SCIS 2012 The 29th Symposium on
Cryptography and Information Security
Kanazawa, Japan, Jan. 30 - Feb. 2, 2012
The Institute of Electronics,
Information and Communication Engineers

Constant-Round Concurrent Zero-Knowledge in Super-Polynomial Simulation Security

Susumu Kiyoshima * Yoshifumi Manabe † Tatsuaki Okamoto ‡

Abstract— In this paper, we construct a constant-round concurrent zero-knowledge protocol that is secure under a relaxed notion of security called *Super-Polynomial Simulation* (SPS) security. Here, we *do not* use any setup assumptions such as common reference strings. The SPS security, introduced by Prabhakaran and Sahai (STOC '04), is identical to Universally Composable (UC) security except that the ideal-world adversary (a.k.a. simulator) is allowed to run in super-polynomial time. The security of our construction is proven under a *standard assumption*, the DDH assumption. In addition, our construction is *practical* since it does not use any inefficient primitives such as general but expensive constructions of zero-knowledge protocols for all $\mathcal{NP}$ statements. To the best of our knowledge, our construction is the first SPS-secure constant-round concurrent zero-knowledge protocol that enjoys both of these properties simultaneously.

A key element of our construction is a concurrently secure commitment scheme with the SPS security. Our construction is obtained by plugging this commitment scheme into the results of Canetti and Fischlin (CRYPTO '01).

Keywords: zero-knowledge, commitment scheme, concurrent security, super-polynomial simulation

1 Introduction

With *zero-knowledge proofs*, the prover can prove the validity of a statement to the verifier without providing any additional knowledge. In other words, the verifier learns nothing other than the validity of the statement. This property is formalized in a simulation paradigm. An interactive proof is said to be zero-knowledge if for any verifier V^* that interacts with the prover, there exists a *simulator* $\mathcal{S}$ that does not interact with the prover such that the output of $\mathcal{S}$ is indistinguishable from the output of V^*. This means that, if the verifier V^* learns something after interacting with the prover, the simulator $\mathcal{S}$ also learns it even without interacting with the prover. Since the notion was introduced in [GMR89], zero-knowledge proofs have been extensively studied. For example, Goldreich and Kahan [GK96] constructed a constant-round zero-knowledge proof for any $\mathcal{NP}$ statement.

Concurrent Zero-Knowledge. In the original setting of zero-knowledge proofs, only one instance of the protocol is executed at a time. A more realistic setting is one where many instances of the protocol are executed concurrently with an arbitrary schedule. *Concurrent zero-knowledge proofs* [DNS98] consider the security in this setting. Unfortunately, concurrent zero-knowledge proofs are significantly harder to construct than the standard zero-knowledge proofs. In fact, Canetti *et al.* [CKPR01] showed that, if no trusted setup is assumed, there exists no constant-round concurrent zero-knowledge proofs for non-trivial languages with so called "black-box" simulators.

UC Security. In order to consider concurrent security of arbitrary protocols, Canetti [Can01] proposed a framework called *Universally Composable* (UC) security. The advantage of the UC security is the UC theorem. This theorem guarantees that, if a protocol is secure in the UC framework, it remains secure even when it is executed concurrently with other protocols in an arbitrary manner.

The security in the UC framework is formalized in the simulation paradigm. In order to define the security of a protocol, we define a *real world* and an *ideal world*. In the real world, the parties carry out some task by communicating with each other and executing the protocol. In the ideal world, the parties do not communicate with each other. Instead, they give inputs to an incorruptible trusted party called an *ideal functionality*. The ideal functionality carries out the task securely and gives the parties the desired outputs. Roughly speaking, the protocol is secure if for any adversary who can perform some attacks in the real world there exists an adversary who can perform essentially the same attacks in the ideal world. In slightly more detail, we consider an additional entity called the *environment*, which oversees either the real world or the ideal world. Then, a protocol is said to be secure (or *securely realize* the ideal functionality) if for any real-world adversary $\mathcal{A}$ there exists an ideal-world adversary (a.k.a. simulator) $\mathcal{S}$ such that any environment $\mathcal{Z}$ cannot distinguish whether it runs in the real world or it

* Graduate School of Informatics, Kyoto University
† NTT Communication Science Laboratories
‡ NTT Information Sharing Platform Laboratories

runs in the ideal world.

Unfortunately, it was known that many useful two-party functionalities (including the zero-knowledge functionality) cannot be securely realized with UC security in the *plain model*, where no setup assumption is assumed other than authenticated communication channels [CF01, CKL03].

Super-Polynomial Simulation. In order to address the above negative results, a relaxed notion of security called *Super-Polynomial Simulation* (SPS) security was proposed [Pas03, PS04]. In the SPS security, the simulator is allowed to run in super-polynomial time. Thus, the SPS security guarantees that, if the adversary can perform some attacks, the simulator can perform essentially the same attacks *in super-polynomial time*. Although the SPS security is weaker than the standard simulation-based security, the SPS security guarantees sufficient security in many applications. For example, in the case of the UC security, the SPS security is sufficient for many ideal functionalities such as commitment, since these functionalities are secure even against computationally unbounded adversaries.

The SPS security was first considered in concurrent zero-knowledge protocols [Pas03], and later introduced into the UC security [PS04]. In fact, [PS04] also proposed the stronger *angel-based UC security*, which implies SPS security. The angel-based UC security is identical to UC security except that both the adversary and the simulator have access to a super-polynomial-time oracle (or *angel*). It was shown that the UC theorem holds in angel-based UC security [PS04]. On the other hand, the UC theorem holds in SPS security only under some special settings (as explained in Section 2.4).

In SPS security, the above negative results do not hold. In the case of concurrent zero-knowledge, there exist constant-round concurrent zero-knowledge protocols with black-box simulators for any $\mathcal{NP}$ language in the SPS security [Pas03]. In the case of UC security, there exist protocols that securely realize any functionality in the plain model with SPS security [PS04, BS05, CLP10, GGJS11][1]. In particular, the latter means that there exist protocols that securely realize the concurrent zero-knowledge functionality in the plain model with SPS security.

However, the above protocols are either based on non-standard assumptions (such as the existence of one-way permutations secure against sub-exponential adversaries) or not practical (they use some inefficient primitives such as general but expensive constructions of zero-knowledge protocols for all $\mathcal{NP}$ statements). Thus, a natural question to ask is

> *In the plain model, does there exist constant-round concurrent zero-knowledge protocol that is practical and SPS-secure under standard assumptions?*

[1] In fact, [PS04, CLP10] consider the angel-based UC security.

Our Results. We answer this question in the affirmative. Namely, we show the following theorem.

Main Theorem (informal)**.** *Assume that the DDH assumption holds. Then, in the plain model, there exists a constant-round protocol that securely realizes the concurrent zero-knowledge functionality for any $\mathcal{NP}$ language with SPS security.*

The formal description of Main Theorem is shown in Section 4. To prove Main Theorem, we first construct an *SPS-UC concurrent commitment*, namely a commitment scheme that securely realizes the concurrent commitment functionality with SPS security. We note that, although it is easy to construct commitment schemes that remain secure under concurrent settings[2], the construction of SPS-UC commitments is not trivial. Using this scheme as a primitive, we construct a protocol that securely realizes the concurrent zero-knowledge functionality.

In this paper, we consider only the SPS security. As future work, it may be interesting to construct similar protocols in the angel-based UC security.

2 Preliminaries

2.1 Notations

Let $\mathbb{N}$ denote the set of all positive integers. For any $q \in \mathbb{N}$, let $\mathbb{Z}_q$ denote the set $\{0, \ldots, q-1\}$. For any set X, let $x \xleftarrow{\mathsf{U}} X$ denote that x is an element of X chosen uniformly at random. For any random variable X, let $x \xleftarrow{\mathsf{R}} X$ denote that x is a value chosen at random according to the probability distribution of X. For any randomized algorithm Algo, let $\mathsf{Algo}(x)$ denote a random variable for the output of Algo on input x with a uniformly-chosen random tape. For any random variable X, let $\mathsf{Algo}(X)$ denote a random variable for the output of Algo on input $x \xleftarrow{\mathsf{R}} X$ with a uniformly-chosen random tape.

Let λ denote a security parameter. Let $\epsilon(\lambda)$ denote an arbitrary negligible function in λ. For any probability ensembles $\mathcal{X} = \{X_k\}_{k\in\mathbb{N}}$ and $\mathcal{Y} = \{Y_k\}_{k\in\mathbb{N}}$, let $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$ denote that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable. That is, we have $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$ if and only if for any probabilistic polynomial-time distinguisher $\mathcal{D}$ we have

$$|\Pr[\mathcal{D}(X_\lambda) = 1] - \Pr[\mathcal{D}(Y_\lambda) = 1]| < \epsilon(\lambda)$$

for sufficiently large λ.

2.2 The Assumption

In this paper, we use the DDH assumption. Let GenG be a probabilistic polynomial-time algorithm that, on input 1^λ, outputs a description of a cyclic group $\mathbb{G}$, its order q, and a generator $g \in \mathbb{G}$.

[2] Non-interactive commitment schemes are clearly secure under concurrent settings.

Definition 1 (DDH assumption). *We say that the DDH assumption holds if there exists an algorithm* GenG *such that for any probabilistic polynomial-time algorithm* $\mathcal{A}$*, we have*

$$\left| \Pr\left[\mathcal{A}(\mathbb{G},q,\vec{g})=1 \;\middle|\; \begin{array}{l}(\mathbb{G},q,g)\xleftarrow{R}\mathsf{GenG}(1^\lambda);\\ x,y\xleftarrow{U}\mathbb{Z}_q;\\ \vec{g}:=(g,g^x,g^y,g^{xy})\end{array}\right] - \Pr\left[\mathcal{A}(\mathbb{G},q,\vec{g})=1 \;\middle|\; \begin{array}{l}(\mathbb{G},q,g)\xleftarrow{R}\mathsf{GenG}(1^\lambda);\\ x,y,z\xleftarrow{U}\mathbb{Z}_q;\\ \vec{g}:=(g,g^x,g^y,g^z)\end{array}\right] \right| < \epsilon(\lambda).$$

2.3 UC Security

In this section, we briefly review UC security. For full details, see [Can01].

The model for protocol execution consists of the *environment* $\mathcal{Z}$, the *adversary* $\mathcal{A}$, and the parties running a protocol π. In the protocol execution, the environment $\mathcal{Z}$ is first invoked on external input z. The environment $\mathcal{Z}$ adaptively gives inputs to the parties and receives outputs from them. In addition, $\mathcal{Z}$ communicates freely with $\mathcal{A}$ throughout the protocol execution. On inputs from $\mathcal{Z}$, the parties execute π by sending messages to each other. The adversary $\mathcal{A}$ sees all communications between the parties and controls the schedule of the communications. In this paper, we assume that there exist authenticated communication channels[3]. Thus, the adversary cannot change the contents of messages sent by the parties. The protocol execution ends when $\mathcal{Z}$ outputs a bit. Let $\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda,z)$ denote a random variable for the output of $\mathcal{Z}$ on security parameter $\lambda\in\mathbb{N}$ and input $z\in\{0,1\}^*$ with a uniformly-chosen random tape. Let $\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda,z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$.

The security of π is defined using the *ideal protocol* for the *ideal functionality* $\mathcal{F}$. In the execution of the ideal protocol, all parties simply hand their inputs to $\mathcal{F}$. The functionality $\mathcal{F}$ carries out the desired task securely and gives outputs to the parties. The parties simply forward these outputs to $\mathcal{Z}$. Let *dummy parties* denote the parties in the ideal protocol. Let $\pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$. Let $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\mathsf{Exec}_{\pi(\mathcal{F}),\mathcal{S},\mathcal{Z}}$.

Then, the security of π is defined by comparing the execution of π (referred to as the *real world*) and the execution of $\pi(\mathcal{F})$ (referred to as the *ideal world*).

Definition 2 (UC-realize). *Let* π *be a protocol and* $\mathcal{F}$ *be an ideal functionality. We say that* π ***UC-realizes*** $\mathcal{F}$ *if for any adversary* $\mathcal{A}$ *there exists a simulator* $\mathcal{S}$ *such that for any environment* $\mathcal{Z}$ *we have*

$$\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \ .$$

$\mathcal{F}$-hybrid protocols are protocols where all parties have access to multiple copies of an ideal functionality $\mathcal{F}$. For an $\mathcal{F}$-hybrid protocol π and a protocol ϕ that UC-realizes $\mathcal{F}$, the composed protocol $\pi^{\phi/\mathcal{F}}$ is constructed by modifying π in such a way that each invocation of $\mathcal{F}$ is replaced with the execution of ϕ. The following theorem was proven in [Can01].

Theorem 1 (UC theorem). *Let* $\mathcal{F}$ *be an ideal functionality,* π *be an* $\mathcal{F}$*-hybrid protocol, and* ϕ *be a protocol that UC-realizes* $\mathcal{F}$*. Then for any adversary* $\mathcal{A}$ *there exists a simulator* $\mathcal{S}$ *such that for any environment* $\mathcal{Z}$ *we have*

$$\mathsf{Exec}_{\pi^{\phi/\mathcal{F}},\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Exec}_{\pi,\mathcal{S},\mathcal{Z}} \ .$$

In this paper, we consider only static adversaries. In other words, we assume that the adversary corrupts parties only at the beginning of the protocol execution.

2.4 SPS-UC Security

SPS-UC security [PS04] is the same as UC security, except that we allow the simulator to run in super-polynomial time.

The UC realization is generalized naturally to the SPS-UC security as follows.

Definition 3 (SPS-UC-realize). *Let* π *be a protocol and* $\mathcal{F}$ *be an ideal functionality. We say that* π ***SPS-UC-realizes*** $\mathcal{F}$ *if for any adversary* $\mathcal{A}$ *there exists a super-polynomial-time simulator* $\mathcal{S}$ *such that for any environment* $\mathcal{Z}$ *we have*

$$\mathsf{Exec}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \ .$$

Unfortunately, the proof of the UC theorem in [Can01] does not work in the SPS-UC security. However, if we restrict our attention to an $\mathcal{F}$-hybrid protocol π that calls only a single instance of $\mathcal{F}$, the proof of the UC theorem in [Can01] also works in SPS-UC security.

Theorem 2 (single-instance UC theorem). *Let* $\mathcal{F}$ *be an ideal functionality,* π *be an* $\mathcal{F}$*-hybrid protocol that calls only a single instance of* $\mathcal{F}$*, and* ϕ *be a protocol that SPS-UC-realizes* $\mathcal{F}$*. Then for any adversary* $\mathcal{A}$ *there exists a super-polynomial-time simulator* $\mathcal{S}$ *such that for any environment* $\mathcal{Z}$ *we have*

$$\mathsf{Exec}_{\pi^{\phi/\mathcal{F}},\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Exec}_{\pi,\mathcal{S},\mathcal{Z}} \ .$$

The single-instance UC theorem is implicitly used in [BS05].

3 SPS-UC Concurrent Commitment

In this section, we show our SPS-UC concurrent commitment and prove its security. We use our SPS-UC concurrent commitment to construct a concurrent zero-knowledge protocol in Section 4.

The concurrent commitment functionality $\mathcal{F}_{\mathsf{cCOM}}$ is shown in Figure 1. With a single run of $\mathcal{F}_{\mathsf{cCOM}}$, the sender can commit to multiple messages for the receiver. Here, *ssid* in $\mathcal{F}_{\mathsf{cCOM}}$ is the *subsession ID*. Subsession IDs are used to distinguish among the different commitments that take place within a single run of

[3] This is not essential since authentication can be realized by a protocol, given a standard authentication infrastructure [Can04].

$\mathcal{F}_{\mathsf{cCOM}}$. We note that $\mathcal{F}_{\mathsf{cCOM}}$ is different from the multi-session commitment functionality $\hat{\mathcal{F}}_{\mathsf{COM}}$ (or $\mathcal{F}_{\mathsf{MCOM}}$) in [CF01, CLOS02]. In particular, $\mathcal{F}_{\mathsf{cCOM}}$ does not capture any kind of non-malleability.

Functionality $\mathcal{F}_{\mathsf{cCOM}}$

$\mathcal{F}_{\mathsf{cCOM}}$ proceeds as follows, running with a sender C, a receiver R, and a simulator $\mathcal{S}$.

Commit Phase: Upon receiving an input $(\mathsf{Commit}, sid, ssid, m)$ from C, record $(ssid, m)$ and send $(\mathsf{Receipt}, sid, ssid)$ to R and $\mathcal{S}$. Ignore subsequent inputs $(\mathsf{Commit}, sid, ssid, \ldots)$.

Open Phase: Upon receiving an input $(\mathsf{Open}, sid, ssid)$ from C, if there is a recorded value $(ssid, m)$ then send $(\mathsf{Open}, sid, ssid, m)$ to R and $\mathcal{S}$. Otherwise, do nothing.

Figure 1: The concurrent commitment functionality $\mathcal{F}_{\mathsf{cCOM}}$.

3.1 Protocols

First, we show a challenge-response based extractable commitment scheme $\langle C, R\rangle$, and then we show our SPS-UC concurrent commitment Π, which uses $\langle C, R\rangle$ as a primitive.

3.1.1 Extractable Commitment Scheme $\langle C, R\rangle$

Let Com be a non-interactive perfectly-binding commitment scheme[4]. Then the extractable commitment scheme $\langle C, R\rangle$, which is used in literature such as [PRS02, PW09], is defined as follows.

Commit Phase. The sender C commits to an element a of a group $\mathbb{G}$ for the receiver R as follows.

(1) $C \Rightarrow R$: For each $i \in \{1, 2, \ldots, k = \omega(\log\lambda)\}$, C chooses $\alpha_i \xleftarrow{\mathsf{U}} \mathbb{G}$ and computes $A_i^{(0)} \xleftarrow{\mathsf{R}} \mathsf{Com}(\alpha_i)$ and $A_i^{(1)} \xleftarrow{\mathsf{R}} \mathsf{Com}(a\alpha_i^{-1})$. Then C sends these $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ to R.

(2) $R \Rightarrow C$: The receiver R chooses $r_1, \ldots, r_k \xleftarrow{\mathsf{U}} \{0,1\}$ and sends them to C.

(3) $C \Rightarrow R$: The sender C opens all of $\{A_i^{(r_i)}\}_{i=1}^k$ to R.

Open Phase. The sender C sends a, and opens all of $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ to R.

It is known that $\langle C, R\rangle$ is a perfectly-binding commitment scheme [PW09].

[4] We can construct an efficient non-interactive perfectly-binding commitment scheme under the DDH assumption using ElGamal encryption.

3.1.2 SPS-UC Concurrent Commitment Π

Our SPS-UC concurrent commitment Π is described below. Here, we use the algorithm GenG described in Section 2.2. We assume that the length of the message m is not long such that we have $m \in \mathbb{Z}_q$ for any q that GenG outputs.

Commit Phase. Upon receiving an input $(\mathsf{Commit}, sid, ssid, m)$ from $\mathcal{Z}$, the sender C commits to a message m for the receiver R as follows.

(1) $R \Rightarrow C$: The receiver R computes $(\mathbb{G}, q, g_0) \xleftarrow{\mathsf{R}} \mathsf{GenG}(1^\lambda)$. Next, R chooses $x, y \xleftarrow{\mathsf{U}} \mathbb{Z}_q$ and sets $h_0 := g_0^x$, $g_1 := g_0^y$. Then the receiver R sends $(sid, ssid, \mathbb{G}, q, g_0, h_0, g_1)$ to C.

(2) $C \Leftrightarrow R$: The sender C chooses $a \xleftarrow{\mathsf{U}} \mathbb{G}$. Then C commits to a for R using $\langle C, R\rangle$. In other words, C and R do the following.

(2.1) $C \Rightarrow R$: For each $i \in \{1, 2, \ldots, k = \omega(\log\lambda)\}$, C chooses $\alpha_i \xleftarrow{\mathsf{U}} \mathbb{G}$ and computes $A_i^{(0)} \xleftarrow{\mathsf{R}} \mathsf{Com}(\alpha_i)$ and $A_i^{(1)} \xleftarrow{\mathsf{R}} \mathsf{Com}(a\alpha_i^{-1})$. Then C sends $(sid, ssid, (A_1^{(0)}, A_1^{(1)}), \ldots, (A_k^{(0)}, A_k^{(1)}))$ to R.

(2.2) $R \Rightarrow C$: The receiver R chooses $r_1, \ldots, r_k \xleftarrow{\mathsf{U}} \{0,1\}$ and sends $(sid, ssid, r_1, \ldots, r_k)$ to C.

(2.3) $C \Rightarrow R$: The sender C opens all of $\{A_i^{(r_i)}\}_{i=1}^k$ to R. If C fails to open one of these commitments, R aborts the protocol.

(3) $R \Rightarrow C$: The receiver R chooses $b \xleftarrow{\mathsf{U}} \mathbb{G}$ and sends $(sid, ssid, b)$ to C.

(4) $C \Rightarrow R$: The sender C opens the commitment of $\langle C, R\rangle$ in step (2). If C fails to open the commitment, R aborts the protocol.

(5) C and R set $h_1 := ab$.

(6) $C \Rightarrow R$: The sender C chooses $s, t \xleftarrow{\mathsf{U}} \mathbb{Z}_q$ and sets $u := g_0^s h_0^t$, $v := g_1^s h_1^t$. Then C sets $c := (u, v g_0^m)$ and sends $(sid, ssid, c)$ to R.

(7) The receiver R outputs $(\mathsf{Receipt}, sid, ssid)$.

Open Phase. Upon receiving an input $(\mathsf{Open}, sid, ssid)$ from $\mathcal{Z}$, the sender C opens the commitment as follows.

(1) $C \Rightarrow R$: The sender C sends $(sid, ssid, m, s, t)$ to R.

(2) The receiver R sets $u' := g_0^s h_0^t$ and $v' := g_1^s h_1^t$. If $c = (u', v' g_0^m)$ then R outputs $(\mathsf{Open}, sid, ssid, m)$. Otherwise, R does nothing.

3.2 Security Proof

In this section, we prove the following theorem.

Theorem 3. *Assume that the DDH assumption holds. Then Π SPS-UC-realizes $\mathcal{F}_{\mathsf{cCOM}}$.*

Proof. We need to show that for any adversary $\mathcal{A}$ there exists a super-polynomial-time simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have

$$\mathsf{Exec}_{\Pi,\mathcal{A},\mathcal{Z}} \overset{\mathsf{c}}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\mathsf{cCOM}},\mathcal{S},\mathcal{Z}} \ . \qquad (1)$$

In the real world, the sender commits to multiple messages using Π. These commitments are executed concurrently and the adversary $\mathcal{A}$ controls the schedule. In the ideal world, the sender commits to multiple messages using $\mathcal{F}_{\mathsf{cCOM}}$. A single run of $\mathcal{F}_{\mathsf{cCOM}}$ consists of multiple subsessions, where a single message is committed to in each subsession.

First, we show the description of the simulator $\mathcal{S}$ for any adversary $\mathcal{A}$. The simulator $\mathcal{S}$ internally invokes $\mathcal{A}$ and forwards every message from $\mathcal{Z}$ to the internal $\mathcal{A}$. For each message that the internal $\mathcal{A}$ outputs to $\mathcal{Z}$, the simulator $\mathcal{S}$ simply forwards it to the external $\mathcal{Z}$. Furthermore, $\mathcal{S}$ internally simulates a real world with $\mathcal{A}$ as follows.

Case 1. Corrupted C and Honest R

Since the internal $\mathcal{A}$ commits to messages on behalf of the corrupted C, the simulator $\mathcal{S}$ needs to interact with $\mathcal{A}$ as a receiver. In addition, $\mathcal{S}$ needs to extract the committed messages and send them to $\mathcal{F}_{\mathsf{cCOM}}$.

For each subsession, $\mathcal{S}$ does the following.

- The simulator $\mathcal{S}$ starts the subsession in the same way as the honest R does. That is, the simulator $\mathcal{S}$ computes $(\mathbb{G}, q, g_0) \xleftarrow{\mathsf{R}} \mathsf{GenG}(1^\lambda)$, chooses $x, y \xleftarrow{\mathsf{U}} \mathbb{Z}_q$, sets $h_0 := g_0^x$, $g_1 := g_0^y$, and sends $(\mathbb{G}, q, g_0, h_0, g_1)$ to the internal $\mathcal{A}$.
- Upon receiving $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ from $\mathcal{A}$, the simulator $\mathcal{S}$ chooses $r'_1, \ldots, r'_k \xleftarrow{\mathsf{U}} \{0,1\}$ and extracts the committed values of $\{A_i^{(r'_i)}\}_{i=1}^k$ by breaking the hiding property of Com in super-polynomial time.
- Then, $\mathcal{S}$ chooses $r_1, \ldots, r_k \xleftarrow{\mathsf{U}} \{0,1\}$ and sends them to $\mathcal{A}$ in the same way as the honest R does.
- If $\mathcal{A}$ opens the commitments of Com correctly in response to the challenge, $\mathcal{S}$ extracts the committed value a of $\langle C, R\rangle$ by combining these opened values with the above extracted values[5]. Then $\mathcal{S}$ sends $b := a^{-1} g_0^{xy}$ to $\mathcal{A}$. Here, if $\mathcal{S}$ finds out that the commitment $\{(A_i^{(0)}, A_i^{(1)})\}_{i=1}^k$ of $\langle C, R\rangle$ is invalid when $\mathcal{S}$ tries to extract a, the simulator $\mathcal{S}$ sends $b \xleftarrow{\mathsf{U}} \mathbb{G}$ instead.
- When $\mathcal{A}$ opens the commitment of $\langle C, R\rangle$, the simulator $\mathcal{S}$ verifies its validity in the same way as the honest R does.
- Upon receiving a commitment $c = (c_0, c_1)$ from $\mathcal{A}$, the simulator $\mathcal{S}$ computes $\tilde{m} := \log_{g_0}(c_1/c_0^y)$ in super-polynomial time. Then, the simulator $\mathcal{S}$ sends $(\mathsf{Commit}, sid, ssid, \tilde{m})$ to $\mathcal{F}_{\mathsf{cCOM}}$.
- If $\mathcal{A}$ opens the commitment correctly in the open phase, $\mathcal{S}$ sends $(\mathsf{Open}, sid, ssid)$ to $\mathcal{F}_{\mathsf{cCOM}}$. If $\mathcal{A}$ fails to open, $\mathcal{S}$ does nothing.

[5] Since the probability that $(r_1, \ldots, r_k) = (r'_1, \ldots, r'_k)$ holds is negligible, we simply assume $(r_1, \ldots, r_k) \neq (r'_1, \ldots, r'_k)$ in what follows.

Case 2. Honest C and Corrupted R

Since the internal $\mathcal{A}$ behaves as a receiver on behalf of the corrupted R, the simulator $\mathcal{S}$ needs to communicate with $\mathcal{A}$ as a sender in the commit phase without knowing what message the honest C sent to $\mathcal{F}_{\mathsf{cCOM}}$.

Upon receiving a message $(\mathsf{Receipt}, sid, ssid)$ from $\mathcal{F}_{\mathsf{cCOM}}$, the simulator $\mathcal{S}$ chooses a random message m and commits to m for $\mathcal{A}$ honestly. Upon receiving a message $(\mathsf{Open}, sid, ssid, m')$ from $\mathcal{F}_{\mathsf{cCOM}}$, the simulator $\mathcal{S}$ computes $x := \log_{g_0} h_0$, $y := \log_{g_0} g_1$, and $z := \log_{g_0} h_1$ in super-polynomial time. If $z = xy$ holds (i.e., if (g_0, h_0, g_1, h_1) is a DDH tuple), $\mathcal{S}$ aborts the simulation. Otherwise, $\mathcal{S}$ sets

$$s' := s - \frac{x}{z - xy}(m - m') \ ,$$
$$t' := t + \frac{1}{z - xy}(m - m')$$

and sends $(sid, ssid, m', s', t')$ to $\mathcal{A}$.

Case 3. Honest C and Honest R

The simulator $\mathcal{S}$ communicates with $\mathcal{A}$ as a sender and a receiver. As a receiver, $\mathcal{S}$ behaves in the same way as the honest receiver. As a sender, $\mathcal{S}$ behaves in the same way as in Case 2.

Next, we show that (1) holds for each case.

Analysis of Case 1

We need to show that for any probabilistic polynomial-time distinguisher $\mathcal{D}$ and any polynomial p, we have

$$\left| \begin{array}{l} \Pr\left[\mathcal{D}(\mathsf{Exec}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)) = 1\right] \\ \quad - \Pr\left[\mathcal{D}(\mathsf{Ideal}_{\mathcal{F}_{\mathsf{cCOM}},\mathcal{S},\mathcal{Z}}(\lambda)) = 1\right] \end{array} \right| < \frac{1}{p(\lambda)} \quad (2)$$

for sufficiently large λ.

Let ℓ be an upper bound of the number of subsessions (i.e., the number of messages that $\mathcal{A}$ commits to) and let $\delta(\lambda) := 3\ell \cdot p(\lambda)$. We define the indices of the subsessions based on the order in which the messages of step (2.2) appear in the interaction between C and R. That is, the message of step (2.2) of subsession 2 appears after the message of step (2.2) of subsession 1, and the message of step (2.2) of subsession 3 appears after the message of step (2.2) of subsession 2, and so on.

Below, we use the hybrid argument by defining machines $B_0, \ldots, B_{2\ell+1}$. Before defining these machines, we describe the idea behind our argument. In the ideal world, the simulator $\mathcal{S}$ extracts the committed value a of $\langle C, R\rangle$ in step (2) of each subsession. Let us call this committed value a the *trapdoor secret* of each subsession. Now, the machine B_0 internally executes the real-world protocol and the machine $B_{2\ell+1}$ internally executes the ideal-world protocol. In the sequence of hybrid machines, we change B_0 into $B_{2\ell+1}$ step by step by increasing the number of subsessions of which the trapdoor secrets are extracted. That is, we will define $B_{2(i-1)}$ so that the trapdoor secrets of subsession j ($j = 1, \ldots, i-1$) are extracted and used as in the ideal

world. Next, we will define B_{2i-1} by modifying $B_{2(i-1)}$ in such a way that the trapdoor secret of subsession i is also extracted (but not used). Then, we will define B_{2i} by modifying B_{2i-1} in such a way that the trapdoor secret of subsession i is used as in the ideal world. Each hybrid machine records these extracted trapdoor secrets in a list $\mathsf{a\text{-}List}$. We note that the hybrid machines, except $B_{2\ell+1}$, do not use their super-polynomial power to extract the trapdoor secrets[6]. Instead, they use rewinding techniques and extract the trapdoor secrets using the extractability of $\langle C, R\rangle$.

Now, let us define the hybrid machines $B_0, \ldots, B_{2\ell+1}$. First, we introduce some notations. The hybrid machines, except $B_{2\ell+1}$, internally executes the real-world protocol repeatedly. That is, they internally invoke machines such as $\mathcal{Z}$ and $\mathcal{A}$, execute the protocol, rewind all the machines, execute the protocol again, rewind all the machines again, and so on. We let *thread* denote a single execution of the protocol. A thread begins when internal $\mathcal{Z}$ receives an input, and the thread ends when internal $\mathcal{Z}$ outputs a bit. Each hybrid machine outputs whatever internal $\mathcal{Z}$ outputs in the last thread. Let us call this last thread the *main thread* of each hybrid machine.

Machine B_0: As its main thread, machine B_0 internally executes the real-world protocol by internally invoking $\mathcal{Z}$, $\mathcal{A}$, C, and R. The machine B_0 simply outputs whatever the internal $\mathcal{Z}$ outputs.

Machine B_{2i-1} ($i = 1, \ldots, \ell$): First, B_{2i-1} runs in the same way as $B_{2(i-1)}$, but B_{2i-1} does not output (and does not halt) even after the main thread of $B_{2(i-1)}$ ends. At the time, the trapdoor secret of subsession j ($j = 1, \ldots, i-1$) on the main thread of $B_{2(i-1)}$ is recorded in the $\mathsf{a\text{-}List}$. After the main thread of $B_{2(i-1)}$, the machine B_{2i-1} rewinds this main thread[7] and executes it δ times with the same random tapes except in step (2.2) of subsession i. Let us call these δ threads the *look-ahead threads*. In each look-ahead thread, the challenge $r_1, \ldots, r_k$ in step (2.2) of subsession i is chosen fleshly. After the look-ahead threads, B_{2i-1} executes the main thread of $B_{2(i-1)}$ once again with exactly the same random tapes. This thread is the main thread of B_{2i-1}.

In the case that $\mathcal{A}$ opens the commitments of Com correctly in step (2.3) of subsession i in the main thread and in at least one of the δ look-ahead threads, B_{2i-1} extracts the trapdoor secret a of subsession i by combining the opened values of these two threads. Then, B_{2i-1} adds a pair (i, a) to the $\mathsf{a\text{-}List}$. If B_{2i-1} finds out that the commitment of $\langle C, R\rangle$ is invalid when it tries to extract a, B_{2i-1} adds $(i, \perp)$ to the $\mathsf{a\text{-}List}$ instead.

In the case that $\mathcal{A}$ did not open the commitments of Com correctly in step (2.3) of subsession i in all δ look-ahead threads but opened them correctly in the main thread, B_{2i-1} outputs $\perp$ and halts. Let us call this event $\mathsf{RewindAbort}_i$.

If $\mathsf{RewindAbort}_i$ does not occur, B_{2i-1} outputs whatever internal $\mathcal{Z}$ outputs in its own main thread.

We note that each look-ahead thread proceeds in exactly the same way as the main thread of $B_{2(i-1)}$ (and of B_{2i-1}) until step (2.2) of subsession i, since the random tapes used in this part are the same. In particular, the message of step (2.1) in subsession j ($j = 1, \ldots, i$) in each look-ahead thread is the same as the message in the main thread of $B_{2(i-1)}$. This means the trapdoor secret of subsession j ($j = 1, \ldots, i$) in each look-ahead thread is the same as the trapdoor secret in the main thread of $B_{2(i-1)}$. Thus, the values recorded in the $\mathsf{a\text{-}List}$ before the rewinding are valid even after the rewinding.

Machine B_{2i} ($i = 1, \ldots, \ell$): B_{2i} runs in the same way as B_{2i-1} except that, in step (3) of subsession i in the main thread, internal R sets $b := a^{-1} g_0^{xy}$ if (i, a) is recorded in the $\mathsf{a\text{-}List}$ for $a \neq \perp$. If $a = \perp$, internal R sets $b \xleftarrow{\mathsf{U}} \mathbb{G}$ as in B_{2i-1}.

Machine $B_{2\ell+1}$: $B_{2\ell+1}$ internally executes the ideal-world protocol by internally invoking $\mathcal{Z}$, $\mathcal{S}$, the dummy party C and R. The machine $B_{2\ell+1}$ outputs whatever the internal $\mathcal{Z}$ outputs.

Next, we show the indistinguishability among hybrid machines. Below, we let $\mathsf{Exec}_i(\lambda)$ denote the random variable for the output of machine B_i.

$B_{2(i-1)}$ and B_{2i-1} ($i = 1, \ldots, \ell$): If $\mathsf{RewindAbort}_i$ does not occur in B_{2i-1}, the output of $B_{2(i-1)}$ and the output of B_{2i-1} are identically distributed. $\mathsf{RewindAbort}_i$ occurs in B_{2i-1} if $\mathcal{A}$ does not open the commitments in step (2.3) on subsession i in all δ look-ahead threads but opens them correctly in the main thread. Since $\mathcal{A}$ opens these commitments correctly in each look-ahead thread with the same probability as in the main thread, we can show that $\mathsf{RewindAbort}_i$ occurs in B_{2i-1} with probability at most $1/\delta$. Thus, for any probabilistic polynomial-time distinguisher $\mathcal{D}$, we have

$$\left| \begin{array}{r} \Pr\left[\mathcal{D}(\mathsf{Exec}_{2(i-1)}(\lambda)) = 1\right] \\ -\Pr\left[\mathcal{D}(\mathsf{Exec}_{2i-1}(\lambda)) = 1\right] \end{array} \right| \leq \frac{1}{\delta(\lambda)} \ . \qquad (3)$$

B_{2i-1} and B_{2i} ($i = 1, \ldots, \ell$): B_{2i} is the same as B_{2i-1} except that B_{2i} sets $b := a^{-1} g_0^{xy}$ instead of $b \xleftarrow{\mathsf{U}} \mathbb{G}$ in step (3) of subsession i on the main thread. Thus, from the DDH assumption, for any probabilistic polynomial-time distinguisher $\mathcal{D}$, we have

$$\left| \begin{array}{r} \Pr\left[\mathcal{D}(\mathsf{Exec}_{2i-1}(\lambda)) = 1\right] \\ -\Pr\left[\mathcal{D}(\mathsf{Exec}_{2i}(\lambda)) = 1\right] \end{array} \right| < \epsilon(\lambda) \ . \qquad (4)$$

$B_{2\ell}$ and $B_{2\ell+1}$: In $B_{2\ell}$, all the trapdoor secrets are extracted as in $B_{2\ell+1}$. However, $B_{2\ell}$ uses rewinding instead of their super-polynomial power. The view of $\mathcal{Z}$ in the main threads in $B_{2\ell}$ and in $B_{2\ell+1}$ are indistinguishable if

[6] If hybrid machines are super-polynomial-time machines, it is difficult to show the indistinguishability between the hybrid machines based on computational assumptions.

[7] That is, B_{2i-1} rewinds all the machines such as $\mathcal{Z}$ and $\mathcal{A}$.

- the computed trapdoor secrets are the same in $B_{2\ell}$ and $B_{2\ell+1}$, and
- in $B_{2\ell+1}$, the simulator $\mathcal{S}$ can open the same messages as the internal $\mathcal{A}$ opened.

First, we show the indistinguishability under the condition that $\mathsf{RewindAbort}_i$ does not occur in $B_{2\ell}$ for all i. In this case, in each subsession, the trapdoor secret a that $B_{2\ell}$ records in the $\mathsf{a\text{-}List}$ and the trapdoor secret a that $\mathcal{S}$ computes in $B_{2\ell+1}$ are identically distributed. To see this, observe that in both machines we can think the trapdoor secret a is computed by combining two responses of $\langle C, R\rangle$ for two different challenges. Additionally, when the internal $\mathcal{A}$ opens the commitment correctly in the open phase with (m, s, t) in $B_{2\ell+1}$, we have $h_1 = g_0^{xy}$, $c_0 = g_0^s h_0^t$, and $c_1 = g_1^s h_1^t g_0^m = (g_0^s h_0^t)^y g_0^m$ and thus we have $m = \log_{g_0}(c_1/c_0^y) = \tilde{m}$. This means that $\mathcal{S}$ can open the same message as the internal $\mathcal{A}$ opened. Therefore, we conclude that the views of $\mathcal{Z}$ in the main threads of $B_{2\ell}$ and in $B_{2\ell+1}$ are identically distributed if $\mathsf{RewindAbort}_i$ does not occur in $B_{2\ell}$ for all i.

Next, we compute the probability that $\mathsf{RewindAbort}_i$ occurs in $B_{2\ell}$ for some i. From (3) and (4), we have

$$\left| \begin{array}{l} \Pr\left[\mathcal{D}(\mathsf{Exec}_0(\lambda)) = 1\right] \\ \quad - \Pr\left[\mathcal{D}(\mathsf{Exec}_{2\ell}(\lambda)) = 1\right] \end{array} \right| \leq \frac{\ell}{\delta(\lambda)} + \epsilon(\lambda) \quad (5)$$

for any probabilistic polynomial-time distinguisher $\mathcal{D}$. Since $\mathsf{RewindAbort}_i$ does not occur in B_0 for all i, we conclude that $\mathsf{RewindAbort}_i$ occurs in $B_{2\ell}$ for some i with probability at most $\ell/\delta(\lambda) + \epsilon(\lambda)$.

Combining the above, we conclude that for any probabilistic polynomial-time distinguisher $\mathcal{D}$ we have

$$\left| \begin{array}{l} \Pr\left[\mathcal{D}(\mathsf{Exec}_{2\ell}(\lambda)) = 1\right] \\ \quad - \Pr\left[\mathcal{D}(\mathsf{Exec}_{2\ell+1}(\lambda)) = 1\right] \end{array} \right| \leq \frac{\ell}{\delta(\lambda)} + \epsilon(\lambda) \ . \quad (6)$$

Finishing the Analysis of Case 1. From (5) and (6), for any probabilistic polynomial-time distinguisher $\mathcal{D}$, we have

$$\left| \begin{array}{l} \Pr\left[\mathcal{D}(\mathsf{Exec}_0(\lambda)) = 1\right] \\ \quad - \Pr\left[\mathcal{D}(\mathsf{Exec}_{2\ell+1}(\lambda)) = 1\right] \end{array} \right| \leq \frac{2\ell}{\delta(\lambda)} + \epsilon(\lambda) \ .$$

By substituting $\mathsf{Exec}_0(\lambda) = \mathsf{Exec}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$, $\mathsf{Exec}_{2\ell+1}(\lambda) = \mathsf{Ideal}_{\mathcal{F}_{\mathsf{cCOM}},\mathcal{S},\mathcal{Z}}(\lambda)$, and $\delta(\lambda) = 3\ell \cdot p(\lambda)$, we have (2).

Analysis of Case 2

The views of $\mathcal{Z}$ in the real and ideal worlds are different in the following.

- In the ideal world, random messages are committed to in the commit phase.
- In the ideal world, open information is not computed honestly in the open phase.

First, we show the indistinguishability under the condition that (g_0, h_0, g_1, h_1) is a non-DDH tuple in each subsession. In this case, the value v in step (6) is uniformly random and independent of the value u. To see this, observe that we have $u = g_0^s h_0^t = g_0^{s+xt}$ and $v = g_1^s h_1^t = g_0^{ys+zt}$ for random s and t, and the expressions $s + xt$ and $ys + zt$ are linearly independent combinations of s and t when $z \neq xy$. Thus, the commitment $c = (u, vg_0^m)$ is uniformly distributed in $\mathbb{G}\times\mathbb{G}$. In addition, since we have $z \neq xy$, the simulation does not abort in the ideal world. The open information (m', s', t') of $\mathcal{S}$ is valid since we have

$$\begin{aligned} g_0^{s'} h_0^{t'} &= g_0^{s-\frac{x}{z-xy}(m-m')} h_0^{t+\frac{1}{z-xy}(m-m')} \\ &= g_0^s g_0^{-\frac{x}{z-xy}(m-m')} h_0^t g_0^{\frac{x}{z-xy}(m-m')} \\ &= g_0^s h_0^t = c_0 \end{aligned}$$

and

$$\begin{aligned} g_1^{s'} h_1^{t'} g_0^{m'} &= g_1^{s-\frac{x}{z-xy}(m-m')} h_1^{t+\frac{1}{z-xy}(m-m')} g_0^{m'} \\ &= g_1^s g_0^{-\frac{xy}{z-xy}(m-m')} h_1^t g_0^{\frac{z}{z-xy}(m-m')} g_0^{m'} \\ &= g_1^s h_1^t g_0^{(m-m')} g_0^{m'} = g_1^s h_1^t g_0^m = c_1 \ . \end{aligned}$$

Therefore, we conclude that the views of $\mathcal{Z}$ in the real and ideal worlds are identically distributed if the tuple (g_0, h_0, g_1, h_1) is a non-DDH tuple in each subsession.

Next, we compute the probability that (g_0, h_0, g_1, h_1) is a DDH tuple in some subsessions. Using the hiding property of $\langle C, R\rangle$, we can show that this probability is negligible in the real world. Moreover, since $\mathcal{S}$ internally behaves in the same way as the honest C (with different message m) in the commit phase and the computation of (g_0, h_0, g_1, h_1) is independent of the message, we can conclude that this probability is also negligible in the ideal world.

Combining the above, we conclude that (1) holds.

Analysis of Case 3

We can show that (1) holds by using the same argument as in Case 2. □

4 Concurrent Zero-Knowledge

In this section, we prove Main Theorem. The concurrent zero-knowledge functionality $\mathcal{F}_{\mathsf{cZK}}$, parameterized by a relation R, is shown in Figure 2. We note that

Functionality $\mathcal{F}_{\mathsf{cZK}}$

$\mathcal{F}_{\mathsf{cZK}}$ proceeds as follows, running with a prover P, a verifier V and a simulator $\mathcal{S}$, and parameterized with a relation R.

- Upon receiving $(\mathsf{Prove}, sid, ssid, x, w)$ from P, if $R(x, w) = 1$ then send $(\mathsf{Proof}, sid, ssid, x)$ to $\mathcal{S}$ and V. Otherwise, do nothing.

Figure 2: The concurrent zero-knowledge ideal functionality $\mathcal{F}_{\mathsf{cZK}}$.

$\mathcal{F}_{\mathsf{cZK}}$ is different from the multi-session zero-knowledge

functionality $\hat{\mathcal{F}}_{\mathsf{zK}}$ in [CLOS02]. In particular, $\mathcal{F}_{\mathsf{cZK}}$ does not capture any kind of non-malleability.

Below, we give the formal description of Main Theorem.

Main Theorem. *Assume that the DDH assumption holds. Then, there exists a constant-round protocol that SPS-UC-realizes $\mathcal{F}_{\mathsf{cZK}}$ for any $\mathcal{NP}$ relation.*

Let H denote the graph Hamiltonicity relation. That is, we have

$$H(G,w) := \begin{cases} 1 & \text{(if } w \text{ is a Hamiltonian cycle in graph } G\text{)} \\ 0 & \text{(otherwise)} \end{cases}$$

Let $\mathcal{F}_{\mathsf{cZK}}^{H}$ be the functionality $\mathcal{F}_{\mathsf{cZK}}$ parameterized with relation H. To prove Main Theorem, we use the three-round protocol HC in [CF01], which UC-realizes $\mathcal{F}_{\mathsf{cZK}}^{H}$ in the $\mathcal{F}_{\mathsf{cCOM}}$-hybrid model[8]. The protocol HC consists of parallel repetitions of the three-round protocol of Blum for graph Hamiltonicity, where the prover commits to values for the receiver with $\mathcal{F}_{\mathsf{cCOM}}$. As noted in [CF01], the protocol HC UC-realizes $\mathcal{F}_{\mathsf{cZK}}$ even if the adversary is a super-polynomial-time machine since the security of HC in the $\mathcal{F}_{\mathsf{cCOM}}$-hybrid model is guaranteed without any computational assumption. Thus, we have the following theorem.

Theorem 4 ([CF01])**.** *For any super-polynomial-time adversary $\mathcal{A}$ there exists a super-polynomial-time simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have*

$$\mathsf{Exec}_{\mathrm{HC},\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Ideal}_{\mathcal{F}_{\mathsf{cZK}}^{H},\mathcal{S},\mathcal{Z}} \ .$$

The protocol HC calls only a single instance of $\mathcal{F}_{\mathsf{cCOM}}$. Therefore, using the SPS-UC commitment Π in Section 3 and the single-instance UC theorem in Section 2.4, we have the following lemma.

Lemma 1. *Assume that the DDH assumption holds. Then, for any adversary $\mathcal{A}$ there exists a super-polynomial-time simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have*

$$\mathsf{Exec}_{\mathrm{HC}^{\Pi/\mathcal{F}_{\mathrm{cCOM}}},\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathsf{Exec}_{\mathrm{HC},\mathcal{S},\mathcal{Z}} \ .$$

Now, we are ready to prove Main Theorem.

Proof (of Main Theorem). Since the Hamiltonian cycle problem is $\mathcal{NP}$-complete, all we have to do is to show that there exists a constant-round protocol that SPS-UC-realizes $\mathcal{F}_{\mathsf{cZK}}^{H}$. Clearly, $\mathrm{HC}^{\Pi/\mathcal{F}_{\mathrm{cCOM}}}$ is a constant-round protocol. In addition, from Theorem 4 and Lemma 1, protocol $\mathrm{HC}^{\Pi/\mathcal{F}_{\mathrm{cCOM}}}$ SPS-UC-realizes $\mathcal{F}_{\mathsf{cZK}}^{H}$. □

[8] Actually, the zero-knowledge functionality in [CF01] is a single-session functionality and has some differences from $\mathcal{F}_{\mathsf{cZK}}$. Nonetheless, as noted in [CLOS02], it is easy to see that the protocol HC UC-realizes $\mathcal{F}_{\mathsf{cZK}}$ in the $\mathcal{F}_{\mathsf{cCOM}}$-hybrid model.

References

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552, 2005.

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

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, pages 219–233, 2004.

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

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT*, pages 68–86, 2003.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\Omega}(\log n)$ rounds. In *STOC*, pages 570–579, 2001.

[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.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, pages 541–550, 2010.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *STOC*, pages 409–418, 1998.

[GGJS12] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Concurrently secure computation in constant rounds. In *EUROCRYPT*, 2012.

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *J. Cryptology*, 9(3):167–190, 1996.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In *TCC*, pages 403–418, 2009.