generates random $(pk_E, sk_E) \leftarrow \mathsf{gen}_E(1^\kappa)$, $(pk_X, sk_X) \leftarrow \mathsf{gen}_X(1^\kappa)$ and the adversary is given pk_E, pk_X, sk_X.
2. The adversary specifies 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 the commitment scheme is extractable if there exists a PPT algorithm extract such that the probability that a PPT adversary wins the extraction game is negligible in κ. We note that extractability implies binding as long as the public keys pk_E, pk_X are generated honestly. This is because an adversary who breaks binding can come up with a commitment C with two openings $(m_1, r_1), (m_2, r_2)$ one of which differs from the extracted m. However, we want to guarantee that binding holds no matter how pk_X is chosen. To do so, we have the following additional property.

2. Binding Under All Extraction Keys: We define an binding game between an adversary and a challenger as follows:

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 $pk_X \in \{0,1\}^n$. 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 the commitment scheme is binding under all extraction keys if the probability that a PPT adversary wins the binding game is negligible in κ.

3. Strong Equivocality/Hiding: We define the commitment game:

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 $(pk_E, w_E) \in \mathcal{R}$, 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.

We define the equivocation game:

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 $(pk_E, w_E) \in \mathcal{R}$, and a message m.
3. The challenger computes $(C, \mathsf{aux}) = \mathsf{ecommit}_{pk_E, pk_X, w_E}()$, $r \leftarrow \mathsf{equivocate}_{pk_E, pk_X, w_E}(C, \mathsf{aux}, m)$ and gives (C, r) to the adversary.

We say that the commitment scheme is strongly equivocal if there exists PPT ecommit and PPT equivocate such that no PPT adversary can distinguish between the commitment game and the equivocation game. We note that strong equivocality implies hiding since $\mathsf{commit}_{pk_E,pk_X}(m) \approx \mathsf{ecommit}_{pk_E,pk_X,w_E}() \approx \mathsf{commit}_{pk_E,pk_X}(m')$.

The commitment scheme of [CLOS02] meets all three of the security requirements and hence is a two key extractable and strongly equivocal commitment scheme. The public key pk_X is simply a public key for a CPA secure encryption scheme (actually for a CRS based scheme, CCA-2 security was needed but for us CPA suffices, as each player has its own key) with pseudorandom ciphertexts. We need to make an additional assumption that the encryption scheme is dense public key. The equivocation secret key, on the other hand, is a random string w and the equivocation public key is $f(w)$ for some one-way function f. The relation $\mathcal{R}$ consisting of pairs $(f(w), w)$ is indeed in NP. Also, any pre-image of $f(w)$ allows for equivocation which is indistinguishable from honestly generated commitments. 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.

For the UC proof we will need to show that a commitment C is binding, even for an adversary who sees an equivocation of C. We define the simulation-sound binding game:

1. The challenger generates a random $(pk_E, sk_E) \leftarrow \mathsf{gen}_E(1^\kappa)$, and gives pk_E to the adversary.
2. The adversary specifies a public key pk_X and a message m
3. The challenger computes $(C, \mathsf{aux}) = \mathsf{ecommit}_{pk_E,pk_X,sk_E}()$, $r \leftarrow \mathsf{equivocate}_{pk_E,pk_X,sk_E}(C, \mathsf{aux}, m)$ and gives (C, r) to the adversary.
4. The adversary specifies (m', r'). If $m' \neq m$ and $C = \mathsf{commit}_{pk_E,pk_X}(m'; r')$, then the adversary wins the game.

We say that the commitment scheme is simulation-sound binding if it holds for all PPT adversaries that they win the simulation-sound binding game with negligible probability.

Lemma 1. *A two key commitment scheme which is extractable and strongly equivocal is also simulation-sound binding.*

Proof. By the assumption that the scheme is strongly equivocal we can replace Step 3 in the simulation-sound binding game by

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.

while changing the success probability of the adversary at most negligibly. Since the adversary knows m, we can however remove this step completely from the game and simply let the adversary compute $C = \mathsf{commit}_{pk_E,pk_X}(m; r)$ itself and output both (m, r) and (m', r'). This will not change its probability of success, and might in fact make it larger as it can now pick (m, r) as it desires, as long as $m \neq m'$. That the adversary wins this last game only with negligible probability follows from the fact that the scheme is binding under all extraction keys.

4.2 Security After WI Proofs

In our setting, the adversary also has access to a prover running a WI ℓ-IPoK of the equivocation secret key. Since this proof is not ZK, it is possible that the adversary might gain some additional

knowledge about the equivocation secret key. An adversary with this additional capability might be able to win the extraction game or the binding game. We show how to convert any 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 equivocation key relation $\mathcal{R}$ into a scheme secure after WI proofs (i.e., secure in the above setting).

We 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

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

It is clear that this is an NP relation and that $(pk'_E, sk'_E) = (pk_E^{(0)}, pk_E^{(1)}, sk_E^{(0)}) \in \mathcal{R}'$. 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 an additively written finite group, like $\{0,1\}^\kappa$ with bitwise xor of strings. To commit to m, let $m^{(0)} = m - m^{(1)}$ for a uniformly random $m^{(1)}$. The sender computes

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

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

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 (1).

Equivocality/Hiding We use a series of games arguments to show that the above scheme is strongly equivocal. Let us define Game 1 to be the commitment game for the above scheme.

In Step 2 of the game, the challenger gets $(pk'_E, w_E) = (pk_E^{(0)}, pk_E^{(1)}, w_E) \in \mathcal{R}'$ such that $(pk_E^{(b)}, w_E) \in \mathcal{R}$ for $b = 0$ or $b = 1$. In addition, since the relation $\mathcal{R}$ 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 accept 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 $(C^{(b)}, \mathsf{aux}) = \mathsf{ecommit}_{pk_E^{(b)}, pk_X, w_E}()$ and $r^{(b)} = \mathsf{equivocate}_{pk_E^{(b)}, pk_X, w_E}(C^{(b)}, m^{(b)}, \mathsf{aux})$. 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 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 $\mathcal{R}$. The argument that binding under all extraction keys holds as well proceeds in almost exactly the same way and hence we skip it.

Let's 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 $\mathcal{R}'$ 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, pk_X, sk_X 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 $pk'_E = (pk_E^{(0)}, pk_E^{(1)}), pk_X, sk_X$ as a challenge to $\mathcal{A}'$. 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

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

and

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

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 with equivocation key relation $\mathcal{R}$ and with extraction keys in $\{0, 1\}^n$. We use such a scheme to UC realize the $\mathcal{F}_{\text{MCOM}}$ functionality in the $\mathcal{F}_{\text{isolate}}$-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. Our protocol has a key registration phase and a commitment phase.

Key Registration: Each party P_i chooses an equivocation public/secret key pair $(pk_{(E,i)}, sk_{(E,i)}) \leftarrow \mathsf{gen}_E(1^k)$. Before a player P_i is able to interact with a player P_j, he needs to register the public key $pk_{(E,i)}$ with the certificate authority CA_j trusted by P_j (and possibly P_j acts as his own CA). To do so, P_i generates the PPT ITM M implementing the prover functionality of a WI ℓ-IPoK protocol for the relation $\mathcal{R}$ using the instance $pk_{(E,i)}$ and the witness $sk_{(E,i)}$. In addition, P_i initializes the random tape of M 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. The player P_i sends $(\texttt{create}, sid, P_i, CA_j, M)$ to the ideal functionality $\mathcal{F}_{\text{isolate}}$ and a key registration request $(\texttt{register}, sid, P_i, CA_j, pk_{(E,i)})$ to CA_j.

The authority CA_j, upon receiving $(\texttt{register}, sid, P_i, CA_j, pk_{(E,i)})$ from P_i, runs the corresponding verifier protocol with the prover functionality M by sending it challenge messages through

the interface provided by $\mathcal{F}_{\text{isolate}}$. Since the certificate authorities get no sensitive information, we let CA_j broadcast the transcript of this protocol to all the players. At the end, if the conversation is accepting, CA_j sends the message $(\texttt{certify}, sid, CA_j, P_j, P_i, pk_{(E,i)})$ to every player P_j in its trust group. Otherwise it does nothing. The party P_j ignores all messages from P_i unless it has received a public key certification message for $pk_{(E,i)}$ from its trusted authority CA_j.

In the ideal world, the 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. In the ideal world a player P_i sends the message $(\texttt{register}, sid, P_i, CA_j)$ to the ideal functionality $\mathcal{F}_{\text{MCOM}}$ which forwards the message to CA_j. For each P_j in the trust group of CA_j, the authority CA_j responds by sending $(\texttt{certify}, sid, CA_j, P_j, P_i)$ to $\mathcal{F}_{\text{MCOM}}$ which in turn forwards it to P_j. An honest player P_i will not produce (resp. receive) a commitment from (resp. to) P_j unless he sent a registration request $(\texttt{register}, sid, P_i, CA_j)$ to CA_j and received a registration confirmation $(\texttt{certify}, sid, P_j, CA_i, P_i)$ from CA_i. Technically speaking, we then show that for any ideal functionality $\mathcal{F}$, we can UC realize the the ideal functionality $\mathcal{F}'$ which includes the dummy registration step prior to any other interaction. In practice this detail is not important.

Commitment: 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 an initiation request to P_j to run a coin-flipping protocol.
2. P_j picks a random $g_1 \leftarrow \{0,1\}^n$ and a random extraction key (to which he does not know the secret key) $pk_{(X,j)} \leftarrow \{0,1\}^n$ and sends $pk_{(X,j)}$ and $C = \mathsf{commit}_{pk_{(E,i)}, pk_{(X,j)}}(g_1; r)$ to P_i
3. P_i sends a random $g_2 \leftarrow \{0,1\}^n$ 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 $pk_{(X,i,j)} = g_1 \oplus g_2$.

Subsequently, whenever P_i wants to commit to a message m, he simply computes $C = \mathsf{commit}_{pk_{(X,i,j)}, pk_{(E,j)}}(m; r)$ and sends C. Later, to open the commitment, P_i sends $(m; r)$

4.4 Outline of Proof of Theorem 1

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 parties using the ideal functionality $\mathcal{F}_{\text{MCOM}}$. The simulator internally runs a copy of $\mathcal{A}$. It passes messages from $\mathcal{E}$ to its internal copy of $\mathcal{A}$ and outputs from $\mathcal{A}$ to $\mathcal{E}$.

To simulate key registration, the simulator picks a random equivocation public key on behalf of honest players and registers it honestly with each certificate authority. The adversary gets to see the transcripts of such proofs and may even be able to choose the challenge messages if the certificate authority is corrupted. When the adversary attempts key registration on behalf of dishonest players the simulator gets the code of the prover machine M. If the prover succeeds in running an accepting proof (with the simulator) while communicating at most ℓ bits with the environment, then the simulator extracts the corresponding equivocation secret key.

To simulate the coin-flip, if P_j is honest and P_i corrupted (at that point) then the simulator can equivocate the commitment made by P_j in Step 2 and open it in Step 4 so that the result of the flip is an extraction public key for which the simulator has the corresponding secret key. If, on the other hand, P_i is honest then the adversary controlling P_j does not have much control over the result of the flip (can only choose to open the commitment or quit). We use the "coin tossing lemma" [CDPW], which says that any indistinguishability task that is hard for a random public key, is also hard for a public key derived through the coin-flip protocol above. In particular, the commitment game and the equivocation game remain indistinguishable even when the extraction key is chosen as above in a protocol with a corrupted P_j rather than randomly.

To simulate a commitment by an honest P_i to a corrupted P_j, the simulator simply uses ecommit and opens the commitment using equivocate when the message is revealed. To simulate a commitment by a corrupted P_i to an honest P_j the simulator simply uses the extraction secret key to extract the contents of the commitment. The security of the extraction game ensures that the adversary will be unable to later open the commitment differently.

The full simulation is written out and proved formally in Appendix B.

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.

[DNW07] Ivan Damgård, Jesper Buus Nielsen, Daniel Wich. Isolated Proofs of Knowledge and Isolated Zero Knowledge. 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.

A A Two Key Extractable and Strongly Equivocal Commitment Scheme

In this section[4] 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 $C_{com} = \mathsf{commit}_{pk_E}(b; r_{com})$, $C_b = \mathsf{Enc}_{pk_X}(C; r_{enc})$ 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 $(C_{com}, \mathsf{aux}) \leftarrow \mathsf{ecommit}_{pk_E, sk_E}()$, $r^{(0)}_{com} \leftarrow \mathsf{equivocate}_{pk_E, sk_E}(C_{com}, \mathsf{aux}, 0)$, $r^{(1)}_{com} \leftarrow \mathsf{equivocate}_{pk_E, sk_E}(C_{com}, \mathsf{aux}, 1)$ and $C_0 = \mathsf{Enc}(r^{(0)}_{com}; r^{(0)}_{enc})$, $C_1 = \mathsf{Enc}(r^{(1)}_{com}; r^{(1)}_{enc})$. To equivocate to a bit b send $(b, r^{(b)}_{com}, r^{(b)}_{enc})$. 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 $r^{(0)}_{com}$, $r^{(1)}_{com}$ which can be decrypted using the encryption secret key. If both equations $C_{com} = \mathsf{commit}_{pk_E}(0; r^{(0)}_{com})$ and $C_{com} = \mathsf{commit}_{pk_E}(1; r^{(1)}_{com})$ 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 under all encryption keys holds because of the computational binding property of the original strongly equivocal commitment scheme.

[4] This section fits within the 18 pages limit in llncs format and has been placed as an appendix only due to readability. This part of the appendix will be included in a possible proceedings version.

B Simulation

In this section[5] we give the full simulation proof for Theorem 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 $(\texttt{register}, P_i, CA_j)$ 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 $(\texttt{create}, sid, P_i, CA_j, M)$ to the functionality $\mathcal{F}_{\text{isolate}}$ and $(\texttt{register}, P_i, CA_j, pk_{(E,i)})$ to CA_j.[6]

We have four cases to consider: CA_j can be corrupted or honest and P_i can be corrupted or honest. If CA_j is corrupted then so are all of the players in the trust group of CA_j. Hence if P_i and CA_j 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_j: In the ideal world, when the simulator sees the message $(\texttt{register}, sid, P_i, CA_j)$ 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_j is then free to interact with the encapsulated prover functionality M by interacting with the $\mathcal{F}_{\text{isolate}}$. This gives the corrupt CA_j nothing more than protocol access to the prover functionality. The simulation here is identical to the real world interaction.

Corrupted P_i, honest CA_j: When the adversary $\mathcal{A}$ attempts key registration on behalf of a corrupted party P_i, it sends $(\texttt{register}, P_i, CA_j, pk_{(E,i,j)})$ to CA_j and $(\texttt{create}, P_i, CA_j, M)$ 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 $pk_{(E,i,j)}$. 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 $w_{(E,i,j)}$ for the instance $pk_{(E,i,j)}$ in the relation $\mathcal{R}$. If the extractor is successful then the simulator sends $(\texttt{register}, P_i, CA_j)$ 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 $w_{(E,i,j)}$ for the equivocation public key $pk_{(E,i,j)}$ registered by any party P_i with any certificate authority CA_j.

[5] This section does not fit within the 18 pages limit in llncs format and has been included only to provide the reviewer with the opportunity to verify Theorem 1. It will not be included in a possible proceedings version.

[6] 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.

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 $(\texttt{commit}, sid, ssid, P_i, P_j, m)$ 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. The coin-flip is simulated as follows:

1. In the first step, P_i sends an initiation request to P_j to run a coin-flipping protocol. If P_i is honest at this point, this request is made by the simulator $\mathcal{S}$, otherwise it is sent by $\mathcal{A}$ on behalf of P_i.
2. In the second step, in the real world, P_j picks a random $g_1 \leftarrow \{0,1\}^n$ and a random extraction key (to which he does not know the secret key) $pk_{(X,j)}$ and sends $pk_{(X,j)}, C = \mathsf{commit}_{pk_{(E,i)},pk_{(X,j)}}(g_1; r)$ to P_i. If P_j is honest at this points, the simulator chooses a random $pk_{(X,j)}$ and computes $(C, \mathsf{aux}) = \mathsf{ecommit}_{pk_{(E,i,j)},pk_{(X,j)},w_{(E,i,j)}}()$. It sends $(pk_{(X,j)}, C)$ to P_i. Otherwise, if P_j is corrupted, the internal adversary $\mathcal{A}$ computes $(pk_{(X,j)}, C)$ on behalf of P_j.
3. In the third step, P_i sends a random $g_2 \leftarrow \{0,1\}^n$ to P_j. If P_i is honest at this point, this step is simulated faithfully by the simulator, otherwise the internal adversary $\mathcal{A}$ generates g_2 on behalf of P_i.
4. In the fourth step P_j sends the opening (g_1, r) to P_i. If P_j is still honest at this point, then the simulator chooses $(pk_{(X,i,j)}, sk_{(X,i,j)}) \leftarrow \mathsf{gen}(1^\kappa)$. It computes $g_1 := pk_{(X,i,j)} \oplus g_2$, $r = \mathsf{equivocate}_{pk_{(E,i,j)},pk_{(X,j)},w_{(E,i,j)}}(C, \mathsf{aux}, g_1)$ and sends (g_1, r) to P_i. In addition, the simulator records $(pk_{(X,i,j)}, sk_{(X,i,j)})$. Otherwise, if P_j is corrupted, the adversary $\mathcal{A}$ produces (g_1, r) on behalf of P_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 ecommit and equivocate instead of sending an honest commitment and opening to simulate an honest P_j. Also, it chooses g_1 using a key generation algorithm rather than randomly. By the equivocality (and the fact that the key generation algorithm produces public keys which are random strings), this modification is indistinguishable. 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 $r \leftarrow \mathsf{equivocate}_{pk_{(E,i,j)},pk_{(X,j)},w_{(E,i,j)}}(C, \mathsf{aux}, g_1)$. 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 $sk_{(X,i,j)}$ even if P_i is corrupted.
- If P_i is honest through the end of the protocol then the resulting extraction public key is essentially random, even if P_j is malicious. We need to argue that the strong equivocality property holds even when the extraction key is chosen through such a coin-flip protocol (with a malicious P_j) rather than randomly. Intuitively, this is true because of the security of the coin-flip which in turn holds by the binding property of the commitment scheme. This intuition is formalized and proved by the "coin tossing lemma" in [CDPW].

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 $(C, \texttt{aux}) = \mathsf{ecommit}_{pk_{(X,i,j)}, pk_{(E,j,i)}, w_{(E,j,i)}}()$ and sends $(\texttt{commit}, sid, ssid, P_i, P_j, C)$ to P_j on behalf of P_i. It remembers $(sid, ssid, P_i, P_j, C, \texttt{aux})$.

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

If the adversary $\mathcal{A}$ corrupts the player P_i, then the simulator recovers the committed messages m and recalls the corresponding stored tuples $(sid, ssid, P_i, P_j, C, \texttt{aux})$. 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.

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 $(\texttt{commit}, sid, ssid, P_j, C)$ and P_j is corrupted then this is internal communication of $\mathcal{A}$ and can be simulated trivially. If P_j is honest, then the simulator knows the extraction secret key $sk_{(X,i,j)}$. The simulator computes $m = \mathsf{extract}_{pk_{(E,j,i)}, pk_{(X,i,j)}, sk_{(X,i,j)}}(C)$. It then sends $(\texttt{commit}, sid, ssid, m)$ to the functionality $\mathcal{F}_{\text{MCOM}}$ in the ideal world.

Simulating "Reveal" Where Sender is Corrupted: When the internal copy of $\mathcal{A}$ generates the decommitment $(\texttt{reveal}, sid, ssid, P_j, m', r')$ on behalf of a corrupted P_i, where the corresponding commitment $C = \mathsf{commit}_{pk_{E,j,i}, pk_{(X,i,j)}}(m', r')$, the simulator simply sends $(\texttt{reveal}, sid, ssid, P_j)$ to $\mathcal{F}_{\text{MCOM}}$. The functionality $\mathcal{F}_{\text{MCOM}}$ will decommit to some message m. There are two options. 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 sent $(\texttt{commit}sid, ssid, m)$ to the functionality $\mathcal{F}_{\text{MCOM}}$. 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 $\mathsf{equivocate}$ 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 $(\texttt{reveal}, sid, ssid, P_j, m', r')$ to C.

In that case, however, it follows that $m' = m$ except with negligible probability from the fact that the commitment scheme is simulation-sound binding (Lemma 1).

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.