On the Necessary and Sufficient Assumptions for UC Computation

Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi

Department of Computer Science, Aarhus University
{ivan, jbn, claudio}@cs.au.dk

Abstract. We study the necessary and sufficient assumptions for universally composable (UC) computation, both in terms of setup and computational assumptions. We look at the common reference string model, the uniform random string model and the key-registration authority model (KRA), and provide new results for all of them. Perhaps most interestingly we show that:

- For even the minimal meaningful KRA, where we only assume that the secret key is a value which is hard to compute from the public key, one can UC securely compute any poly-time functionality if there exists a passive secure oblivious-transfer protocol for the stand-alone model. Since a KRA where the secret keys can be computed from the public keys is useless, and some setup assumption *is* needed for UC secure computation, this establishes the best we could hope for the KRA model: any non-trivial KRA is sufficient for UC computation.
- We show that in the KRA model one-way functions are sufficient for UC commitment and UC zero-knowledge. These are the first examples of UC secure protocols for non-trivial tasks which do not assume the existence of public-key primitives. In particular, the protocols show that non-trivial UC computation is possible in Minicrypt.

1 Introduction

We study the necessary and sufficient assumptions for universally composable (UC) computation [Can01], both in terms of which setup models are needed and how strong assumptions on the setup are needed, and in terms of necessary and sufficient computational assumptions.

One of the motivation is to study the minimal setup required for UC computation. It is known that some kind of setup *is* required, which makes it a theoretically interesting question exactly how strong an assumption must be made on the setup. We study both the common reference string model (CRS) and the key registration authority model (KRA), and some variations.

The goal of the study is also to determine the relationships between one way functions (OWF), passive secure stand-alone oblivious transfer (SA-OT)[1], UC

[1] If a passive secure SA-OT exists then also an active secure SA-OT exists via standard compilation techniques.

commitments (UC-Com) and UC oblivious transfer (UC-OT) in different set-up models[2]: for stand-alone security, we know that OWF are sufficient for other cryptographic tasks such as commitments and zero-knowledge proofs. Are OWF equivalent to any of these tasks when it comes to UC security? For the CRS models, Damgård and Groth [DG03] answered the question negatively showing that UC-Com implies key agreement in the CRS models and therefore, given the black-box separation between OWF and key agreement [IR89,RTV04], OWF are not sufficient to realize UC-Com. We find it interesting to study if this is inherent or associated to the particular setup model. We include SA-OT because it is complete for stand alone computation [Kil88], and therefore a natural question is whether SA-OT is sufficient also for UC computation. On the other hand it is interesting to know whether this assumption is minimal, i.e. whether SA-OT is also necessary to implement UC-Com and UC-OT. The motivation for including UC-OT is that it is complete for general UC computation: it is possible to implement any well-formed ideal functionality given the UC-OT functionality, see [CLOS02,IPS08]. Finally the motivation for including UC-Com is that it is potentially weaker than UC-OT but still implies a number of non-trivial tasks like coin-flip and zero-knowledge.

Highlights. We highlight some of the new findings which we find particularly interesting: In the KRA model, we provide the first construction of UC commitment from one-way functions—all previous constructions, to the best of our knowledge, used special assumptions or assumed at least public-key encryption. A consequence of it is that zero-knowledge and coin-flip can be UC securely implemented in Minicrypt. Until now it was not known if any non-trivial UC computation was possible in Minicrypt.

Remembering that UC-Com implies SA-OT in the CRS models we get another new result: The choice of the setup model can make a difference in which ideal functionalities can be implemented under a given computational assumption. In the CRS model we need SA-OT for UC-Com, but in the KRA model we can do with just OWF. This seems to be the first such separation of the setup models.

It turns out that SA-OT is sufficient for UC-OT in any setup model we considered, in particular in the minimal version of both the CRS and KRA model. Since some setup assumption is needed for general UC computation, this seems a very positive addition to the UC theory: Some setup is needed, *but even the most trivial setup will allow to implement any well-formed functionality.*

Finally, we show how to implement authenticated channels given a minimal meaningful KRA: Implementing authentication in a public-key setting is of course trivial if one can choose the structure of the public keys—one includes a verification key in the public key and signs all messages. It is by far trivial in our relaxed KRA model as we make no assumption on the public keys except that they are in the range of some one-way function, which might itself be maliciously

[2] When writing UC-Com or UC-OT we mean the multi-party, multi-session version of the protocols.

chosen. Standard constructions of signature schemes from one-way functions use verification keys with much more structure than this. This seems to be the first result which shows that *any value you could hope for to act as a public key can actually be used to implement an authenticated channel*.

Setup models: We look at five setup models:

- In the *uniform common random string (U-CRS)* model we assume that a single uniformly random ℓ-bit string crs is chosen by a trusted party and made public. Here ℓ is chosen by the protocol.
- In the *chosen common reference string (C-CRS)* model the trusted party samples crs using a poly-time one-way[3] distribution $D : \{0,1\}^\kappa \to \{0,1\}^\ell$, which allows crs to have a particular form. We assume that the trusted party samples a single $crs = D(r)$ for uniformly random $r \in \{0,1\}^\kappa$ and makes crs public. The function D might be given by the protocol π.
- The *any common reference string (A-CRS)* model is like the C-CRS model, except that we let the adversary pick D, under the only restriction that D is one-way. The trusted party samples $crs = D(r)$ and makes (D, crs) public. [4]
- In the *chosen key registration authority (C-KRA)* model we assume that the protocol contains a poly-time function f_i for each party P_i. A trusted party will sample $pk_i = f_i(s_i)$ for each P_i and give s_i to P_i and pk_i to all other parties. This models a key-registration authority with public keys pk_i, secret keys s_i and where the parties are guaranteed to know their secret keys. [5]
- The *any key registration authority (A-KRA)* model is like the C-KRA model, except that we allow the adversary to specify each f_i, under the only restriction that f_i is a one-way function when P_i is honest.

In the CRS models we in addition assume the presence of authenticated channels, as the existence of a CRS clearly does not allow authentication: All parties know the CRS and nothing else, so nothing distinguishes an honest party from the adversary. In the KRA models we start from the unauthenticated channels model, as the existence of a public-key infrastructure has the potential to allow authentication. In the A-CRS model the protocol π does not choose D, and the security of π should hold for any one-way function D. Another way of phrasing the model is to say that the protocol π, parametrized by D, should be secure in the C-CRS model for any one-way function D. In some sense this models the minimal meaningful common random string: we do not make assumptions on how random it is, but the parties can agree on the fact that there is something about the string which neither of them knows.

[3] If D is invertible then the C-CRS and the U-CRS model are trivially equivalent.

[4] Note that the A-CRS model generalizes both the U-CRS and the C-CRS, for computational security: the U-CRS is computationally indistinguishable from a setup where the trusted party picks a random seed and expands it using a pseudo random generator.

[5] This is essentially the key registration with knowledge (KRK) model from [BCNP04]. We cannot start from the KR model from [BCNP04] as we need that parties know their secret keys to implemented authenticated channels.

The A-KRA model in some sense is the minimal meaningful assumption on a key registration authority: Each party has a publicly known public key pk_i and there is some secret about pk_i which only P_i knows. A protocol for the A-KRA can therefore be run given any meaningful key registration authority, with no assumption whatsoever about the form of pk_i or the exact hardness of finding s_i. Note that one can think of simpler public key models, as the bare public key model (BPK) introduced in [CGGM00], that does not require corrupted parties to know their secret keys. However, even if the BPK model has been successfully used to break some impossibility results about concurrent ZK (see [OPV08] and reference therein), it was shown in [KL07] that UC computation is impossible even in the BPK model or any other public key model "without knowledge".

Results: Let $C_1, C_2 \in \{$ OWF, SA-OT, UC-Com, UC-OT $\}$ and M one of the described setup model. Then we can ask ourselves questions of the form *"is C_1 necessary/sufficient for C_2 in the M model?"* Some of these are of course trivial, like *is OWF necessary for UC-OT in the U-CRS model?*, but many are non-trivial and theoretically interesting, like *is SA-OT sufficient for UC-OT in the A-CRS model?*. The answers to these questions are pictorially illustrated in Figures 1 and 2. The main results are proved through the text while the implications that were already known, or that easily follow from our results and known facts are presented in Sec. 8.

$$
\begin{array}{ccccc}
\multicolumn{5}{c}{\text{U/C/A-CRS, C/A-KRA}} \\
\hline
 & \Swarrow & \text{UC-COM} & \Nwarrow & \\
\text{OWF} & & \Uparrow & & \text{SA-OT} \\
 & \Nwarrow & \text{UC-OT} & \Swarrow &
\end{array}
$$

Fig. 1. One direction of the relationship between the primitives holds in any setup model.

$$
\begin{array}{ccccc|ccccc|ccccc}
\multicolumn{5}{c|}{\text{U-CRS, A-CRS, A-KRA}} & \multicolumn{5}{c|}{\text{C-KRA}} & \multicolumn{5}{c}{\text{C-CRS}} \\
\hline
 & \not\Nearrow & \text{UC-COM} & \Searrow & & & \Nearrow & \text{UC-COM} & \not\Searrow & & & \not\Nearrow & \text{UC-COM} & \overset{?}{\Searrow} & \\
\text{OWF} & & \Downarrow & & \text{SA-OT} & \text{OWF} & & \not\Downarrow & & \text{SA-OT} & \text{OWF} & & \Downarrow_{?} & & \text{SA-OT} \\
 & \not\Searrow & \text{UC-OT} & \Nearrow & & & \not\Searrow & \text{UC-OT} & \Nearrow & & & \not\Searrow & \text{UC-OT} & \overset{?}{\Nearrow} &
\end{array}
$$

Fig. 2. The other direction differs in different setup models.

When SA-OT appears on the left side of the implication, the assumption is that there exist a protocol that implement SA-OT in the stand alone mode. When UC-Com and UC-OT appear on the left side of the implication, the assumption is that there exist a protocol that implement that functionality *in the* *model in question*, i.e., we are not referring to an hybrid world where the functionality is given to the party: We assume that the parties know a protocol to implement the functionality.

Note that the figure states that UC-Com is not sufficient for UC-OT in the C-KRA model, while the answer is yes in the A-KRA model. This may seem surprising because the C-KRA model (unlike A-KRA) allows the protocol to choose how public keys are computed and so it seems that anything that is possible in the A-KRA model should also possible in C-KRA. The catch is that the UC-Com assumption is *not* the same in the two models, in particular, having a UC-commitment scheme that works in the A-KRA model is a much stronger tool than one that needs C-KRA.

For all our negative answers to *sufficient* questions and positive answers to *necessary* questions, we mean that there is no black-box construction. We cannot answer whether OWF is sufficient for UC-Com with our current understanding of complexity theory: It might be that one-way functions do not exists, in which case the assumption OWF is false, and then OWF $\Rightarrow$ UC-Com is true. The result $\mathrm{UC-Com} \overset{(\mathrm{C-KRA})}{\not\Rightarrow} \mathrm{UC-OT}$ therefore means that there is no black-box construction which takes an implementation of UC commitment for the C-KRA model and gives an implementation of UC OT for the C-KRA model. For some of our positive answers to *sufficient* questions and negative answers to *necessary* questions, we appeal to non-black box constructions. As an example, the result $\mathrm{OWF} \overset{(\mathrm{C-KRA})}{\Rightarrow} \mathrm{UC-Com}$ uses a description of the circuit for the OWF.[6]

Related work. In [CLOS02] the main feasibility result for UC computation in the CRS model can be found. [CLOS02] needs to assume enhanced trapdoor permutation in order to achieve their results, while we use the strictly weaker assumption SA-OT. On the other hand this comparison is not quite fair as [CLOS02] tries to achieve adaptive security, and consider static security just as a special case, while our focus is entirely on static security. In [LPV09] a general framework for UC feasibility results is presented, showing how different setup assumptions (including timing model, tamper proof hardware, etc.) can be seen as different implementations of what the authors call *UC puzzles*. While in [LPV09] the results are proved assuming the existence of enhanced trapdoor permutation, we look at strictly weaker assumptions as OWF and SA-OT.

In a recent series of papers [PR08,MPR09], the classification of the cryptographic complexity of UC functionalities is studied. Perhaps most interestingly with respect to our work, in [MPR09] it is shown the SA-OT assumption is equivalent to any UC functionality being either trivial or complete. There is a clear overlap between these results and some of ours, however we focus on *setup* functionalities - that are invoked just once at the beginning of the protocol, while

[6] It might be possible that $A \not\Rightarrow B$ and $A \Rightarrow B$ at the same time. However, if for any of the $\not\Rightarrow$ separations in Figure 1, 2 this is the case, then one would have a non-black-box construction of SA-OT using OWF. Such a construction is unlikely to exist - or at least requires completely new cryptographic techniques - see also [RTV04].

the constructions in [MPR09] use the ideal functionalities during the protocol in an on-line fashion.

2 The KRA Model

In this section we give our model of minimal public-key setup, where each party knows a secret which is not known by the other parties. We associate these secrets to public values which we distribute via a key generator G. When sampled it outputs $((R_1, s_1), \ldots, (R_n, s_n))$, where each R_i is a description of a PPT set and s_i is the *secret of* P_i, and $s_i \in R_i$ for $i \in [n] = \{1, \ldots, n\}$. We call $R = (R_1, R_2, \ldots, R_n)$. For a "normal" KRA we would have that the description of R contains the parties' public keys $pk_1, \ldots, pk_n$ and that $s_i \in R_i$ if s_i is the secret key associated to pk_i; in this case we will write $(pk_i, s_i) \in R_i$. To model that a party's secret s_i is hard to find for the other parties we require that it is hard to find any s'_i such that $s'_i \in R_i$. This should hold even if one is given $R \cup \{s_j\}_{j \in [n] \smallsetminus \{i\}}$.

To allow corrupted parties to use secrets different from those of the honest parties (maybe fixed instead of random or even of another form), we let G depend on the set C of corrupted parties, and we let the adversary $\mathcal{A}$ influence the key generation as follows: Both G and $\mathcal{A}$ are ITMs. First G is given input n and C, where n defines the number of parties and $C \subset [n]$ defines the set of corrupted parties. Then G and $\mathcal{A}$ interact and at some point G outputs $(R, s_1, \ldots, s_n)$; we write $(R, s_1, \ldots, s_n) \leftarrow (G(n, C), \mathcal{A})$. For G to be meaningful we require that $s_j \in R_j$ for *all* parties $P_{j \in [n]}$. We only require that the secrets of *honest* parties, $P_{i \in [n] \smallsetminus C}$, are hard to find, as it is not necessarily meaningful to require that corrupted parties keep their secrets hidden.

We introduce some convenient notation for the case where all public keys are generated using the same function f. For a function $f : \{0,1\}^\kappa \to \{0,1\}^\ell$ we define the key generator G^f as follows: For each honest party P_i it samples $s_i \in \{0,1\}^\kappa$ and computes $pk_i = f(s_i)$. Then it outputs $\{pk_i\}_{i \in [n] \smallsetminus C}$ to the adviser. It interprets the next message from the adviser as a set $\{s_i\}_{i \in C}$ and computes $pk_i = f(s_i)$ for $i \in C$. It defines R^f by $(pk, s) \in R^f$ iff $pk = f(s)$ and then outputs $((pk_1, s_1, R^f), \ldots, (pk_n, s_n, R^f))$.

- Let n be the number of parties and C the set of corrupted parties, and run $G(n, C)$ with the UC adversary $\mathcal{A}$ as adviser.
- When $G(n, C)$ outputs $(R, s_1, \ldots, s_n)$, then send $(R, \{s_i\}_{i \in C})$ to $\mathcal{A}$ and send (R, s_i) to each P_i, letting $\mathcal{A}$ determine the delivery time.

Fig. 3. The KRA ideal functionality $\mathcal{F}^G_{\text{KRA}}$ for a generator G.

Definition 1. *We call G* meaningful *if $\forall$ $\mathcal{A}$, $\mathcal{A}$ wins the following game with negligible probability: Run $\mathcal{A}$ to get (n, C), with n polynomially bounded and $C \subset [n]$. Then sample $(R, s_1, \ldots, s_n) \leftarrow (G(n, C), \mathcal{A})$. At this point $\mathcal{A}$* wins *if $\exists j \in [n]$ $s_j \notin R_j$. If $\mathcal{A}$ did not win here, run $\mathcal{A}$ on R to get $i \in [n] \smallsetminus C$, and run $\mathcal{A}$ on $s_{-i} = \{s_j\}_{j \in [n] \smallsetminus \{i\}}$ to get an output (i, s'_i). If $s'_i \in R_i$, then $\mathcal{A}$* wins.

Definition 2. *Let $\mathcal{G}$ be a set of key generators, let π be a protocol and $\mathcal{F}$ an ideal functionality. We say that π is a UC secure implementation of $\mathcal{F}$ with a $\mathcal{G}$ KRA if π is a UC secure implementation of $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{KRA}}^{G}$-hybrid model (Fig. 3) for all $G \in \mathcal{G}$. We say that π is a UC secure implementation of $\mathcal{F}$ with any meaningful KRA (A-KRA) if the above holds for $\mathcal{G}$ being the set of all meaningful key generators.*

3 Authentication in the A-KRA Model given OWF

We show here how to implement authentication with any meaningful KRA. We first construct a system for identification secure under concurrent composition, using Σ-protocols in a more or less standard manner. Then we extend this identification system to a UC secure authentication system in a novel manner.

Implementing authentication in a public-key setting is of course trivial if one can choose the structure of the public keys—one includes a verification key in the public key and signs all messages. It is by far trivial in our relaxed KRA model as we make no assumption on the public keys except that they are in the range of some one-way function, which might itself be maliciously chosen. Standard constructions of signature schemes from one-way functions use verification keys with much more structure than this.

3.1 Σ-Protocols

For details on the following brief introduction see [CDS94]. Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. A Σ-protocol for R consists of (A, E, Z, J, W, S), where A is a poly-time algorithm which for all $(x, w) \in R$ and sufficiently long randomness r outputs a *commit message* $a = A(x, w, r)$; $E = \{0,1\}^\ell$ is a set of *challenges*; Z is a poly-time algorithm which given $(x, w) \in R$ and $e \in E$ and randomness r outputs a *reply* $z = Z(x, w, e, r)$; J is a poly-time algorithm, called the *judgment*, which given any (x, a, e, z) outputs $J(x, a, e, z) \in \{\texttt{accept}, \texttt{reject}\}$; and W is a poly-time algorithm called the *witness extractor* and S is a PPT algorithm called the *simulator*. Furthermore:

completeness: For all $(x, w) \in R$, all randomness r and $a = A(x, w, r)$ and $z = Z(x, w, e, r)$ it holds that $J(x, a, e, z) = \texttt{accept}$.

special soundness: For all (x, a, e, z) and (x, a, e', z') with $e \neq e'$, $V(x, a, e, z) = \texttt{accept}$ and $J(x, a, e', z') = \texttt{accept}$ it holds that $(x, W(x, a, e, z, e', z')) \in R$.

honest verifier zero-knowledge: For all $(x, w) \in R$ and all $e \in E$ the simulator outputs $(a, z) \leftarrow S(x, e)$ such that $J(x, a, e, z) = \texttt{accept}$ and such that the distribution of (x, a, e, z) is computationally indistinguishable from $(x, A(x, w, r), e, Z(x, w, e, r))$ for a uniformly random r. This holds even when the distinguisher is given w.

One round of the standard zero-knowledge protocol for Hamiltonian Cycle using a statistically binding commitment scheme is a Σ-protocol for Hamiltonian Cycle with $E = \{0,1\}$. Since Σ-protocols are closed under parallel composition, this gives a Σ-protocol for any NP relation R based on one-way function, with $E = \{0,1\}^\ell$ for any polynomial ℓ.

3.2 The OR-Construction

Let R_0 and R_1 be binary relations and define $R = R_0 \vee R_1$ by $((x_0, x_1), w) \in R$ iff $(x_0, w) \in R_0$ or $(x_1, w) \in R_1$. Let $\Sigma_0 = (A_0, E, Z_0, J_0, W_0, S_0)$ be a Σ-protocol for R_0 and let $\Sigma_1 = (A_1, E, Z_1, J_1, W_1, S_1)$ be a Σ-protocol for R_1 with the same challenge space as Σ_0. We construct a Σ-protocol, $\Sigma = \Sigma_0 \vee \Sigma_1$, for R as follows:

- The commit message $a = A(x, w_0, r)$ is computed as follows given w_0: Let $(x_0, x_1) = x$, compute $a_0 = A_0(x_0, w_0, r_0)$ for uniformly random r_0, sample a uniformly random $e_1 \in E$ and sample $(a_1, z_1) \leftarrow S_1(x_1, e_1)$. Let $a = (a_0, a_1)$. The randomizer is $r = (r_0, e_1, \rho)$, where ρ is the randomness used to sample S_1. Similarly with w_1, with all indices swapped.
- The verifier sends the prover a challenge e from the challenge space E.
- The reply $z = Z(x, w_0, e, r)$ is computed as follows: Let $e_0 = e \oplus e_1$. Let $z_0 = Z_0(x_0, w_0, e_0, r_0)$, $(a_1, z_1) = S_1(x_1, e_1; \rho)$ and $z = (e_0, z_0, e_1, z_1)$.
- The judgement $J(x, a, e, z)$ is performed as follows: Output $\texttt{accept}$ iff $J_0(x_0, a_0, e_0, z_0) = \texttt{accept}$ and $J_1(x_1, a_1, e_1, z_1) = \texttt{accept}$ and $e = e_0 \oplus e_1$.
- The witness extractor $w = W(x, a, e, z, e', z')$ works as follows: Let $z = (e_0, z_0, e_1, z_1)$ and $z' = (e'_0, z'_0, e'_1, z'_1)$. From $e \neq e'$ and $e = e'_0 \oplus e'_1$ and $e = e'_0 \oplus e'_1$ it follows that $e_0 \neq e'_0$ or $e_1 \neq e'_1$. Say it is $e_1 \neq e'_1$. From construction of the judgement J it follows that $J_1(x_1, a_1, e_1, z_1) = \texttt{accept}$ and $J_1(x_1, a_1, e'_1, z'_1) = \texttt{accept}$. So, let $w = W_1(x_1, a_1, e_1, z_1, e'_1, z'_1)$ and $(x_1, w) \in R_1$ and thus $((x_0, x_1), w) \in R$.
- The simulator $(a, z) \leftarrow S(x, e)$ works as follows: Sample $e_0 \in E$ uniformly at random and let $e_1 = e \oplus e_0$. (Equivalently, sample $e_1 \in E$ uniformly at random and let $e_0 = e \oplus e_1$.) Sample $(a_0, z_0) \leftarrow S_0(x_0, e_0)$ and $(a_1, z_1) \leftarrow S_1(x_1, e_1)$, and let $a = (a_0, a_1)$ and $z = (e_0, z_0, e_1, z_1)$.

It is straight-forward to verifiy that $\Sigma_0 \vee \Sigma_1$ is a Σ-protocol for $R_0 \vee R_1$.

Then the OR-construction is *witness indistinguishable* in the following sense: Consider a PPT adversary $\mathcal{A}$. Give it $(x, (w_0, w_1))$ and give it access to a proof oracle $\mathcal{O}_b$, which on input $\texttt{prv}$ picks a fresh identifier I, samples $a^{(I)} \leftarrow A(x, w_b, r)$, stores the prover intermediate state $P^{(I)} = (I, b, r)$ and returns $a^{(I)}$ to $\mathcal{A}$. On an input $(\texttt{chl}, I, e^{(I)})$ when some $P^{(I)} = (I, b, r)$ is stored, it deletes $P^{(I)}$, computes $z^{(I)} = Z(x, w_b, e^{(I)}, r)$ and returns $z^{(I)}$ to $\mathcal{A}$. At the end $\mathcal{A}$ outputs a guess at b. Then $|\Pr[\mathcal{A}^{\mathcal{O}_0}(x, (w_0, w_1)) = 0] - \Pr[\mathcal{A}^{\mathcal{O}_1}(x, (w_0, w_1)) = 0]|$ is negligible.

To see that it is witness-hiding, consider the game with oracles $\mathcal{O}_0$ and $\mathcal{O}_1$ and consider the hybrid oracle $\mathcal{O}_{\frac{1}{2}}$ which computes the proofs as follows: The commit message $a = A(x, w_0; r)$ is computed as follows: Let $(x_0, x_1) = x$, compute $a_0 = A_0(x_0, w_0; r_0)$ for uniformly random r_0 and $a_1 = A_1(x_1, w_1; r_1)$ for uniformly random r_1. Let $a = (a_0, a_1)$. The reply $z = Z(x, w_0, e, r)$ is computed as follows: Sample $e_1 \in E$ uniformly at random and let $e_0 = e \oplus e_1$. Let $z_0 = Z_0(x_0, w_0, e_0, r_0)$, $z_1 = Z_1(x_1, w_1, e_1, r_1)$ and $z = (e_0, z_0, e_1, z_1)$. It is easy to see that the probability that $\mathcal{A}^{\mathcal{O}_0}(x, (w_0, w_1))$ outputs 0 is negligible close to the probability that $\mathcal{A}^{\mathcal{O}_{\frac{1}{2}}}(x, (w_0, w_1))$ outputs 0, as the difference is whether a conversation is simulated (in $\mathcal{O}_0$) or computed correctly (in $\mathcal{O}_{\frac{1}{2}}$). In the same way it follows that the probability that $\mathcal{A}^{\mathcal{O}_1}(x,(w_0,w_1))$ outputs 0 is negligible close to the probability that $\mathcal{A}^{\mathcal{O}_{\frac{1}{2}}}(x,(w_0,w_1))$ outputs 0.

3.3 Hard Double-Witness Generator

We call G a *hard double-witness generator* for $R = R_0 \vee R_1$ if it is PPT and a random sample $(x, w_0, w_1) \leftarrow G$ has the property that $(x, w_0) \in R$ and $(x, w_1) \in R$ and that it is hard to compute w_0 from (x, w_1) and hard to compute w_1 from (x, w_0), i.e., a PPT algorithm given a random (x, w_b) outputs (x, w_{1-b}) with negligible probability. If G is a hard double-witness generator for $R = R_0 \vee R_1$, then $\Sigma = \Sigma_0 \vee \Sigma_1$ is witness hiding for G, i.e., an adversary cannot compute a witness after seeing any number of proofs. Since Σ-protocol are proofs of knowledge, the adversary cannot give a proof without knowing the witness. Putting these two observations together we get that the adversary cannot give a proof for a statement x even after seeing any number of proofs for x, in the following sense: We say that $\mathcal{A}$ *wins the reprove game* in Fig. 4 if at the end of the game there is a stored value (a, e, z) (from **reply verifier**), where $J(x, a, e, z) = \texttt{accept}$ and where $\mathcal{A}$ did not challenge a prover (in **reply verifier**) between receiving e and returning z. I.e., $\mathcal{A}$ did not challenge a prover while it had to compute its own challenge.

initialize Let $I = 0$. Sample $(x, w_0, w_1) \leftarrow G$ and give x to $\mathcal{A}$.
start prover Whenever $\mathcal{A}$ inputs $(\texttt{prv}, b)$, let $I = I+1$, sample $a^{(I)} \leftarrow A(x, w_b, r)$, store the prover intermediate state $P^{(I)} = (I, b, r)$ and return $a^{(I)}$ to $\mathcal{A}$.
challenge prover Whenever $\mathcal{A}$ inputs $(\texttt{chl}, I, e^{(I)})$ and some $P^{(I)} = (I, b, r)$ is stored, delete $P^{(I)}$, compute $z^{(I)} = Z(x, w_b, e^{(I)}, r)$ and return $z^{(I)}$ to $\mathcal{A}$.
start verifier On input $(\texttt{verify}, a)$ from $\mathcal{A}$, sample a uniformly random $e \in_R E$, store (a, e) and return e to $\mathcal{A}$.
reply verifier On input $(\texttt{reply}, a, e, z)$ from $\mathcal{A}$, where (a, e) is stored, delete (a, e) and store (a, e, z).

Fig. 4. The reprove game for $\mathcal{A}$, Σ and G

Theorem 1. *Let Σ_0 be a Σ-protocol for R_0, Σ_1 be a Σ-protocol for R_1 and G be a hard double-witness generator for $R = R_0 \vee R_1$. Then for all $\mathcal{A}$ PPT verifiers, $\mathcal{A}$ wins the reprove game with $\Sigma = \Sigma_0 \vee \Sigma_1$ and G with negligible probability.*

The intuition behind the proof is that an adversary $\mathcal{A}$ which wins the game can be used to extract a witness by rewinding the winning conversation and sending a new challenge, to get two valid conversations. Since $\mathcal{A}$ did not challenge a prover between getting e and sending its reply, the rewinding does not give problems. Which of the two witnesses w_0 and w_1 is extracted by $\mathcal{A}$ does not change significantly if we give all the proofs to $\mathcal{A}$ using a random fixed witness w_b instead of letting $\mathcal{A}$ choose b from proof to proof: If it did, it would clearly allow us to break witness indistinguishability. So, with a non-negligible probability $\mathcal{A}$ computes the witness not used to give the proofs. This allows to break G. (Full proof in Appendix A.)

3.4 Authentication

We now turn our focus to authentication. Given that we are in the A-KRA model, the sender S knows a secret s_S for his public key pk_S s.t. $(pk_S, s_S) \in R_S$ for some poly-time relation R_S. In the same way, the receiver R knows s_R s.t. $(pk_R, s_R) \in R_R$. Construct a Σ–protocol $\Sigma = \Sigma_0 \vee \Sigma_1$ for the relation $R = R_R \vee R_S$, i.e. the verifier V accepts if the prover P knows a secret key for pk_S or for pk_R. Now the parties can identify to each other using this Σ–protocol.

The way we build an authenticated channel from this identification protocol is as follows: S wants to send R a message $m \in \{0,1\}^\ell$, where ℓ is a fixed message length. We essentially let the receiver simulate a clock by identifying towards the sender ℓ times. In each "time period" the sender will then either identify itself or not. This defines the ℓ bits of the message. At the end the sender does a number of identifications to bring up the total number of identifications given by the sender to ℓ. The receiver will accept only if it sees a total of ℓ identifications. This is done to make it impossible for an adversary to drop identifications from the sender to the receiver. At the end, we add two last rounds where S identify to R and then R identifies to S. This is to inform the other party that the message was accepted.

For $m \in \{0,1\}^\ell$ define $\sigma(m) \in \{\mathtt{R}, \mathtt{S}\}^{2\ell+2}$ to be $\mathtt{S}^{m_1} \| \mathtt{R} \| \mathtt{S}^{m_2} \| \mathtt{R} \| \cdots \| \mathtt{S}^{m_\ell} \| \mathtt{R} \| \mathtt{S}^{\ell - \sum_{i=1}^{\ell} m_i} \| \mathtt{S} \| \mathtt{R}$, where m_i is the i-th bit of m. Note that $m \neq m' \Rightarrow \sigma(m) \neq \sigma(m')$, that $\sigma(m)$ contains exactly $\ell + 1$ symbols of each type, and that the last symbols are always $\mathtt{S} \| \mathtt{R}$. These are sufficient properties for the protocol to be secure. The protocol is given in Fig. 5.

setup: Sender S knows s_S, pk_R, R_S and R_R and receiver R knows pk_S, s_R, R_S and R_R such that $(pk_S, s_S) \in R_S$ and $(pk_R, s_R) \in R_R$.

sender: The first time S gets an input $m \in \{0,1\}^\ell$ it computes $\sigma = \sigma(m)$, sends m to R and runs the following:

1. Let $I = 1$.
2. If $\sigma_I = \mathtt{S}$, then instantiate a prover $P = P((pk_S, pk_R), s_S)$ and let it interact with R.
3. If $\sigma_I = \mathtt{R}$, then instantiate a verifier $V = V(pk_S, pk_R)$ and let it interact with R. If V rejects, then terminate the protocol with output `reject`.
4. When the above instance closes (either P or V), then let $I = I + 1$. If $I \leq 2\ell + 2$, then go to Step 2. If $I > 2\ell + 2$, then output `accept`.

receiver: The first time R receives $m \in \{0,1\}^\ell$ from S it computes $\sigma = \sigma(m)$ and runs the following:

1. Let $I = 1$.
2. If $\sigma_I = \mathtt{R}$, then instantiate a prover $P = P((pk_S, pk_R), s_R)$ and let it interact with S.
3. If $\sigma_I = \mathtt{S}$, then instantiate a verifier $V = V(pk_S, pk_R)$ and let it interact with S. If V rejects, then terminate the protocol with output `reject`.
4. When the above instance closes (either P or V), then let $I = I + 1$. If $I \leq 2\ell + 2$, then go to Step 2. If $I > 2\ell + 2$, then output $(\mathtt{accept}, m)$.

Fig. 5. The authentication protocol $\pi_{\mathsf{au}}((pk_S, pk_R), s_S, s_R)$.

Theorem 2. *If the public keys are set up as in Fig. 3, then the following holds except with negligible probability: If S outputs* `accept` *at the end of $\pi_{\mathtt{au}}$ then R outputs $(\mathtt{accept}, m)$, where m was the message input by S.*

The intuition is as follows: For $I = 1, \ldots, 2\ell+2$ we match the i'th instance run by S to the i'th instance run by R. If S and R open a prover respectively a verifier, or a verifier respectively a prover, then they might both continue to $I+1$ without rejecting. If S and R both open a verifier, then one of them will be terminated without a prover were running. Therefore this verifier will reject (by Thm. 1), which makes the party running that verifier reject. If S and R both instantiate a prover, then one of these provers will close without a verifier having been running at the other party.

Wlog, say that a prover was running at S while no verifier was running at R (one can repeat the argument switching the role of R and S). This prover will not make any verifier accept at R, therefore S will run more provers than the number of accepting verifiers that R runs. Since S starts $\ell+1$ provers, by construction of σ, it follows that R sees at most ℓ accepting verifiers. Therefore R will not output `accept`. It follows that if S and R have different σ, then one of them does not output `accept`. In other words, if both parties output accept, then they saw the same message m, as σ is a unique encoding of m.

Second, assume that R did not accept. This implies that R rejected when $I < 2\ell+2$ or at least R never reached $I = 2\ell+2$, as $\sigma_{2\ell+2} = \mathtt{R}$ implies that R cannot reject while $I = 2\ell+2$. Therefore R ran at most ℓ provers and thus S saw at most ℓ verifiers accept. Therefore S did not accept either. In other words, if S accepts, then R accepts.

Putting these two observations together, we conclude that if S accepts, then both parties accept, and then S and R saw the same message m, as desired. (Full proof in Appendix B.) This symmetric guarantee makes the protocol suitable also to authenticate messages from R to S, and we will use this property in Thm. 3.

init: First it lets $\mathrm{init}_i = 1$ for all P_i, and then it runs $\mathcal{F}^G_{\text{KRA}}$ with adversary $\mathcal{A}$, to generate $(R_1, pk_1, s_1), \ldots, (R_n, pk_n, s_n)$.
init done: If the adversary inputs $(\mathtt{done}, i)$ at a point where $\mathrm{init}_i = 1$ and after $\mathcal{F}^G_{\text{KRA}}$ terminated, then output (pki, s_i) to P_i, where $pki = ((R_1, pk_1), \ldots, (R_n, pk_n))$, and set $\mathrm{init}_i = 0$.
authenticated transfer, send: On input (j, m) from P_i where $\mathrm{init}_i = 0$, store (i, j, m) and output (i, j, m) to the adversary.
authenticated transfer, receive: On input (i, j, m) from the adversary, if (i, j, m) was previously stored, wait until $\mathrm{init}_j = 0$ and then output (i, m) to P_j.

Fig. 6. The ideal functionality $\mathcal{F}^G_{\text{MAU}}$ for multiparty authenticated communication.

3.5 Multiparty Authentication

Our ideal functionality for authenticated transmission is given in Fig. 6. We have it do a key setup as $\mathcal{F}^G_{\text{KRA}}$ and output the generated keys before the authenticated transfer phase begins. This is for compositional reasons—it allows outer protocols to use the same secrets, which we exploit in later sections. Here we focus on the phase after the keys are generated: The functionality allows to deliver only messages which were actually sent, which models authentication. It can deliver a message several times and reorder them. This can be handled outside $\mathcal{F}_{\text{MAU}}$ using e.g. sequence numbers. Any message sent is leaked to the adversary to model that the transmission is only authenticated, not private.

Our implementation of $\mathcal{F}^G_{\text{MAU}}$ runs in the $\mathcal{F}^G_{\text{KRA}}$ hybrid model, see Fig. 7.

setup: When party P_i receives (pki, s_i) from $\mathcal{F}^G_{\text{KRA}}$, it parses pki as $((R_1, pk_1), \ldots, (R_n, pk_n))$ and sets init = 1.

key generation: On its first activation P_i generates a random verification key vk_i for a digital signature scheme, along with the corresponding signing key sk_i and stores $(\texttt{keys}, vk_i, sk_i)$. Then P_i sends vk_i to all other parties.

key authentication: After **key generation** each ordered pair of parties (P_i, P_j) with $i < j$ runs the following in parallel:

- The parties P_i and P_j run the protocol $\pi_{i,j} = \pi_{\texttt{au}}((pk_i, pk_j), s_i, s_j)$ from Fig. 5.
- Party P_i uses the input $m = (vk_i, vk'_j)$, where vk'_j is the value it received from P_j in **key generation**. Party P_j uses the input $m = (vk'_i, vk_j)$, where vk'_i is the value it received from P_i in **key generation**.
- If P_i accepts in $\pi_{i,j}$, then it stores $(\texttt{vk}, j, vk'_j)$. If P_j accepts in $\pi_{i,j}$ then it stores $(\texttt{vk}, i, vk'_i)$.

KRA propagation: When P_i stored $(\texttt{keys}, vk_i, sk_i)$ and $(\texttt{vk}, j, vk'_j)$ for all P_j with $j \neq i$, then P_i outputs (pki, s_i) and sets init = 0.

authenticated transfer, send: When P_i gets input (j, m), where init = 0, P_i computes $S = \text{sig}_{sk_i}(i\|j\|m)$ and sends (i, j, m, S) to P_j.

authenticated transfer, receive: On a message (i, j, m, S) the party P_j waits until init = 0. Then it looks up $(\texttt{vk}, i, vk_i)$ and outputs (i, m) if $\text{ver}_{vk_i}(i\|j\|m, S) = \texttt{accept}$.

Fig. 7. The protocol π^G_{MAU} for multiparty authenticated communication.

Theorem 3. *If G is a meaningful key generator, then π^G_{MAU} UC securely implements $\mathcal{F}^G_{\text{MAU}}$ against a static, active adversary.*

The proof is essentially a reduction to Thm. 2. If π^G_{MAU} is not secure it is possible to make an honest P_j output (i, m) for an honest P_i without giving input (j, m) to P_i. We can reduce that to an attack on the protocol in Fig. 5. First of all, we can assume that all other parties than P_i and P_j are corrupted, as this can only help the adversary. Then, whenever P_i or P_j have to interact with any $P_k \notin \{P_i, P_j\}$, they run the protocol honestly, but use the secret s_k of P_k as witness. By witness indistinguishability (WI) this changes the probability that P_j outputs (i, m) without P_i having received input (j, m) at most negligibly. But now all interaction involving other parties than P_i and P_j can be run by the adversary in its head, as it knows s_k for all corrupted parties—whatever messages P_i would send to P_k can be computed using s_k. But this modified adversary is carrying out an attack on Fig. 7 with $n = 2$. This is essentially an attack on Fig. 5. The only difference is that in Fig. 7, during **KRA propagation**, the environment gets s_i and s_j from P_i, P_j. This happens after the protocol $\pi_{i,j}$ was run, and therefore it is not needed to run the adversary against Fig. 5. (Full proof in Appendix C.)

4 UC-OT in the A-KRA Model given SA-OT

Suppose we are given an UC commitment functionality, $\mathcal{F}_{\text{MCOM}}$ as defined in [CF01]: then we can implement UC zero-knowledge, $\mathcal{F}_{\text{MZK}}$, for all NP relations, which in turn allows us to implement a static, active UC secure OT from the passive secure OT. We can therefore focus on implementing $\mathcal{F}_{\text{MCOM}}$ using SA-OT.

The main idea of the protocol in Fig. 8 is to "compile" the SA-OT into a UC-OT using the WI proof for statements of the kind "I followed to protocol or I know your secret key".

The following describes a commitment to m from party S to party R. In the full protocol different instances use session identifiers to separate executions. Here commit$(\cdot)$ is a statistically binding commitment.

1. All communication is authenticated using $\mathcal{F}^{G}_{\text{MAU}}$. Use sequence numbers to guarantee that no identical messages are ever sent, and thus never accept the same message twice from any party.
2. R samples a uniformly random string u and sends $U \leftarrow \text{commit}(u)$ to S.
3. S samples a uniformly random string v, sets $m_0 = 1^{|m|}$, sets $m_1 = m$ and sends $V \leftarrow \text{commit}(v)$, $M_0 \leftarrow \text{commit}(m_0)$ and $M_1 \leftarrow \text{commit}(m_1)$ to R.
4. Then S and R run the SA-OT, where S takes inputs m_0 and m_1 and uses randomness v while R gives input $c = 0$ and uses randomness u. After sending each message in the SA-OT R shows that it knows an opening of U to u such that the message it sent is consistent with having run the SA-OT with randomness u, selection bit $c = 0$ and the messages received from R so far. After sending each message in the SA-OT S shows that it knows an opening of V, M_0 and M_1 to v, m_0 respectively m_1 such that the messages it sent are consistent with the execution of the SA-OT with randomness v, inputs m_0, m_1 and the messages received from S. The proofs are given via a Σ-protocol for NP and use the OR-construction to prove either knowledge of the openings mentioned above or the secret of the other party.
5. To open S sends m to R and shows that m is the message inside M_1 or that S knows s_R such that $(pk_R, s_R) \in R_R$. The proof is given using two Σ-protocols and the OR-construction.

Fig. 8. The protocol π_{MCOM} for UC commitments using SA-OT.

Theorem 4. *The protocol π_{MCOM} is a UC secure implementation of $\mathcal{F}_{\text{MCOM}}$ in the $\mathcal{F}^{G}_{\text{MAU}}$ hybrid model secure against a static, active adversary.*

The simulator extracts a commitment from a corrupted sender S^* to an honest receiver R by using selection bit $c = 1$ to learn m from the SA-OT. If the sender manages to send $m' \neq m$ in the opening phase for some commitment, we can extract the proofs in the SA-OT for this commitment and learn a secret s'_R for R's public key pk_R. Since R never uses s_R in the protocol, this contradicts the hardness of computing a witness for pk_R. To be able to use selection bit $c = 1$, the simulator gives the proof in the run of the SA-OT using the secret s'_S of the sender. This goes unnoticed by the computational hiding of the commitment scheme, the computational hiding of the SA-OT and the WI of the OR-construction. To trapdoor open a commitment to some m' the simulator simply sends m' and simulates the proof that this is the correct message, by using the secret of the receiver as witness. This goes unnoticed as for $c = 1$. (Full proof in Appendix C.)

Corollary 1. *If there exists a passive secure OT protocol, then any well-formed functionality $\mathcal{F}$ can be UC implemented in the A-KRA model, against a static, active adversary.*

Proof: By Thm. 4 we can implement $\mathcal{F}_{\text{MCOM}}$ in the $\mathcal{F}^G_{\text{MAU}}$-hybrid model, which implies that we can implement any well-formed $\mathcal{F}$ in the $\mathcal{F}^G_{\text{MAU}}$-hybrid model, if there exists a passive secure SA-OT protocol. By Thm. 3 we can implement $\mathcal{F}^G_{\text{MAU}}$ in the unauthenticated $\mathcal{F}^G_{\text{KRA}}$-hybrid model for any meaningful G. It then follows from the UC composition theorem that we can implement any well-formed $\mathcal{F}$ in the unauthenticated $\mathcal{F}^G_{\text{KRA}}$-hybrid model for any meaningful G, if there exists a passive secure SA-OT protocol. $\square$

5 UC Commitment in the C-KRA Model given OWFs

In a nutshell, to construct UC-Com in the C-KRA model, we let the public keys to be commitments of the secret keys. Then to commit the sender send an encryption of the message under his secret key. To open, he sends the message m together with a WI proof for a statement "m is the committed message or I know your secret key".

Theorem 5. *If one-way functions exist, then there exists a UC commitment scheme for the C-KRA model secure against a static, active adversary.*

Proof: The public key is an unconditionally binding commitment $pk_i = \text{commit}(K_i; r_i)$ to a uniformly random value $K_i \in_R \{0,1\}^\kappa$. Let $F_{\{0,1\}^\kappa} : \{0,1\}^{2\kappa} \to \{0,1\}^{2\kappa}$ be a pseudo-random permutation (PRP). Both can be instantiated using one-way functions.

To commit to $m \in \{0,1\}^\kappa$ with session identifier $sid \in \{0,1\}^\kappa$ towards P_j, P_i sends $M = F_{K_i}(sid \| m)$. To open the commitment to P_j, the sender sends m and gives a proof that it knows K and r such that "$pk_j = \text{commit}(K; r) \vee (pk_i = \text{commit}(K; r) \wedge M = F_K(sid \| m))$". The proof is given using two Σ-protocols combined with the OR-construction.

To extract, the simulator computes $m = F^{-1}_{K_i}(M)$, where K_i is found as part of the secret $s_i = (K_i, r_i)$ of the sender P_i. By pk_i binding the sender to K_i unconditionally and the soundness of the proof and the fact that the sender cannot open the commitment pk_j, this will yield the only m that the sender can open the commitment to later.

To equivocate the simulator sends a uniformly random M. When given m it sends m and gives the proof using the secret s_j of the receiver as witness. By computational hiding of the commitment scheme, pseudo-randomness of F and WI of the proof, this will go unnoticed. □

6 UC OT in the A-CRS Model given SA-OT

Here we implement UC OT from SA-OT in any CRS model. We prove it for the A-CRS model, and hence for the U-CRS and C-CRS models too. We already know how to do UC OT in the A-KRA model given SA-OT, so it is sufficient to implement $\mathcal{F}_{\text{KRA}}^{G}$ in the $\mathcal{F}_{\text{CRS}}^{D}$ for any meaningful G and all one-way D.

The protocol runs in the $\mathcal{F}_{\text{CRS}}^{D}$-hybrid model.

- All parties P_i receive (D, crs) from $\mathcal{F}_{\text{CRS}}^{D}$.
- Each P_i samples $pk_i = D(s_i)$ for a uniformly random s_i and sends pk_i to all parties. All parties resend the received value pk_i to all parties.
- Then in round-robin, for $i = 1, \dots, n$, each P_i proves knowledge of s_i to all other parties. It does this in round robin, for $j = 1, \dots, n$. With each P_j it runs the proof as in Fig. 8: It inputs $m_0 = 0^{|s_i|}$ and $m_1 = s_i$ to the SA-OT and P_j inputs $c = 0$. During the run of the SA-OT, P_i proves that either 1) its messages are consistent with a run of the SA-OT protocol and $pk_i = D(m_1)$ or 2) its messages are consistent with a run of the SA-OT protocol and $crs = D(m_1)$. Party P_j proves that either 1) its messages are consistent with a run of the SA-OT protocol with $c = 0$ or 2) it knows s such that $crs = D(s)$. The proofs are given via a Σ-protocol for NP and the OR-construction. When P_i and P_j are done, they both send `done` to the other parties. Parties only begin their proof when they received `done` from all previous pairs.
- If and when a party P_k received crs from $\mathcal{F}_{\text{CRS}}^{D}$, a value pk_i from each P_i and a resent value pk_i' from all other parties P_j with $pk_i' = pk_i$, and saw an accepting proof from each P_i, it outputs $((pk_1, R^D), \dots, (pk_n, R^D)), s_k$.

Fig. 9. The protocol π_{KRA}^{D} that implements a KRA in the A-CRS model.

Theorem 6. *If D is OWF, then G^D is meaningful, and if the used OT protocol is a SA-OT, then π_{KRA}^{D} in Fig. 9 is a UC secure implementation of $\mathcal{F}_{\text{KRA}}^{G^D}$ in the $\mathcal{F}_{\text{CRS}}^{D}$-hybrid model against a static, active adversary.*

The proof is very similar to the proof of Thm. 4. The simulator extracts the secret of corrupted parties using selection bit $c = 1$. It simulates proofs using the secret s of crs. We run the proofs in round-robin to ensure that the simulator will not give a simulated proof (using s) while a corrupted party has to give a proof. If it did so, we could not show that a corrupted party cannot give an acceptable proof unless it used m_1 such that $pk_i = D(m_1)$ in the SA-OT. When the proofs are run in round-robin, we can. (Full proof in Appendix C.)

7 UC-Com in the A-KRA model implies SA-OT

Theorem 7. *SA-OT is necessary for UC-Com in the A-KRA model.*

Proof: We show how a UC secure commitment scheme for the A-KRA model can be turned into a SA-OT. Note that this UC-Com protocol needs to work for any KRA, so we can choose a special KRA that it's possible to "simulate" in some sense without using any setup assumptions.

To simplify the proof, consider the AND primitive, where A inputs $a \in \{0,1\}$ and B inputs a bit $b \in \{0,1\}$ and where A has no output and B gets output $c = ab$. It is well-known that if there exists passive, stand-alone secure AND (SA-AND), then there also exist SA-OT. Then it is sufficient to show how to implement SA-AND from UC-Com in the A-KRA model.

The existence of UC-Com clearly implies OWFs, so we can assume that we have a PRG $g : \{0,1\}^\kappa \to \{0,1\}^{\kappa+1}$. Consider the key generator G^f, where $f : \{0,1\}\times\{0,1\}^\kappa\times\{0,1\}^{\kappa+1} \to \{0,1\}^{\kappa+1}\times\{0,1\}^{\kappa+1}$ and $f(b, r^b, pk^{1-b}) = (pk^0, pk^1)$ for $pk^b = g(r^b)$. This is clearly a meaningful generator, as a PRG $g : \{0,1\}^\kappa \to \{0,1\}^{\kappa+1}$ is one-way.

From the assumption that there exist UC-Com in the A-KRA model, we have a protocol π which UC implements $\mathcal{F}_{\text{MCOM}}$ in the $\mathcal{F}^{G}_{\text{KRA}}$-hybrid model with sender S and receiver R. The sender gets key material (pk_S, s_S) and pk_R and the receiver gets key material pk_S and (pk_R, s_R). Here $pk_i = f(s_i)$ for $i = S, R$.

Consider the following adversary $\mathcal{A}$ against π for the case when the sender is corrupted: It samples uniformly random $c \in \{0,1\}$, $r^c_S \in \{0,1\}^\kappa$ and $r^{1-c}_S \in \{0,1\}^\kappa$ and lets $pk^c_S = g(r^c_S)$ and $pk^{1-c}_S = g(r^{1-c}_S)$. Then it inputs $s'_S = (1 - c, r^{1-c}_S, pk^c_S)$ to $\mathcal{F}^{G^f}_{\text{KRA}}$. Then it commits to some $m \in \{0,1\}$ by honestly running the commitment phase of the protocol π with key material (pk_S, s_S) and pk_R, where $s_S = (c, r^c_S, pk^{1-c}_S)$. It's clear here that $s_S \neq s'_S$ as the first bit is different, and $f(s_S) = f(s'_S)$. Later it decommits by honestly running the opening phase of the protocol π.

Lemma 1. *When running with $\mathcal{A}$, the honest receiver R will accept the commitment and will later accept the opening to m, except with negligible probability.*

The proof follows from the fact that R cannot distinguish $\mathcal{A}$ from the honest sender S (Full proof in Appendix C.). By π being UC secure, and the above lemma, it follows that there exists a UC simulator $\mathcal{S}$ which can extract m from the conversation with $\mathcal{A}$ already in the commitment phase. Since $\mathcal{S}$ is simulating $\mathcal{F}^{G^f}_{\text{KRA}}$ to $\mathcal{A}$, it follows that $\mathcal{S}$ learns s'_S and chooses the value of pk_R.

Consider then the SA-AND in Fig. 10. If $b = 1$, then all values are distributed as in the simulation of π with $\mathcal{A}$ and $\mathcal{S}$, so B computes a, except with negligible probability. This established the correctness, hence it only remains to prove the following lemma.

Lemma 2. *1) When A and B are honest, then the view of A when $b = 0$ and $b = 1$ are computationally indistinguishable. 2) When A and B are honest and $b = 0$, then the views of B when $a = 0$ and $a = 1$ are computationally indistinguishable.*

1. First B samples $c \in \{0,1\}$ uniformly at random. Then, if $b = 1$, it uses $\mathcal{S}$ to sample pk_R, samples $r_S^{1-c} \in \{0,1\}^\kappa$ uniformly at random and lets $pk_S^{1-c} = g(r_S^{1-c})$. If $b = 0$, then B samples $pk_R = f(s_R)$ for uniformly random s_R and samples uniformly random $pk_S^{1-c} \in_R \{0,1\}^{\kappa+1}$. In both cases it sends (c, pk_S^{1-c}) and pk_R to A.
   At the same time A samples uniformly random $r_S^c \in_R \{0,1\}^\kappa$, lets $pk_S^c = g(r_S^c)$ and sends pk_S^c to B.
2. Both parties let $pk_S = (pk_0^S, pk_1^S)$. A sets $s_S = (c, r_S^c, pk_S^{1-c})$. If $b = 1$ then B lets $s'_S = (1-c, r_S^{1-c}, pk_S^c)$. *Note that in this case* $f(s_S) = pk_S = f(s'_S)$.
3. A inputs a by committing to $m = a$ by honestly running the commitment phase of π, playing the role of the sender S with key material (pk_S, s_S) and pk_R.
   If $b = 1$, then B runs $\mathcal{S}$ to extract a from the conversation with A, and outputs a. If $b = 0$, then B honestly runs the commitment phase of π, playing the role of the receiver R with key material (pk_R, s_R) and pk_S, and outputs 0.

Fig. 10. SA-AND protocol

Part 1) follows readily from the fact that by UC security R and $\mathcal{S}$ cannot be distinguished by $\mathcal{A}$. Part 2) follows readily from the fact that a commitment hides the message when both parties are honest. (Full proof in Appendix C.) □

8 Conclusions

Combining our findings with some previous results it is possible to fill the rows of Table 8. We will make use of the following:

Theorem 8. *[IR89] There is no black-box construction of SA-OT from OWF.*

Theorem 9. *[IR89] There is no black-box construction of key-agreement (KA) from OWF.*

Theorem 10. *[DG03] UC-Com in the U/A-CRS model implies SA-OT.*

Theorem 11. *[DG03] UC-Com in the C-CRS model implies KA.*

The answer to (a) follows directly from Thm. 11 and Thm. 9 for the CRS models; in the same way it follows from (j) and Thm. 8 for the A-KRA model; (b) is shown in Thm. 5; (c) follows from Thm. 9 and the fact that UC-OT in any model implies KA; the answer to (d) is built from the fact that UC-Com in those models implies SA-OT (see (j)), and that we can compile this into a UC-OT using the UC-Com, as it implies UC-ZK; the answer to (f) goes as follows: (m) tells us that UC-OT in the C-KRA model implies SA-OT while (b) tells us that OWF are sufficient for UC-OT in the C-KRA model. Therefore UC-Com is not sufficient for UC-OT, or we will get a contradiction with Thm. 8; (g) is proved in Thm. 4 and 6; (h) is trivial as OWF are minimal for cryptography, and (i) is trivial as UC-OT is complete for UC computation; (j) is proved in Thm. 7 and 10; (l) follows from (b) and Thm. 8; finally (m) follows from the following observation: semi-honest parties can efficiently simulate the U-CRS (or the A-CRS) setup model by letting one party pick a random string without learning the trapdoor and make the *crs* public. Then the parties will run the UC-OT

	Assumption		Functionality	Model	Answer
(a)	OWF	suf.	UC-Com	U/C/A-CRS, A-KRA	N
(b)	OWF	suf.	UC-Com	C-KRA	Y
(c)	OWF	suf.	UC-OT	U/C/A-CRS, C/A-KRA	N
(d)	UC-Com	suf.	UC-OT	U/A-CRS, A-KRA	Y
(e)	UC-Com	suf.	UC-OT	C-CRS	open
(f)	UC-Com	suf.	UC-OT	C-KRA	N
(g)	SA-OT	suf.	UC-Com, UC-OT	U/C/A-CRS, C/A-KRA	Y
(h)	OWF	nec.	UC-Com, UC-OT	U/C/A-CRS, C/A-KRA	Y
(i)	UC-Com	nec.	UC-Com, UC-OT	U/C/A-CRS, C/A-KRA	Y
(j)	SA-OT	nec.	UC-Com	U/A-CRS, A-KRA	Y
(k)	SA-OT	nec.	UC-Com	C-CRS	open
(l)	SA-OT	nec.	UC-Com	C-KRA	N
(m)	SA-OT	nec.	UC-OT	U/A/C-CRS, A-KRA	Y
(n)	SA-OT	nec.	UC-OT	C-CRS	open

Table 1. Questions and answers: If a cell contains more than one element it means that the answer, Y(es) or N(o), in the row is true for all elements in the cell. As an example, row (g) says that the answer to the question *is SA-OT sufficient for UC-OT in the A-CRS model?* is yes.

protocol using this string as the CRS, therefore achieving a SA-OT. As for the C-KRA (or the A-KRA) models, they can be efficiently simulated by letting every party generate his own public/secret key pair and sending the public key to all other parties. Now the parties can run the UC-OT using those public keys, and they'll achieve a SA-OT.

8.1 The C-CRS setup assumption

In this section we discuss the C-CRS model, and the open questions (e), (k) and (n) left in the table. Consider (n): is SA-OT necessary for UC-OT in the C-CRS model? The way we positively answered the question for the other setup models is by letting one party honestly pick a random CRS and publish it, therefore simulating the setup model. We don't know how to do it in the C-CRS model: in fact, we don't know whether it is possible, for any chosen OWF f, to sample an image $y = f(x)$ *without* learning the pre-image x. For instance, if $x \in \mathbb{Z}_q$ and $f(x) = (g^x, h^x)$ for g, h elements in group of large prime order q, then it is believed that one *cannot* sample from the image of f without learning x, to the extent that people construct protocols based on this belief (the so-called knowledge of exponent assumption [Dam91]). This suggests very strongly that the open questions cannot be solved using the techniques we have used here.

It could of course be possible to approach (n) in some other way. It seems counter-intuitive to think that it would be possible to implement UC-OT in a world where SA-OT does not exist: how much power does a symmetric setup as the C-CRS give to the parties? However, if it turns out that the answer to (n) is affirmative, then we could use (g) to turn any UC-OT in the C-CRS model into a UC-OT in the U/A-CRS model, and this would also be a surprising result. Similar considerations can be made for (e) and (k).

Acknowledgements We would like to thank Yuval Ishai and the anonymous reviewers for many valuable comments.

References

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. 1994.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[DG03] Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *STOC*, pages 426–437, 2003.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *STOC*, pages 44–61, 1989.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[KL07] Dafna Kidron and Yehuda Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. Cryptology ePrint Archive, Report 2007/478, 2007.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC*, pages 179–188, 2009.

[MPR09] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. A zero-one law for deterministic 2-party secure computation. In *Manuscript*, 2009.

[OPV08] Rafail Ostrovsky, Giuseppe Persiano, and Ivan Visconti. Constant-round concurrent non-malleable zero knowledge in the bare public-key model. In *ICALP (2)*, pages 548–559, 2008.

[PR08] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multiparty computation problems: Classifications and separations. In *CRYPTO*, pages 262–279, 2008.

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In *TCC*, pages 1–20, 2004.

A Proof of Thm. 1

The theorem follows directly from the following five lemmas.

Let Game 0 denote the reprove game in Fig. 4 and let $\text{SUCCESS}_0(\mathcal{A})$ denote the probability that $\mathcal{A}$ wins it. Let Game 1 be the reprove game in Fig. 4, but where we restrict the adversary $\mathcal{A}'$ to use each of the commands **start verifier** and **reply verifier** at most once. Let $\text{SUCCESS}_1(\mathcal{A}')$ denote the probability that such $\mathcal{A}'$ wins this alternative reprove game.

Lemma 3. *For all PPT adversaries $\mathcal{A}$ for Game 0 there exists a PPT adversary $\mathcal{A}'$ for Game 1 such that $|\text{SUCCESS}_1(\mathcal{A}') - \text{SUCCESS}_0(\mathcal{A})/p(\kappa)|$ is negligible, for a polynomial $p(\kappa)$.*

Proof: Let $p(\kappa)$ be a polynomial upper bound on the number of verifiers started by $\mathcal{A}$. The adversary $\mathcal{A}'$ samples a uniformly random $v \in [p(\kappa)]$. It then runs as $\mathcal{A}$, except that when $\mathcal{A}$ outputs $(\texttt{verify}, a)$, then $\mathcal{A}'$ samples a uniformly random $e \in_R E$, stores (a, e) and returns e to $\mathcal{A}$. When $\mathcal{A}$ outputs $(\texttt{reply}, a, e, z)$ and some (a, e) is stored, delete (a, e) and store (a, e, z). All verifiers are handled like this one, except that when $\mathcal{A}$ outputs $(\texttt{verify}, a)$ for the v'th time, then $\mathcal{A}'$ outputs $(\texttt{verify}, a)$ too, and when $\mathcal{A}$ outputs $(\texttt{reply}, a, e, z)$, then $\mathcal{A}$ outputs $(\texttt{reply}, z)$. It is clear that $\mathcal{A}'$ wins if $\mathcal{A}$ would have won because the v'th conversation (a, e, z) fulfilled the winning condition, which proves the lemma. □

initialize Let $I = 0$ and phase = $\texttt{proof}$. Sample $(x, w_0, w_1) \leftarrow G$ and give x to $\mathcal{A}$.

start prover Whenever $\mathcal{A}$ inputs $(\texttt{prv}, b)$ and phase = $\texttt{proof}$, let $I = I+1$, sample $a^{(I)} \leftarrow A(x, w_b, r)$, store the prover intermediary state $P^{(I)} = (I, b, r)$ and return $a^{(I)}$ to $\mathcal{A}$.

challenge prover Whenever $\mathcal{A}$ inputs $(\texttt{chl}, I, e^{(I)})$ and phase = $\texttt{proof}$ and some $P^{(I)} = (I, b, r)$ is stored, delete $P^{(I)}$, compute $z = Z(x, w_b, e^{(I)}, r)$ and return $z^{(I)}$ to $\mathcal{A}$.

start verifier On input $(\texttt{verify}, a)$ from $\mathcal{A}$ when phase = $\texttt{proof}$, set phase = $\texttt{verify}$ and return a uniformly random $e \in_R E$ to $\mathcal{A}$.

reply verifier On input $(\texttt{reply}, z)$ from $\mathcal{A}$ when phase = $\texttt{verify}$, the game proceeds as follows: If $V(x, a, e, z) = \texttt{accept}$ for the a and e from **start verifier**, then $\mathcal{A}$ wins the game, otherwise it loses. If $\mathcal{A}$ terminates before the **reply verifier** phase it also loses.

Fig. 11. Game 2

Consider Game 2, defined as Game 1, except that $\mathcal{A}$ is not allowed to start or challenge a prover after receiving the e in the **start verifier** command—see Fig. 11 for details. Let $\text{SUCCESS}_2(\mathcal{A})$ denote the probability that $\mathcal{A}$ wins Game 2.

Lemma 4. *For all PPT adversaries $\mathcal{A}$ for Game 1 there exists a PPT adversary $\mathcal{A}'$ for Game 2 such that $|\text{SUCCESS}_2(\mathcal{A}') - \text{SUCCESS}_1(\mathcal{A})|$ is negligible.*

Proof: Let $\mathcal{A}$ be an adversary winning Game 1 with probability $\text{SUCCESS}_1(\mathcal{A})$. Since $\mathcal{A}$ (by definition of Game 1) only calls **start verifier** and **reply verifier** once, and since $\mathcal{A}$ (by definition of the reprove game) does not win if it challenges a prover between receiving its own challenge e in **start verifier** and returning its reply z in **reply verifier**, we can assume that $\mathcal{A}$ does not challenge any provers between using the **start verifier** and **reply verifier** commands. After it uses the **reply verifier** command, it is already determined if it wins or not, so we can in fact assume that $\mathcal{A}$ does not challenge any provers after using the **start verifier** commands. Now let adversary $\mathcal{A}'$ run $\mathcal{A}$, except that when phase = `verify` and $\mathcal{A}$ outputs $(\texttt{prv}, b)$, then $\mathcal{A}'$ samples $(a^*, z^*) \leftarrow S(x, e)$ for $e = 0^\ell$ and returns a^* to $\mathcal{A}$—here S is the simulator of the Σ-protocol. Since $\mathcal{A}$ never asks to challenge this prover, it is not a problem that $\mathcal{A}'$ only can reply to $e = 0^\ell$. If using simulated a values would noticeably change the probability with which $\mathcal{A}$ wins, then it can distinguish simulated a values from real ones, which would contradict honest verifier zero-knowledge. □

Consider now Game 3, defined as Game 2, except that $\mathcal{A}$ wins iff it outputs w such that $(x, w) \in R$.

Lemma 5. *For all PPT adversaries $\mathcal{A}$ for Game 2 there exists a PPT adversary $\mathcal{A}'$ for Game 3 such that* $\textsc{Success}_3(\mathcal{A}') \geq \textsc{Success}_2(\mathcal{A})^2 - \epsilon(\kappa)$ *for negligible ϵ.*

Proof: Since $\mathcal{A}$ is for Game 2, it does not start or challenge any provers after using the **start verifier** command. This means, the end of the game is of the form: $\mathcal{A}$ sends $(\texttt{verify}, a)$ to the game, gets back e and then returns $(\texttt{reply}, z)$. This allows $\mathcal{A}'$ to run and rewind $\mathcal{A}$ and extract a witness when it wins: It rewinds $\mathcal{A}$ to the point where it got e and reruns with a new uniformly random e'. If $\mathcal{A}$ wins again and $e' \neq e$, then $\mathcal{A}'$ uses $w = W(x, a, e, z, e', z')$ as the witness for x. An application of Jensen's inequality shows that if $\mathcal{A}$ wins with probability p, then $\mathcal{A}'$ wins with probability $\geq p^2 - 2^{-\ell}$, where $-2^{-\ell}$ comes from the fact that $e' = e$ with probability $2^{-\ell}$. □

Consider Game 4, defined as Game 3, except that $\mathcal{A}$ before it starts any verifiers must specify a bit $b^* \in \{0, 1\}$. Then it is only allowed to use $b = 1 - b^*$ when it starts a prover, and it only wins if it outputs w_{b^*}.

Lemma 6. *For all PPT adversaries $\mathcal{A}$ for Game 3 there exists a PPT adversary $\mathcal{A}'$ for Game 4 such that* $\textsc{Success}_4(\mathcal{A}') \geq \frac{1}{2}\textsc{Success}_3(\mathcal{A}) - \epsilon(\kappa)$ *for negligible ϵ.*

Proof: Use a uniformly random b^* and then run $\mathcal{A}$, but replace each b in $(\texttt{prv}, b)$ by $b = 1 - b^*$. If we did not replace the b, then $\mathcal{A}'$ would win with probability $\frac{1}{2}\textsc{Success}_3(\mathcal{A})$. The only difference introduced by $b = 1 - b^*$ is in which witness is used. If the winning probability of $\mathcal{A}'$ would change noticeably away from $\frac{1}{2}\textsc{Success}_3(\mathcal{A})$, then a simple reduction allows to use $\mathcal{A}'$ to break the WI of the Σ-protocol. □

Lemma 7. *It holds for all PPT verifiers $\mathcal{A}$ that* $\textsc{Success}_4(\mathcal{A})$ *is negligible.*

Proof: Given any $\mathcal{A}$ for Game 4, consider the following attack contradicting that G is a hard double-witness generator: Run $\mathcal{A}$ to get b^* and output $1 - b^*$ to get (x, w_{1-b^*}) sampled by G. Then use x and w_{1-b^*} to run $\mathcal{A}$ in Game 4. If $\mathcal{A}$ wins Game 4, then let w_b^* be the witness produced by $\mathcal{A}$, and output w_b^* to win the game against G. □

B Proof of Thm. 2

For $i = 1, \ldots, \ell + 2$, let $P_S^{(i)}$ ($P_R^{(i)}$) denote the i'th prover instantiated at S (R), and let $V_S^{(i)}$ ($V_R^{(i)}$) denote the i'th verifier instantiated at S (R).

We match verifiers at S to provers at R as follows: Define a map $\pi_S : [\ell+1] \to [\ell+1] \cup \{\top, \bot\}$. For $i \in [\ell+1]$, define $\pi_S(i)$ as follows: If there is no i'th verifier at S, then let $\pi_S(i) = \top$. Otherwise, if there exists a prover $P_R^{(j)}$ such that $P_R^{(j)}$ received its challenge e' between $V_S^{(i)}$ received its commit message a and received its reply message z, then let $\pi_S(i) = j$ for the highest such index j. Otherwise, let $\pi_S(i) = \bot$. We define a map $\pi_R : [\ell+1] \to [\ell+1] \cup \{\top, \bot\}$ completely symmetrically.

Lemma 8. *If S and R both accept, they hold the same message if the following two conditions hold:*

matching *For all $i \in [\ell+1]$ it holds that $\pi_S(i) = i$ and $\pi_R(i) = i$.*

no crossing *If $\pi_R(j) = j$ and $\pi_S(i) = i$ and $P_R^{(i)}$ ran before $V_R^{(j)}$, then $V_S^{(i)}$ ran before $P_S^{(j)}$.*

Proof: Put the $2\ell+2$ instances run by S in the Euclidean plane at coordinates $(x, y) = (I, 1)$ for $I = 1, \ldots, 2\ell+2$. Put the $2\ell+2$ instances run by R at coordinates $(x, y) = (I, 2)$ for $I = 1, \ldots, 2\ell+2$ as in Fig. B. Put a line between $V_S^{(i)}$ and $P_R^{(j)}$ if $j = \pi_S(i)$, and put a line between $V_R^{(i)}$ and $P_S^{(j)}$ if $j = \pi_R(i)$. If the **matching** property holds, then all $2\ell+2$ points with $y = 1$ have a line to one of the points with $y = 2$, and vice versa. If in addition the **no crossing** property holds, then none of the lines going from a point with $y = 1$ to a point with $y = 2$ will cross. This implies that all $2\ell+2$ lines are vertical, which implies that S and R had the same σ sequences: when one ran a prover the other ran a verifier, and vice versa. □

$y \backslash x$	1	2	3		2ℓ
2	$V_R^{(1)}$	$P_R^{(1)}$	$P_R^{(2)}$		$V_R^{(\ell)}$
	$\uparrow$	$\downarrow$	$\downarrow$	$\cdots$	$\uparrow$
1	$P_S^{(1)}$	$V_S^{(1)}$	$V_S^{(2)}$		$P_S^{(\ell)}$

Fig. 12. Picturing the conditions for Lemma 8 for a message $m = 10\ldots$.

We prove the two conditions below, in Lemma 12 and Lemma 13, going via some smaller lemmas for clarity.

Lemma 9. *It cannot happen that $\pi_S(i_1) = j = \pi_S(i_2)$ for $i_1 \neq i_2$, and it cannot happen that $\pi_R(i_1) = j = \pi_R(i_2)$ for $i_1 \neq i_2$.*

Proof: We prove the first claim. The second follows by symmetry. Assume that $\pi_S(i_1) = j = \pi_S(i_2)$. Since $V_S^{(i_1)}$ and $V_S^{(i_2)}$ were running at the same party, they did not run at the same time. Say $V_S^{(i_2)}$ ran after $V_S^{(i_1)}$. By definition of π_S, $P_R^{(j)}$ was challenged after $V_S^{(i_2)}$ received its commit message. Since $V_S^{(i_2)}$ ran after $V_S(i_1)$, it follows that $P_R^{(j)}$ was challenged *after* $V_S^{(i_1)}$ was given its reply z. So, $\pi_S(i_1) \neq j$, a contradiction. $\square$

Lemma 10. *If S accepts, then $\pi_S(i) \neq \top$ for $i = 1, \ldots, \ell + 1$, and if R accepts, then $\pi_R(i) \neq \top$ for $i = 1, \ldots, \ell + 1$.*

Proof: If S does not see ℓ accepting verifiers, then it does not output `accept`, and if R does not see ℓ accepting verifiers, then it does not output `accept`. $\square$

Lemma 11. *If S accepts, then $\pi_S(i) \neq \bot$ for $i = 1, \ldots, \ell+1$ except with negligible probability, and if R accepts, then $\pi_R(i) \neq \bot$ for $i = 1, \ldots, \ell + 1$ except with negligible probability.*

Proof: If $\pi_S(i) = \bot$, then between $V_S^{(i)}$ received its commit message a and it received the reply z, there was no prover which was challenged at R. Since S runs only one instance at a time it follows that no prover was challenged between $V_S^{(i)}$ received its commit message a and $V_S^{(i)}$ received the reply z. So, if $\pi_S(i) = \bot$ and $V_S^{(i)}$ accepts, then the adversary broke Thm. 1. It follows that when $\pi_S(i) = \bot$, then $V_S^{(i)}$ accepts with negligible probability. The proof for R is symmetric. $\square$

Lemma 12 (Matching). *If S accepts, then $\pi_S(i) = i$ for $i = 1, \ldots, \ell + 1$, and if R accepts, then $\pi_R(i) = i$ for $i = 1, \ldots, \ell + 1$. In both cases this holds except with negligible probability.*

Proof: We do the proof for S. The proof for R follows by symmetry. If S accepts, then by Lemma 10 and Lemma 11 $\pi_S(i) \in [\ell + 1]$ for $i = 1, \ldots, \ell + 1$. By construction, $\pi_S(i_1) \geq \pi_S(i_2)$ if $i_1 > i_2$. By Lemma 9 $\pi_S(i_1) \neq \pi_S(i_2)$ if $i_1 > i_2$. So, $\pi_S(i_1) > \pi_S(i_2)$ if $i_1 > i_2$. From $\pi_S(i) \in [\ell + 1]$ and $\pi_S(i_1) > \pi_S(i_2)$ if $i_1 > i_2$ it follows that $\pi_S(i) = i$. $\square$

Lemma 13 (No crossing). *If $\pi_R(j) = j$ and $\pi_S(i) = i$ and $P_R^{(i)}$ ran before $V_R^{(j)}$, then $V_S^{(i)}$ ran before $P_S^{(j)}$.*

Proof: Let $e_R^{(i)}$ be the challenge received by $P_R^{(i)}$ and $a_R^{(j)}$ the commit message received by $V_R^{(j)}$. Given that $V_R^{(j)}$ ran after $P_R^{(j)}$, $a_R^{(j)}$ was received after $e_R^{(i)}$ was received. Given $\pi_S(i) = i$, we know that $V_S^{(i)}$ received its commit message $a_S^{(i)}$ before $P_R^{(i)}$ received its challenge $e_R^{(i)}$. Now let $e_S^{(j)}$ be the challenge received by $P_S^{(j)}$. As $\pi_R(j) = j$, $e_S^{(j)}$ was received after $a_R^{(j)}$ was received by $V_R^{(j)}$. Then it follows that $a_S^{(i)}$ was received by $V_S^{(i)}$ before $e_S^{(j)}$ was received by $P_S^{(j)}$, and given that S runs $P_S^{(j)}$ and $V_S^{(i)}$ sequentially, it follows that $V_S^{(i)}$ ran before $P_S^{(j)}$. $\square$

Combining Lemma 12, Lemma 13 and Lemma 8 we know that the R does not accept a wrong message if both S and R accept. It is therefore sufficient to argue that if S accepts, then R accept, except with negligible probability.

Lemma 14. *If S accepts, then except with negligible probability R accepts.*

Proof: If S accepts, then $\pi_S(i) = i$ for $i = 1, \dots, \ell + 1$, (except with negligible probability) This means that R ran $\ell + 1$ provers. Since $\sigma_{2\ell+2} = \mathtt{R}$ and σ contains exactly $\ell + 1$ positions with $\sigma_I = \mathtt{R}$, it follows that R reached $I = 2\ell + 2$ and that at this iteration it ran a prover, called $P_R^{(\ell+2)}$. By definition of $\pi_S(i)$ it follows that $P_R^{(\ell+2)}$ was challenged before $V_S^{(\ell+2)}$ received its reply message. Given that S accepts, $V_S^{(\ell+2)}$ did receive a reply message, and therefore $P_R^{(\ell+2)}$ did receive its challenge, making $P_R^{(\ell+2)}$ accept and therefore making R output `accept`. □

C Proofs

Proof of Thm. 3: The simulator $\mathcal{S}$ simply runs the protocol honestly with $\mathcal{A}$: First it runs $\mathcal{F}^G_{\text{KRA}}$ with $\mathcal{A}$ and distributes the keys as specified. It records the secrets s_j of all parties. Then, when it receives (i, j, m) from $\mathcal{F}^G_{\text{MAU}}$, it runs P_i from π^G_{MAU} with input (j, m). If an honest P_j in the simulated run of π^G_{MAU} outputs (i, m') (because it received (i, m') with $\text{ver}_{vk'_i}(i\|j\|m', S) = \mathtt{accept}$), then $\mathcal{S}$ inputs (i, j, m') to $\mathcal{F}^G_{\text{MAU}}$. This gives a perfect simulation unless $\mathcal{S}$ at some point inputs (i, j, m') to $\mathcal{F}^G_{\text{MAU}}$ without party i on $\mathcal{F}^G_{\text{MAU}}$ having received input (j, m').

Since inputting (j, m') to party i on $\mathcal{F}^G_{\text{MAU}}$ is the only way for $\mathcal{S}$ to receive (i, j, m') from $\mathcal{F}^G_{\text{MAU}}$, which is in turn the only way $\mathcal{S}$ will send $S = \text{sig}_{sk_i}(i\|j\|m')$, it follows that the simulation is perfect unless it happens that an honest P_j receives (i, j, m', S) with $\text{ver}_{vk'_i}(i\|j\|m', S) = \mathtt{accept}$ without $\mathcal{S}$ having sent $S = \text{sig}_{sk_i}(i\|j\|m')$, we can focus on proving that this happens with negligible probability.

Let $\mathcal{A}$ (and $\mathcal{Z}$) be any PPT adversary (and environment) which makes an honest P_j receive (i, j, m', S) with $\text{ver}_{vk'_i}(i\|j\|m', S) = \mathtt{accept}$ for an honest P_i without $\mathcal{S}$ having sent $S = \text{sig}_{sk_i}(i\|j\|m')$. Say this happens with probability p. We show that p is negligible.

Suppose $\mathcal{A}$ breaks the protocol by guessing i and j uniformly at random before running the attack. Then we can turn $\mathcal{A}$ into an adversary $\mathcal{A}'$ which first fixes i and j and then makes P_j output (i, m) without giving the input (j, m) to P_i. It will succeed with probability at least $p' = p/n^2$.

We then change the protocol as follows: For each P_k with $k \neq i$ and $k \neq j$ we make the following change: In **key authentication** we let P_i and P_k run the protocol $\pi_{i,k} = \pi_{\mathsf{au}}((pk_i, pk_k), s_i, s_k)$ (or $\pi_{k,i}$ if $k < i$) with the cheat that in all proofs from P_i to P_k we let P_i use the witness s_k, and we let P_j and P_k run the protocol $\pi_{j,k} = \pi_{\mathsf{au}}((pk_j, pk_k), s_j, s_k)$ (or $\pi_{k,j}$ if $k < j$) with the cheat that in all proofs from P_j to P_k we let P_j use the witness s_k. By WI, this would not change the probability that $\mathcal{A}'$ breaks the protocol, except negligibly, even if $\mathcal{A}'$ knew all witnesses. In particular, this first hybrid is indistinguishable from the original protocol also for corrupted P_k, as long as P_i and P_j are honest.

Consider then the following attack on π_{au}. First we set up pk_S and pk_R: Specifically we let G' (the key generator for π_{au}) depend on G (the key generator for π^G_{MAU}) with $(pk_S, s_S) = (pk_i, s_i)$, $(pk_R, s_R) = (pk_j, s_j)$. This is clearly a hard generator when G is a hard generator. We run the attack against the protocol in

π_{au} using this G'. We let $\mathcal{A}$ interact with G' as it would interact with G. We get back pk_i and pk_j and $(pk_k, s_k)_{k \notin \{i,j\}}$ —we can simply discard these last keys. Then we run the protocol π_{au} with $\mathcal{A}$, except that we input $m_S = (vk_i, vk'_j)$ to S and $m_R = (vk'_i, vk_j)$ to R in the game against π_{au}. And in the game against π_{au} we send the message to S that $\mathcal{A}$ sends to P_i and we send the message to R that $\mathcal{A}$ sends to P_j, and the messages sent by S we show to $\mathcal{A}$ as if coming from P_i and the messages sent by R we show to $\mathcal{A}$ as if coming from P_j. It follows by security of π_{au} that if R accepts then $m_S = m_R$. I.e., if P_j accepts, then we can assume that $(vk_i, vk'_j) = (vk'_i, vk_j)$, except with negligible probability. Since P_j accepts a message from P_i only if it accepts and stores vk'_i, we can conclude that P_j accepts and that therefore $vk'_i = vk_i$, except with negligible probability. Yet we know that $\mathcal{A}$ with probability p makes P_j output $(\texttt{accept}, m')$ without giving input (j, m) to P_i. This means that $\mathcal{A}$ sends S a message (i, j, m', S) such that $\mathrm{ver}_{vk_i}(i \| j \| m', S) = \texttt{accept}$ without P_i ever signing $i \| j \| m'$ under sk_i. This can trivially be turned into an attack on the enforceability of the signature scheme under chosen message attack, with success probability negligibly close to p/n^2. This shows that p/n^2 is negligible, and hence p is negligible.

Given that π_{au} is completely symmetric i.e. R accepts iff S accepts, and in this case they both output $(\texttt{accept}, m)$, one can repeat the previous argument by exchanging P_i with P_j to see that the probability that P_i outputs (j, m) when P_j didn't get (i, m) as input is negligible. $\square$

Proof of Thm. 4: The simulator $\mathcal{S}$ will extract commitments by inputting $c = 1$ to learn m and input m to $\mathcal{F}_{\text{MCOM}}$ on behalf of the corrupted sender. It gives the proofs by using the secret of the corrupted sender as witness. The simulator $\mathcal{S}$ trapdoor opens the commitment to some m' simply by sending m'. Then it gives the proof using the secret of the corrupted receiver as witness. For a proof from an honest party to an honest party it uses no witness: It generates a simulated conversation (a, e, z) and then lets the party playing verifier send the challenge e. This is done to ensure that $\mathcal{S}$ never uses s_i for an honest P_i.

To show that this simulation works it is sufficient to show that it generates a view indistinguishable from that of the real protocol.

The indistinguishability follows from WI of the OR-construction and the security of the OT: Starting from the real protocol we can first consider the hybrid where $c = 0$ is still used but the proofs given by the receiver are produced using the secret of the corrupted sender as witness. By WI of the proofs this will not change the output of the environment, except negligibly, or the environment could distinguish between proofs generated with different witnesses. Then we go to a hybrid where the honest receiver uses $c = 1$. By security of the OT, this will not change the output of the environment, except negligibly. This hybrid is now the simulation, except that honest parties output the message m' to which a commitment is opened, whereas in the simulation $\mathcal{F}_{\text{MCOM}}$ outputs the message m which $\mathcal{S}$ receives from the OT. It is therefore sufficient to argue that these two messages are the same except with negligible probability. If not, we can run this hybrid and find a commitment where $m' \neq m$. Note that in running it, we do not need to use the s_i of an honest party. We then use rewinding on all the proofs given by the sender and extract a witness. Except with negligible probability the extracted witnesses consist of openings of V, M_0 and M_1 to v, m_0 and m_1 such that $V = \mathrm{commit}(v)$, $M_0 = \mathrm{commit}(m_0)$ and $M_1 = \mathrm{commit}(m_1)$ and $m_0 = m'$ and the view of the receiver in the OT protocol is consistent with the randomness v and inputs m_0 and m_1; if this is not the extracted witness, then the extracted witness contains s_i' such that $(pk_i, s_i') \in R$. Since s_i is never used to produce the view of this hybrid distribution, i.e. the hybrid has been executed, rewound and rerun without using s_i', it follows that the witness contains s_i' with negligible probability, as G is a hard key generator. Therefore the OT protocol is consistent with an execution with input $m_0 = m'$. By correctness of the OT it therefore follows that $m = m'$, as m is the output of the OT and the selection bit was $c = 0$. □

Proof of Thm. 6: The simulator $\mathcal{S}$ simulates $\mathcal{F}_{\text{CRS}}^{D}$ by sampling $crs = D(s)$ and giving crs to $\mathcal{A}$. From $\mathcal{F}_{\text{KRA}}^{G^D}$ it receives pk_i for $i \in H$, where $H = [n] \setminus C$ and C are the indices of the corrupted parties. Then for $i = 1, \ldots, n$ and $j = 1, \ldots, n$, $j \neq i$ it simulates the proofs. If P_i and P_j are both corrupted there is nothing to simulate. If P_i and P_j are both honest $\mathcal{S}$ uses $m_0 = m_1 = 0^\kappa$ (where $D : \{0,1\}^\kappa \to \{0,1\}^\ell$) and $c = 0$. It simulates all proofs by sampling a simulated conversation (a, e, z) using the honest verifier simulator and letting P_i send a, P_j send e and P_i send z. When P_i is corrupted and P_j is honest, then the simulator inputs $c = 1$ to the SA-OT and learns m_1. If the proof succeeds and $pk_i = D(m_1)$, then $\mathcal{S}$ inputs m_1 to $\mathcal{F}_{\text{KRA}}^{G^D}$ on behalf of P_i, making pk_i the output of P_i. If the proof succeeds and $pk_i \neq D(m_1)$, then $\mathcal{S}$ terminates. If the proof fails, then the protocol π_{KRA}^{D} will not give any outputs, so $\mathcal{S}$ simply does not deliver the outputs of $\mathcal{F}_{\text{KRA}}^{G^D}$ either. To allow to run with input $c = 1$ it gives the proofs using s as witness, which is possible as $crs = D(s)$. If P_i is honest and P_j is corrupted, then $\mathcal{S}$ received pk_i from $\mathcal{F}_{\text{KRA}}^{G^D}$. In the SA-OT it takes $m_0 = 0^\kappa$ and $m_1 = s$ and gives the proof using s as witness.

That the simulation is indistinguishable from the real protocol follows from the computational hiding of the commitment scheme used in proving the execution of the SA-OT consistent, the special honest verifier zero-knowledge of the Σ-protocol (for the honest/honest case), the computational hiding of the OT and WI of the OR-construction. What remains is to show that there is a negligible probability that a corrupted P_i gives acceptable proofs to $\mathcal{S}$ and yet $pk_i \neq D(m_1)$.

Consider a pair P_i, P_j where P_i is corrupt, P_j honest and where the proofs of P_i are acceptable and $pk_i \neq D(m_1)$. Here P_i is controlled by the environment and the adversary. We can take the entire execution of the simulation, including the environment, the adversary and $\mathcal{S}$ and rewind all proofs given by P_i to $\mathcal{S}$, and extract a witness for the proof. The witness will include m_1 such that $pk_i = D(m_1)$ or $crs = D(m_1)$. Since $pk_i \neq D(m_1)$ it follows that $crs = D(m_1)$.

Since $\mathcal{S}$ is itself using s such that $crs = D(s)$ in *its* proofs, it is not necessarily a contradiction with the one-wayness of D that we can extract m_1 such that $crs = D(m_1)$ from P_i. We can, however, define an hybrid where $\mathcal{S}$ does not use s: For each pk_k for an honest P_k, we inspect $\mathcal{F}_{\text{KRA}}^{G^D}$ to find s_k such that $pk_k = D(s_k)$.[7] Then we run all honest parties correctly, using the witness s_k in the proofs. By WI of the OR-construction, this will only negligibly change the probability that the extraction of $\mathcal{A}$ gives m_1 such that $crs = D(m_1)$. Now that s is not used anymore in the execution, we set crs to be any given value. But then the expected poly-time process of running the execution and then extracting P_i can take crs as input and with non-negligible probability output m_1 such that $crs = D(m_1)$, which contradicts that D is a OWF.

In proving that when $\mathcal{S}$ changes witness, then the witness extracted from the proof of P_i does not change, follows the proof of Thm. 1. For the proof to go through it is important that while P_i is giving its proof, the simulator will not have to simulate any proof of honest parties: by construction of the protocol only P_i and P_j are giving proofs at this phase, and either P_i acts as prover or P_j acts as prover, and never at the same time. □

Proof of Lemma 1: Let $\mathcal{A}'$ be the adversary which runs exactly as $\mathcal{A}$, except that it inputs $s_S = (c, r_S^c, pk_S^{1-c})$ to $\mathcal{F}_{\text{KRA}}^{G^f}$. Since $f(s'_S) = f(s_S)$, the view of the honest receiver R does not change, so it is enough to prove the claim for the case where R runs with $\mathcal{A}'$. For this purpose, let $\mathcal{A}''$ be the adversary which runs exactly as $\mathcal{A}'$, except that it samples $pk_S^{1-c} \in \{0,1\}^{\kappa+1}$ uniformly at random. Since $\mathcal{A}''$ corresponds to the honest sender, it follows that R accepts the commitment when running with $\mathcal{A}''$, and that R later accept the opening to m. Consequently, the honest R will also accept the commitment when running with $\mathcal{A}'$, and will later accepts the opening to m, or R could be used to distinguish $pk_S^{1-c} = g(r_S^{1-c})$ with $r_S^{1-c} \in_R \{0,1\}^\kappa$ from a uniformly random $pk_S^{1-c} \in_R \{0,1\}^{\kappa+1}$. □

Proof of Lemma 2: Note first that we can make B sample pk_S^{1-c} as $pk_S^{1-c} = g(r_S^{1-c})$ also when $b = 0$. By g being a PRG, this will not change whether the views of A when $b = 0$ and $b = 1$ are computationally indistinguishable. Note then that even if we give r_S^{1-c} to A (both when $b = 0$ and $b = 1$) we can show that the views when $b = 0$ and $b = 1$ are computationally indistinguishable: then it clearly holds also when A is not given r_S^{1-c}. But when A is given r_S^{1-c} we can let A sample c and r_S^{1-c} and send pk_S^{1-c} to B, instead of the other way around. This is not going to change the view of A, only the direction of a message. But now, when $b = 0$, then A runs $\mathcal{A}$ and B runs $\mathcal{S}$. Furthermore, when $b = 1$, then A still runs $\mathcal{A}$, but B runs the honest R. By UC security, $\mathcal{A}$ cannot distinguish whether it interacts with $\mathcal{S}$ or R.

Note that when $b = 0$, we can change the protocol such that A samples $pk_S^{1-c} \in_R \{0,1\}^{\kappa+1}$ uniformly at random and sends it to B. This is not going to change the view of B, only the direction of a message. Hence it is sufficient to prove the claim for this protocol. We can then change the protocol such that A

[7] It should be noted that $\mathcal{S}$, of course, cannot inspect $\mathcal{F}_{\text{KRA}}^{G^D}$ in a UC simulation. We are, however, not in a UC simulation in the above argument. We are showing how the entire execution, including the adversary and the environment can be rewound to compute a secret s for a given crs.

instead samples $r_S^{1-c} \in_R \{0,1\}^\kappa$ uniformly at random and sends $pk_S^{1-c} = g(r_S^{1-c})$. By g being a PRG, this does not change whether the views of B when $a = 0$ and $a = 1$ are computationally indistinguishable. But now A is running the honest sender S from π and B is running the honest receiver R from π. So by π being computationally hiding it follows that the views of B are computationally indistinguishable when $b = 0$ and $b = 1$. □