Constant-Round Leakage-Resilient Zero-Knowledge Arguments of Knowledge for NP

Hongda Li, Qihua Niu, Guifang Huang

1 The Data Assurance and Communication Security Research Center
2 State Key Lab of Information Security, Institute of Information Engineering
Chinese Academy of Sciences, Beijing 100093, China

Abstract. Garg, Jain, and Sahai first consider zero knowledge proofs in the presence of leakage on the local state of the prover, and present a leakage-resilient-zero-knowledge proof system for HC (Hamiltonian Cycle) problem. Their construction is called \((1 + \varepsilon)\)-leakage-resilient zero-knowledge, for any constant \(\varepsilon > 0\), because the total length of the leakage the simulator needs is \((1 + \varepsilon)\) times as large as that of the leakage received by the verifier. In recent, Pandey provides a constant-round leakage-resilient zero-knowledge argument satisfying the ideal requirement of \(\varepsilon = 0\). Whether there exist constant round leakage-resilient zero-knowledge arguments of knowledge for all NP languages is an interesting problem. This paper focuses on this problem and presents a constant-round construction of leakage-resilient zero-knowledge arguments of knowledge for the HC problem.

Key word: zero-knowledge proofs, proofs of knowledge, leakage-resilient, non-black-box simulation, constant-round.

1 Introduction

The zero-knowledge proofs, first introduced by Goldwasser et al. in [4], allow the prover to convince the verifier of the validity of a statement \(x \in L\), while providing no additional knowledge to the verifier apart from the fact that the statement is indeed true. A fundamental variant, known as zero knowledge arguments, relaxed the soundness condition only with respect to efficient provers. The zero-knowledge property is formalized by requiring that for any probabilistic polynomial-time (PPT) verifier \(V^*\) there exists a PPT simulator such that the view of \(V^*\) interaction with the prover can be “indistinguishably simulated” by the simulator on input \(x\).

The conventional zero knowledge property assume that the prover’s internal state, including the witness and the random coins, is perfectly hidden from the verifier. The verifier is given only black-box access to the honest prover’s algorithm during the interaction. After a class of attack (known as side-channel attacks) is noticed, this idealized assumption becomes unreasonable and is often hard to satisfy in many real setting where an malicious verifier has the ability to obtain leakage of prover’s internal

\(^0\) Corresponding author e-mail: lihongda@iie.ac.cn
state by launching a side-channel attacks. In order to deal with this problem, Garg et al. [13] recently investigated zero knowledge proofs in such a leaking setting (known as leakage-resilient zero-knowledge proof, LR-ZKP), where the malicious verifier is able to learn an arbitrary amount of leakage on the internal state of the honest prover. Such a leakage attack is modeled as leakage-queries in the execution of the protocol, denoted as a series of polynomial-time computable functions \( f_1, f_2, \ldots \). There exist several model to restrict leakage functions. The most general one requires that leakage functions are hard-to-invert. Ideally, LR-ZKP require that no such malicious verifier can learn anything beyond the validity of the assertion and the leakage. Concretely, it is required that, for any malicious verifier \( V^* \), there exists a simulator which is given access to a leakage oracle, such that the output of the simulator and the view of \( V^* \) are computationally indistinguishable. [13] defined \( \lambda \)-leakage-resilient zero knowledge, which allows the simulator to obtain at most \( \lambda \cdot l \)-bits leakage from the leakage oracle if \( V^* \) obtains \( l \)-bit leakage. [13] provided super-constant-round \( (1+\varepsilon) \)-leakage-resilient-zero-knowledge proof for HC problem for any \( \varepsilon > 0 \). Very recently, Pandey in [20] presented a constant-round leakage-resilient zero-knowledge argument satisfying the ideal requirement of \( \varepsilon = 0 \).

If zero knowledge proofs or arguments are also “proof of knowledge”, it is known as zero knowledge proofs or arguments of knowledge (ZKPoK or ZKAoK). In fact, [13] discussed how to modify their protocol such that the modified protocol is a proof of knowledge. The presented method needs a public-coin zero knowledge proof of knowledge, in which the verifier proves to the prover that the previous commitment is “valid”. Concretely, the verifier first commit to a random challenge \( ch \). At a later decommitment stage, the verifier prove, by a public-coin ZKPoK, that the committed value is indeed \( ch \). [7] first presented a simple construction of super-constant-round LR-ZKPoK for HC problem. An interesting problem left by [13, 20, 7] is how to construct constant-round leakage-resilient ZKPoK (or ZKAoK) for NP. In this paper, We focus on how to construct constant-round LR-ZKAoK for NP and give a construction for HC problem by means of non-black-box simulation techniques.

### 1.1 Related Works

In the past few years there have been a few works on leakage-resilient interactive protocols. Bitansky et. al. [9] considered leakage-resilient protocols for general functionality which are secure against semi-honest adversaries in the UC framework. Boyle et. al. [11] studied leakage-resilient multi-party coin tossing protocol. Damgard et. al. [12] considered leakage-resilient two-party secure protocols against semi-honest adversary. Two types of leakage attack are considered in [12]: global leakage model (the adversary use a leakage function on the input and the entire randomness of an honest party) and local leakage model (the adversary chooses different leakage functions at different points of time during the protocol execution). Very recently, Boyle et. al. [10] constructed a general leakage-resilient multiparty computation (MPC) protocol with more strong security notion: an adversary obtaining leakage on the honest party’s secret state is guaranteed to learn nothing beyond the input and output of corrupted parties. Garg et. al. [13] first formulated leakage-resilient zero-knowledge proofs and provided a super-constant-round construction for HC problem. Following
showed a construction of leakage-resilient zero-knowledge proofs of knowledge. [20] presented a constant-round leakage-resilient zero-knowledge argument.

1.2 Our results

In this work, we adopt the basic model of [13] and present the first constant-round construction of LR-ZKAoK protocol.

**Main result:** Assume that collision-resistant hash functions and pseudorandom permutations exist, and DDH assumption holds. Then, there exists constant-round LR-ZKAoK system for HC.

The key problem in constructing LR-ZK protocol is how to respond to leakage queries in simulation such that the responses are consistent to the simulated view. To solve this problem, [13] adds a preamble to Blum’s 3-round zero knowledge proof. In the preamble, the verifier first commits to its random challenge $ch$ and a random string $r'$ using an extractable commitment scheme originally proposed by Prabhakaran et al. in [21], and then the prover sends a random string $r''$. In subsequent Blum’s protocol, the prover uses $r = r' \oplus r''$ as common random reference of Naor’s non-interactive commitment scheme to compute its commitments. There are many challenge-response slots in the extractable commitment scheme. This provides chances for the simulator to learn $ch$ and $r'$ by rewinding the verifier’s program. By Learning $ch$ in advance the simulator can prepare its commitment in Blum’s protocol according to $ch$ and then can answer this challenge correctly. By obtaining $r'$ the simulator can control $r$ and then have the ability to respond leakage queries under the help of the leakage oracle. The construction in [20] use the same idea but the preamble use non-black-box methods. In this approach, however, black-box knowledge extractors $K$ cannot work efficiently. Therefore, the protocols in [13, 20] are no longer a proof of knowledge.

To obtain LR-ZKAoK, we modified the construction in [20] such that the following two condition hold: (1) the challenge $ch$ is generated after the prover sends its commitments, as in [15, 16]; (2) the simulator has chance to control over the generated challenge and the prover cannot. Thus, the simulator can simulate the prover’s commitments according to a-priori randomly selected challenge and the prover cannot break the soundness. On the other hand, black-box knowledge extractors $K$ can work efficiently. To this end, we use a jointly coin-flipping protocol to generate $ch$ after the prover sends its commitments. In order to let the simulator have ability to control over $ch$, a preamble in which the simulator obtains the trapdoors is needed.

As in [20], we use Barak’s non-black-box techniques to construct such a preamble. The difference is that we use a new variant of Barak’s relation to define a NP-relation $R_{\text{sim}}$. In general Barak’s non-black-box simulation, the simulator needs the description of the verifier’s strategy, denoted as $desc(V)$, to accomplish a task that the prover cannot achieve no matter what it does. That is, the simulator’s advantage over the prover is that it is given the verifier’s program.

To improve the simulator’s ability, we permit the simulator to receive leakage queries apart from this advantage. We define a variant of Barak’s relation by simultaneously using $desc(V)$ and the received leakage queries. The simulator can take advantage of knowing $desc(V)$ and the leakage queries to obtain an instance $\sigma \in L_{R_{\text{sim}}}$ and a corresponding witness $w$ such that $R_{\text{sim}}(\sigma, w) = 1$, and no prover can obtain
σ ∈ L_{R_{sim}}. This advantage over the prover enables the simulator to control over ch and r. In the jointly coin-flipping protocols, we use the modified Yao’s garbled-circuit method, known as augmented garbled-circuit method, to achieve the conditional disclosure of a random secret selected by the verifier. Thus, the verifier needs not use a zero-knowledge proof to convince the prover that the coin-flipping protocol is executed honestly.

2 Preliminaries

In this paper, we use some standard notations. If \( A(\cdot) \) is a probabilistic algorithm, \( A(x) \) is the result of running \( A \) on input \( x \) and \( y = A(x) \) (or \( y \leftarrow A(x) \)) denote that \( y \) is set to \( A(x) \). For a finite set \( S \), we denote by \( y \in R S \) that \( y \) is uniformly selected from \( S \). \( U_l \) denotes uniform distribution on \( \{0, 1\}^l \). We write \([n] \) for any \( n \in \mathbb{N} \) to denote the set \( \{1, \cdots, n\} \) and \( poly(\cdot) \) to denote an unspecified polynomial.

We use the following standard definitions and tools.

2.1 Leakage-resilient zero-knowledge proof

We first recall the definitions of leakage-resilient zero-knowledge.

Let \( P \) and \( V \) be a pair of randomized interactive Turing machines, \( \langle P, V \rangle(x) \) be a random variable representing the local output of Turing machine \( V \) when interacting with machine \( P \) on common input \( x \), when the random input to each machine is uniformly and independently chosen. Customarily, machine \( P \) is called the prover and machine \( V \) is called the verifier. We denote by \( \langle P, V \rangle(x) = 1 \) (\( \langle P, V \rangle(x) = 0 \)) that machine \( V \) accepts (rejects) the proofs given by machine \( P \).

Definition 1. A pair of interactive Turing machines \( \langle P, V \rangle \) is called an interactive proof system for a language \( L \) if \( V \) is polynomial-time and the following two conditions hold:

- Completeness: there exists a negligible function \( c \) (known as completeness error) such that for every \( x \in L \),
  \[ \Pr[\langle P, V \rangle(x) = 1] > 1 - c(|x|) \]

- Soundness: there exists a negligible function \( s \) (known as soundness error) such that for every \( x \notin L \) and every interactive machine \( B \),
  \[ \Pr[\langle B, V \rangle(x) = 1] < s(|x|) \]

In the execution of interactive proofs, \( P \) has the ability to select a random coins \( r_i \) at the beginning of round \( i \) and uses it in round \( i \). In the interaction \( P \) synchronously update his current secret state. That is, let \( state \) denote \( P \)'s secret state (initialized to the private auxiliary input \( z \)), \( P \) updates \( state \) by setting \( state = state || r_i \) in round \( i \). The cheating verifier \( V^* \) launches a leakage attack by means of any number of arbitrary leakage queries throughout the interaction. Without loss of generality, we assume that the cheating verifier launches one and only one leakage query after receiving a message from the prover. The leakage query in round \( i \) is denoted as a
leakage function \( f_i \), to which the prover responds with \( f_i(state) \). We denote by \( \ell \) the total number of leaking bits, i.e. \( \ell = \sum_j |f_i(state)| \).

General zero knowledge is formalized by requiring that for any polynomial-time verifier \( V^* \) there exists a polynomial-time algorithm \( S \) (a.k.a. the simulator) such that the view of \( V^* \), which consists of the random tosses of \( V \) and the messages received by \( V \), can be simulated by \( S \). Under the leakage attack setting, leakage information \( f_i(state) \) obtained by \( V^* \) cannot be simulated by any polynomial-time algorithm \( S \) only upon common input \( x \). To formulate leakage-resilient zero-knowledge, \( S \) is given black-box access to an leakage oracle to respond the verifier’s leakage queries. The leakage oracle, denoted by \( L_n^\sigma(\cdot) \), is determined by the security parameter \( n \) and the auxiliary input \( z \) (the witness). A query to the oracle is an efficiently computable function \( f'_j(\cdot) \), to which \( L_n^\sigma(\cdot) \) responds with \( f'_j(z) \). We denote by \( \ell' \) the number of bits that \( S \) receives from the leakage oracle, i.e. \( \ell' = \sum_j |f'_j(state_j)| \). The leakage resilient zero-knowledge property essentially requires the condition \( \ell' \leq \ell \) holds. \cite{13} defined \((1+\varepsilon)\)-LR-ZKP that require a relaxed condition, \( \ell' \leq (1+\varepsilon)\ell \), holds for a-priori fixed \( \varepsilon > 0 \). This is reasonable and necessary for black-box zero knowledge. Recently, \cite{20} presented a non-black-box zero knowledge protocol that satisfies the condition \( \ell' \leq \ell \).

The leakage-resilient zero knowledge proofs require that no malicious verifier, launching leakage attack, can learn anything beyond the validity of the assertion and the leakage. This is formulated by requiring that for any polynomial-time verifier \( V^* \) with any auxiliary input \( aux \) there exists a polynomial-time algorithm \( S \) such that the output of \( S^{L_n^\sigma(\cdot)}(x, aux) \) is indistinguishable from the view of \( V^* \). Since this paper focuses on non-black-box zero knowledge, we refer to the definition of LR-ZKP in \cite{20}.

**Definition 2 (Leakage-Resilient Zero Knowledge).** An interactive proof system \((P,V)\) for a language \( L \) with a witness relation \( \mathcal{R} \) is said to be leakage-resilient zero knowledge if for every PPT machine \( V^* \) that makes any arbitrary polynomial number of leakage queries on \( P \)'s state (in the manner as described above), there exists a PPT algorithm \( S \) such that the following two conditions hold:

- For every \((x,z) \in \mathcal{R}, and every aux \in \{0,1\}^*\), \( \{View_V(\cdot, x, aux)\}_{x \in L, aux \in \{0,1\}^*} \) and \( \{S^{L_n^\sigma(\cdot)}(x, aux)\}_{x \in L, aux \in \{0,1\}^*} \) are computationally indistinguishable.

- For every \((x,z) \in \mathcal{R}, every aux \in \{0,1\}^*\), and sufficiently long \( r \in \{0,1\}^*\), it holds that \( \ell_S(r) \leq \ell_{V^*}(v) \), where \( \ell_S(r) \) is the number of bits that \( S \) receives from \( L_n^\sigma \) when generating the view \( v \), \( \ell_{V^*}(v) \) is total length of leakage answers that \( V^* \) receives in the view \( v \).

### 2.2 Proof of knowledge

In a proof of knowledge for a relationships \( R \), the prover, holding a secret input \( z \) such that \((x,z) \in R\), and the verifier interact on a common input \( x \). The goal of the protocol is to convince the verifier that the prover indeed knows such \( z \). This is in contrast to a regular interactive proof, where the verifier is just convinced of the validity of the statement.
The concept of “knowledge” for machines is formalized by saying that if the prover can convince the verifier, then there exists an efficient procedure that can “extract” a witness from this prover (thus the prover knows the witness because it could run the extraction procedure on itself).

**Definition 3.** An interactive protocol \( \langle P, V \rangle \) is a system of proofs of knowledge for a (poly-balanced) relation \( R \) with knowledge error \( \kappa \) if the following conditions hold:

- **(efficiency):** \( \langle P, V \rangle \) is polynomially bounded, and \( V \) is computable in probabilistic polynomial time.
- **(non-triviality):** There exists an interactive machine \( P \) such that for every \( (x, z) \in R \) all possible interactions of \( V \) with \( P \) on common input \( x \) and auxiliary \( y \) are accepting.
- **(validity with knowledge error \( \kappa \)):** Denote by \( p(x, y, r) \) the probability that the interactive machine \( V \) accepts, on input \( x \), when interacting with the prover specified by \( P^* \). There exists an expected polynomial-time oracle machine \( K \) and a polynomial \( q \) such that if \( p(x, y, r) > \kappa \), on input \( x \) and access to oracle \( P^* \), \( K(x) \) outputs \( z \), satisfying \( (x, z) \in R \), with probability of at least \( (p(x, y, r) - \kappa(|x|))/q(|x|) \)

### 2.3 Universal arguments

The notion of universal arguments, introduced by Barak and Goldreich in [3], is used to present interactive proofs for language

\[ L_{U_t} = \{(M, x, t) : \text{non-deterministic machine } M \text{ accepts } x \text{ within } t \text{ steps}\} \]

The corresponding relation is denoted by \( R_{U_t} \), that is, \( R_{U_t}((M, x, t), w) = 1 \) if and only if \( M \) (viewed here as a two-input deterministic machine) accepts \( (x, w) \) within \( t \) steps. We can handle all NP language by a universal argument for \( L_{U_t} \), because every language in NP is linear time reducible to \( L_{U_t} \).

For a super-polynomial function \( T : N \to N \), define the relation \( R_{U_t}^T \) as follows: \( R_{U_t}^T(y, w) = 1 \) if \( R_{U_t}(y, w) = 1 \) and \( t \leq T(|y|) \), where \( y = (M, x, t) \). The corresponding language is denoted by \( L_{U_t}^T \).

**Definition 4.** (Universal arguments)[3]. A universal-argument system for \( \text{Ntime}(T(n)) \) is a pair of strategies, denoted \( \langle P, V \rangle \), that satisfies the following properties:

- **Efficient verification:** There exists a polynomial \( p \) such that for any \( y = (M, x, t) \), the total time spent by the (probabilistic) verifier strategy \( V \), on common input \( y \), is at most \( p(|y|) \). In particular, all messages exchanged in the protocol have length smaller than \( p(|y|) \). (Here, \( |y| \) is assumed polynomial bounded).

- **Completeness by a relatively-efficient prover:** For every \( ((M, x, t), w) \) in \( R_{U_t}^T \),

\[ \Pr[(P(w), V)(M, x, t) = 1] = 1 \]

Furthermore, there exists a polynomial \( p \) such that the total time spent by \( P(w) \), on common input \( (M, x, t) \), is at most \( p(T_M(x, w)) \leq p(t) \), where \( T_M(x, w) \) denote the number of steps made by \( M \) on input \( (x, w) \).
Commitment schemes are used to enable a party, known as the sender, to commit itself to a value while keeping it secret from another party, called the receiver. This property is called hiding. Furthermore, the commitment is binding, and thus at a later stage when the commitment is opened, it is guaranteed that the “opening” can yield only a single value determined in the committing phase. We sketch the two properties. General definitions can be found in [5].

- **Statistically binding commitments**: the binding property holds against unbounded senders, while the hiding property only holds against computationally bounded receivers.
- **Statistically hiding commitments**: the hiding property holds against unbounded receivers, while the binding property only holds against computationally bounded senders.

In this paper, we use two types of commitment schemes. One is Naor’s statistically binding commitment scheme[18], denoted by $Comm_{sh}$. This scheme is constructed from pseudorandom generator $g : \{0,1\}^n \rightarrow \{0,1\}^3n$. Recall that in the commitment phase of this scheme, the receiver first selects $\tau \in_R \{0,1\}^{3n}$ and sends it to the sender. To commit to bit $b$, the sender selects $s \in_R \{0,1\}^n$, and then sends to the receiver $c = g(s)$ if $b = 0$, or $c = \tau \oplus g(s)$ otherwise. In the decommitment phase, the sender reveals $b$ and $s$.

Obviously, when $\tau \in \{g(s_0) \oplus g(s_1) : s_0, s_1 \in \{0,1\}^n\}$, the commitment to $b$ by sending $g(s_0)$ to the receiver can be decommitted to two different values. Concretely, the commitment $c$ can decommit to 0 by revealing $(0, s_0)$, or to 1 by revealing $(1, s_1)$. For convenience, we denote by $Comm_{sh}^\tau(v; r)$ the commitment to $v$ with the random coin $r$ when the receiver’s message is $\tau$. To commit to a string $r = r_1 \cdots r_k \in \{0,1\}^k$, the sender and the receiver run $k$ instances of this scheme in parallel. Concretely speaking, the receiver sends $\tau = \tau_1 \cdots \tau_k \in \{0,1\}^{k \cdot 3n}$, and the sender computes and sends $c_i = Comm_{sh}^\tau(r_i; s_i)$ to the receiver, where $s_i \in \{0,1\}^n$, $i = 1, \cdots, k$. Similarly, we denote by $Comm_{sh}^\tau(r; s = s_1 \cdots s_k)$ the commitment to $r = r_1 \cdots r_k \in \{0,1\}^k$.

The other is a two-round statistically hiding commitment scheme $Comm_{sh}(\cdot; \cdot)$. In such a scheme, the first message denoted by $m$ is from the receiver, and a corresponding commitment algorithm is denoted by $Comm_{sh}^m(\cdot; \cdot)$. In particular, this scheme can be constructed based on claw-free collections[14].
3 Tools

3.1 Barak’s non-black-box preamble

Barak’s non-black-box simulation method enable the simulator to have access to “trapdoor” without using rewinding technique. By this “trapdoor”, the simulator can simulate the real interaction. Concretely, the simulator take the advantage of holding the description of the verifier’s strategy $\text{desc}(V)$ to generate such a “trapdoor” by means of “generation protocol” and then it can work efficiently. The actual prover without $\text{desc}(V)$, however, cannot generate any “trapdoor”, and then the soundness of the protocol holds.

In the leaking setting, except fore interacting with $P$ according to the protocol, $V^*$ has the ability to obtain additional information $f(\text{state})$ (where state consists of the witness and all used random value so far) by launching a leakage attack. The prover, however, is not aware of the leakage attack. The leakage query $f(\cdot)$ is a part of the strategy of $V^*$, and so the messages sent by the verifier may depend on the leaking information $f(\text{state})$. Therefore, to show leakage-resilient zero knowledge property, the simulator should “correctly” respond $V^*$’s leakage queries. However, owing to without any witness, the simulator does not have the ability to reply directly to any leakage queries by itself. In order to solve this problem, [13] introduced the leakage-oblivious simulation, in which the simulator directly receives $V^*$’s leakage queries and responds each leakage query $f(\cdot)$ under the help of the leakage oracle holding the auxiliary input of $P$ (a witness for $x \in L$). More precisely, after receiving a leakage query $f(\cdot)$, the simulator constructs a new function $f'(\cdot)$ only on the witness, and query the leakage oracle with $f'(\cdot)$. Finally, $V^*$ receives $f'(z)$ directly from the leakage oracle.

Notice that, the simulator in leakage-oblivious simulation is authorized to receive the leakage query, and so it obtains an advantage over the prover who is not aware of the existence of the leakage attack. To use such the advantage, we consider a new variant of Barak’s relation.

For any $x \in L$, let $\Psi$ be the verifier’s interactive strategy. Let $\{H_n\}$ be a family of hash functions, where $h \in H_n : \{0, 1\}^* \rightarrow \{0, 1\}^n$. Let $T_n$ be a family of pseudorandom permutations on $\{0, 1\}^n$. Define relation $R$ as follows (depicted in Figure 1).

Figure 1: Relation $R$

Denote by $L_R$ the language corresponding to the variant of Barak’s relation $R$. Notice that $R$ is dependent on a given instance $x \in L$. Next, we present an instance
generation protocol, denoted by $\Phi$, for $L_R$ (depicted in Figure 2). It is similar to Barak’s generation protocol.

<table>
<thead>
<tr>
<th>Instance generation protocol $\Phi$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- $V$ selects $h \in R \mathcal{H}_n$, $t \in R T_n$, $\tau \in R {0,1}^{2n^3n}$, and then sends $(h,t,\tau)$ to $P$.</td>
</tr>
<tr>
<td>- $P$ selects $\eta \in R {0,1}^{2n^2}; s \in R {0,1}^{2n^2}$ and sends $c =\text{Comm}_{sb}(\eta; s)$ to $V$.</td>
</tr>
<tr>
<td>- $V$ selects $d \in R {0,1}^{6n^2}$ and sends it to $P$.</td>
</tr>
</tbody>
</table>

Figure 2: Generation protocol $\Phi$

The generation protocol $\Phi$ is sound if $V$ honestly executes the protocol.

**Lemma 1.** For any $T(n)^{O(1)}$-size $P$, let $\sigma = (h, t, \tau, c, d)$ be the transcript of execution of $\Phi$. If $V$ is honest, then the probability that $P$ outputs $w$ such that $R(\sigma, w) = 1$ is negligible.

**Proof.** Define the event $E$ to be “$P$ outputs a witness for $\sigma \in L_R$”, and let $\varepsilon = Pr[E]$. Suppose $H_{\text{set}} = \{(h, t, \tau) : Pr[E|(h, t, \tau)] \geq \varepsilon/2\}$. That is, for any $(h, t, \tau) \in H_{\text{set}}$, it holds that $Pr[E|(h, t, \tau)] \geq \varepsilon/2$. Then, the probability that $V$ randomly selects $(h, t, \tau)$ such that $(h, t, \tau) \in H_{\text{set}}$ is at least $\varepsilon/2$. Without loss of generality, we can assume that $P$ is deterministic. Thus, for any fixed $(h, t, \tau)$, $c$ and $z$ (a witness for $x$) are also fixed. It follows that $V$’s leakage query $f$ is determined if it exists. By the assumption that $P$ obtains a witness for $\sigma$ with probability $\varepsilon$, we have that $P^*$, with probability $\varepsilon^2/4$, obtains two witness respectively for $\sigma = (h, t, \tau, c, d)$ and $\sigma' = (h, t, \tau, c, d')$, where $d \neq d'$. This means that $P^*$ can find two different programs $\Psi, \Psi'$ and random values $\eta, \eta' \in \{0,1\}^n$, such that

$$\eta = (t(h(\Psi) \oplus u), u), \eta' = (t(h(\Psi') \oplus u'), u')$$

$$\text{Comm}_{sb}(\eta; s) = \text{Comm}_{sb}(\eta'; s')$$

By the statistically binding property of $\text{Comm}_{sb}()$, we have that $Pr[\eta \neq \eta']$ is negligible. This means that $Pr[t(h(\Psi) \oplus u) \neq t(h(\Psi') \oplus u')]$ and $Pr[u \neq u']$ all are negligible. It follows that $Pr[h(\Psi) \neq h(\Psi')]$ is negligible since $t$ is a permutation. Therefore, $P^*$ can find a collision for $h$ with a non-negligible probability if $\varepsilon$ is non-negligible. This contradicts with the assumption for $H_n$. $\blacksquare$

In a leaking setting where $V$ carries out leakage attacks, $P$ is not ware of any leakage attacks. If the simulator is allowed to see $V$’s leakage queries, it will get an extra advantage than $P$. In other words,

**Lemma 2.** For any $V^*$ that possibly launches leakage attack, there exists an oracle PPT algorithm $S_{V^*}$, on inputting the description of $V^*$ and being allowed to obtain $V^*$’s leakage queries, outputs $(v, w)$ such that the following conditions hold:

- $v$ is indistinguishable from the view of $V^*$ interacting with an honest $P$.
- Let $\sigma$ be the transcript that is contained in $v$. $w$ is a witness for $\sigma \in L_R$.

**Proof.** For any $V^*$, let $\Psi$ denote the description of $V^*$. Define $S_{V^*}$, which has black-box access to the leakage oracle, as follows:

- Uniformly select random coins $R_{V^*}$ for $V^*$.
Obtain \((h, t, \tau)\) by invoking \(V^*\).
- Randomly select \(u \in \{0,1\}^n\) and \(s \in \{0,1\}^{2n-n}\), compute \(c = \text{Comm}_{ab}(\eta; s)\), where \(\eta = (t(h(\Psi) \oplus u), u)\).
- Execute \(\Psi(c)\) and then obtain leakage query \(f_\Psi = \Psi(c)\). (If \(V^*\) does not launch leakage query, \(f_\Psi\) is the empty.)
- Query the leakage oracle with \(f_\Psi(\cdot, \eta, s)\) (if \(f_\Psi\) is not empty) and obtain \(\zeta = f_\Psi(\cdot, \eta, s)\).
- Invoke \(V^*\) with \((c, \zeta)\) and it returns \(d\).
- Output \(v = ((h, \tau, c, d, R_{V^*}), w = (\Psi, \zeta, s)\).

It follows from the hiding property of \(\text{Comm}_{ab}\) that \(v\) is indistinguishable from the view of \(V^*\). Obviously, \((h, t, \tau, c, d) \in L_R\) and the corresponding witness is \(w = (\Psi, u, \zeta, s)\). \hfill \blacksquare

**Remark:** Since the auxiliary input \(z\) (witness for \(x \in L\)) is not given to \(S_{V^*}\), \(S_{V^*}\) cannot compute \(f_\Psi(z, \eta, s)\) by itself even if \(S_{V^*}\) has obtained \(f_\Psi(\cdot)\). Fortunately, \(S_{V^*}\) here is given access to the leakage oracle. After obtaining \(f_\Psi(\cdot)\), \(S_{V^*}\) queries the leakage oracle with \(f_\Psi(\cdot, \eta, s)\) and then receives \(f_\Psi(z, \eta, s)\) from the leakage oracle.

The hardness condition of \(R\) holds against \(T(n)^{O(1)}\)-size adversaries. Barak in [3] used an error correcting code (ECC) to relax the assumption such that the hardness condition holds under the existence of the standard collusion-resistant hash functions. Let ECC be a polynomial-time computable function with polynomial-time encoding and with constant distance. If one replaces the condition \(\eta = (t(h(\Psi) \oplus u), u)\) with \(\eta = (t(h(ECC(\Psi))) \oplus u), u)\), the conclusion of lemma 1 holds for polynomial sized \(P^*\).

### 3.2 A new preamble

The fact that \(R\) can be verified in \(T(n) = n\log^\log n\) under the help of the leakage oracle implies that \(L_R\) does not lie in NP. Therefore, all the traditional proof systems for NP cannot deal with \(L_R\). Barak and Goldreich in [3] introduced universal arguments for the languages \(\text{Ntime}(T(n))\) and presented a 4-round public-coin protocol, denoted by \(\langle P_{ua}, V_{ua}\rangle\), for such languages. Their construction is as follows:

**Common input:** \((h, t, \tau, c, d)\).
- Security parameter \(1^n\).
- \(V_{ua}\) sends its first random massage \(\alpha\) to \(P_{ua}\).
- \(P_{ua}\) computes the response, denoted by \(\beta\), and sends it to \(V_{ua}\). Let \(l_1 = |\beta|\).
- \(V_{ua}\) sends its second random massage \(\gamma\) to \(V_{ua}\).
- \(P_{ua}\) computes the response, denoted by \(\delta\), and sends it to \(V_{ua}\). Let \(l_2 = |\delta|\).

The above system \(\langle P_{ua}, V_{ua}\rangle\) can be used to deal with \(L_R\) although the verification of the relation \(R\) needs access to the leakage oracle. Suppose that \(\text{Comm}(\cdot; \cdot)\) is a non-interactive statistically binding commitment scheme. By combining \(\text{Comm}(\cdot; \cdot)\) with universal argument \(\langle P_{ua}, V_{ua}\rangle\), one can construct an “encrypted” universal argument, denoted by \(\langle P_{ua}, V_{ua}\rangle\) for \(L_R\) (Figure 3).
‘Encrypted” universal argument \( \langle \tilde{P}_{ua}, \tilde{V}_{ua} \rangle \) for the statement that \( (h, \tau, c, d) \in L_R \).
- \( \tilde{V}_{ua} \) executes \( V_{ua} \) and sends \( \alpha \) to \( \tilde{P}_{ua} \).
- \( \tilde{P}_{ua} \) executes \( P_{ua} \) to computes \( \beta \), and then sends \( \tilde{\beta} = (t_\beta, \text{Comm}(t_\beta(r_\beta))) \) to \( \tilde{V}_{ua} \), where \( t_\beta \in R T_1, r_\beta \in \{0,1\}^* \).
- \( \tilde{V}_{ua} \) executes \( V_{ua} \) and then sends \( \gamma \) to \( \tilde{P}_{ua} \).
- \( \tilde{P}_{ua} \) executes \( P_{ua} \) to computes \( \delta \) and then sends \( \tilde{\delta} = (t_\delta, \text{Comm}(t_\delta(r_\delta))) \) to \( \tilde{V}_{ua} \), where \( t_\delta \in R T_2, r_\delta \in \{0,1\}^* \).

Figure 3: “Encrypted” universal argument \( \langle \tilde{P}_{ua}, \tilde{V}_{ua} \rangle \)

Next, we define the following relation \( R_{sim} \) corresponding to \( \langle \tilde{P}_{ua}, \tilde{V}_{ua} \rangle \) (depicted in Figure 4).

| Instance: \( \sigma = ((h, t, \tau, c, d), (\alpha, \tilde{\beta} = (t_\beta, \tilde{\beta}_1), \gamma, \tilde{\delta} = (t_\delta, \tilde{\delta}_1))) \). |
| Witness: \( (\beta, r_\beta, \delta, r_\delta) \). |
| Relation: \( R_{sim}(\sigma, (\beta, r_\beta, \delta, r_\delta)) = 1 \) if and only if the following conditions hold. |
| 1. \( \tilde{\beta}_1 = \text{Comm}(t_\beta(r_\beta)) \). |
| 2. \( \tilde{\delta}_1 = \text{Comm}(t_\delta(r_\delta)) \). |
| 3. \( V_{ua}(h, \tau, c, d), (\alpha, \beta, \gamma, \delta) = 1 \). |

Figure 4: Relation \( R_{sim} \)

Note that \( V_{ua} \) is efficient. It follows that \( R_{sim} \) is an NP-relation. The corresponding language is denoted \( L_{R_{sim}} \in NP \). We construct the following preamble \( \Sigma \) for \( L_{R_{sim}} \) (depicted in Figure 5).

| Preamble \( \Sigma \): |
| - \( P \) and \( V \) run \( \Phi \). Let \( \sigma' = (h, t, \tau, c, d) \) be the transcript. |
| - \( P \) and \( V \) run \( \langle \tilde{P}_{ua}, \tilde{V}_{ua} \rangle \) for \( \sigma' \). |
| - \( V \) sends \( V_{ua} \)’s first massage \( \alpha \) to \( P \). |
| - \( P \) sends \( \beta = (t_\beta, \text{Comm}(\beta; r_\beta)) \) to \( V \), where \( \beta \in R \{0,1\}^{k_1}, t_\beta \in R T_1, r_\beta \in \{0,1\}^* \). |
| - \( V \) sends \( V_{ua} \)’s second massage \( \gamma \) to \( P \). |
| - \( P \) sends \( \delta = (t_\delta, \text{Comm}(\delta; r_\delta)) \) to \( V \), where \( \delta \in R \{0,1\}^{k_2}, t_\delta \in R T_2, r_\delta \in \{0,1\}^* \). |

The transcript is denoted by \( \sigma'' = (\alpha, \tilde{\beta}, \gamma, \tilde{\delta}) \).

Figure 5: Preamble \( \Sigma \)

The preamble \( \Sigma \) is sound for \( L_{R_{sim}} \). For any \( P^* \) interacting with \( V \), let \( \sigma \) be the transcript of the protocol, then the probability that \( P^* \) obtains a witness for \( \sigma \in L_{R_{sim}} \) is negligible.

**Lemma 3.** No matter how \( P^* \) does, the probability that \( P^* \) obtains a witness \( w \) such that \( R_{sim}(\sigma, w) = 1 \) is negligible when \( V \) executes the protocol honestly, where \( \sigma \) is the transcript of protocols.

**Proof.** Let \( \sigma = (\sigma', \sigma'') \), where \( \sigma' \) and \( \sigma'' \) are the transcript \( \Phi \) and \( \langle \tilde{P}_{ua}, \tilde{V}_{ua} \rangle \), respectively. Assume the probability that, at the end of \( \Sigma \), \( P^* \) obtains \( w \) such that \( R_{sim}(\sigma, w) = 1 \) is non-negligible. Notice that \( w = (\beta, r_\beta, \delta, r_\delta) \), meeting with
efficiently verifiable range. Such an encryption scheme can be constructed from a family under chosen-plaintext attacks. In addition, assume that \((G,E,D)\) is the idea of garbled-circuits is that one party generates an "garbled-circuit" of every gate in the circuit 
\(\mathcal{P}\) such that \(\mathcal{R}(\sigma', \omega') = 1\) with a non-negligible probability. By lemma 1, this is impossible.

3.3 Augmented garbled-circuit

Garbled circuit method allows two parties, \(P_0\) and \(P_1\), to evaluate an arbitrary function \(f(x,y)\) on their respective inputs \(x = x_1 \cdots x_n\) and \(y = y_1 \cdots y_n\), without leaking any information about their inputs beyond what is implied by \(f(x,y)\).

Let \(C\) be a acyclic boolean circuit such that \(C(x,y) = f(x,y)\) for any \(x, y\). The basic idea of garbled-circuits is that one party generates an "garbled-circuit" \(\hat{C}\) of \(C\), the second party then obliviously computes the output without learning any intermediate values.

Suppose that \(e = \text{poly}(n)\) is the number of wires (including input wires of \(C\) and output wires of every gate in the circuit \(C\)). All the wires are labeled by \(W_1, \ldots, W_e\), where \((W_1, \ldots, W_n)\) and \((W_{n+1}, \ldots, W_{2n})\) correspond to \(x\) and \(y\) respectively. Let \(\text{OUT} = \{W_{e-k+1}, \ldots, W_e\}\) be all the output wires and \(C_g\) be all gates in \(C\). Let \((G, E, D)\) be a symmetric encryption scheme that has indistinguishable encryptions under chosen-plaintext attacks. In addition, assume that \((G, E, D)\) has elusive and efficiently verifiable range. Such a encryption scheme can be constructed from a family of pseudorandom functions. One party garbles the circuit \(C\) as follows:

1. For each wire \(W_i\), choose two independent keys \((k_0^i, k_1^i)\) by a algorithm \(G(1^n)\).
2. Given these keys, the four garbled values of each gate \(g\), with input wires \(W_i, W_j\) and output wire \(W_m\), are computed as follows:

\[
c_{a, b} = E_{k_0^g}(E_{k_0^h}(k_0^{a,b}, a = 0, 1; b = 0, 1)
\]

where the inputs of \(W_i\) and \(W_j\) are \(a\) and \(b\) respectively, the output of \(g\) is \(g(a, b)\).

Let \(T_g = (c_{0,0}, c_{0,1}, c_{1,0}, c_{1,1})\).

3. For each \(T_g\), randomly choose a permutation \(\pi_g\) and set \(T^*_g = \pi_g(T_g) = (c_0, c_1, c_2, c_3)\).

Let \(T^* = \{T^*_g\}_{g \in C_g}\).

4. Construct decryption tables \(T_{\text{dec}} = \{(0, k_0^i, 1, k_1^i)\}_{i = e-k+1}^{e}\), and define garbled-circuit \(\hat{C} = (T^*, T_{\text{dec}})\).

Garbled circuit method is formally denoted by two PPT algorithms \((\text{Garble, Eval})\). On inputting \(1^n\) and \(C\), algorithm \(\text{Garble}\) first executes the above process and then outputs \((\hat{C}, K_0, K_1)\), where \(K_0 = \{k_{0,1}^i\}_{i=1}^n\) is the keys of the input wires of \(x\), \(K_1 = \{k_{0,1}^i\}_{i=1}^n\) is the keys of the input wires of \(y\). Let \(K_x = \{k_{0,1}^i\}_{i=1}^n, K_y = \{k_{1,1}^i\}_{i=n+1}^{2n}\). Algorithm \(\text{Eval}\), on inputting \(K_x, K_y\) and \(\hat{C}\), outputs \(C(x,y) = f(x,y)\). Roughly speaking, for every gate \(g\) in \(C\) with input wires \(W_i, W_j\) and output wire \(W_m\), \(\text{Eval}\) uses the keys \(K_g^0\) and \(K_g^1\) respectively corresponding \(W_i, W_j\) to decrypt.
\( T^*_y = \pi_y(T_y) = (c_0, c_1, c_2, c_3) \). If more than one decryption or no decryption succeed, then fails and stops. Otherwise, obtain a key \( K^{(a,b)}_m \) corresponding \( W_m \). Using \( K_x, K_y \) and starting from the input gates, \( Eval \) can obtain the keys of all the output wires and return \( C(x, y) \) by requiring the decryption tables.

Notice that if \( \pi_y \) is omitted in garbling \( C \), that is, \( T^*_y = (c_0, 0, c_1, 0_1, c_1, 1_1) \), the keys \( K^a_i \) and \( K^b_j \) (corresponding the inputs wires of the gate \( g \)) in fact will reveal the real inputs \( a \) and \( b \), since the subscript of the cipher that can be decrypted is \((a, b)\).

Here, assumed that \(|x| = |y| \) for simplicity. In fact, this condition is unnecessary.

Garbled circuit method presents an approach for secure two-party computation. Let \( OT \) be 2-round 1-out-of-2 oblivious transfer protocol. Yao's two-party protocol is described as follows:

<table>
<thead>
<tr>
<th>Yao's protocol ( \Pi_{cd} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common input:</strong> ( C(x, y) = f(x, y), ) security parameter ( n ) and ( m ) of ( Comm^{n}_{sh}(\cdot) ).</td>
</tr>
<tr>
<td><strong>Private inputs:</strong> ( P_0 )'s input: ( x ); ( P_1 )'s input: ( y ).</td>
</tr>
<tr>
<td>(- P_0 ) prepares a garbled-circuit ((\hat{C}, K_0, K_1)) by ((\hat{C}, K_0, K_1) \leftarrow Garble(C, 1^n)) for the ( C(x) ) as above. Then ( P_0 ) sends ( \hat{C} ) and ( K_x ) to ( P_1 ).</td>
</tr>
<tr>
<td>(- P_0 ) and ( P_1 ) execute ( OT ) such that ( P_1 ) obtains ( K_y ).</td>
</tr>
<tr>
<td>(- P_1 ) computes ( v = (v_1, \ldots, v_n) ), where ( v_i ) is the first message of ( OT ) protocol using the input ( y_i ) and fresh randomness ( r_i ) for ( i \in [n] ). It then sends ( v ).</td>
</tr>
<tr>
<td>(- P_0 ) prepares ( v' = (v'<em>{1,1}, \ldots, v'</em>{n,1}) ), where ( v'<em>{i,1} ) is the second message of ( OT ) when taking ((k^{0}</em>{1,1}, k^{1}_{i,1})) as sender's input and ( v_i ) as receiver's first message. Finally, ( P_0 ) sends ( v' ).</td>
</tr>
<tr>
<td>(- P_1 ) computes ( K_y ) from ( v, v' ).</td>
</tr>
<tr>
<td>(- P_1 ) obtains ( f(x, y) ) by computing ( Eval(\hat{C}, K_x, K_y) ).</td>
</tr>
</tbody>
</table>

The construction of LR-ZKA in [20] used Yao’s protocol to compute a conditional disclosure function such that the verifier reveals a random string only if the prover holds some special “witness”. In order to force the verifier to compute conditional disclosure function honestly, a public-coin statistical ZKAoK, in which the verifier plays the role of a prover, follows this conditional disclosure.

Let \( R_{sim} \) be as above and \( \sigma \in \{0, 1\}^* \) be the transcript of \( \Sigma \). As in [20], we use Yao’s protocol \( \Pi_{cd} \) to compute the following conditional disclosure function:

\[ f_{\sigma, R_{sim}}(r, w) = \begin{cases} r, & R_{sim}(\sigma, w) = 1 \\ 0|v|, & \text{others} \end{cases} \]

The common input to \( P_0 \) and \( P_1 \) is the description of the function \( f_{\sigma, R_{sim}} \) or a circuit \( C \) computing \( f_{\sigma, R_{sim}} \).

\( P_0 \) (known as a sender, denoted by \( S_{cd} \)) randomly selects input \( r \in \{0, 1\}^{poly(n)} \) and want to disclose \( r \) to \( P_1 \) (known as a receiver, denoted by \( R_{cd} \)). The receiver holding \( w \) obtains \( r^* \) after the computation has been performed. If \( R(\sigma, w) = 1 \) then \( r^* = r \), otherwise \( r^* = 0|v| \). That is, the sender discloses \( r \) to the receiver if and only if \( R_{sim}(\sigma, w) = 1 \). Here, we will use Naor’s 2-round OT protocol (presented in [19], denoted as \( OT_{NP} \)) in \( \Pi_{cd} \). \( OT_{NP} \) is secure if the DDH assumption holds. In addition, \( OT_{NP} \) holds the following properties: 1) the sender’s security is statistical; and 2)
the receiver’s first message is indistinguishable from one selected uniformly from the specific message space. It follows from [17] that $\Pi_{ed}$ securely computes the conditional disclosure function $f_{\pi,R_{sim}}(r, w)$. Specially, the receiver taking $w$ as the input does not obtain any information about the sender’s input $r$ except for $f_{\pi,R_{sim}}(r, w)$. Therefore, the receiver can only obtain $0^{\lvert r \rvert}$ when $w$ is not a witness for $\sigma \in L_{R_{sim}}$.

The computation of the conditional disclosure function $f_{\pi,R_{sim}}(r, w)$ in our construction of LR-ZKAoK, in fact, is a commitment stage, where the verifier acts as a sender to commit to a random value $r$. If the prover does not have any witness for $\sigma \in L_{R_{sim}}$, it only obtains $0^{\lvert r' \rvert}$ at the end of this commitment stage. So, the hiding property holds when the prover does not have any witness. In later stage when receiving a random value $r''$ from the prover, the verifier will reveal $r'$. Finally, the prover and the verifier obtain a common random value $r = r' \oplus r''$. Obviously, After receiving the revealment to $r'$ from the verifier the prover cannot verify whether it is correct. That is, the binding property is in doubt.

The binding property is unnecessary to real interaction, since the prover completely ignores the computation of $f_{\pi,R_{sim}}(r', w)$ and accepts any $r'$ revealed by the verifier. In simulation, however, simulator needs to learn $r'$ by computing $f_{\pi,R_{sim}}(r', w)$ such that $r' \oplus r''$ equals a specified value by selecting a special $r''$. Therefore, if the verifier does not correctly reveal $r'$, the simulator will fail even though the prover does not abort in that case. In order to ensure that the simulator fails if and only if the prover aborts in real interaction, the prover needs to check weather the verifier correctly reveal $r'$ and aborts when the verifier does not. Therefore, we need to modify the algorithm Garble by replacing the step 3 of garbling circuit with the following process.

1. For each $T_g$, choose randomly permutation $\pi_g$, and computes $C_{T_g} = \text{Comm}^{\text{un}}_{s_i}(\pi_g)$ and $T_g^* = \pi_g(T_g) \triangleq (c_0, c_1, c_2, c_3)$. Let $T^* = \{T_g^*\}_{g \in C_0}$ and $C_T = \{C_{T_g}\}_{g \in C_0}$.

The augmented garbled-circuit is $\hat{C} = (T^*, C_T, T_{dec})$. It is easy to see that the modified garbled-circuit can be evaluated by the algorithm $\text{Eval}$ when given $K_x, K_y$.

Therefore, Yao’s protocol $\Pi_{ed}$ works well in the context of $\hat{C} = (T^*, C_T, T_{dec})$ and the proved security has not been impacted.

On the other hand, $C_T$ in $\hat{C} = (T^*, C_T, T_{dec})$ contains the commitments of all $\pi_g$’s and can be used to verify the computation of $C(x, y)$. Recall that the receiver computes the garbled-circuit $\hat{C}$ of $C$ by executing $\text{Eval}(\hat{C}, K_x, K_y)$. Starting from $K_x, K_y$, every gate in $C$ can be “decrypted” correctly. Concretely, for every gate $g$ $\text{Eval}$ uses the keys corresponding the input wires of $g$ to decrypt $T_g^* = \pi_g(T_g) = (c_0, c_1, c_2, c_3)$ and then obtain the key corresponding the output wires of $g$. Notice $\pi_g$ is a permutation over $T_g = (c_{0,0,0,0,1,1,0,1,1})$, where $c_{a,b} = E_{k_{\pi}^a}(E_{k_j^b}(k_{\pi}^{a,b}))$. Therefore, if the receiver obtains $\pi_g$, it can recover $T_g = (c_{0,0,0,0,1,1,0,1,1})$ by computing $\pi_g^{-1}(T_g^*)$. By requiring which one of $T_g$ can be decrypted correctly, the receiver can obtain the inputs of $g$. Therefore, if the sender opens correctly all $\pi_g$’s, the receiver can verifies the sender’s input $x$ and $C(x, y)$.

In our construction, the prover and the verifier (playing the role of sender) jointly compute $f_{\pi,R_{sim}}(r', w)$ by the augmented garbling circuit. This sub-protocol, in substance, is a commitment scheme in which the verifier commits to a random value $r'$. 

In subsequent opening phase, the verifier is required to reveal $r'$ and all the commitments to $\pi_g$'s. Since by $\pi_g$'s the prover can verify weather the sender correctly reveal $r'$, the binding property holds since $Comm_{sh}^n$ is computationally binding.

4 constant-round LR-ZKAoK for HC

Let $HC$ be all directed graphs containing Hamiltonian cycles. Our goal in this section is to construct LR-ZKAoK for NP-relation $R_{HC}$. Recall Blum's protocol for $HC$.

Common input: $G = (V, E) \in HC$ (containing a Hamiltonian cycle $H$), $|V| = n$.

- Prover’s first step(P1): The prover selects a random permutation $\pi$ over $V$, and sends the commitmentsto the adjacency of $G_i = \pi(G)$ to the verifier.
- Verifier’s first step(V1): The verifier uniformly selects a challenge $ch \in_R \{0, 1\}$ and sends it to the prover.
- Prover’s second step(P2): If $ch = 1$, the prover reveals some commitments corresponding to the edges of the Hamiltonian cycle $\pi(H)$. If $ch = 0$ then the prover reveals all the commitments and $\pi$.
- Verifier’s second step(V2): The verifier $V$ checks $P$’s revealment.

Garg et al. in [13] first presented a LR-ZKP system for NP by adding a preamble to the paralleled Blum’s protocol. In the preamble, the verifier first commits to its random challenge $ch$ and a random string $r'$ using a extractable commitment scheme, and then the prover sends a random string $r''$. In subsequent Blum’s protocol, the prover uses $r = r' \oplus r''$ as the first random message of Naor’s commitment scheme to compute its commitments. From the extractability of the commitment scheme, the simulator can learn $ch$ and $r'$ in advance by rewinding the verifier’s strategy. Therefore, the simulator can answer this challenge correctly and has the ability to respond leakage queries under the help of the leakage oracle. More precisely, their construction consist of three stages. In Stage 1, $V$ commits to its challenge $ch$ and a random string $r'$ using a public-coin statistically hiding commitment scheme, followed by many challenge-response slots. In Stage 2, $P$ selects a random string $r''$ and $V$ reveals $r'$. And then, $P$ and $V$ compute $r = r' \oplus r''$. Finally, in Stage 3, $P$ and $V$ run $n$ parallel repetitions of Blum’s protocol. In the first round, $P$ uses Naor’s two-round commitment scheme, using $r$ as the first message, to commit to the adjacency of the permuted graphs. In the second round, $V$ reveals the commitment to its challenge $ch$. Finally, $P$ responds to this challenge and $V$ verifies the response. The construction of [13] is a black-box LR-ZKP, since the simulator needs to rewind the verifier’s program $V^*$ such that it can extract the challenge $ch$ and force $r$ to a special distribution.

In the model of [13], cheating verifiers are allowed to launch a leakage query every time the prover sends its message. To obtain $ch$ and $r'$, the (black-box) simulator needs to rewind the challenge-response many times. In each rewinding, the simulator must inquire the leakage oracle again. Therefore, rewinding will increase the number of bits that the simulator receives from the leakage oracle. In order to allow the simulator to obtain $ch$ and $r'$ with minimal increase of the number of bits from the leakage oracle, the number of challenge-response slots in Stage 1 is not a constant. This results in the presented construction is super-constant-round.
The LR-ZK property requires that for any $V^*$ there exists a simulator (with a leakage oracle) that can simulate the view of $V^*$, while proofs of knowledge require that there exists a (black-box) knowledge extractor $K$ to extract a witness from $P^*$ if $P^*$’s proof is accepted. In the protocol of [13], $V$ commit to its challenges $ch$ before $P$ sends its commitment. This implies that the permutation that $P$ use to compute its commitments may depend on the challenge selected by $V$. This results in that the black-box knowledge extractors cannot work efficiently. Therefore, the protocol in [13] is no longer a proof of knowledge. The approach of requiring the verifier to commit to its challenges $ch$ in advance can ensure LR-ZK property, but then seemingly destroy the proof of knowledge property.

Recently, [20] presented a constant round leakage-resilient zero-knowledge proof for HC by combining the idea of [13] with Barak’s non-black-box simulation technique. Roughly speaking, the protocol of [20] replaced the stage 1 of the protocol of [13] with a constant round preamble consisting of four stage such that the simulator can extract the challenge $ch$ and force $r$ to a special distribution of its choice without rewinding the verifier’s algorithm $V^*$. Unfortunately, the protocol of [20] is not a proof of knowledge because of the same reason as mentioned above.

The key problem in constructing LR-ZKP is how to respond leakage queries in simulation such that the responses are consistent to the simulated view. To obtain LR-ZK AoK, we modified the construction in [20] such that the following two conditions hold: (1) the challenge $ch$ is generated after the prover sends its commitments, as in [15, 16]; (2) the simulator have chance to control over the generated challenge and the prover cannot. Thus, the simulator can simulate the prover’s commitments according to a-priori randomly selected challenge. On the other hand, black-box knowledge extractors $K$ can work efficiently. To this end, we use a jointly coin-flipping protocol to generate $ch$ after the prover sends its commitments. In addition, we also need a preamble in which the simulator obtains “trapdoors”. As in [20], we use Barak’s non-black-box techniques to construct such a preamble. The difference between this preamble and that in [20] is that it use a new variant of Barak’s relation to define a NP-relation $R_{sim}$. In general Barak’s non-black-box simulation, the simulator needs the description of the verifier’s strategy to accomplish a task that the prover cannot. That is, the simulator’s advantage over the prover is that it is given the verifier’s program. Notice that apart from the description of the verifier strategy is allowed to see the verifier’s leakage queries. We define a variant of Barak’s relation by simultaneously using the description of the verifier’s strategy and leakage queries. The simulator can take advantage of knowing the description of the verifier’s strategy and leakage queries to obtain an instance $\sigma \in L_{R_{sim}}$ and a corresponding witness $w$ such that $R_{sim}(\sigma, w) = 1$, and no prover can obtain $\sigma \in L_{R_{sim}}$. Therefore, the protocol is sound and meanwhile the simulator have the ability to control over $ch$ and $r$.

Our protocol, denoted by $\Pi$, consists of 5 stages (depicted in Figure 6). In Stage 0, $P$ and $V$ run preamble $\Sigma$. Stage 1 is a coin-flipping protocol to produce a random string $r = r' \oplus r''$, where $r'$ and $r''$ are randomly picked by $P$ and $V$ respectively. In this stage, $V$ first commits to a random string $r'$ by means of computing the conditional disclosure function $f_{\sigma, R_{sim}}(r', w)$. Here, the augmented garbled-circuit $\tilde{C} = (T^*, C_T, T_{dec})$ is used to achieve the conditional disclosure of a random secret $r'$. 
Subsequently, $P$ sends a random string $r''$, and then $V$ sets $r = r' \oplus r''$ and reveals $r'$ along with and the all permutations committed in $C_T$. If $r'$ pass the verification, $P$ compute $r = r' \oplus r''$. In Stage 2, the prover commits $n$ random permuted graphs using Naor’s commitment scheme with $r$ as the first message. Stage 3 is another coin-flipping protocol to produce a random challenge $ch$. Similarly as in Stage 1, the augmented garbled-circuit method is used to achieve the conditional disclosure of a random secret $ch_V$ selected by $V$. And then, $P$ randomly selects $ch_P$ and $V$ reveals $ch_V$. If $ch_V$ is passed the verification, the two parties obtain a common random string $ch = ch_P \oplus ch_V$. In Stage 4, $P$ uses $ch$ as challenge to execute prover’s second step of Blum’s protocol P2. Then, $V$ executes verifier’s second step of Blum’s V2 with $ch$.

<table>
<thead>
<tr>
<th>II: LR-ZK AoK for HC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common input:</strong> $G=(V,E) \in HC$, $</td>
</tr>
<tr>
<td><strong>Auxiliary input to the prover:</strong> A Hamilton cycle $H$ in $G$.</td>
</tr>
<tr>
<td>- Stage 0: Preamble.</td>
</tr>
<tr>
<td>- $P$ and $V$ run $\Sigma$. The transcript is denoted by $\sigma=(\sigma',\sigma'')$.</td>
</tr>
<tr>
<td>- Stage 1: Jointly coin-tossing.</td>
</tr>
<tr>
<td>- $P$ sends the first random message $m$ to $V$. (The first message of the statistically hiding commitment scheme $Comm_{sh}^n(\cdot)$).</td>
</tr>
<tr>
<td>- $V$ (the sender) selects $r' \in {0,1}^t$, and run $\Pi_{cd}$ with $P$ to compute $f_{\pi,\tau_{im}}(r',\rho)$, where $\rho$ is selected by $P$. Finally, $P$ obtains $r$.</td>
</tr>
<tr>
<td>- $P$ selects and sends $r'' \in_R {0,1}^t$ to $V$.</td>
</tr>
<tr>
<td>- $V$ opens $r'$ and the commitments in $C_T$ computed in $\Pi_{cd}$. $V$ sets $r = r' \oplus r''$.</td>
</tr>
<tr>
<td>- $P$ verifies $r'$ (verifying the decommitment to $C_T$ and that $r'$ is V’s input in $\Pi_{cd}$) and aborts if the verification fails. Otherwise, $P$ sets $r = r' \oplus r''$.</td>
</tr>
<tr>
<td>- Stage 2: The prover’s commitment.</td>
</tr>
<tr>
<td>- Suppose $r = r_1 \cdots r_n$, where $r_i = r_{i,1} \cdots r_{i,n^2}$ satisfying $</td>
</tr>
<tr>
<td>- Stage 3: Jointly coin-tossing:</td>
</tr>
<tr>
<td>- $V$ (the sender) selects $ch_V \in {0,1}^n$ and and run $\Pi_{cd}$ with $P$ (the receiver) to compute $f_{\pi,\tau_{im}}(ch_V,\rho)$, where $\rho$ is selected by $P$. Finally, $P$ obtains $ch_V$.</td>
</tr>
<tr>
<td>- $P$ selects and sends $ch_P \in_R {0,1}^n$ to $V$.</td>
</tr>
<tr>
<td>- $V$ opens $ch_V$ and the commitments in $C_T$. $V$ sets $ch = ch_P \oplus ch_V$.</td>
</tr>
<tr>
<td>- $P$ verifies $ch_V$ (as the verification of $r'$ in Stage 1) and aborts if the verification fails. Otherwise, $P$ sets $ch = ch_P \oplus ch_V \overset{\Delta}{=} ch_1 \cdots ch_n$.</td>
</tr>
<tr>
<td>- Stage 4: $P$ responds to the challenge $ch$, and $V$ verifies this response.</td>
</tr>
<tr>
<td>- $P$ reveals some commitments in $C_i$ corresponding to the edges of the Hamiltonian cycle $\pi_i(H)$ if $ch_i = 1$, or reveals all the commitments in $C_i$ and $\pi_i$ if $ch_i = 0$.</td>
</tr>
<tr>
<td>- $V$ verifies the decommitments according to $ch$.</td>
</tr>
</tbody>
</table>

Figure 6: LR-ZK AoK
The presented protocol uses non-black-box techniques such that the total of the leakage information received by the simulator is as much as the verifier. Concretely, since the simulator holds the description of the verifier’s strategy $V^*$ and is allowed to receive leakage queries from $V^*$, it can obtain a witness for $\sigma \in L_{R_{sim}}$, where $\sigma$ is the transcript of running $\Sigma$. By this witness, the simulator has the ability to force $r$ to a special distribution in Stage 1 and ensure $ch$ equal to a-priori randomly selected value. This results in that the simulator can interact with $V^*$ in later stages as an honest prover and correctly respond to $V^*$’s leakage queries.

**Theorem 1.** Let $\mathcal{H}$ be a family of collision-resistant pseudorandom hash function and $\mathcal{T}$ be a family of pseudorandom permutation. Assume that DDH assumption holds. Protocol $\Pi$ is a LR-ZK AoK for $HC$ with knowledge error $\kappa = 2^{-n}$, if $Comm_{sb}(\cdot;\cdot)$ is a Naor’s 2-round statistically binding commitment scheme and $Comm_{sh}(\cdot;\cdot)$ is 2-round statistically hiding commitment scheme.

**Proof.** Completeness: Completeness is obvious. Soundness: Suppose $G \notin HC$. For any $P^*$, let $\varepsilon^*$ be the probability that $P^*$ successfully cheats $V$. By Lemma 3, we have that for any $P^*$ the probability that $P^*$ holds a witness for $\sigma \in L_{R_{sim}}$ at the end of Stage 0 is negligible. Next, we need to show that the protocol is statistically binding.

To show this, define
$$\mathcal{B} = \{ r : \exists s_0, s_1 \in \{0,1\}^n, \text{ such that } r = g(s_0) \oplus g(s_1) \}$$
for $g : \{0,1\}^n \rightarrow \{0,1\}^{3n}$ (pseudorandom generator used in Naor’s commitment). Let $r''$ be a string such that $P^*$ sends to $V$ in Stage 1. Then, after $V$ open $r'$, $P^*$ obtains $r = r' \oplus r'' = r_{1,1} \cdots r_{n,n^2}$ at the end of Stage 1. We show the following lemma holds.

**Lemma 4.** No matter what $P^*$ does, the probability that there exist $i \in [n]$ and $j \in [n^2]$ such that $r_{i,j} \in \mathcal{B}$ is negligible, if $V$ honestly executes the protocol.

**Proof.** Denote by $E_1$ the event that there exist $i \in [n]$ and $j \in [n^2]$ such that $r_{i,j} \in \mathcal{B}$. Denote by $E_2$ the event that $P^*$ holds a witness for $\sigma \in L_{R_{sim}}$ at the end of Stage 0.

$$Pr[E_1] = Pr[E_1 \land E_2] + Pr[E_1 \land \overline{E_2}] \leq Pr[E_2] + Pr[E_1|\overline{E_2}]$$

First, it follows from Lemma 3 that $Pr[E_2]$ is negligible. Next, under the condition that $P^*$ does not obtain a witness for $\sigma \in L_{R_{sim}}$ in Stage 0, $P^*$ only obtains 0 by means of executing $II_{cd}$ and cannot learn $r'$ before sending $r''$ to $V$. Therefore, $r = r' \oplus r''$ is uniformly distributed on $\{0,1\}^t$ except for a negligible probability if $V$ is honest. It follows that $Pr[E_1|\overline{E_2}]$ is negligible. Therefore, $Pr[E_1]$ is negligible. □

By the same reason as above, the following lemma holds.

**Lemma 5.** For any priori fixed $ch' \in \{0,1\}^n$, the probability that $P^*$ successfully selects $ch_P$, such that $ch = ch'$ at the end of Stage 3 is negligible, if $V$ honestly executes the protocol.

By Lemma 4, $r$ generated in Stage 1 is uniformly distributed except for a negligible probability. So, the commitment scheme Naor’s $Comm_{sb}(\cdot;\cdot)$ is statistically binding. If $G \notin HC$, by the binding property of $Comm_{sb}(\cdot;\cdot)$, we have that, except for a
must determine all random value of Naor’s commitment scheme in order to response to the leakage query in Stage 2. The reason for this is that, in Stage 2, S (Stage 2) can be opened in two different manner. In Stage 2 and Stage 4, S obtains the response to it is denoted as $\delta_S$. process between $\delta_S$ and $\delta_P$ such that the generated challenge $\delta_S$ is equal to a specified one which depend on $\delta_P$. Let $\delta_S$ and $\delta_P$ jointly generate $\delta, \delta_P, \delta_V$ before the challenge $\delta$ is generated. Therefore, if V is convinced of $G \in HC$ with a noticeable probability, then $\delta^*$ is able to select $\delta_P$ such that $\delta_P \oplus \delta_V = \delta'$ in Stage 3. By Lemma 5, this is impossible. And then we have that $\epsilon^*$ is negligible.

**Leakage-resilient zero knowledge:** To prove leakage-resilient zero knowledge property, we need to construct a simulator which not only output a simulated transcript but also responds $V^*$’s leakage query. The key problem is that the leakage must conform to the simulated transcript.

We first show how the simulator answers the verifier’s leakage queries. The simulator is permitted to directly receive leakage query $f(state)$ from $V^*$, where state consists of the witness and all used random value so far. Without holding any witness, however, the simulator cannot respond to leakage query $f$ directly. It is just well that the simulator has access to a leakage oracle $L^*_z$ that holds the witness $z$. To respond to a leakage query $f$, the simulator construct a new function $f'$ such that $f'(z)$ is indistinguishable from $f(state)$, and then sends $f'$ to the leakage oracle $L^*_z$. The leakage oracle $L^*_z$ leaks $f'(z)$ directly to verifier $V^*$, and does not leak any information to the simulator.

Notice that leakage query $f$ takes the form of $f(z,R)$, where $z$ is the prover’s witness (a Hamiltonian cycle $H$ in $G$), $R = R(z)$ is a function that, on inputting the prover’s witness $z$, outputs the prover’s random coins. To create a new query $f'$ after receiving a query $f(z,R(z))$ from the verifier, the simulator first selects an appropriate random coins $\tilde{R}$, and then presents a strategy based on $z$ to select $R'$ from $\tilde{R}$ such that $f(z,R')$ is indistinguishable from $f(z,R)$. Finally, define $f'(z) = f(z,R')$.

**Non-black-box simulator:** given desc($V^*$) and permitted to receive $V^*$’s leakage query, the non-black-box simulation is very straightforward. Roughly speaking, the simulator $S$ plays the role of prover to interact with $V^*$ and then outputs a simulated transcript. Concretely, in Stage 0, $S$ can obtain a instance $\sigma = (\sigma', \sigma'')$ and a corresponding witness $w = (\beta, r_\beta, \delta, r_\gamma)$, where $\sigma' = (h, \tau, t, c, d), \sigma'' = (\alpha, \beta, \gamma, \delta), \sigma = (\sigma', \sigma'') \in L_{R,sim}$. In Stage 1, $S$ and $V^*$ jointly generate $r$, which will be used as the first message of Naor’s commitment scheme used in Stage 2. Using the witness $w$, $S$ is able to select a special $r$ in Stage 1, such that all the commitments generated in Stage 2 can be opened in two different manner. In Stage 2 and Stage 4, $S$ acts as an honest prover. $S$ completes Stage 3 using the same strategy as that in Stage 1, such that the generated challenge $\delta$ is equal to a specified one which depend on $S$’s response to the leakage query in Stage 2. The reason for this is that, in Stage 2, $S$ must determine all random value of Naor’s commitment scheme in order to response $V^*$’s leakage query. This results in that $S$ must specify the challenge $\delta$ in advance.

Let $t = 3n^4$. We denote by $S \leftarrow V^*$ that $S$ receives a message from $V^*$, by $S \rightarrow V^*$ that $S$ sends a message to $V^*$. For simplicity, we use $S \equiv V^*$ to denote a interactive process between $S$ and $V^*$. The process that $V^*$ carries out a leakage query and obtains the response to it is denoted as $S \leftrightarrow V^*$. 
On inputting common $n$-vertex graph $G$ and the description of the code of the strategy of $V^*$ (denoted by $\text{desc}(V^*)$), the simulator $S^{L_n}(\text{desc}(V^*), G, \text{aux})$ (where aux is $V^*$’s auxiliary. We omit aux sometimes) first selects uniformly random string $R_{V^*}$ for $V^*$ and then proceeds as follows.

If $V^*$ aborts at any point, $S$ outputs $V^*$’s current View and stops.

**Simulating Stage 0:** $S^{L_n}(\text{desc}(V^*), G)$ deals with $V^*$’s leakage queries as follows. Initially, let $R_0 = \emptyset$ be the empty bit string. At any point, $S$ resets $R_0 = R_0 || R'$, where $R'$ is the random coins used in the current process. When receiving a leakage query $f(\cdot, \cdot)$ from $V^*$, $S^{L_n}(\text{desc}(V^*), G)$ sets $f'(z) \triangleq f(z, R_0)$, and then queries $L_n^0(\cdot)$ with $f'(\cdot)$. Finally, $V^*$ obtains $f'(z)$. It is easy to see that the leakage information, $f'(z) = f(z; R_0)$, is indistinguishable from one obtained by $V^*$ in the real execution.

1. $S \Rightarrow V^*$: $S$ runs $\Phi$ with $V^*$: $S$ first invokes $V^*$ and receives $(h, t, \tau)$. Thus, $S$ randomly selects $u \in \{0, 1\}^n$, $s \in \{0, 1\}^{2n^2}$ and computes $c = \text{Comm}_\beta^* (\eta, s)$, where $\eta = (t(h(desc(V^*)) \oplus u), u)$ Finally, $S$ invoke $V^*$ with $c$.

2. $S \leftrightarrow V^*$: After receiving a new query $f$, $S$ sets $R_0 = t(h(desc(V^*)) \oplus u) || u$. Thus, $S$ queries $L_n^0(\cdot)$ with $f'(\cdot) \triangleq f(\cdot, R)$. Finally, $V^*$ obtains $\zeta = f'(z)$ returned from $L_n^0(\cdot)$.

**Remark 1:** Notice that $R = t(h(desc(V^*)) \oplus u) || u$ is indistinguishable from $\theta || u$ (corresponding random value generated in real interaction), where $\theta \in R \{0, 1\}^n$. It follows that $\zeta = f'(z)$ is indistinguishable from the leakage query in real interaction.

3. $S \leftarrow V^*$: $S$ receives $d$ from $V^*$.

4. $S \Rightarrow V^*$: $S$ executes $P_{\text{au}}$ of $(P_{\text{au}}, \tilde{V}_{\text{au}})$ with the witness $(\text{desc}(V^*), u, \zeta, s)$. First, after receiving $a$ from $V^*$, $S$ computes $\beta$ as an honest $P_{\text{au}}$. Then, sends $\tilde{\beta} = (t_\beta, \tilde{\beta}_1)$ to $V^*$, where $\tilde{\beta}_1 = \text{Comm}(t_\beta(\beta); r_\beta)$, $t_\beta \in R_{\text{t}_1}, r_\beta \in R \{0, 1\}^n$.

5. $S \leftrightarrow V^*$: After receiving a new query $f$ from $V^*$, $S$ prepares random string $R_0 = R_0 || t_\beta || t_\beta(\beta) || r_\beta$. Then, $S$ queries the leakage oracle $L_n^0(\cdot)$ with $f'(\cdot) \triangleq f(\cdot, R)$, and $L_n^0(\cdot)$ returns $\zeta = f'(z)$ to $V^*$.

**Remark 2:** In real interaction, the prover uses $t_\beta || \beta' || r_\beta$ as random value to response leakage query, where $\beta' \in R \{0, 1\}^{11}$. Obviously, it is indistinguishable from $t_\beta || t_\beta(\beta) || r_\beta$ used by $S$.

6. $S \Rightarrow V^*$: $S$ receives $\gamma$ from $V^*$, and then computes $\delta$ as an honest $P_{\text{au}}$. Finally, $S$ sends $\tilde{\delta} = (t_\delta, \tilde{\delta}_1)$ to $V^*$, where $\tilde{\delta}_1 = \text{Comm}(t_\delta(\delta); r_\delta)$, $t_\delta \in R_{\text{t}_2}, r_\delta \in R \{0, 1\}^n$.

7. $S \leftrightarrow V^*$: After receiving a new leakage query $f$ from $V^*$, $S$ sets $R_0 = R_0 || t_\delta || t_\delta(\delta) || r_\delta$. Then, $S$ queries $L_n^0(\cdot)$ with $f'(\cdot) \triangleq f(\cdot, R)$. Finally, $L_n^0(\cdot)$ returns $\zeta = f'(z)$ to $V^*$.

**Remark 3:** Similar to Remark 2.

Let $\sigma$ be the simulated transcript of Stage 0, $\text{fake} = (\beta, r_\beta, \delta, r_\delta)$.

**Simulating Stage 1:** $S^{L_n}(\text{desc}(V^*), G)$ deals with $V^*$’s leakage queries as in simulating Stage 0. Initially, let $R_1 = R_0$. At any point, $S$ resets $R_1 = R_1 || R'$, where
RL is the random coins used in the current process. To respond to V∗’s query f, 
Lf(z, f(L∗), G) defines f′(⋅) ≜ f(⋅, R1) and then queries the leakage oracle L∗(⋅) with f′(⋅). Finally, V∗ obtains f′(z) from L∗(⋅). The details are as follows.
1. S → V∗: S sends a random value m to V∗.
2. S ↔ V∗: After receiving a leakage query f from V∗, S sets R1 = R1||m. Thus, S queries L∗(⋅) with f′(⋅) ≜ f(⋅, R). Thus, V∗ obtains f′(z) returned from L∗(⋅).
3. S = V∗: S honestly executes Πcd taking fake = (β, rβ, δ, rδ) as the input. S first receives ̂C = (T∗, C, Tdesc) and K, r. Then, S prepares the first message v1 = (v1, v2, · · ·, v1) of OTNP and sends it to V∗.
4. S ↔ V∗: After receiving V∗’s response v1 = (v1, v2, · · · , v1), S first computes K, r, from v, v′. Finally, S computes Eval( ̂C, K, f, K, r) and then obtains r∗.
5. S → V∗: Assume r∗ = r′ 1 · · · r′ n, where r′ 1 = r′ 2 · · · r′ n, and |r′ 1 | = 3n. For i ∈ [n], j ∈ [n2], S selects z′ i,j ∈ {0, 1} and sets
   r′ i,j = r′ i,j ⊕ g(z′ i,j) ⊕ g(z′ i,j), r′ i,j = r′ i,j · · · r′ i,j.
   Then, S sends r″ = r″ 1 · · · r″ n to V∗ and receives r′ along with the decommitments to all C∗, i’s commitments in C∗ from V∗.
6. S → V∗: Assume r∗ = r′ 1 · · · r′ n, where r′ 1 = r′ 2 · · · r′ n, and |r′ 1 | = 3n. For i ∈ [n], j ∈ [n2], S selects z′ i,j, z′ i,j ∈ {0, 1} and sets
   r′ i,j = r′ i,j ⊕ g(z′ i,j) ⊕ g(z′ i,j), r′ i,j = r′ i,j · · · r′ i,j.
   Then, S sends r″ = r″ 1 · · · r″ n to V∗ and receives r′ along with the decommitments to all C∗, i’s commitments in C∗ from V∗.
7. Output: S verifies r as an honest prover. If the verification fails, S outputs (rV ∗, c, β, δ, v, r′′) and then aborts. Otherwise, S sets r = r′′ ⊕ r∗ and proceeds the next step.

Simulating Stage 2: Assume r = r1 · · · rn, where r1 = r1, · · · , r1,2 and |r1,2 | = 3n. Obviously, r1,2 = g(z0,0,1 ⊕ g(z1,1) for (i, j) ∈ [n] × [n2] if S does not abort. In this stage, SL∗(desc(V∗), G) completes the commitments as follows.
1. S → V∗: For every i ∈ [n], S picks a random permutation πi and sets Gi = πi(G). Then, S sends to V∗ the commitments to the adjacency of G, using Naor’s commitment scheme with r1 as the first message, i ∈ [n]. Accurately, S sets Ci = (g(z0,1)j=1n | j = 1, · · · , n) for i ∈ [n], and sends C = (C1, · · · , Cn) to V∗.
2. S ↔ V∗: S uses L∗(⋅) to answer the leakage queries. The detail is as follows.

Response to query leakage in simulating Stage 2: It is not easy to deal with leakage queries in this stage although S acts seemingly as the same as an honest prover. The witness for x ∈ L is a Hamiltonian cycle H in G. For simplicity, we replace z with H in what follows. After receiving V∗’s leakage query f(⋅), S needs to construct a new leakage query f′ such that f′(H) is indistinguishable from f(state). To this end, S selects random permutation πi and ti ∈ {0, 1} , i = 1, · · · , n, such that the following two conditions hold.
1) C′ = Comm(xi)(z′ (G); θi).
2) \( \mathcal{S} \) is able to answer the challenge in Stage 4.

It is easy to see that if \( \mathcal{S} \) has selected such \((\pi_1^\ast, \tilde{\theta}_1), \ldots, (\pi_n^\ast, \tilde{\theta}_n)\) and \( \tilde{\theta}_i \in \{0,1\}^n \), it can construct \( f' \) by letting \( f'(\cdot) = f(\cdot, R_1||\pi_1'||\tilde{\theta}_1||\cdots||\pi_n'||\tilde{\theta}_n) \).

The condition 1) is easy to meet. But the condition 2), which is related to the witness \( H \), is not easy to meet owing to lack of the witness. To solve this problem, \( \mathcal{S} \) specifies in advance the challenge \( w \) and the witness \( H \).

\[ \text{Simulating Stage 3:} \]

\( \mathcal{S} \) honestly executes \( \Pi_{cd} \) taking \( \text{fake} \) as the inputs of the garbled-circuit. \( \mathcal{S} \) handles leakage queries as in simulating Stage 1. Initially, let \( R_3 = R_2(H) \).
1. $S = V^*$: $S$ honestly executes $H_{cd}$ taking fake as the inputs of the garbled-circuit. $S$ first receives $\tilde{C} = (T^*, C_T, T_{dec})$ and $K_{chV}$ (keys corresponding to $ch_{V^*}$). Then, $S$ prepares the first message $v_2 = (v_{2,1}, \cdots, v_{2,i})$ of $OT_N^*$ and sends it to $V^*$.

2. $S \leftrightarrow V^*$: After receiving a leakage query $f$ from $V^*$, $S$ sets $R_3 = R_3(H)||R'$, where $R'$ is random coins used in preparing $v$. Thus, $S$ queries $L_n^*$ with $f'(\cdot) \triangleq f(\cdot, R)$. Then, $V^*$ obtains $f'(z)$ returned from $L_n^*$.  

3. $S \leftarrow V^*$: After receiving $V^*$’s response $v'_2 = (v'_{2,1}, \cdots, v'_{2,i})$, $S$ first computes $K_{fake}$ from $v, v'$. Finally, $S$ computes $\text{Eval}(\tilde{C}, K_{fake}, K_{chV})$ and then obtains $ch_{V^*}$.

4. $S \rightarrow V^*$: $S$ sets $ch_P = ch \oplus ch_{V^*}$, and sends $ch_P$ to $V^*$. $V^*$ returns the commitments to $C_{T_2}$ to $S$.

5. Output: $S$ verifies $ch_{V^*}$ as an honest prover. If the verification fails, $S$ outputs the view $(R_{V^*}; c, \tilde{\beta}, \tilde{\delta}, v_1, r'', \tilde{C}, v_2, ch_P)$ and then aborts. Otherwise, $S$ sets $ch = ch_P \oplus ch_{V^*}$ and proceeds the next step.

Simulating Stage 4:

1. $S \rightarrow V^*$: $S$ reveals $\pi_i$ and all the commitments of $C_i$ when $ch_i = 0$, or reveals some commitments of $C_i$ corresponding to the edges to $H_i$ when $ch_i = 1$, for every $i \in [n]$. Formally, $S$ sets

$$\text{Ans}_i = \begin{cases} \{\pi_i, \{g'_{i,j}\}^2\} \cup \{z_{i,j} : h'_{i,j} = 1, 1 \leq j \leq n^2\}, & ch_i = 0 \\ \{z_{i,j} \} \cup \{h'_{i,j} : 1 \leq j \leq n^2\}, & ch_i = 1 \end{cases}$$

where $\{g'_{i,j}\}^2$ and $\{h'_{i,j}\}^2$ are the adjacencies of $G_i = \pi(G)$ and $H_i$, respectively. $S$ sends $\text{Ans} = (\text{Ans}_1, \cdots, \text{Ans}_n)$ to $V^*$.

Outputting: Finally, $S$ outputs $(R_{V^*}; c, \tilde{\beta}, \tilde{\delta}, v_1, r'', \tilde{C}, v_2, ch_P, \text{Ans})$ and stops.

Simulator’s output distribution: Next, we use the hybrid argument method to prove that $\{\mathcal{S}^{L_n^*}(G, \text{desc}(V^*), aux)\}_{x \in \{0,1\}^*}$ is indistinguishable from the real view $\{\text{View}_{V^*}(G, aux)\}_{x \in \{0,1\}^*}$.

For simplicity, we will omit $V^*$’s auxiliary aux in what follows.

First, the prover may abort in real interaction when $V^*$ fails in revealing $r'$ correctly in Stage 1 (or $ch_{V^*}$ in Stage 3). Notice that $S$ aborts in simulating Stage 1 (or Stage 3) under the same condition, and meanwhile outputs the current simulated view. For simplicity, we assume $S$ does not abort in simulating Stage 1 or Stage 3 in what follows.

Suppose there is a hybrid simulator $\widehat{S}$ which receives a Hamiltonian cycle $H \subseteq G$ as the input. We define hybrid experiments as follows:

$Exp_0$: $\widehat{S}$ holding a hamiltonian cycle $H$, executes the protocol with $V^*$ just like the honest prover.
Exp$_1$: This experiment is the same as Exp$_0$ except that in Stage 0 $\hat{S}$ interacts with $V^*$ as $\mathcal{S}$. Specially, $\hat{S}$ computes $c = Comm^r_{ab}(\eta; s)$ after receiving $(h, t, \tau)$ from $V^*$, where $u \in_R \{0, 1\}^n$, $\eta = t(h(desc(V^*))) \oplus u, u$, and then sends $c$ to $V^*$. After receiving a leakage query $f$ from $V^*$, $\hat{S}$ computes the leakage information $\zeta = f(H, R)$. Finally, $\hat{S}$ can compute $\beta, \delta$ as an honest $P_{ua}$.

Exp$_2$: This experiment is the same as Exp$_1$ except that in Stage 1 $\hat{S}$ interacts with $V^*$ and responds to leakage queries as $\mathcal{S}$.

Exp$_3$: This experiment is the same as Exp$_2$ except in Stage 2 and Stage 4. In Stage 2, $\hat{S}$ interact with $V^*$ as $\mathcal{S}$ to commit to $\pi_i(G)$ by $C_i = \{g(z_{i,j}^0)\}_{j=1}^{n^2}$ for a randomly selected $\pi_i$. Let $ch = ch\_1 \cdots ch\_n$ be the challenge selected by $\hat{S}$ in Stage 2, $ch' = ch\_1' \cdots ch\_n'$ be the challenge generated in Stage 3. The probability that $ch = ch'$ is negligible since $\hat{S}$ proceeds as the honest prover in Stage 3. Therefore, $\hat{S}$ should use a special strategy to answer the challenge $ch'$. More precisely, $\hat{S}$ proceeds in Stage 4 as follows.

- When $ch_i = 0 \land ch_i' = 0$, $\hat{S}$ simply reveals $\pi_i$ and $\{z_{i,j}^g\}_{j=1}^{n^2}$ to show $C_i$ is the commitments to $\pi_i(G) = (g_{i,j}^i)$.
- When $ch_i = 0 \land ch_i' = 1$, using $H$ and $\pi_i$ (selected in Stage 2), $\hat{S}$ can open some commitments of $C_i$ according to the Hamiltonian cycle $\pi_i(H)$ in Stage 4. (Because $\hat{S}$ replies to the leakage query in Stage 2 as an honest prover ($ch_i = 0$), $\hat{S}$'s respond to $ch_i'$ is coincident with its leakage answer in Stage 2.)
- When $ch_i = 1 \land ch_i' = 0$, $\hat{S}$ first invokes $\rho(1, H, H_i)$ and $\varphi(1, H, H_i, \{z_{i,j}^0, z_{i,j}^1\})$ with $H$ and $H_i$ (selected by $\mathcal{S}$ in Stage 2) to determine $\pi_i'$ and the random coins $z_{i,j}'$ for each $j \in [n^2]$, where $\{g_{i,j}'\}_{j=1}^{n^2}$ be the adjacency matrix of $G_i = \pi_i'(G)$, and then reveals $\pi_i$ and $\{z_{i,j}^g\}_{j=1}^{n^2}$ to show that $C_i$ is the commitment to $G_i$. (Here, $\hat{S}$ uses the same strategy as in Stage 2 to determine $\pi_i'$ and the random coins $z_{i,j}'$)
- if $ch_i = 1, ch_i' = 1$, $\hat{S}$ opens some commitments of $C_i$ according to the Hamiltonian cycle $H_i$ (the Hamiltonian cycle picked in Stage 2).

Exp$_4$: This experiment is the same as Exp$_3$ except that in Stage 3 $\hat{S}$ interact with $V^*$ and responds to leakage queries as $\mathcal{S}$.

Exp$_5$: This experiment is the same as Exp$_4$ except that $\hat{S}$ interact with $V^*$ like $\mathcal{S}$ in Stage 4. That is, $\hat{S}$ in Exp$_5$ is the same as $\mathcal{S}$.

Let $D_i$ be the output of $\hat{S}$ in Exp$_1$ ($i = 0, 1, 2, 3, 4, 5$). In Exp$_i$, $V^*$'s leakage queries are dealt with as $\mathcal{S}$ from Stage 0 to Stage $i - 1$. It is easy to see that $D_0 = View_{V^*}(G)$ and $D_5 = SELc^{\tau}(G, desc(V^*))$.

Notice the differences between Exp$_0$ and Exp$_1$ are that the values committed by $c, \beta_1, \delta_1$ respectively are different. Thus, it follows from the hiding property of the commitment schemes that $D_0$ and $D_1$ are indistinguishable.

Exp$_2$ is different from Exp$_1$ in two aspects: (1) the input to $II_{cd}$ in Stage 1, and (2) the distribution of $r'$ (and then $r$). More precisely, $\hat{S}$ computes $II_{cd}$ with a random input in Exp$_1$, whereas with a witness of $\sigma \in L_{\mathcal{S}, \mathcal{M}}$ in Exp$_2$. Additionally,
instead of selecting uniformly \(r''\) in \(\text{Exp}_1\), \(\hat{S}\) randomly selects \(z_{i,j}^0, z_{i,j}^1 \in R \{0, 1\}^n\) and then constructs \(r''\) such that \(r = r' \oplus r'' = r_1 \cdots r_n = r_{1,1} \cdots r_{n,n^2}\) meeting with \(r_{i,j} = g(z_{i,j}^0) \oplus g(z_{i,j}^1)\) for every \(i \in [n], j \in [n^2]\), where \(r'\) be the output of \(\Pi_{cd}\) received by \(\hat{S}\). Therefore, the fact that \(D_2\) and \(D_1\) are indistinguishable is implied by the receiver’s security of \(\text{OT}_{NP}\) and the pseudo-randomness of \(g\).

The differences between \(\text{Exp}_3\) and \(\text{Exp}_2\) are in two aspects. One is \(\hat{S}\) computes \(C_i = \text{Comm}_{\hat{S}}^i(\pi_i(G); \theta_i)\) (the commitment to \(\{g_{i,j}\}_{j=1}^{n^2}\), the adjacency of \(G_i = \pi_i(G)\) for a randomly selected \(\pi_i\)) using different \(\theta_i = \theta_{i,1} \cdots \theta_{i,n^2}\). In \(\text{Exp}_2\), \(\pi_i\) is uniformly selected is Stage 2 and \(\theta_{i,j} \in R \{0, 1\}^n\) is independent with \(z_{i,j}^b (b = 0, 1)\), whereas in \(\text{Exp}_3\), \(\pi_i\) is determined from \(H\) and \(H_i\) (a random Hamiltonian cycle selected in Stage 2), \(\theta_{i,j} = z_{i,j}^0\) if \(g_{i,j} = 0\) or \(z_{i,j}^1\) if \(g_{i,j} = 1\). So, \(C_i = \{g(z_{i,j}^b)\}_{j=1}^{n^2}\) in \(\text{Exp}_3\).

The other is the approach of replying to the challenge \(ch' = ch_1' \cdots ch_n'\) generated in Stage 3. In \(\text{Exp}_2\), \(\hat{S}\) honestly replies \(ch'\) in Stage 4. In \(\text{Exp}_3\), \(\hat{S}\) (in Stage 4) first determines \(\pi_i\) from \(H\) and \(H_i\) by requiring \(H_i = \pi_i(H)\). Next, \(\hat{S}\) sets \(\theta_{i,j} = z_{i,j}^g\), \(j = 1, \ldots, n^2\). After then, \(\hat{S}\) reply \(ch'_t\) as in \(\text{Exp}_2\), \(\hat{S}\) needs the witness \(H\) in Stage 4.

Notice \(\pi_i\) is also uniformly distributed in \(\text{Exp}_3\) although it is determined from \(H\) and \(H_i\). It is easy to see that the difference between \(D_3\) and \(D_2\) results only from \(C_i\)’s. \(D_3\) and \(D_2\) are distinguishable means \(g(z_{i,j}^1)\) and \(g(\theta_{i,j}) \oplus g(z_{i,j}^0) \oplus g(z_{i,j}^1)\) (the commitments to 1 in \(\text{Exp}_3\) and \(\text{Exp}_2\), respectively) are distinguishable for \(z_{i,j}^b, \theta_{i,j} \in R \{0, 1\}^n\). This is contradict with the pseudo-randomness of \(g\). It follows that \(D_3\) and \(D_2\) are computationally indistinguishable.

The difference between \(\text{Exp}_4\) and \(\text{Exp}_3\) lies in Stage 3. Notice that Stage 3 is aimed at generating a uniform challenge \(ch'\). In \(\text{Exp}_4\), \(\hat{S}\) first randomly selects \(ch\) in Stage 2, and then take fake = (\(\beta, r, \delta, r_3\)) the witness of \(\sigma \in L_{\text{sim}}\) as the input to run \(\Pi_{cd}\) in Stage 3, such that the generated challenge \(ch' = ch_P \oplus ch_V\) = \(ch\). In \(\text{Exp}_3\), however, \(\hat{S}\) honestly executes Stage 3 and generates an uniformly independent \(ch' = ch_P \oplus ch_{V'}\). Obviously, the difference between \(\text{Exp}_4\) and \(\text{Exp}_3\) in essential is that \(\hat{S}\) computes \(\Pi_{cd}\) with the different inputs. Therefore, the fact that \(D_4\) and \(D_3\) are indistinguishable is implied by the receiver’s security of \(\text{OT}_{NP}\).

Because of \(ch = ch'\) in \(\text{Exp}_4\), it is not hard to see that \(\text{Exp}_5\) is the same as \(\text{Exp}_4\). So, \(D_5\) and \(D_4\) are identical.

Overall, \(D_5\) and \(D_0\) are indistinguishable. That is, \(\{S^{\text{U}^{t}}_{\text{L}}(G, \text{desc}(V^{*}))\}_{x \in L}\) is indistinguishable from \(\{\text{View}_{V'}(G)\}_{x \in L'}\).

**Proof of Knowledge:** By Definition 3, we need to construct a knowledge extractor \(K\) with knowledge error \(\kappa\) such that \(K\), given access to the prover-strategy oracle \(P^*\), can output a witness at least with the probability \(p' = p - \kappa\) if \(P^*\) is accepted with the probability \(p > \kappa\). Here, \(\kappa = 2^{-n}\).

On input \(G, K^P^*\) first interacts with \(P^*\) to execute the protocol as an honest verifier. Let \(ch\) the challenge generated in Stage 3. If the proof of \(P^*\) is rejected, \(K^P^*\) output \(\perp\) and aborts. Otherwise, \(K\) rewinds \(P^*\) to the beginning of Stage 3 to rerun the residual protocol with a random \(ch_V^*\). \(K^P^*\) repeats this until another acceptable proof occurs. Assume the second accepting proof occurs in \(j\)th iteration and the corresponding challenge is denoted by \(ch'\). Thus, for the same commitments
Finally, such that $\chi_i \neq \chi'_i$, and $\mathcal{K}^{P^*}$ outputs $\pi_i$ and Hamiltonian cycle $H_i$ of $G_i = \pi_i(G)$. Finally, $\mathcal{K}^{P^*}$ outputs $H = \pi^{-1}(H_i)$ and stops. The details are in Figure 7.

Next, we show that $\mathcal{K}^{P^*}$ runs in the expected polynomial time.

In fact, Let $p$ the probability that the proof of the first execution of the protocol is accepted by $V$, $p'$ be the probability that $P^*$ successfully responds to the query in Stage 4. From that fact that the $\chi, \chi'$ are uniformly distributed if $V$ is honest, it is easy to see that $P$ correctly responds to $\chi$ and $\chi'$ with the same probability. Therefore, it follows that $p' \geq p$. (Because the fact the proof is accepted means that $P^*$ correctly replies $\chi_i$.) Notice that $\mathcal{K}^{P^*}$ clearly runs in strict polynomial time when $p = 0$. So we assume that $p > 0$ in what follows. Therefore, the expected running time of $\mathcal{K}^{P^*}$ is given by

$$\left(1 - p\right) \cdot \text{poly}(n) + p \cdot \frac{1}{p'} \cdot \text{poly}(n) = \text{poly}(n)$$

That is, $\mathcal{K}^{P^*}$ an expected polynomial-time algorithm.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\mathcal{K}$, playing the role of the honest verifier, interacts with $P^<em>$ to perform the first three stages. Then, $\mathcal{K}$ obtains the commitment $C_i$ to the adjacency of $G_i (i \in [n])$ from $P^</em>$.</td>
</tr>
<tr>
<td>2</td>
<td>$\mathcal{K}$ honestly executes Stage 3. Concretely, $\mathcal{K}$ first randomly selects $\chi_V \in R {0, 1}^n$ to run $\Pi_{cd}$. After receiving $\chi_P$ from $P^*$, $\mathcal{S}$ computes $\chi = \chi_P \oplus \chi_V$.</td>
</tr>
<tr>
<td>3</td>
<td>$\mathcal{K}$ honestly executes Stage 4 with $P^<em>$. After receiving the response to $\chi$ from $P^</em>$, $\mathcal{K}$ verifies the response. If the verification fails, $\mathcal{K}$ outputs $\perp$ and aborts. Otherwise, $\mathcal{K}$ proceeds next step.</td>
</tr>
<tr>
<td>4</td>
<td>$\mathcal{K}$ rewinds $P^*$ to run Stage 3, and obtains $\chi'$ at the end of Stage 3.</td>
</tr>
<tr>
<td>5</td>
<td>$\mathcal{K}$ executes Stage 4 with $P^<em>$. After receiving the response to $\chi'$ from $P^</em>$, $\mathcal{K}$ verifies this response. If the verification fails, $\mathcal{K}$ returns to the point of the beginning of Step 4. Otherwise, $\mathcal{K}$ proceeds next step.</td>
</tr>
<tr>
<td>6</td>
<td>If $\chi = \chi'$, $\mathcal{K}$ outputs $\perp$ and aborts. Otherwise, let $i$ be such that $\chi_i \neq \chi'_i$, $\mathcal{K}$ obtains two accepting response corresponding to two different queries $\chi_i$ and $\chi'_i$ respectively, e.g., $\chi_i = 0, \chi'_i = 1$. Thus, for given commitment $C_i$, $\mathcal{K}$ obtains a Hamiltonian cycle $H_i$ of $G_i$ from the response to $\chi_i$ and a permutation $\pi_i$ meeting $G_i = \pi_i(G)$ from the response to $\chi'_i$. Finally, $\mathcal{K}$ outputs $H = \pi^{-1}(H_i)$ and stops.</td>
</tr>
</tbody>
</table>

Figure 7: Knowledge extractor

Next, we show that the probability for $\mathcal{K}$ to output a Hamiltonian cycle is at least $p - 2^{-n}$ when $p > 2^{-n}$.

Let $\text{ErrorFail}$ be the event that the first proof is reject. Then, we have

$$p = \Pr[\neg \text{ErrorFail}] = 1 - \Pr[\text{ErrorFail}]$$

Let $\text{CollisionFail}$ denote the event that $\mathcal{K}^{P^*}$ outputs $\perp$ because of $\chi = \chi'$ in Step 6. Thus, $\mathcal{K}^{P^*}$ fails if $\text{ErrorFail}$ or $\text{CollisionFail}$ takes place. We have
Pr[K^P*(G) = ⊥] = Pr[ErrorFail] + Pr[¬ErrorFail ∨ CollisionFail]

We denote by CollisionFail_j the event that K^P* outputs ⊥ at jth repetition, that is, P^* fails to reply the corresponding challenge in the first j−1 times repetition and correctly responds to ch' in jth iteration but ch = ch'. Thus, using Pr[ch = ch'] = 2^{−n}, we have the following:

Pr[¬ErrorFail ∨ CollisionFail] = Pr[¬ErrorFail ∨ (∨_{j=1}^l CollisionFail_j)]
= Pr[¬ErrorFail] Pr[(∨_{j=1}^l CollisionFail_j)]|¬ErrorFail|
= Pr[¬ErrorFail] ∑_{j=1}^l Pr[CollisionFail_j]|¬ErrorFail|
= p ∑_{j=1}^l (1−p')^{j−1} · 2^{−n}
= 2^{−n} p · \frac{1}{p'} \leq 2^{−n}

Therefore, the probability that K^P* succeeds in computing a Hamiltonian cycle of G is

Pr[K^P*(G) = H : (G, H) ∈ R_{HC}] = 1 − Pr[¬ErrorFail ∨ CollisionFail]
= 1 − Pr[ErrorFail] − Pr[¬ErrorFail ∨ CollisionFail]
≥ p − 2^{−n}

The proof is completed.

Acknowledgements

This work was partially supported the National Natural Science Foundation of China (Grant No. 60970139,61003276), Strategic Priority Program of Chinese Academy of Sciences (Grant No. XDA06010702).

References

7. Hongda Li, Qihua Niu, Bei Liang. Leakage-Resilient Zero-Knowledge Proofs of Knowledge for NP. (to appear)