Theoretical Computer Science 410 (2009) 1099–1108

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A note on universal composable zero-knowledge in the common reference string model[☆]

Andrew C.C. Yao[a], Frances F. Yao[b], Yunlei Zhao[c,*]

[a] *Institute for Theoretical Computer Science, Tsinghua University, Beijing, China*
[b] *Department of Computer Science, City University of Hong Kong, China*
[c] *Software School, Fudan University, Shanghai, China*

A R T I C L E   I N F O

Keywords:
Cryptographic protocols
Universal composability
Zero-knowledge
Proof of knowledge
Common reference string

A B S T R A C T

Pass observed that universal composable zero-knowledge (UCZK) protocols in the common reference string (CRS) model lose deniability that is a natural security property and implication of the ZK functionality in accordance with the UC framework. An open problem (or, natural query) raised in the literature is: are there any other essential security properties, other than the well-known deniability property, that could be lost by UCZK in the CRS model, in comparison with the ZK functionality in accordance with the UC framework? In this work, we answer this open question (or, natural query), by showing that when running concurrently with other protocols UCZK in the CRS model can lose proof of knowledge (POK) property that is very essential and core security implication of the ZK functionality. This is demonstrated by concrete attack against naturally existing UCZK protocols in the CRS model. Then, motivated by our attack, we make further clarifications of the underlying reasons beneath the concrete attack, and investigate the precise security guarantee of UC with CRS.

1. Introduction

Universal composability (UC) is a powerful notion proposed by Canetti [6] to describe cryptographic protocols that behave like ideal functionality, and can be composed *in arbitrary way*. The salient feature of UC secure protocols is that their security preserves even when it is composed with any arbitrary protocols (captured by unpredictable environment) concurrently in asynchronous networks (like the Internet). In such settings, a protocol execution may run concurrently with an unknown number of other protocols. *These arbitrary protocols may be executed by the same parties or other parties, they may have potentially related inputs and the scheduling of message delivery may be adversarially coordinated. Furthermore, the local outputs of a protocol execution may be used by other protocols in an unpredictable way* [19].

In the framework of UC security, a generic definition is given for what it means for a protocol to "securely realize a given ideal functionality". Here, an "ideal functionality" is a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs. Informally, a protocol securely carries out a given ideal functionality if no adversary can gain more advantages from an attack on a real execution of the protocol, than from an attack on an ideal process where the parties merely hand their inputs to a trusted party with the appropriate functionality

☆ This work was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CityU 122105), CityU Research Grant (No. 9380039) and a grant from the Basic Research Development (973) Program of China (No. 2007CB807901). The third author is also supported by NSFC (No. 60703091), the Pujiang Program of Shanghai and a grant from MSRA.

* Corresponding author.
E-mail addresses: andrewcyao@tsinghua.edu.cn (A.C.C. Yao), csfyao@cityu.edu.hk (F.F. Yao), ylzhao@fudan.edu.cn (Y. Zhao).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.10.027

and obtain their outputs from it (without any other interactions). In other words, it is required that a real execution can be *emulated* in the ideal process.

Traditionally, emulation means that for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ attacking a real protocol execution, in which $\mathcal{A}$ controls the communication channels and potentially corrupts parties, there should exist an "ideal process adversary" or simulator $\mathcal{S}$ that causes the outputs of the parties in the ideal process to be essentially the same as the outputs of the parties in a real execution. In the UC framework, an additional *adversarial* entity $\mathcal{Z}$, called the environment, is introduced. As is hinted by its name, $\mathcal{Z}$ represents the external environment that consists of arbitrary protocol executions running concurrently with the given protocol. This environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. Then, a protocol is said to UC realize a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties interacting with $\mathcal{F}$ in the ideal process. In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$. *One salient and frequently claimed security goal of UC security is the implication of concurrent general composability (CGC), i.e., composability concurrently with arbitrary protocols.*

Zero-knowledge (ZK) protocols allow a prover to validate theorems to a verifier without giving away any other knowledge other than the theorems being true (i.e., existing witnesses). This notion was introduced by Goldwasser, Micali and Rackoff [24] and its generality was demonstrated by Goldreich, Micali and Wigderson [23]. Since its introduction ZK has found numerous and extremely useful applications, and by now has been playing the central role in modern cryptography.

The concept of "proof of knowledge (POK)" was informally introduced in [24], and was formally treated in [4,20,5]. POK systems, especially zero-knowledge POK (ZKPOK) systems, play a fundamental role in the designing of cryptographic schemes and protocols, and enable a formal complexity theoretic treatment of what does it mean for a machine to "know" something. Very roughly, by "proof of knowledge" we mean that a possibly malicious prover can convince that an $\mathcal{NP}$ statement is true if and only if it, in fact, "knows" (i.e., possesses) a witness to the statement (rather than only convincing the language membership of the statement, i.e., the fact that a corresponding witness exists).

Clearly, achieving US secure protocols, in particular UCZK protocols, would be highly desirable in modern cryptography, especially for cryptographic protocols running over Internet. In general, it has been shown that any ideal functionality can be UC realized, as long as a majority of players are assumed to be honest [6]. But, for the more general case where a majority of players may be corrupted (in particular, for the important case of two-party protocols where each player wishes to maintain its security even if the other player is corrupted), it is shown that large classes of functionalities, in particular most two-party protocols, cannot be UC realized in the plain model where no trusted setup is assumed [6,9,10,29,31]. The impossibility results of [6,9,10] are further shown to be hold for *any* definition that implies security under the composition operation considered by the UC framework. Therefore, in the natural setting of no trusted setup and no honest majority (*including the important two-party case*), it is impossible to obtain security in a setting where protocols are run concurrently with arbitrary other protocols. Therefore, whenever this level of security is desired, some setup assumptions (or relaxed definitions of security) are necessary.

The typical setup assumption (in particular, considered in this work) is the common reference string (CRS) model. In the CRS model all parties are given a common (public) reference string that is *ideally and trustily* chosen from a given distribution. A large number of round-efficient UC-secure protocols have been developed in the CRS model. In this work, we focus on UC security for (round-efficient) ZK protocols in the CRS model.

Pass observed that (not necessarily universal composable) zero-knowledge protocols in the common reference string model lose deniability that is a natural security property and implication of the ZK functionality in accordance with the UC framework. An open problem (or, natural query) raised in the literature is: are there any other essential security properties, other than the well-known deniability property, that could be lost by UCZK in the CRS model, in comparison with the ZK functionality in accordance with the UC framework? In this work, we answer this open question (or, natural query), by showing that when running concurrently with other protocols UCZK in the CRS model can lose the proof-of-knowledge property that is very essential and core security implication of the ZK functionality. This is demonstrated by concrete attack against naturally existing UCZK protocols in the CRS model. Then, motivated by our attack, we make further clarifications of the underlying reasons beneath the concrete attack, and investigate the precise security guarantee of UC with CRS.

1.1. Related works

Very recently, we noted the related *independent* work of [8]. The work of [8] clarifies the potential weakness of UC security with global setup in general, with deniability loss as an illustrative example for UCZK in the CRS model. In a sense, our work could also be viewed to exemplify, in another *essential* way (other than the well-known deniability loss), the general theme observed in the independent work of [8] on UC with global setup. Some further investigations about the precise interpretation of the GUC security guarantee are presented in [35].

2. Preliminaries

We briefly recall preliminaries in this section. We assume the reader is familiar with some basic definitions: witness indistinguishability, argument/proof of knowledge, commitments, public-key encryption and signatures, etc. Some of them are recalled in Appendix.

2.1. Σ-protocols, Ω-protocols and the OR-proofs

Definition 2.1 (*Σ-Protocol [13]*). A 3-round public-coin protocol $\langle P, V\rangle$ is said to be a Σ-protocol for a relation R if the following hold:

- Completeness. If P, V follow the protocol, the verifier always accepts.
- Special soundness. From any common input x of length n and any pair of accepting conversations on input x, (a, e, z) and (a, e', z') where $e \neq e'$, one can efficiently compute out w such that $(x, w) \in R$. Here a, e, z stand for the first, the second and the third message, respectively, and e is assumed to be a string of length k (that is polynomially related to n) selected uniformly at random in $\{0, 1\}^k$.
- Special honest verifier zero-knowledge (SHVZK). There exists a PPT simulator S, which on input x (where there exists a w such that $(x, w) \in R$) and a random challenge string $\hat{e}$, outputs an accepting conversation of the form $(\hat{a}, \hat{e}, \hat{z})$, with the probability distribution indistinguishable from that of the real conversation (a, e, z) between the honest $P(w)$ and V on input x. A Σ-protocol is called *perfect/statistical* Σ-protocol, if it is of perfect/statistical SHVZK, i.e., the distribution of the simulated transcript is identical or statistically close to that of the real conversation.

Σ-protocols are very useful cryptographic tools. A very large number of Σ-protocols have been developed in the literature. In particular, (the parallel repetition of) Blum's protocol for DHC [3] is a computational Σ-protocol for $\mathcal{NP}$, and most practical Σ-protocols for number-theoretical languages (e.g., DLP and RSA [34,25], etc.) are of *perfect* SHVZK property. More details about Σ-protocols and their applications can be found in [16].

The OR-proof of Σ-protocols [14]. One basic construction with Σ-protocols allows a prover to show that given two inputs x_0, x_1, it knows a w such that either $(x_0, w) \in R_0$ or $(x_1, w) \in R_1$, but without revealing which is the case (i.e., witness indistinguishable WI). Specifically, given two Σ-protocols $\langle P_b, V_b\rangle$ for R_b, $b \in \{0, 1\}$, with random challenges of, without loss of generality, the same length k, consider the following protocol $\langle P, V\rangle$ which we call Σ_{OR}. The common input of $\langle P, V\rangle$ is (x_0, x_1) and P has a private input w such that $(x_b, w) \in R_b$.

- P computes the first message a_b in $\langle P_b, V_b\rangle$, using x_b, w as private inputs. P chooses e_{1-b} at random, runs the SHVZK simulator of $\langle P_{1-b}, V_{1-b}\rangle$ on input (x_{1-b}, e_{1-b}), and lets $(a_{1-b}, e_{1-b}, z_{1-b})$ be the output. P finally sends (a_0, a_1) to V.
- V chooses a random k-bit string e and sends it to P.
- P sets $e_b = e \oplus e_{1-b}$ and computes the answer z_b to challenge e_b using (x_b, a_b, e_b, w) as input. It sends $((e_0, z_0), (e_1, z_1))$ to V.
- V checks that $e = e_0 \oplus e_1$ and that conversations (a_0, e_0, z_o), (a_1, e_1, z_1) are accepting conversations with respect to inputs x_0, x_1, respectively.

Theorem 2.1 ([14]). *The protocol Σ_{OR} above is a Σ-protocol for R_{OR}, where $R_{OR} = \{((x_0, x_1), w) | (x_0, w) \in R_0$ or $(x_1, w) \in R_1\}$. Moreover, Σ_{OR}-protocols are witness indistinguishable (WI) proof of knowledge systems.*

Ω-protocols [19]. An Ω-protocol is a Σ-protocol in the common reference string (CRS) model, with a special straight-line simulation/extraction property. Specifically, an Ω-protocol $\langle P, V\rangle_{[\sigma]}$ for an $\mathcal{NP}$-relation R and common reference string σ, is a Σ-protocol for relation R with the following additional properties:

- For a given distribution ensemble $\mathcal{D}$, on security parameter 1^n a common reference string σ is drawn from $\mathcal{D}_n$. The players take σ as an additional input (to generate messages from them). Naturally, the simulator S in the definition of Σ-protocol may also take σ as an additional input.
- There exists a polynomial-time extractor $E = (E_1, E_2)$ such that the first element of the output of $E_1(1^n)$ is statistically indistinguishable from $\mathcal{D}_n$. Furthermore, given $(\sigma, \tau) \leftarrow E_1(1^n)$, if there *exist* two accepting conversations (a, e, z) and (a, e', z') with $e \neq e'$ on common input x and CRS σ, then $E_2(x, \tau, (a, e, z))$ outputs w such that $(x, w) \in R$.

Notice that the above second property is similar to the special soundness of Σ-protocols. For a Σ-protocol, there could exist an accepting conversation even for an invalid proof, but two accepting conversations (with the same first-round message but different second-round challenges) guarantee that the proof is valid. Here, for a Ω-protocol, the extractor E can always extract something from any conversation, but it might not be the witness if there is only one accepting conversation. However, having two different accepting conversations guarantees the extracted value is indeed a witness.

A natural way to construct Ω-protocols is as follows: the common reference string will consist of a random public-key pk for a semantically-secure encryption scheme. Then for $(x, w) \in R$, we construct an encryption c of w under public-key pk, and then construct a Σ-protocol to prove that the value encrypted in c is indeed a witness w such that $(x, w) \in R$.

As with Σ-protocol, we can construct the OR-proof combining a Ω-protocol and a Σ-protocol.

2.2. The universal composability framework and ZK functionalities

We briefly summarize the UC framework (the material in this section is almost verbatim from [19,6,12,7], and the reader is referred to these references for further details).

Communication model: We assume an asynchronous network, without guaranteed delivery of messages. Further, we assume that the messages are authenticated, since authentication can be added in standard ways (i.e., the $\mathcal{F}_{AUTH}$-model [6]).

Entities: The basic entities involved are n parties $P_1, \ldots, P_n$, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. All entities are modelled as probabilistic interactive Turing machines.

Session IDs and sub-session IDs: Each message also carries a *session ID* (sid), and (if the message is for a multi-session functionality) an additional *sub-session ID* (ssid). These IDs are used to ensure the uniqueness of the sessions. It is required that no two instances of protocols have same ID, and this is enforced by protocols at a higher level. In other words, only when the uniqueness of sid/ssid are established is the security of the protocols guaranteed.

Corruptions: We will specify either *static* or *adaptive* corruptions. In the static case, the adversary corrupts parties only at the onset of the computation; in the adaptive case, the adversary chooses which parties to corrupt as the computation evolves. Once the adversary corrupts a party, it learns all its internal information, including the private input, the communication history, and the random bits used, *except* the information explicitly erased by the party before the corruption. Once they are corrupted, the behavior of the corrupted parties is arbitrary or malicious.

Real-life execution: At a high level, the execution of a protocol π, run by the parties in the presence of $\mathcal{A}$ and an environment $\mathcal{Z}$ with input z, is modelled as a sequence of *activities* of the entities, with $\mathcal{Z}$ activated first. When $\mathcal{Z}$ is activated, it may write messages on the other entities' input tapes (and thus activate them next), and read messages from the other entities' output tapes. When $\mathcal{A}$ is activated, it may read messages from a party's outgoing communication tapes, and write a message to a party's incoming communication tapes, thus activating the party. It may also corrupt parties, as discussed above. When a party is activated, it runs the protocol π. Finally, the environment $\mathcal{Z}$ outputs one bit and halts.

For security parameter 1^n, and input $z \in \{0, 1\}^*$ to $\mathcal{Z}$, let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble of random variables describing $\mathcal{Z}$'s output when interacting with adversary $\mathcal{A}$ and parties running protocol π, with input z, security parameter 1^n, and uniformly-chosen random tapes for all the entities.

Ideal process: The security of the protocol is defined by comparing the real execution of the protocol (as described above) to an ideal process in which an additional entity, the ideal functionality $\mathcal{F}$, is introduced; Essentially, $\mathcal{F}$ is an incorruptible trusted party that is programmed to produce the desired functionality of the given task. Additionally, the parties are replaced by dummy parties, who do not communicate with each other, but instead have access to $\mathcal{F}$. In this idealized execution, again the environment is activated first, generating the inputs. Whenever a dummy party is activated, it forwards its input to $\mathcal{F}$. Let $\mathcal{S}$ denote the adversary in this idealized execution. $\mathcal{S}$ can see the destinations of the messages between the parties and $\mathcal{F}$, but not the contents. As in the real-life execution, at some point the environment outputs one bit and halts.

Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the distribution ensemble of random variables describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{S}$ in the ideal process for $\mathcal{F}$, with input z, security parameter 1^n, and uniformly-chosen random tapes for all the participating entities.

Security: In this framework, a protocol π *securely realizes* an ideal functionality $\mathcal{F}$, if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π in the real-life execution, or with $\mathcal{S}$ in the ideal process for $\mathcal{F}$. More precisely, two corresponding binary distribution ensembles, $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, are indistinguishable.

The hybrid model: Protocols typically would invoke other sub-protocols. The hybrid model is like a real-life execution, except that some invocations of the sub-protocols are replaced by the invocation of an instance of an ideal functionality $\mathcal{F}$; this is called the "$\mathcal{F}$-hybrid model". Specifically, the model is identical to the real-life model, with the addition that besides sending messages to each other, the parties may exchange messages with an unbounded number of copies of $\mathcal{F}$, where each copy is identified via a unique *session identifier (sid)*. The communication between the parties and each one of those copies mimics the ideal execution.

Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble of random variables describing the output of $\mathcal{Z}$, after interacting with $\mathcal{A}$ and parties running protocol π in the $\mathcal{F}$-hybrid model. Now, let ρ be a protocol that securely realizes $\mathcal{F}$. The composed protocol π^{ρ} is constructed by replacing the first message to $\mathcal{F}$ in π by an invocation of a new copy of ρ, with fresh random input, the same *sid*, and with the contents of that message as input; each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of the message as new input to ρ.

The universal composition theorem: The composition theorem [6] basically says that if ρ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, for some functionality $\mathcal{G}$, then an execution of the composed protocol π^{ρ}, running in the $\mathcal{G}$-hybrid model, "emulates" an execution of protocol π in $\mathcal{F}$-hybrid model. That is, no environment machine $\mathcal{Z}$ can distinguish whether it is interacting with $\mathcal{A}$ and π^{ρ} in the $\mathcal{G}$-hybrid model, or it is interacting with $\mathcal{S}$ and π in the $\mathcal{F}$-hybrid model. In other words, the two distribution ensembles, $\text{HYB}^{\mathcal{G}}_{\pi^{\rho},\mathcal{A},\mathcal{Z}}$ and $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}$, are indistinguishable.

The zero-knowledge functionality: The ZK functionality $\mathcal{F}^{R}_{\text{ZK}}$, parameterized by a relation R, is presented in Fig. 1. In the functionality, the prover sends to the functionality the input x together with a witness w. If $R(x, w)$ holds, then the functionality forwards x to the verifier. As pointed in [6], this is actually a *proof of knowledge* in that the verifier is assured that the prover actually knows w.

Functionality $\mathcal{F}_{\textbf{ZK}}^{R}$

$\mathcal{F}_{\text{ZK}}^{R}$ proceeds as follows, running with security parameter 1^n, a prover P_i, a verifier P_j, and an adversary $\mathcal{S}$:

- Upon receiving $(\textsf{zk-prover}, sid, P_i, P_j, x, w)$ from P_i: If $R(x, w)$ then send $(\textsf{ZK-PROOF}, sid, P_i, P_j, x)$ to P_j and $\mathcal{S}$ and halts. Otherwise, ignore.

Fig. 1. The zero-knowledge functionality (for relation *R*).

Functionality $\hat{\mathcal{F}}_{\textbf{ZK}}^{R}$

$\hat{\mathcal{F}}_{\text{ZK}}^{R}$ proceeds as follows, running with security parameter 1^n, parties $P_1, \cdots, P_n$, and an adversary $\mathcal{S}$:

- Upon receiving $(\textsf{zk-prover}, sid, ssid, P_i, P_j, x, w)$ from P_i: If $R(x, w)$ then send $(\textsf{ZK-PROOF}, sid, ssid, P_i, P_j, x)$ to P_j and $\mathcal{S}$ and halts. Otherwise, ignore.

Fig. 2. The multi-session ZK functionality (for relation *R*).

One shortcoming of the above formulation is that we will be designing and analyzing protocols in the common reference string model, and so they will be operating in the $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$-hybrid model, where $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ is the CRS generation functionality that, for a given security parameter 1^n, chooses a string from distribution $\mathcal{D}_n$ and hands it to all parties and the adversary (*but not directly to the environment*). However, directly realizing $\mathcal{F}_{\text{ZK}}^{R}$ in the $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$-hybrid model and using the universal composition theorem would result in a composed protocol where a new instance of the reference string is needed for each proof. This is extremely inefficient and does not reflect the notion of the CRS model, where an unbounded number of protocol instances would use the same copy of the string. Canetti and Rabin [12] suggested the following notion to cope with this problem:

- **Universal composition with joint state:** Let $\mathcal{F}$ and $\mathcal{G}$ be ideal functionalities, and let $\hat{\mathcal{F}}$ denote the "multi-session extension of $\mathcal{F}$", in which $\hat{\mathcal{F}}$ will run multiple copies of $\mathcal{F}$, where each copy is identified by a special *sub-session identifier* (*ssid*). Now, let π be a protocol in the $\mathcal{F}$-hybrid model, and let $\hat{\rho}$ be a protocol that securely realizes $\hat{\mathcal{F}}$ in the $\mathcal{G}$-hybrid model. Then, construct the composed protocol $\pi^{[\hat{\rho}]}$ by replacing all the copies of $\mathcal{F}$ in π by a single copy of $\hat{\rho}$. The universal composition with joint state theorem of [12] states that $\pi^{[\hat{\rho}]}$, running in the $\mathcal{G}$-hybrid model, correctly emulates π in the $\mathcal{F}$-hybrid model.

The definition $\hat{\mathcal{F}}_{\text{ZK}}^{R}$, the multi-session extension of $\mathcal{F}_{\text{ZK}}^{R}$, is presented in Fig. 2. Note that there are two types of indices: the *sid* differentiates messages to $\hat{\mathcal{F}}_{\text{ZK}}^{R}$ from messages sent to other functionalities; and the sub-session ID *ssid* is unique per input message (or proof).

3. Concurrent general composition attack on UCZK in the common reference string model

In this section, we present a concurrent general composition attack on the protocol of [19] that is UCZK in the common reference string model.

3.1. The protocol structure of UCZK of [19]

We first recall the protocol structure of the UCZK protocol of [19].

Common reference string: $(verk, \sigma')$, where $verk$ is a random verification key of a signature scheme secure against chosen message attacks, σ' is the public reference string for the underlying Ω-protocol (typically, σ' is a random public-key of semantically-secure PKE).

Common input: $x \in L$, where L is an $\mathcal{NP}$-language with $\mathcal{NP}$-relation R_L.

Auxiliary date: An auxiliary data *aux* that may contain any arbitrary public values.

Prover's private input: w s.t. $(x, w) \in R_L$.

Main-proof stage: consists of three phases (in real implementation, phases are combined):

Phase-1: Prover P generates a key-pair (vk, sk) for a one-time strong signature, sends vk to the verifier V.

Phase-2: Give a OR-proof: one Ω-proof for showing the knowledge of w (typically, send a encryption c of w using σ', and prove by Σ-protocol that the encrypted value is indeed a witness for $x \in L$); one Σ-protocol for showing the knowledge of a signature on vk under $verk$. We denote by $a = (a_L, a_{vk})$, e, and $z = ((e_L, z_L), (e_{vk}, z_{vk}))$ the first-round, second-round and the third-round message of the OR-proof, respectively, where (a_L, e_L, z_L) constitute the (partial) conversation of the Ω-protocol (specifically, the conversation of the Σ-protocol in the Ω-protocol for showing the knowledge of the value encrypted in c is indeed a valid witness for $x \in L$), and (a_{vk}, e_{vk}, z_{vk}) constitutes the conversation of the Σ-protocol for showing the knowledge of the signature on vk under $verk$, and $e = e_L \oplus e_{vk}$.

Phase-3: The prover P applies sk on the whole transcript to get a one-time strong signature s, and sends s to V.

Notes: The above protocol is shown to be UCZK in the common reference string model, assuming static corruptions [19]. For UCZK with adaptive corruptions, the above protocol is augmented as follows: In Phase-2, the prover does not send $a = (a_L, a_{vk})$ directly. Rather, it first commits to a and the auxiliary information aux by using a special trapdoor commitment scheme, called simulation-sound trapdoor commitments (SSTC), following the paradigm of [15]. Then, in the third-round of the OR-proof of Phase-2, the prover decommits accordingly and reveals a. The following CGC attack is described against the above UCZK with static corruption, but it can be trivially extended to work on the augmented adaptive-corruption version as well.

3.2. The CGC attack

To present a CGC attack, we need to first design a (different) protocol, and then show that when composed with the designed protocol the UCZK protocol of [19] is not secure. We present a natural and also very useful protocol, and show that when composed with this natural and practical protocol, a malicious adversary can convince the honest verifier of any statement in the original UCZK protocol of [19] *but without knowing any witness for the statement being proved*. This shows that, when concurrently composing with other protocols, UCZK protocols in the common reference string model can lose the POK property that is the very essential and core security implication of the ZK functionality in accordance with the UC framework. We suggest that such a security loss might be more harmful, in comparison with the loss of deniability observed in [33].

The encrypt/commit-then-proof protocol. The protocol to be composed with the UCZK of [19] is the natural and very useful *encrypt/commit-then-proof* protocol $\langle P', V'\rangle$, described as follows.

Common input: $x \in L$.
Prover's private input: w s.t. $(x, w) \in R_L$.
Main-proof stage: consists of two phases:

Phase-1: The verifier V' generates and sends to the prover P' a random public-key σ' for a semantically-secure PKE scheme. Here, σ' can also be viewed as the first-round message of a commitment scheme.

Phase-2: The prover P' encrypts (i.e., commits) w to c using the public-key σ'. Then, P' proves to V' that the value committed is indeed a witness for $x \in L$, by executing a Σ-protocol with V'. We denote by a_L, e_L, z_L the first-round, second-round and third-round message of the Σ-protocol.

We remark that the above encrypt/commit-then-proof protocol is a natural and very useful protocol in practice. The encrypt/commit-then-proof paradigm has been employed in a number of works for various cryptographic tasks and settings (e.g., [28,30,11,1], etc.). When the protocol works in the public-key model with σ' as the verifier's public-key, such protocol is also a common paradigm for achieving *plaintext-aware* (interactive and verifiable) encryption (e.g., [27]), which is also used in group signature and group encryption systems.

Message schedule of the CGC attack. We now describe the message schedule of the CGC attack, that enables an adversary to convince the honest verifier of any statement in the original UCZK protocol of [19] but without knowing any corresponding $\mathcal{NP}$-witness.

The adversary $\mathcal{A}$ runs the UCZK protocol of [19] and the above commit-then-proof protocol concurrently, by playing the role of prover in the UCZK protocol of [19] and playing the role of verifier in the commit-then-proof protocol. In other words, the adversary $\mathcal{A}$ corrupts and controls the prover P of UCZK of [19] and the verifier V' of the commit-then-proof protocol at the onset of the computation. $\mathcal{A}$ schedules the messages as follows.

(1) $\mathcal{A}$ first executes the UCZK with V on common input x and the common reference string $(verk, \sigma')$. For presentation simplicity, we refer to such execution of UCZK as *the first session*. Specifically, it generates a key-pair (vk, sk) for a one-time strong signature, sends vk to the verifier V, just as the honest prover does. When it moves into Phase-2 of the UCZK, $\mathcal{A}$ suspends the first session.

(2) $\mathcal{A}$ executes the commit-then-prove protocol with P' on common input x (x could be set by $\mathcal{A}$ via the environment). For presentation simplicity, we call the execution of the commit-then-prove protocol *the second session*. Specifically, $\mathcal{A}$ sends σ' (*got from the CRS of the first session*) to P' as the Phase-1 message of the second session. After receiving from P' the first-round message of Phase-2 of the second session, $\mathcal{A}$ suspends the second session. Note that the first-round message of Phase-2 of the second session from P' consists of c (that encrypts w) and the first-round message a_L of the underlying Σ-protocol executed in Phase-2 of the second session.

(3) Now, $\mathcal{A}$ continues the first session, and works as follows. On $(vk, verk)$, it generates a simulated conversation (a_{vk}, e_{vk}, z_{vk}) for the Σ-protocol of Phase-2 of the first session (that is used to prove the knowledge of a signature of vk under $verk$), by running the underlying SHVZK simulator. Then, $\mathcal{A}$ sends (c, a_L, a_{vk}) to V as the first-round message of Phase-2 of the first session. After receiving from V the random challenge e (i.e., the second-round message of the OR-proof of Phase-2 of the first session), $\mathcal{A}$ sets $e_L = e \oplus e_{vk}$ and suspends the first session again. *Note that (c, a_L) are got from the second session.*
(4) $\mathcal{A}$ continues the second session again, sends $e_L = e \oplus e_{vk}$ to P' as the second-round message of Phase-2 of the second session. After receiving from P' the last-round message e_L of the second session, $\mathcal{A}$ stops the second session.
(5) $\mathcal{A}$ continues the first-session again, sends $z = ((e_L, z_L), (e_{vk}, z_{vk}))$ to V as the last-round message of the OR-proof of Phase-2 of the first session.
(6) Finally, $\mathcal{A}$ applies the one-time strong signing key sk on the whole transcript of the first session to get a valid signature s, and sends s to V. Note that $\mathcal{A}$ can do this, as the one-time strong key pair (vk, sk) are generated by itself.

Note that (c, a_L, e_L, z_L) is an accepting conversation of the Ω-protocol for showing $x \in L$, (a_{vk}, e_{vk}, z_{vk}) is an accepting conversation for showing the knowledge of the signature of vk under $verk$, and also $e = e_L \oplus e_{vk}$. Furthermore, the one-time strong signature s is also valid. This means that, from the viewpoint of V, $\mathcal{A}$ has successfully convinced V of the statement "$x \in L$" in the first session with the UCZK protocol of [19], *but $\mathcal{A}$ actually does not know any corresponding $\mathcal{NP}$-witness*! It is also easy to see that the above CGC attack schedule can be trivially extended to the augmented adaptive corruption version of the UCZK of [19].

Notes: The adversary $\mathcal{A}$ does not use the same CRS in the second session, but a part of the CRS. Also, the commit-then-proof protocol is run in the plain model. We remark that $\mathcal{A}$ can potentially use a completely different (but maliciously related to CRS) message in Phase-1 of the second session. *In general, $\mathcal{A}$ can potentially malleate the CRS of one session into some message of another concurrent session that is completely different from (but maliciously related with) the CRS.* We also note that it is impossible to prevent transparent adversaries. Specifically, an adversary runs the same protocol twice in two sessions (in one session, the *same* CRS could be sent by a player in the plain model), and forwards the messages from one session to another session (i.e., the transcripts of the two sessions are identical). Such transparent adversary is impossible to prevent, and is not viewed as a harmful adversarial activity *by definition*, analogue to the definition of non-malleability [17].

4. On the *precise* security guarantee of UC with CRS

We remark that the above concrete attack contradicts our intuition, as well as some (informal) interpretations frequently stated in existing works, about the security guarantee of universal composability. We remark that the UCZK functionality does guarantee the POK property. The problem with UC in the common reference string model is that: in security analysis the simulator in the ideal-process world can set the CRS by itself, thus learning the corresponding trapdoor information. This does not capture the real ability of the adversary in the real-life world. But, this does not violate the security formulation of UC, as the environment does not directly access and invoke the CRS and thus it is oblivious of the cases of real CRS and simulated CRS. In other words, in contrast to the common intuition and expectation for UC, the adversary considered in the UC framework actually has access to very limited (external) arbitrary protocols, in the sense that the arbitrary (external) protocols are implicitly required to be "independent" of the challenge protocol (i.e., sharing no state information with the challenge protocol). This is clearly unrealistic to reflect the actual adversarial activities conducted in asynchronous and open networks like the Internet.

Our work may trigger the curiosity about the *precise* security guarantee of UC in the CRS model. Due to the high system complexity and subtle nature of UC both for security formulation and for security analysis, from our view, it is important to interpret the precise security guarantee of existing UC feasibility results, especially for non-experts of UC and practitioners who apply the theory and implementations of UC in practice. It turns out that the interpretation itself can be subtle and error-prone. For example, the goal of "composability with arbitrary protocols" and the reuse feature of CRS are *both* stated in many existing works establishing UC feasibility with CRS, which may raise confusion to the literature (especially for non-experts of UC).

For the precise security guarantee of UC with CRS, our work (and the independent work [8]) show that nothing may be guaranteed, *in general*, when the CRS is reused (which is however the natural scenario and practice for cryptography with CRS). In particular, we note that UC with reusable CRS might not, *automatically and generally*, imply concurrent self composability (where only the same protocol is run concurrently). Furthermore, even with fresh and independent CRS for each session of the challenge protocol, UC with fresh CRS still does not achieve the goal of "composability with arbitrary external protocols". Specifically, even for UC with fresh CRS, the arbitrary external protocols are still implicitly assumed to be "independent" of the challenge protocol, i.e., sharing no arbitrary (other than some well-predefined) state information with the challenge protocol.[1] This situation may, more or less, violate the common intuition and expectation, as well as some frequent informal interpretations, about the UC security. Our this work is thus helpful for a precise understanding of UC with CRS, and for preventing potential misinterpretations and/or misuses in practice.

[1] As noted in [8], the approach proposed in [12] for handling universal composition with joint state (JUC) also does not fully work in this case. Specifically, the JUC approach only allows for the constructions of protocols that share state information *amongst themselves* (rather than sharing arbitrary state information with arbitrary external protocols).

Acknowledgements

We thank Ran Canetti for kind clarifications about UC in the CRS model, and referring us to [8].

Appendix. Basic definitions

We use standard notations and conventions below for writing probabilistic algorithms, experiments and interactive protocols. If A is a probabilistic algorithm, then $A(x_1, x_2, \ldots; r)$ is the result of running A on inputs $x_1, x_2, \ldots$ and coins r. We let $y \leftarrow A(x_1, x_2, \ldots)$ denote the experiment of picking r at random and letting y be $A(x_1, x_2, \ldots; r)$. If S is a finite set then $x \leftarrow S$ is the operation of picking an element uniformly from S. If α is neither an algorithm nor a set then $x \leftarrow \alpha$ is a simple assignment statement. By $[R_1; \ldots; R_n : v]$ we denote the set of values of v that a random variable can assume, due to the distribution determined by the sequence of random processes $R_1, R_2, \ldots, R_n$. By $\Pr[R_1; \ldots; R_n : E]$ we denote the probability of event E, after the ordered execution of random processes $R_1, \ldots, R_n$.

Let $\langle P, V\rangle$ be a probabilistic interactive protocol, then the notation $(y_1, y_2) \leftarrow \langle P(x_1), V(x_2)\rangle(x)$ denotes the random process of running interactive protocol $\langle P, V\rangle$ on common input x, where P has private input x_1, V has private input x_2, y_1 is P's output and y_2 is V's output. We assume w.l.o.g. that the output of both parties P and V at the end of an execution of the protocol $\langle P, V\rangle$ contains a transcript of the communication exchanged between P and V during such execution.

Definition A.1 (*(Public-Coin) Interactive Argument/Proof System*). A pair of interactive machines, $\langle P, V\rangle$, is called an interactive argument system for a language L if both are probabilistic polynomial-time (PPT) machines and the following conditions hold:

- Completeness. For every $x \in L$, there exists a string w such that for every string z, $\Pr[\langle P(w), V(z)\rangle(x) = 1] = 1$.
- Soundness. For every polynomial-time interactive machine P^*, and for all sufficiently large n's and every $x \notin L$ of length n and every w and z, $\Pr[\langle P^*(w), V(z)\rangle(x) = 1]$ is negligible in n.

An interactive protocol is called a *proof* for L, if the soundness condition holds against any (even power-unbounded) P^* (rather than only PPT P^*). An interactive system is called a public-coin system if at each round the prescribed verifier can only toss coins and send the coin-tossing outcome to the prover.

Definition A.2 (*Statistically/Perfectly Binding Bit Commitment Scheme*). A pair of PPT interactive machines, $\langle P, V\rangle$, is called a statistically/perfectly binding bit commitment scheme, if it satisfies the following:

Completeness. For any security parameter n, and any bit $b \in \{0, 1\}$, it holds that $\Pr[(\alpha, \beta) \leftarrow \langle P(b), V\rangle(1^n); (t, (t, v)) \leftarrow \langle P(\alpha), V(\beta)\rangle(1^n) : v = b] = 1$.

Computationally hiding. For all sufficiently large n's, any PPT adversary V^*, the following two probability distributions are computationally indistinguishable: $[(\alpha, \beta) \leftarrow \langle P(0), V^*\rangle(1^n) : \beta]$ and $[(\alpha', \beta') \leftarrow \langle P(1), V^*\rangle(1^n) : \beta']$.

Statistically/perfectly binding. For all sufficiently large n's, and *any* adversary P^*, the following probability is negligible (or equals 0 for perfectly-binding commitments): $\Pr[(\alpha, \beta) \leftarrow \langle P^*, V\rangle(1^n); (t, (t, v)) \leftarrow \langle P^*(\alpha), V(\beta)\rangle(1^n);$ $(t', (t', v')) \leftarrow \langle P^*(\alpha), V(\beta)\rangle(1^n) : v, v' \in \{0, 1\} \bigwedge v \neq v']$. That is, no (*even computational power unbounded*) adversary P^* can decommit the same transcript of the commitment stage both to 0 and 1.

Below, we recall some classic perfectly-binding commitment schemes.

One-round perfectly-binding (computationally-hiding) commitments can be based on any one-way permutation OWP [2,23]. Loosely speaking, given a OWP f with a hard-core predict b (cf. [20]), on a security parameter n one commits a bit σ by uniformly selecting $x \in \{0, 1\}^n$ and sending $(f(x), b(x) \oplus \sigma)$ as a commitment, while keeping x as the decommitment information.

Statistically-binding commitments can be based on any one-way function (OWF) but run in two rounds [32,26]. On a security parameter n, let $PRG : \{0, 1\}^n \longrightarrow \{0, 1\}^{3n}$ be a pseudorandom generator, the Naor's OWF-based two-round public-coin perfectly-binding commitment scheme works as follows: In the first round, the commitment receiver sends a random string $R \in \{0, 1\}^{3n}$ to the committer. In the second round, the committer uniformly selects a string $s \in \{0, 1\}^n$ at first; then to commit a bit 0 the committer sends $PRG(s)$ as the commitment; to commit a bit 1 the committer sends $PRG(s) \oplus R$ as the commitment. Note that the first-round message of Naor's commitment scheme can be fixed once and for all and, in particular, can be posted as a part of public-key in the public-key model.

Definition A.3 (*Witness Indistinguishability WI [18,20]*). Let $\langle P, V\rangle$ be an interactive system for a language $L \in \mathcal{NP}$, and let R_L be the fixed $\mathcal{NP}$ witness relation for L. That is, $x \in L$ if there exists a w such that $(x, w) \in R_L$. We denote by $view^{P(w)}_{V^*(z)}(x)$ a random variable describing the transcript of all messages exchanged between a (possibly malicious) PPT verifier V^* and the honest prover P in an execution of the protocol on common input x, when P has auxiliary input w and V^* has auxiliary input z. We say that $\langle P, V\rangle$ is witness indistinguishable for R_L if for every PPT interactive machine V^*, and any two sequences $W^1 = \{w^1_x\}_{x\in L}$ and $W^2 = \{w^2_x\}_{x\in L}$ for sufficiently long x, so that $(x, w^1_x) \in R_L$ and $(x, w^2_x) \in R_L$, the following two probability distributions are computationally indistinguishable by any non-uniform polynomial-time algorithm: $\{x, view^{P(w^1_x)}_{V^*(z)}(x)\}_{x\in L, z\in\{0,1\}^*}$ and $\{x, view^{P(w^2_x)}_{V^*(z)}(x)\}_{x\in L, z\in\{0,1\}^*}$. Namely, for every non-uniform polynomial-time distinguishing algorithm D, every polynomial $p(\cdot)$, all sufficiently long $x \in L$, and all $z \in \{0, 1\}^*$, it holds that

$$|\Pr[D(x, z, view^{P(w^1_x)}_{V^*(z)}(x) = 1] - \Pr[D(x, z, view^{P(w^2_x)}_{V^*(z)}(x) = 1]| < \frac{1}{p(|x|)}.$$

Definition A.4 (*System for Argument/Proof of Knowledge [4,20,5]*). Let R be a binary relation and $\kappa : N \to [0, 1]$. We say that a probabilistic polynomial-time interactive machine V is a knowledge verifier for the relation R with knowledge error κ if the following two conditions hold:

- Non-triviality: There exists an interactive machine P such that for every $(x, w) \in R$ all possible interactions of V with P on common input x and auxiliary input w are accepting.
- Validity (with error κ): There exists a polynomial $q(\cdot)$ and a probabilistic oracle machine K such that for every interactive machine P^*, every $x \in L_R$, and every $w, r \in \{0, 1\}^*$, machine K satisfies the following condition:

  Denote by $p(x, w, r)$ the probability that the interactive machine V accepts, on input x, when interacting with the prover specified by $P^*_{x,w,r}$ (where $P^*_{x,w,r}$ denotes the strategy of P^* on common input x, auxiliary input w and random-tape r). If $p(x, w, r) > \kappa(|x|)$, then, on input x and with oracle access to $P^*_{x,w,r}$, machine K outputs a solution $w' \in R(x)$ within an expected number of steps bounded by

  $$\frac{q(|x|)}{p(x, w, r) - \kappa(|x|)}.$$

  The oracle machine K is called a knowledge extractor.

An interactive argument/proof system $\langle P, V\rangle$ such that V is a knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called a system for argument/proof of knowledge (AOK/POK) for the relation R.

Blum's protocol for DHC [3]. The n-parallel repetitions of Blum's basic protocol, for proving the knowledge of directed Hamiltonian cycle (DHC) on a given directed graph G [3], is just a 3-round public-coin witness indistinguishable proof of knowledge (WIPOK) system for $\mathcal{NP}$ (with knowledge error 2^{-n}) under any one-way permutation (as the first round of it involves one-round perfectly-binding commitments of a random permutation of G). But it can be easily modified into a 4-round public-coin WIPOK for $\mathcal{NP}$ under any OWF by employing Naor's two-round (public-coin) perfectly-binding commitment scheme [32]. The following is the description of Blum's *basic* protocol for DHC:

Common input. A directed graph $G = (V, E)$ with $q = |V|$ nodes.
Prover's private input. A directed Hamiltonian cycle C_G in G.
Round-1. The prover selects a random permutation, π, of the vertices V, and commits (using a perfectly-binding commitment scheme) the entries of the adjacency matrix of the resulting permutated graph via π. That is, it sends a q-by-q matrix of commitments so that the $(\pi(i), \pi(j))$th entry is a commitment to 1 if $(i, j) \in E$, and is a commitment to 0 otherwise.
Round-2. The verifier uniformly selects a bit $b \in \{0, 1\}$ and sends it to the prover.
Round-3. If $b = 0$ then the prover sends π to the verifier along with the revealing of all commitments (and the verifier checks that the revealed graph is indeed isomorphic to G via π); If $b = 1$, the prover reveals to the verifier only the commitments to entries $(\pi(i), \pi(j))$ with $(i, j) \in C_G$ (and the verifier checks that all revealed values are 1 and the corresponding entries form a simple q-cycle).

We remark that the WI property of Blum's protocol for DHC relies on the hiding property of the underlying perfectly-binding commitment scheme used in its first-round.

References

[1] B. Barak, M. Prabhakaran, A. Sahai, Concurrent non-malleable zero-knowledge, Cryptology ePrint Archive, Report No. 2006/355, Extended abstract appears in FOCS 2006.
[2] M. Blum, Coin flipping by telephone, in: Proc. IEEE Spring COMPCOM, 1982, pp. 133–137.
[3] M. Blum, How to prove a theorem so no one else can claim it, in: Proceedings of the International Congress of Mathematicians, Berkeley, California, USA, 1986, pp. 1444–1451.

[4] M. Bellare, O. Goldreich, On defining proofs of knowledge, in: E.F. Brickell (Ed.), Advances in Cryptology-Proceedings of CRYPTO 1992, in: LNCS, vol. 740, Springer-Verlag, 1992, pp. 390–420.
[5] M. Bellare, O. Goldreich, On probabilistic versus deterministic provers in the definition of proofs of knowledge, Electronic Colloquium on Computational Complexity, 13 (136), 2006, Available also from Cryptology ePrint Archive, Report No. 2006/359.
[6] R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in: IEEE Symposium on Foundations of Computer Science, 2001, pp. 136–145.
[7] R. Canetti, Security and composition of cryptographic protocols: A tutorial, Distributed Computing Column of SIGACT News 37 (3–4) (2006) Available also from Cryptology ePrint Archive, Report 2006/465.
[8] R. Canetti, Y. Dodis, R. Pass, S. Walfish, Universal composable security with global setup, in: S. Vadhan (Ed.), Theory of Cryptography (TCC) 2007, in: LNCS, vol. 4392, Springer-Verlag, 2007, pp. 61–85.
[9] R. Canetti, M. Fischlin, Universal composable commitments, in: Advances in Cryptology-Proceedings of CRYPTO 2001, in: LNCS, vol. 2139, Springer-Verlag, 2001, pp. 19–40.
[10] R. Canetti, E. Kushilevitz, Y. Lindell, On the limitations of universal composition without set-up assumptions, in: E. Biham (Ed.), Advances in Cryptology-Proceedings of EUROCRYPT 2003, in: LNCS, vol. 2656, Springer-Verlag, 2003, pp. 68–86.
[11] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai, Universally composable two-party and multi-party secure computation, in: ACM Symposium on Theory of Computing, 2002, pp. 494–503.
[12] R. Canetti, T. Rabin, Universal composition with joint state, in: Advances in Cryptology-Proceedings of CRYPTO 2002, in: LNCS, vol. 2729, Springer-Verlag, 2003, pp. 265–281.
[13] R. Cramer, Modular design of secure, yet practical cryptographic protocols, Ph.D. Thesis, University of Amsterdam, 1996.
[14] R. Cramer, I. Damgard, B. Schoenmakers, Proofs of partial knowledge and simplified design of witness hiding protocols, in: Y. Desmedt (Ed.), Advances in Cryptology-Proceedings of CRYPTO 1994, in: LNCS, vol. 893, Springer-Verlag, 1994, pp. 174–187.
[15] I. Damgard, Efficient concurrent zero-knowledge in the auxiliary string model, in: B. Preneel (Ed.), Advances in Cryptology-Proceedings of EUROCRYPT 2000, in: LNCS, vol. 1807, Springer-Verlag, 2000, pp. 418–430.
[16] I. Damgard, Lecture notes on cryptographic protocol theory, BRICS, Aarhus University, 2003.
[17] D. Dolev, C. Dwork, M. Naor, Non-malleable cryptography, SIAM Journal on Computing 30 (2) (2000) 391–437. Preliminary version in ACM Symposium on Theory of Computing, pages 542–552, 1991.
[18] U. Feige, A. Shamir, Witness indistinguishable and witness hiding protocols, in: ACM Symposium on Theory of Computing, 1990, pp. 416–426.
[19] J.A. Garay, P. MacKenzie, K. Yang, Strengthening zero-knowledge protocols using signatures, Journal of Cryptology 19 (2) (2006) 169–209. Preliminary version appears in E. Biham (Ed.), Advances in Cryptology-Proceedings of EUROCRYPT 2003, LNCS 2656, Springer-Verlag, 2003, pp. 177–194.
[20] O. Goldreich, Foundation of Cryptography-Basic Tools, Cambridge University Press, 2001.
[21] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, in: IEEE Symposium on Foundations of Computer Science, 1986, pp. 174–187.
[22] O. Goldreich, S. Micali, A. Wigderson, How to prove all $\mathcal{NP}$-statements in zero-knowledge, and a methodology of cryptographic protocol design, in: A.M. Odlyzko (Ed.), Advances in Cryptology-Proceedings of CRYPTO 1986, in: LNCS, vol. 263, Springer-Verlag, 1986, pp. 104–110.
[23] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity or all language in $\mathcal{NP}$ have zero-knowledge proof systems, Journal of the Association for Computing Machinery 38 (1) (1991) 691–729. Preliminary version appears in [21,22].
[24] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof-systems, in: ACM Symposium on Theory of Computing, 1985, pp. 291–304.
[25] L. Guillou, J.J. Quisquater, A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory, in: C.G. Gnther (Ed.), Advances in Cryptology-Proceedings of EUROCRYPT 1988, in: LNCS, vol. 330, Springer-Verlag, 1988, pp. 123–128.
[26] J. Hastad, R. Impagliazzo, L.A. Levin, M. Luby, Construction of a pseudorandom generator from any one-way function, SIAM Journal on Computing 28 (4) (1999) 1364–1396.
[27] J. Katz, Efficient and non-malleable proofs of plaintext knowledge and applications, in: E. Biham (Ed.), Advances in Cryptology-Proceedings of EUROCRYPT 2003, in: LNCS, vol. 2656, Springer-Verlag, 2003, pp. 211–228.
[28] J. Kilian, Uses of Randomness in Algorithms and Protocols, MIT Press, Cambridge, MA, 1990.
[29] Y. Lindell, General composition and universal composability in secure multi-party computation, in: IEEE Symposium on Foundations of Computer Science, 2003, pp. 394–403.
[30] Y. Lindell, Parallel coin-tossing and constant-round secure two-party computation, Journal of Cryptology 16 (3) (2003) 143–184. Preliminary version appears in Crypto 2001, LNCS 2139, pages 171-189, Springer-Verlag, 2001.
[31] Y. Lindell, Lower bounds for concurrent self composition, in: M. Naor (Ed.), Theory of Cryptography (TCC) 2004, in: LNCS, vol. 2951, Springer-Verlag, 2004, pp. 203–222.
[32] M. Naor, Bit commitment using pseudorandomness, Journal of Cryptology 4 (2) (1991) 151–158.
[33] R. Pass, On deniability in the common reference string and random oracle models, in: D. Boneh (Ed.), Advances in Cryptology-Proceedings of CRYPTO 2003, in: LNCS, vol. 2729, Springer-Verlag, 2003, pp. 316–337.
[34] C. Schnorr, Efficient signature generation by smart cards, Journal of Cryptology 4 (3) (1991) 24.
[35] A.C.C. Yao, F.F. Yao, Y. Zhao, A note on the feasibility of generalized universal composability, in: J. Cai, S.B. Cooper, H. Zhu (Eds.), Theory and Applications of Models of Computation-Proceedings of TAMC 2007, in: LNCS, vol. 4484, Springer-Verlag, 2007, pp. 474–485.