Improved Universally Composable Multi-party Computation using Tamper-proof Hardware

Nishanth Chandran Vipul Goyal Amit Sahai

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

Abstract

Katz, in [Kat07], introduced the model of tamper resistant hardware as a new setup assumption for universally composable multi-party computation. 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 a protocol for universally composable multi-party computation tolerating any number of dishonest parties. Unfortunately, the protocol proposed by Katz has the following drawbacks. Firstly, the protocol requires the tamper-resistant hardware to remember state. In particular, the parties need to execute a two-round interactive protocol with the tamper-resistant hardware. This might make the hardware susceptible to resettability attacks [CGGM00]. It seems quite reasonable that the tamper resistant hardware can be reset and executed again. Secondly, the protocol requires an adversary that creates a hardware token implementing a functionality, to "know" the code that implements the functionality. In particular, it does not model adversaries who may pass on (possibly malicious) hardware tokens from one party to another.

In this paper, using completely different techniques from [Kat07], we improve the results of Katz [Kat07]. In particular, we give a construction for Universally composable multi-party computation in the tamper resistant hardware model that models the interaction with the tamper-resistant hardware as a simple request-reply protocol. This implies resetability of our scheme and also that the tamper-resistant hardware is stateless. Furthermore, the protocol does not require rewinding of the device in the security proof. This implies that the party P creating the hardware token need not "know" the code corresponding to the actions that the token will take. This models adversaries who may simply pass on hardware obtained from a source to someone else without actually knowing the functionality of the hardware token. Our protocol is also based on more 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 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.

In [Kat07], Katz 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. The assumption is that tamper-proof hardware exists. 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 a protocol for universally composable multi-party computation tolerating any number of dishonest parties. Unfortunately, the protocol proposed by Katz has the following drawbacks. Firstly, the protocol requires the tamper-resistant hardware to remember state. In particular, the parties need to execute a two-round interactive protocol with the tamper-resistant hardware. This might make the hardware susceptible to resettability attacks [CGGM00]. It seems quite reasonable that the tamper resistant hardware can be reset and executed again. Secondly, the protocol requires an adversary that creates a hardware token implementing a functionality, to "know" the code that implements the functionality. In particular, it does not model adversaries who may pass on (possibly malicious) hardware tokens from one party to another.

In this paper, using completely different techniques from [Kat07], we improve the results of Katz [Kat07]. In particular, we give a construction for Universally composable multi-party computation in the tamper resistant hardware model that models the interaction with the tamper-resistant hardware as a simple request-reply protocol. This implies resetability of our scheme and also that the tamper-resistant hardware is stateless. Furthermore, the protocol does not require rewinding of the device in the security proof. This implies that the party P creating the hardware token need not "know" the code corresponding to the actions that the token will take. This models adversaries who may simply pass on hardware obtained from a source to someone else without actually knowing the functionality of the hardware token. Our protocol is also based on more general assumptions (namely enhanced trapdoor permutations) [1].

In [Kat07], it is assumed that once P creates a hardware token and hands it over to P', then P cannot send messages to the token. We relax this assumption slightly, and assume that once P creates a hardware token and hands it to P' then P can neither send nor receive messages from the token. We believe that if communication can be prevented in one direction, then it is reasonable to assume that communication from both sides can be prevented.

To summarize, we give UC-secure multi-party computation in the tamper-resistant hardware model (proposed by [Kat07]) with the following properties:

1. The construction is based on general assumptions (namely, enhanced trapdoor permutations).

2. Interaction with the tamper-proof hardware is a simple request-reply protocol (one round protocol). This implies resetability of the protocol. The tamper-resistant hardware is also hence stateless in the protocol.

3. The protocol also does not need require rewinding of the hardware token in the security proof. Hence, the party creating the hardware token need not "know" the code corresponding to the actions that the token will take.

2 Model

Our results are in a model largely similar to the model proposed by [Kat07]. We modify the model to allow for adversaries who may pass on hardware tokens to other parties without knowing the code of the functionality implemented by the hardware token. At a high level, the Ideal/Real model for multi-party computation in the tamper-proof hardware model is as follows. 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 adversaries who will create the code for these tokens. The functionality $\mathcal{F}_{wrap}$ implements the tamper-resistant hardware.

In the real world, during the setup for multi-party computation, all parties give out a hardware token to every other party. Honest parties create the code for the hardware tokens honestly and send the program code of the hardware tokens to $\mathcal{F}_{wrap}$. Malicious parties interact with $\mathcal{F}_{Adv}$ in an arbitrary manner. $\mathcal{F}_{Adv}$ on behalf of each malicious party, then sends the program code of the tokens that are to be given to all

[1]Our commitment scheme is based on one-way permutations

other parties, to $\mathcal{F}_{wrap}$. During protocol execution, all queries made to tamper-resistant hardware tokens are made to the $\mathcal{F}_{wrap}$ functionality.

In the ideal world, there exists an ideal functionality $\mathcal{F}$ computing the function f which the parties wish to compute. The simulator simulates the view of the adversary. When the adversary interacts with $\mathcal{F}_{Adv}$, the simulator forwards messages from the adversary to $\mathcal{F}_{Adv}$ and vice-versa. As in the real model, $\mathcal{F}_{Adv}$ on behalf of each malicious party, then sends the program code of the tokens that are to be given to all other parties, to $\mathcal{F}_{wrap}$. Program codes of the tokens created by honest parties are created by the simulator using the honest protocol for token creation. When the adversary queries a token corresponding to an honest party, S replies with the response to the adversary. When the adversary queries a token corresponding to a malicious party, S forwards the query to $\mathcal{F}_{wrap}$ and then upon receiving the response from $\mathcal{F}_{wrap}$, forwards it to the adversary. Honest parties send their inputs to the trusted functionality $\mathcal{F}$. Simulator extracts inputs from adversarial parties and sends them to $\mathcal{F}$. $\mathcal{F}$ returns the function output to all honest parties and to the simulator who forwards it to the malicious parties.

We first formally define the $\mathcal{F}_{wrap}$ functionality modelled on the $\mathcal{F}_{wrap}$ functionality of [Kat07]. An honest user can create a hardware token T_F implementing any polynomial time functionality F, but an adversary given the token T_F can do no more than observe the input/output characteristics of the token. We modify the "wrapper" functionality $\mathcal{F}_{wrap}$ of [Kat07]. The functionality models the hardware token encapsulating an interactive protocol M. The only change we make is that $\mathcal{F}_{wrap}$ now models M as a 1-round interactive Turing machine (instead of a 2-round interactive Turing machine). It models the following sequence of events: (1) a party (also known as *creator*) takes software implementing a particular functionality and seals this software into a tamper-resistant hardware token (2) The creator then gives this token to another party (also known as the *user*) who can use the hardware token as a black-box to the functionality. Figure 1 shows the formal description of $\mathcal{F}_{wrap}$ based on a 1-round interactive protocol M (modified from [Kat07]).

$\mathcal{F}_{wrap}$ 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 1-round interactive 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). Choose random $w \leftarrow \{0,1\}^{p(k)}$. Run $M(msg; w)$ 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: $\mathcal{F}_{wrap}$ functionality based on a 1-round interactive Turing machine M

We now formally describe the Ideal/Real model 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 let the honest parties be denoted by $\mathcal{H} = \mathcal{P} - \mathcal{M}$. We consider only static adversaries. 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 adversaries who will create the code for these tokens. $\mathcal{F}$ is the 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 model. The real model is the $(\mathcal{F}_{Adv}, \mathcal{F}_{wrap})$-hybrid model. The adversary interacts with $\mathcal{F}_{Adv}$ in an arbitrary manner before protocol execution. 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 $\mathcal{F}_{wrap}$. Each honest party sends a list of n-1 program codes 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 MODEL. The ideal model is the $(\mathcal{F}_{Adv}, \mathcal{F}_{wrap}, \mathcal{F})$-hybrid model. The simulator S simulates the view of the adversary. When the adversary initially communicates with $\mathcal{F}_{Adv}$, S simply forwards messages from the adversary to $\mathcal{F}_{Adv}$ and vice-versa. 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 $\mathcal{F}_{wrap}$. S generates the program code for all tokens that are to be created by honest parties (S does this according to the honest protocol for creating the program code). When the adversary queries a token corresponding to an honest party, S replies with the response to the adversary. When the adversary queries a token corresponding to a malicious party, S forwards the query to $\mathcal{F}_{wrap}$ and then upon receiving the response from $\mathcal{F}_{wrap}$, forwards it to the adversary. Honest parties send their inputs to the trusted functionality $\mathcal{F}$. Simulator extracts inputs from adversarial parties and sends them to $\mathcal{F}$. $\mathcal{F}$ returns the output to all honest parties and to S who forwards it to the malicious parties. Figure **??** shows the representation of the Ideal model.

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 $\mathsf{Com}(a)$.

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 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 is to commit to a string a (of length n bits) to P_2.

At a high level, the protocol is as follows: P_2 sends a tamper resistant hardware token that will take a commitment and its opening from P_1 as input. If the opening to the commitment is valid, then the token outputs a signature of this commitment under P_2's public key. This is the initial setup phase.

In the commitment phase, P_1 generates n commitments to 0 and n commitments to 1 and using the hardware token obtains signatures on this set of $2n$ commitments. To commit to an n-bit string a, P_1 selects commitments along with their signatures from this set corresponding to the value of a. P_1 now commits to these signatures and sends the commitment of a as well as the commitment of the signatures to P_2. P_1, finally proves using a zero knowledge proof of knowledge that the commitments were valid and represent commitments to signatures of commitments to the bits of a.

In the decommitment phase, P_1 sends a to P_2. We note that P_1 does not send the actual opening of the commitment. This is to allow equivocation of the commitment by the simulator during protocol simulation. P_1 then gives a zero knowledge proof that a was the string committed to in the commitment phase. Note that 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. P_2 then uses randomness from R in the zero knowledge protocol and gives zero knowledge proofs at every round to prove this. 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! We describe the protocol more formally below:

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 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 signature on these commitments. This is because, P_2's hardware could be programmed to respond only if some condition is satisfied by the input string a. If P_1 continues with the protocol, then P_2 gains 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.

   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 $C_i = \mathsf{Com}(\sigma_i)$.

   - 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 output some information about a.

   Let $Com_i = (B_i, C_i)$. Now, $A = \mathsf{COM}(a) = \{Com_1, Com_2, ..., Com_n\}$. P_1 sends A to P_2.

2. P_1 now gives a zero knowledge proof of knowledge to P_2 that 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. 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 $\mathsf{UC\text{-}Com}(P_2, P_1, R)$.

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. More formally, the statement P_1 proves to P_2 is "$L_1 =$ There exists randomness such that $A = \mathsf{COM}(\mathsf{a})$." Let the commitment of R in $\mathsf{UC\text{-}Com}(P_2, P_1, R)$ be $Z = COM(R)$, where $COM(\cdot)$ is as defined in the commitment phase. The statement P_2 proves to P_1 is "$L_2 =$ There exists string R, such that

   - There exists an opening of Z to R under the commitment scheme $COM(\cdot)$ of $\mathsf{UC\text{-}Com}(P_2, P_1, R)$
   - R was the randomness used to compute this message"

4. P_2 accepts the decommitment 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 commitment 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). When simulating the adversary's view during its interaction with $\mathcal{F}_{Adv}$, the simulator simply forwards messages from the adversary to $\mathcal{F}_{Adv}$ and vice-versa. Whenever an adversary interacts with a token created by a malicious party, 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 adversary and the program code of the token.

Extraction during Commitment phase. The simulator (simulating honest party P_2's interaction with P_1) runs protocol $\mathsf{UC\text{-}Com}(P_1, S, a)$ with adversarial party P_1. S executes the protocol honestly as a receiver in the commitment phase.

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 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 $\mathsf{Verify}(PK, B_i, \sigma_i) = 1$."

S accepts the commitment if it accepts the zero-knowledge proof. If the zero knowledge proof was accepted, S looks up the commitments of the bits of a in the list of commitments that were queried by P_1 to the device created by honest party P_2. Note that since P_2 is honest, the queries made by P_1 to the device created by P_2 were actually made to the simulator in the ideal world and S has a list of these commitments along with their openings.

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, the commitments to the bits of a were 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 the openings to these commitments and hence has extracted the value of a by looking up these commitments.

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.

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. 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 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 adversary using randomness R. Now, S sends messages to the adversary according to the simulated zero knowledge protocol transcript. It sends the adversary the messages of the prover and as response receives exactly the same messages as in the simulated transcript (because the adversary uses randomness R to execute this protocol). The adversary 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 the zero knowledge proof, except with negligible probability.

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 *(*S *simulates honest party* P_2*). Let* E *be the event that* S *accepts the zero knowledge proof of knowledge given by* P_1 *in the commitment phase of* $\mathsf{UC\text{-}Com}(P_1, S, a)$*. Let the bits of* a *be denoted by* $a_1, a_2, \cdots, a_n$*. Let* $(Com(a_i), Open(Com(a_i)))$ *denote valid commitment/opening pairs to the bit* a_i *for all* i*. Let* F *be the event that* P_1 *queried the device sent by* P_2 *with* $(Com(a_i), Open(Com(a_i)))$ *for all* i*. Then,*

$$|\Pr[E] - \Pr[F]| < \epsilon$$

where ϵ *is negligible in the security parameter* k*.*

Proof. Details will be given in the full version of the 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.

[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

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

- **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