Improved UC Secure Computation using Tamper-proof Hardware

Nishanth Chandran Vipul Goyal Amit Sahai

Department of Computer Science, UCLA
{nishanth,vipul,sahai}@cs.ucla.edu

Abstract

The Universal Composability framework was introduced by Canetti to study the security of protocols which are concurrently executed with other protocols in a network environment. Unfortunately it was shown that in the so called plain model, a large class of functionalities cannot be securely realized. These severe impossibility results motivated the study of other models involving some sort of setup assumptions, where general positive results can be obtained. Until recently, all the setup assumptions which were proposed required some trusted third party (or parties).

Katz recently proposed using a *physical setup* to avoid such trusted setup assumptions. In his model, the physical setup phase includes the parties exchanging temper proof hardware tokens implementing some functionality. The temper proof hardware is modeled so as to assume that the receiver of the token can do nothing more than observe its input/output characteristics. It is further assumed that the sender *knows* the program code of the hardware token which it distributed. Based on the DDH assumption, Katz gave general positive results for universally composable multi-party computation tolerating any number of dishonest parties making this model quite attractive.

In this paper, we improve the results of Katz in several directions using completely different techniques. Interestingly, our security proofs do not rely on being able to rewind the hardware tokens created by malicious parties. This means that we are able to relax the assumptions that the parties *know* the code of the hardware token which they distributed. This allows us to model real life attacks where, for example, a party may simply pass on the token obtained from one party to the other without actually knowing its functionality. Furthermore, our construction models the interaction with the tamper-resistant hardware as a simple request-reply protocol. Thus, we show that the hardware tokens used in our construction can be *resettable*. In fact, it suffices to use token which are completely stateless (and thus cannot execute a multi-round protocol). Our protocol is also based on general assumptions (namely enhanced trapdoor permutations).

1 Introduction

The universal composability (UC) framework of security, introduced by Canetti [Can01], provides a model for security when protocols are executed multiple times in a network where other protocols may also be simultaneously executed. Canetti showed that any polynomial time computable multi-party functionality can be realized in this setting when a strict majority

of the players are honest. Canetti and Fischlin [CF01] then showed that without an honest majority of players, there exists functionalities that cannot be securely realized in this framework. Canetti, Kushilevitz and Lindell [CKL06] later characterized the two-party functionalities that cannot be securely realized in the UC model ruling out almost all non-trivial functions. These impossibility results are in a model without any setup assumptions (referred to as the "plain" model). These results can be bypassed if one assumes a setup in the network. Canetti and Fischilin suggest the use of common reference string (CRS) and this turns out to be a sufficient condition for UC-secure multi-party computation for any polynomial time functionality, for any number of dishonest parties [CLOS02]. Some other "setup" assumptions suggested have been trusted "public-key registration services" [BCNP04, CDPW07], government issued signature cards [HMQU05] and so on.

UC Secure Computation based on Temper Proof Hardware. Recently, Katz [Kat07] introduced the model of tamper resistant hardware as a setup assumption for universally composable multi-party computation. An important attraction of this model is that it eliminates the need to trust a party, and instead relies on a physical assumption. In this model, a party P creates a hardware token implementing a functionality and sends this token to party P'. Given this token, P' can do nothing more than observe the input/output characteristics of the functionality. Based on the DDH assumption, Katz gave general feasibility results for universally composable multi-party computation tolerating any number of dishonest parties.

Our Contributions. In this paper, we improve the results of Katz in several directions using completely different techniques. Our results can be summarized as follows:

- Knowing the Code: A central assumption made by Katz [Kat07] is that all parties (including the malicious ones) *know* the program code of the hardware token which they distributed. This assumption is precisely the source of extra power which the simulator gets in the security proofs [Kat07]. The simulator gets the power of *rewinding the hardware token* which is vital for the security proofs to go through. However we argue that this assumption might be undesirable in practice. For example, it does not capture real life adversaries who may simply pass on hardware tokens obtained from one party to another. As noted by Katz [Kat07], such attacks may potentially be prevented by making the creator of a token easily identifiable (e.g., the token could output the identity of the creator on certain fixed input). However, we note that a non-sophisticated fix of this type might to susceptible to attacks where a malicious party builds a *wrapper* around the received token to create a new token and passes it on to other parties. Such a wrapper would use the token inside it in a black-box way while trying to answer the user queries. Secondly, one can imagine more sophisticated attacks where tokens of one type received as part of one protocols may be used as tokens of some other type in other protocols. Thus, while it may be possible to design constructions based on this assumption, it seems like significant additional analysis might be needed to show that this assumption holds.

  We relax this assumption in this work. In other words, we make no assumptions on how malicious parties create the hardware token which they distribute.

- Resettability of the token: The security of the construction in [Kat07] also relies on the

ability of the tamper-resistant hardware to maintain state (even when, for example, the power supply is cut off)[1]. In particular, the parties need to execute a two-round interactive protocol with the tamper-resistant hardware. It is explicitly assumed that the hardware cannot be *reset* [CGGM00]. In contrast, our construction models the interaction with the tamper-resistant hardware as a simple one round request-reply protocol. Thus, we are able to show that the hardware tokens used in our construction can be *resettable*. In fact, it suffices to use token which are completely stateless (and thus cannot even execute a multi-round protocol). We argue that relaxing this assumption about the capability of the temper resistant tokens is desirable and may bring down their cost considerably.

- Cryptographic Assumptions: An open problem left in [Kat07] was to construct a protocol in this model which is based on *general assumptions*. We settle this problem by presenting a construction which is based on enhanced trapdoor permutations previously used in CLOS [CLOS02] and other works.

Our model also has an interesting technical difference from the one in [Kat07]. In [Kat07], it is assumed that once P creates a hardware token and hands it over to P', then P cannot send any messages to the token (but can receives messages from it). We require the opposite assumption; once the token has been handed to P', it cannot send any messages to P (but can potentially receive messages from it). It is easy to see that if the communication is allowed in both directions, then this model degenerates to the so called plain model which is the subject of severe impossibility results [CF01, CKL06].

Concurrent Independent Work. Independent of our work, Damgard et al [DNW07] proposed a new construction for UC secure computation in the temper proof hardware model. The main thrust of their work seems to obtain a scheme where the hardware tokens only need to be *partially isolated*. In other words, there exists a pre-defined threshold on the number of bits that the token can exchange with the outside world (potentially in both directions). Their construction is also based on general assumptions (albeit their assumptions are still stronger than ours).

Damgard et al [DNW07] however do not solve the main problems addressed by this work. In particular, their work is in the same *rewinding based simulator* paradigm as Katz [Kat07] and thus requires the same assumption that the sender is aware of the program code of the hardware token which it distributed. Furthermore, the security of their construction relies upon the assumption that the hardware token is able to keep state (i.e., is not resettable).

2 Our Model

Our model is a modification of the model in [Kat07]. The central modifications we need are to allow for adversaries who may supply hardware tokens to other parties without knowing the

[1] As Katz [Kat07] noted, this assumption can be relaxed if the token has an inbuild source of randomness and thus messages sent by the token in the protocol are different in different execution (even if the other party is sending the same messages). Note that a true randomness source is needed to relax this assumption and cryptographic techniques such as pseudo random functions do not suffice.

code of the functionality implemented by the hardware token. To model adversaries who give out tokens without actually "knowing" the code of the functionality of the tokens, we consider an ideal functionality $\mathcal{F}_{Adv}$ that models the adversarial procedure used to create these tokens. The ideal functionality $\mathcal{F}_{wrap}$ implements the tamper-resistant hardware as in [Kat07].

We first formally define the $\mathcal{F}_{wrap}$ functionality which is a modification of the $\mathcal{F}_{wrap}$ functionality of [Kat07]. This functionality formalizes the intuition that an honest user can create a hardware token T_F implementing any polynomial time functionality, but an adversary given the token T_F can do no more than observe its input/output characteristics. $\mathcal{F}_{wrap}$ models the hardware token encapsulating a functionality M. The only changes from [Kat07] we make is that M is now a Turing machine (instead of a 2-round interactive Turing machine) and does not require any externally supplied randomness. $\mathcal{F}_{wrap}$ models the following sequence of events: (1) a party (also known as *creator*) takes software implementing a particular functionality M and seals this software into a tamper-resistant hardware token, (2) The creator then gives this token to another party (also known as the *receiver*) who can use the hardware token as a black-box implementing M. Figure 1 shows the formal description of $\mathcal{F}_{wrap}$ based on an algorithm M (modified from [Kat07]). Note that M could make black box calls to other tokens (to model the tokens created by an adversarial party).

$\mathcal{F}_{wrap}$ is parameterized by a polynomial p and an implicit security parameter k. There are 2 main procedures:

Creation. Upon receiving $(create, sid, P, P', M)$ from P or from $\mathcal{F}_{Adv}$, where P' is another user in the system and M is a Turing machine, do:

1. Send $(create, sid, P, P')$ to P'.
2. If there is no tuple of the form $(P, P', \star)$ stored, then store (P, P', M).

Execution. Upon receiving (run, sid, P, msg) from P', find the unique stored tuple (P, P', M) (if no such tuple exists, then do nothing). Run $M(msg)$ for at most $p(k)$ steps and let *out* be the response (set $out = \perp$ if M does not respond in the allotted time). Send (sid, P, out) to P'.

Figure 1: The $\mathcal{F}_{wrap}$ functionality

We now formally describe the Ideal/Real world for multi-party computation in the tamper-proof hardware model. Let there be n parties $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ (P_i holding input x_i) who wish to compute a function $f(x_1, x_2, \cdots, x_n)$. Let the adversarial parties be denoted by $\mathcal{M} \subset \mathcal{P}$ and the honest parties be denoted by $\mathcal{H} = \mathcal{P} - \mathcal{M}$. We consider only static adversaries. As noted before, to model adversaries who give out tokens without actually "knowing" the code of the functionality of the tokens, we consider an ideal functionality $\mathcal{F}_{Adv}$ that models the adversarial procedure used to create these tokens. $\mathcal{F}$ is the ideal functionality that computes the function f that the parties $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ wish to compute, while $\mathcal{F}_{wrap}$ (as discussed earlier) models the tamper-resistant device.

Real World. Our real world is the $(\mathcal{F}_{Adv}, \mathcal{F}_{wrap})$-hybrid world. In the real world, during

the setup for multi-party computation, a party has to give out a hardware token to every other party. This is modeled as follows. The adversarial parties send arbitrary messages to $\mathcal{F}_{Adv}$ functionality before protocol execution ($\mathcal{F}_{Adv}$ could use this information for the code creation of the adversarial tokens). At the end of this interaction, for each adversarial party $P_i \in \mathcal{M}$, $\mathcal{F}_{Adv}$ sends a list of $n-1$ program codes (corresponding to tokens that are to be given to the other $n-1$ parties) to $\mathcal{F}_{wrap}$. These program codes can make black calls to tokens of other (possibly honest) parties. Each honest party sends a list of $n-1$ program codes directly to $\mathcal{F}_{wrap}$. During protocol execution, all queries made to tamper-resistant hardware tokens are made to the $\mathcal{F}_{wrap}$ functionality. The parties execute the protocol and compute the function $f(x_1, x_2, \cdots, x_n)$.

IDEAL WORLD. The ideal world is the $(\mathcal{F}_{Adv}, \mathcal{F}_{wrap}, \mathcal{F})$-hybrid world. The simulator S simulates the view of the adversarial parties. As in the real world, the adversarial parties initially sends arbitrary messages to $\mathcal{F}_{Adv}$. For each adversarial party $P_i \in \mathcal{M}$, $\mathcal{F}_{Adv}$ sends a list of $n-1$ program codes (corresponding to tokens that are to be given to the other $n-1$ parties) to $\mathcal{F}_{wrap}$. These program codes can make black calls to tokens of other parties. The simulator S generates the program code for all tokens that are to be created by honest parties (S does this honestly according to the protocol specifications for creating the program code). S sends these program codes to $\mathcal{F}_{wrap}$. When an adversarial party queries a token created by another adversarial party, the simulator S forwards the query to $\mathcal{F}_{wrap}$ and then upon receiving the response from $\mathcal{F}_{wrap}$, forwards it to the querying party. When an adversarial party queries a token created by an honest party, the simulator S replies with the response to the querying party on its own. Honest parties send their inputs to the trusted functionality $\mathcal{F}$. Simulator extracts inputs from adversarial parties and sends them to $\mathcal{F}$. The ideal functionality $\mathcal{F}$ returns the output to all honest parties and to the simulator S who then uses it to complete the simulation for the malicious parties.

3 Preliminaries

As in [Kat07], we will show how to securely realize the multiple commitment functionality $\mathcal{F}_{mcom}$ in the $(\mathcal{F}_{Adv}, \mathcal{F}_{wrap})-$ hybrid model for static adversaries. This will imply the feasibility of UC-secure multi-party computation for any well formed functionality ([CF01, CLOS02]). The primitives we need for the construction of the commitment functionality are non-interactive perfectly binding commitments, a secure signature scheme, pseudorandom function and a zero-knowledge proof of knowledge protocol (that are all implied by one-way permutations [GL89, NY89, HILL99, Gol01, Gol04]).

Non-interactive perfectly binding bit commitment. We denote the non-interactive perfectly binding commitment to a string or bit a (from [GL89]) by $\mathsf{Com}(a)$. $\mathsf{Open}(\mathsf{Com}(a))$ denotes the opening to the commitment $\mathsf{Com}(a)$ (which includes a as well as the randomness used to create $\mathsf{Com}(a)$).

Secure signature scheme. We use a secure signature scheme (security as defined in [GMR88]) with public key secret key pair (PK, SK) that can be constructed from one-way permutations ([NY89]). By $\sigma_{PK}(m)$ we denote a signature on message m under the public key

PK. We denote the verification algorithm by $\mathsf{Verify}(PK, m, \sigma)$ that takes as input a public key PK, message m and purported signature σ on message m. It returns 1 if and only if σ is a valid signature of m under PK.

Zero knowledge proof of knowledge. Informally, a zero knowledge proof is a proof of knowledge protocol, if it has the additional property that the witness to the statement being proven can be extracted by a simulator that interacts with the prover. For completeness, a more formal description is given in Appendix A. Refer [Gol01, Gol04] for more details.

4 The Construction

We show how to securely realize the multiple commitment functionality $\mathcal{F}_{mcom}$ in the $(\mathcal{F}_{Adv}, \mathcal{F}_{wrap})-$ hybrid model for static adversaries. We will first give a construction that realizes the single commitment functionality in the $(\mathcal{F}_{Adv}, \mathcal{F}_{wrap})-$ hybrid model for static adversaries and then note how this can be extended to realize $\mathcal{F}_{mcom}$. P_1 wishes to commit to a string a (of length n bits) to P_2.

Setup phase. P_2 generates a public-key/secret-key pair (PK, SK) for a secure signature scheme, a seed s for a pseudorandom function $F_s(\cdot)$ and sends a token to P_1 encapsulating the following functionality M:

- Wait for message $I = (\mathsf{Com}(b), \mathsf{Open}(\mathsf{Com}(b)))$. Check that the opening is a valid opening to the commitment. If so, generate signature $\sigma = \sigma_{PK}(\mathsf{Com}(b))$ and output the signature. The randomness used to create these signatures is obtained from $F_s(I)$.

We note that this setup is done between all pairs of parties P_i and P_j in the protocol for multi-party computation.

Commitment phase. We denote the protocol in which P_1 commits to a string a (of length n bits) to P_2 by $\mathsf{UC\text{-}Com}(P_1, P_2, a)$. The parties perform the following steps:

1. For every commitment to a string a of length n, P_1 generates n commitments to 0 and n commitments to 1. P_1 interacts with the token sent to it by P_2 and obtains signatures on these $2n$ commitments. In order to commit to the i^{th} bit of a string a (denoted by a_i), P_1 selects a commitment to either 0 or 1 whose signature it had obtained from the device sent by P_2 (depending on what a_i is).

    - We note that P_1 cannot give the hardware token commitments to the bits of a alone and obtain the signatures on these commitments. Doing this would allow P_2's hardware token to perform a selective failure attack. In other words, P_2's hardware could be programmed to respond and output signatures only if some condition is satisfied by the input string a (e.g., all its bits are 0). Thus if P_1 still continues with the protocol, P_2 gains some non-trivial information about a. Hence, P_1 obtains signatures on n commitments to 0 and n commitments to 1 and then selects commitments (and their signatures) according to the string a. This makes sure that the interaction of P_1 with the hardware token is independent of the actual input a.

Let $B_i = \mathsf{Com}(a_i)$ and let the signature obtained by P_1 from the device on this commitment be $\sigma_i = \sigma_{PK}(B_i)$. P_1 now computes a commitment to σ_i for all $1 \leq i \leq n$ denoted by $C_i = \mathsf{Com}(\sigma_i)$.

Let $Com_i = (B_i, C_i)$. Now $A = \mathsf{COM}(a) = \{Com_1, Com_2, ..., Com_n\}$ (in other words, A is the collection of commitments to the bits of a and commitments to the obtained signatures on these commitments). P_1 sends A to P_2.

- Note here that P_1 does not send the obtained signatures directly to P_2, but instead sends a commitment to these signatures. This is because the signatures could have been maliciously generated by the hardware token created by P_2 to leak some information about a.

2. P_1 now gives a zero knowledge proof of knowledge to P_2 that, for all i, C_i is a commitment to a valid signature of B_i under P_2's public key PK and that B_i is a valid commitment to a bit. More formally, P_1 proves the following statement using a zero knowledge proof of knowledge protocol: "For all i,

   - There exists a valid opening of B_i to a bit a_i under the commitment scheme $\mathsf{Com}(\cdot)$
   - There exists a valid opening of C_i to a string σ_i under the commitment scheme $\mathsf{Com}(\cdot)$ such that $\mathsf{Verify}(PK, B_i, \sigma_i) = 1$."

Decommitment phase. The parties perform the following steps:

1. P_1 sends P_2 the string that was initially committed to. In particular, P_1 sends a to P_2.

   - Note that P_1 does not send the actual opening to the commitment. P_1 will later prove in zero knowledge that a was the string committed to in the commitment phase. This is to allow equivocation of the commitment by the simulator during protocol simulation.

2. P_2 picks a string R uniformly at random from $\{0,1\}^{p(k)}$ and executes the commitment protocol $\mathsf{UC\text{-}Com}(P_2, P_1, R)$.

   - P_1 will prove in zero knowledge that a was the string committed to in the commitment phase. Since we require straight-line simulation, the simulator would have to know in advance the challenge queries made by P_2 in this zero knowledge proof. Hence before this zero knowledge proof is given, P_2 commits to his randomness R using the UC-secure commitment protocol.

   - We note that the decommitment to R need not be equivocable and hence we avoid having to use the UC-secure decommitment protocol itself, which would have lead to circularity!

3. P_1 gives a zero knowledge proof that a is the string that was committed to in the commitment phase of the protocol. The randomness used by P_2 in this zero knowledge proof is R and along with every message sent in the zero knowledge protocol, P_2 proves in zero knowledge that the message uses randomness according to the string R.

Denote by R_i and a_i the i^{th} bits of R and a respectively. More formally, the statement P_1 proves to P_2 is "There exists randomness such that for all $i, B_i = \mathsf{Com}(a_i)$, where B_i is as sent in the commitment phase." Let the value $\mathsf{COM}(R)$ sent during $\mathsf{UC\text{-}Com}(P_2, P_1, R)$ be denoted by Z. Note that Z is of the form $\{(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)\}$ where $X_i = \mathsf{Com}(R_i)$ and Y_i is a commitment to the signature of X_i under P_1's public key. The statement P_2 proves to P_1 is "There exists string R, such that

- For all i, there exists an opening of X_i to R_i under the commitment scheme $\mathsf{Com}(\cdot)$
- R was the randomness used to compute this message."

4. P_2 accepts the decommitment if and only if the proof given by P_1 was accepted.

5 Security Proofs

5.1 Description of Simulator

In order to prove UC security of the commitment functionality, we will need to construct a straight-line simulator that extracts the committed value in the commitment phase of the protocol and that can equivocate a commitment to a given value in the decommitment phase of the protocol. Below, we describe such a simulator that runs straight-line both while extracting the committed string when interacting with a committer P_1, as well as when equivocating a commitment to a receiver P_2.

Setup phase. In this phase, the simulator S creates the program code for all the tokens to be created by honest parties (according to the honest token creation protocol) and sends a copy of the program codes to $\mathcal{F}_{wrap}$. The adversarial parties create their tokens by interacting with $\mathcal{F}_{Adv}$ as described before.

Handling token queries. Whenever an adversarial party queries a token created by another adversarial party, the simulator S forwards the request to $\mathcal{F}_{wrap}$. When simulating the view during the adversary's interaction with a token created by an honest party, S generates the response according to the request by the adversarial party and the program code of the token. For every pair of parties (P_i, P_j) such that $P_i \in \mathcal{M}$ and $P_j \in \mathcal{H}$, S creates a table T_{ij}. When a malicious party P_i queries the token of an honest party P_j, S stores the query in table T_{ij}. In other words, the simulator S builds a list of all the commitments (along with their openings) that the malicious party queries to a token created by an honest party (for getting a signature).

Extraction during Commitment phase. The simulator (acting on behalf of honest party P_2) runs protocol $\mathsf{UC\text{-}Com}(P_1, S, a)$ with the adversarial party P_1. S executes the protocol honestly as a receiver in the commitment phase. In more detail:

1. Let $A = \mathsf{COM}(a) = \{Com_1, Com_2, ..., Com_n\}$ according to the commitment protocol described earlier. P_1 sends A to S (Of course, P_1 may not follow the protocol).

2. P_1 now gives a zero knowledge proof of knowledge to S that for all i, C_i is a commitment to a valid signature of B_i under P_2's public key PK and that B_i is a valid commitment

to a bit. That is, P_1 proves the following statement using a zero knowledge proof of knowledge protocol: "For all i,

- There exists a valid opening of B_i to a bit a_i under the commitment scheme $\mathsf{Com}(\cdot)$
- There exists a valid opening of C_i to a string σ_i under the commitment scheme $\mathsf{Com}(\cdot)$ such that $\mathsf{Verify}(PK, B_i, \sigma_i) = 1$."

The simulator S accepts the commitment if it accepts the zero-knowledge proof. If the zero knowledge proof was accepted, S looks up the commitments B_i's to the bits of a in the table T_{12}. Note that T_{12} contains a list of all commitments that were queried by P_1 to the token created by honest party P_2.

By a reduction to the security of the underlying signature scheme, we prove in Lemma 1 that if the zero knowledge proof was accepted by S, then except with negligible probability, for all i, the commitment B_i was queried by P_1 to the device sent by P_2. Now, since the commitments to the bits of a were queried by P_1 to the device, the simulator S has already recorded the openings to these commitments and hence can extract a by looking up for the opening of all these commitments B_i's in the table T_{12}.

Equivocation during Decommitment phase. The simulator (on behalf of honest party P_1 interacting with malicious party P_2) is given a string a' to which it needs to decommit a commitment given earlier. The simulator proceeds as follows:

1. S sends a' to P_2.
2. P_2 picks a string R uniformly at random from $\{0,1\}^{p(k)}$ and executes the commitment protocol $\mathsf{UC\text{-}Com}(P_2, S, R)$. Again, P_2 may not execute the protocol honestly.

   By the property of extraction of commitments (shown earlier), if the commitment was accepted by S, then except with negligible probability S would have extracted R at the end of this stage in the protocol.
3. The simulator S now has to give a zero knowledge proof that a' is the string that was committed to in the commitment phase of the protocol. Now given R, all of P_2's messages in this zero knowledge proof protocol are deterministic.

   - S internally runs the simulation of the zero knowledge protocol (using the simulator S_{zk} for the underlying zero knowledge protocol) and obtains the simulated transcript of the protocol. Note that S can do this by interacting with prover S_{zk} and generating all messages of the verifier using randomness R.
   - Now S sends messages to the party P_2 according to the simulated zero knowledge protocol transcript. It sends to P_2 the messages of the prover and as response receives exactly the same verifier messages as in the simulated transcript (because P_2 uses randomness R to execute this protocol).
   - The party P_2 is forced to use the randomness R because P_2, along with every message sent in the zero knowledge protocol, has to prove in zero knowledge that the message uses randomness according to the string R. By the soundness property of this zero knowledge proof (given by P_2), if P_2 sends a message that is not according to randomness R, it will fail in the zero knowledge proof.

Hence S will be successful in simulating the zero knowledge proof, except with negligible probability.

5.2 The Soundness Lemma

Lemma 1 *(Soundness Lemma)*
Let $\mathsf{UC\text{-}Com}(P_1, S, a)$ *be the commitment protocol in which* P_1 *commits to string* a *(of length* n *bits) to the simulator* S *(acting on behalf of the honest party* P_2*). Suppose* S *accepts the zero knowledge proof of knowledge in the commitment phase of* $\mathsf{UC\text{-}Com}(P_1, S, a)$*.* S *then looks up table* T_{12} *for the commitments* $B_i = \mathsf{Com}(a_i)$ *for all* i*. Let* E *be the event that there exists* i *such that* $B_i \notin T_{12}$*. If the signature scheme used in the creation of the token by* P_2 *is secure, then* $\Pr[E] \leq \epsilon$*, where* ϵ *is negligible in the security parameter* k*.*

A proof of the above Soundness Lemma will be provided in the full version of this paper.

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.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, pages 61–85, 2007.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, Lecture Notes in Computer Science, pages 19–40. Springer, 2001.

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

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[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.

[DNW07] Ivan Damgaard, Jesper Buus Nielsen, and Daniel Wichs. Universally composable multiparty computation with partially isolated parties. Cryptology ePrint Archive, 2007. http://eprint.iacr.org/2007/332.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, UK, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HMQU05] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *5th Central European Conference on Cryptology*, page A version is available at http://homepages.cwi.nl/ hofheinz/card.pdf., 2005.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, Lecture Notes in Computer Science, pages 115–128. Springer, 2007.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *STOC*, pages 33–43, 1989.

Appendix

A Zero knowledge proof of knowledge

A protocol with a prover P and a verifier V is a zero knowledge proof of knowledge for a language L if the following properties hold:

- **Correctness.** $\forall x \in L, \exists$ witness w, such that

$$\Pr[P(1^k, x, w) \leftrightarrow V(1^k, x) \rightarrow \text{Accept}] = 1$$

- **Zero Knowledge.** Let $PV(x)$ denote the view of the conversation from the verifiers point of view on input x. $\forall$ PPT V', $\exists$ PPT S, such that $\forall x \in L, \forall$ witnesses w are indistinguishable:

$$D_0 = \{P(1^k, x, w) \leftrightarrow V'(1^k, w) \rightarrow PV(x)\}$$
$$D_1 = \{S(1^k, x) \leftrightarrow V'(1^k, x) \rightarrow SV'(x)\}$$

- **Extraction.** For all PPT P', $\exists$ PPT E such that $\forall x \in L$, additional input w', there exists a negligible function ϵ such that

$$\Pr[P'(1^k, x, w') \leftrightarrow V(1^k, x) \rightarrow \mathit{Accept}] -$$

$$\Pr[P'(1^k, x, w') \leftrightarrow E(1^k, x) \rightarrow w : w \text{is a valid witness for } x \in L] \leq \epsilon(k)$$

  Here, E is allowed to rewind P' and run P' on polynomially many inputs.

Below, we give a protocol that is a zero knowledge proof of knowledge [Gol01, Gol04]. Let $l(k)$ be a super-logarithmic function of the security parameter k and let $p(k)$ be a polynomial in k. Both prover (P) and verifier (V) are given the instance x and the language L. The prover is given the witness w that proves the statement $x \in L$. Without loss of generality, we can assume that the statement is "A given graph G has a Hamiltonian cycle". Hence w is a witness to the Hamiltonian cycle of G. Let $\mathsf{Com}(a)$ denote a non-interactive perfectly binding commitment to a string or bit a. $\mathsf{Open}(\mathsf{Com}(a))$ denotes the opening to $\mathsf{Com}(a)$ (which is a along with the randomness used to create the commitment). Figure 2 shows a zero knowledge proof of knowledge protocol.

1. P generates $l(k)$ pairs of the form $\{w_i^0, w_i^1\}$ such that $w_i^0 \oplus w_i^1 = w$ for all $1 \leq i \leq l(k)$. P then sends $\mathsf{Com}(w_i^0), \mathsf{Com}(w_i^1)$ for all i to V.
2. V generates a $l(k)$-bit challenge and sends the challenge to P. Let the i^{th} bit of the challenge be denoted by q_i.
3. P responds with $w_i^{q_i}$, for all i.
4. P picks a random permutation π of graph G and sends commitments of the adjacency matrix $H = \pi(G)$. P will also commit to the permutation π.
5. V flips a bit b at random and sends it to P.
6. If $b = 0$, then P responds with the opening of the commitment to the adjacency matrix and the opening of the commitment to the permutation π. If $b = 1$, then P responds with the opening of the commitments to the edges in H forming a Hamiltonian cycle.
7. If $b = 0$, V checks if the openings are valid and if $H = \pi(G)$. If $b = 1$, V checks if the openings are valid and that they correspond to a Hamiltonian cycle in H. If the checks succeed, V accepts this round. Steps $4 - 7$ of the protocol are repeated $p(k)$ times. V accepts if all rounds were accepted.

Figure 2: Zero-knowledge proof of knowledge protocol