Precise Bounded-Concurrent Zero-Knowledge in Almost Constant Rounds

Ning Ding[1] Dawu Gu[1,2] Bart Preneel[2]

[1]Department of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai, 200240, P.R. China
[2]ESAT-COSIC, Katholieke Universiteit Leuven (KUL), Leuven 3001, Belgium
Email address: {cs.dingning@gmail.com, dawu.gu@esat.kuleuven.be, Bart.preneel@esat.kuleuven.be}

Abstract

Precise concurrent zero-knowledge is a new notion introduced by Pandey et al. [23] in Eurocrypt'08 (which generalizes the work on precise zero-knowledge by Micali and Pass [19] in STOC'06). This notion captures the idea that the view of any verifier in concurrent interaction can be reconstructed in the almost same time. [23] constructed some (private-coin) concurrent zero-knowledge argument systems for **NP** which achieve precision in different levels and all these protocols use at least $\omega(\log n)$ rounds. In this paper we investigate the feasibility of reducing the round complexity and still keeping precision simultaneously. Our result is that we construct a public-coin precise bounded-concurrent zero-knowledge argument system for **NP** only using almost constant rounds, i.e., $\omega(1)$ rounds. Bounded-concurrency means an a-priori bound on the (polynomial) number of concurrent sessions is specified before the protocol is constructed. Our result doesn't need any setup assumption. We stress that this result cannot be obtained by [23] even in bounded-concurrent setting.

Keywords: Zero-Knowledge, Precise Zero-Knowledge, Concurrent Zero-Knowledge, Interactive Proofs and Arguments, Proofs of Knowledge

Contents

List of Tables

1 Introduction

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff [15]. Their definition essentially states that an interactive proof of $x \in L$ provides zero (additional) knowledge if, for any efficient verifier V, the view of V in the interaction can be "indistinguishably reconstructed" by an efficient simulator S-interacting with no one- on just input x. Since efficiency is formalized as polynomial-time, a worst-case notion, zero-knowledge too automatically becomes a worst-case notion. The refinement of [11] calls for a tighter coupling between the expected running-time of V and that of S: a proof is zero-knowledge with tightness $t(\cdot)$ if there exists a fixed polynomial $p(\cdot)$ such that the expected running-time of $S(x)$ is upper-bounded by $t(|x|)$ times the expected running-time of $V(x)$ plus $p(|x|)$.

Micali and Pass [19] argued, however, that such coupling may still be insufficient, even when the tightness function is a constant and the polynomial $p(\cdot)$ is identically 0. Consider a malicious verifier V that, on input an instance $x \in \{0,1\}^n$, takes n^{10} computational steps with probability $\frac{1}{n}$, and n steps the rest of the time. The expected running-time of V is $\Omega(n^9)$, and thus zero-knowledge with optimal tightness only requires that V be simulated in expected time $\Omega(n^9)$. They thought that it is doubtful to take indifference for V to get out and interact with the prover or to stay home and run S for granted. Since by interacting with P, V will almost always execute n steps of computation, while (in absence of extra guarantees) running the simulator might always cause him to invest n^9 steps of computation. This discussion shows that we need a stronger notion of zero-knowledge.

1.1 Precise Zero-Knowledge

Hence [19] put forward a stronger notion of precise zero-knowledge. This notion captures the idea that prover provides a zero-knowledge proof of $x \in L$ if the view v of any verifier in an interaction with the prover about x can be reconstructed in the almost same time. Thus, precise zero-knowledge bounds the knowledge of the verifier in terms of its actual computation. Precisely, by [19] a proof system is zero-knowledge with precision $p(n, y)$ if for every V^* the simulator's running-time in outputting a simulated view is bounded by $p(n, \mathsf{T})$ whenever V^*'s running-time on this view is T. Following this notion [19] constructed some (private-coin) zero-knowledge proofs or arguments with polynomial (resp. linear) precisions, i.e. $p(n, y) = \text{poly}(n, y)$ (resp. $p(n, y) = O(y)$).

For convenience of statement, we say a zero-knowledge protocol is precise (resp. imprecise) if the (known) simulator for it can (resp. cannot) provide polynomial precision. [19] showed there don't exist black-box precise zero-knowledge protocols for any non-trivial language. Further, [26] showed Barak's non-black-box zero-knowledge arguments [1] are also imprecise due to the imprecise simulation strategy.

As all protocols in [19] uses at least $\omega(1)$ rounds, a natural question arises if there exist constant-round precise zero-knowledge proofs or arguments. However, if such precise proof systems exist, a longstanding question has been solved automatically, i.e. the existence of constant-round zero-knowledge proofs for **NP** with strict polynomial-time simulators, as the latter is weaker than the former. On the other hand, Barak's protocols [1] is the known unique construction able of realizing constant rounds property and strict polynomial-time simulation simultaneously for arguments. Thus it seems very hard to construct constant-round precise zero-knowledge proofs or arguments. Another non-trivial question is whether there exist $\omega(1)$-round *public-coin* precise zero-knowledge proofs or arguments as all protocols in [19] are private-coin. We will try to find out the answer to this question in this paper.

1.2 Precise Concurrent Zero-Knowledge

Although the notion in [19] is quite strong, it deals with precise zero-knowledge only in stand-alone setting. The more realistic setting for zero-knowledge is concurrent setting, introduced by [9]. A zero-knowledge protocol is concurrent zero-knowledge if for every polynomial-time verifier there exists a polynomial-time simulator that can output an indistinguishable view in concurrent execution of the protocol. So concurrent zero-knowledge is also formalized as a worst-case notion, which also suffers the same problems [19] proposed. [28][16][27] presented the constructions of black-box concurrent zero-knowledge protocols. However, by the conclusion that black-box zero-knowledge protocols are all imprecise, these protocols in [28][16][27] are all imprecise.

A recent work [23] formalizes the notion of precise concurrent zero-knowledge and constructed some precise concurrent zero-knowledge argument systems which obtain sub-quadratic precision in different levels. The best round complexity of these protocols is $\omega(\log n)$. To obtain these results [23] introduced a new simulation technique for protocols in [27], based on known recursive simulation strategy in [27]. That is, the simulator in [23] is not only oblivious of the contents of verifier's messages as the simulator in [27], but also oblivious to when verifier sends these messages.

As shown in [8], logarithmic rounds are necessary to construct (black-box) concurrent zero-knowledge. Since the simulator in [23] is based on the construction paradigm of the black-box simulator in [27], it inherits some characters from this black-box one, e.g., it inherits the probability analysis of simulator's not getting "stuck". (According to the probability analysis in [23] to avoid simulator's getting "stuck" $\omega(\log n)$ rounds are required.) Hence it seems $\omega(\log n)$ is the lower bound of round complexity for the protocols in [23], even in bounded-concurrent setting introduced below. On the other hand, as we know for concurrent zero-knowledge, constant-round property can be achieved in a specific setting. That is the bounded-concurrent setting which means that an a-priori bound on the polynomial number of concurrent sessions is specified before the protocol is constructed. The bounded-concurrent setting was first put forward explicitly in [1], which presented a breakthrough of (bounded-concurrent) non-black-box zero-knowledge arguments which have many properties such as constant-round, public coins and polynomial-time simulation which cannot be achieved by any black-box (bounded-concurrent) zero-knowledge simultaneously. Following [1] many works such as [24][25][17] studied the concurrent composition of protocols in bounded-concurrent setting. In this paper we will try to find out whether better round complexity (i.e. lower than $\omega(\log n)$) and precision can be achieved simultaneously in bounded-concurrent setting.

1.3 Our Result

In this paper we first construct a public-coin $\omega(1)$-round precise zero-knowledge argument for **NP** with polynomial precision. Second, based on the first result, we construct a public-coin $\omega(1)$-round precise bounded-concurrent zero-knowledge argument for **NP**, which is our main result. We stress that this result cannot be obtained by [23] even in bounded-concurrent setting. Formally, our results can be shown as follows.

Theorem 1.1. *Assuming the existence of collision-resistent hash functions against polynomial-sized circuits, there exist $\omega(1)$-round public-coin precise zero-knowledge arguments for* **NP** *with polynomial precision.*

Theorem 1.2. *Assuming the existence of collision-resistent hash functions against polynomial-sized circuits, there exist $\omega(1)$-round public-coin precise bounded-concurrent zero-knowledge arguments for* **NP** *with polynomial precision.*

Our Technique. The starting point of our work is Barak's constant-round (non-black-box) zero-knowledge arguments for **NP**, which consist of two phases: a general protocol GenProt and a WI universal argument. As shown in [26], Barak's protocols are imprecise (due to the imprecise simulation strategy, in particular, in the simulation of GenProt). We obtain our results by modifying GenProt of his protocols from 3 rounds to $\omega(1)$ rounds and construct precise simulators for the modified protocols.

Actually we first obtain the results as Theorem 1.1 and 1.2 show except assuming the existence of collision-resistent hash functions against circuits of some super-polynomial bounds (e.g. $n^{\log \log n}$). Then using the tree-hashing and error-correcting codes [2][3] this complexity assumption can be reduced to the more standard assumption that the existence of collision-resistent hash functions against polynomial-sized circuits. Hence in this paper our emphases is to prove the same results under the stronger assumption. Since it is straightforward to apply the technique [2][3] in our cases, we will just sketch the technique of reducing the assumption.

1.4 Outline of This Paper

The rest of the paper is arranged as follows. Section 2 presents preliminaries this paper needs. In Section 3 we prove the result as Theorem 1.1 shows but under the stronger assumption. In Section 4 we prove the result as Theorem 1.2 shows still under the stronger assumption. In Section 5, we sketch the technique in [3] to reduce the complexity assumption and thus Theorem 1.1 and Theorem 1.2 follow.

2 Preliminaries

This section contains the notations and definitions used throughout this paper.

2.1 Basic Notions

A function $\mu(\cdot)$, where $\mu : \mathbb{N} \to [0,1]$ is called *negligible* if $\mu(n) = n^{-\omega(1)}$ (i.e., $\mu(n) < \frac{1}{p(n)}$ for all polynomial $p(\cdot)$ and large enough n's). We will sometimes use neg to denote an unspecified negligible function. We say that two probability ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are *computationally indistinguishable* if for every polynomial-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$ it holds that $|\Pr[C_n(X_n) = 1] - \Pr[C_n(Y_n) = 1]| = \mathsf{neg}(n)$. We will sometimes abuse notation and say that the two random variables X_n and Y_n are computationally indistinguishable when each of them is a part of a probability ensemble such that these ensembles $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are computationally indistinguishable. We will also sometimes drop the index n from a random variable if it can be inferred from the context. In most of these cases, the index n will be the security parameter.

2.2 Cryptographic Primitives

2.2.1 Commitment Schemes

Definition 2.1. (Perfectly Binding Commitment) A (non-interactive perfectly binding computationally hiding) commitment scheme is a polynomial-time computable sequence of functions $\{C_n\}_{n\in N}$ where $C_n : \{0,1\}^n \times \{0,1\}^{p(n)} \to \{0,1\}^{q(n)}$, and $p(\cdot), q(\cdot)$ are some polynomials, that satisfies:
Perfect Binding For every $x \neq x' \in \{0,1\}^n$, $C(x, \{0,1\}^{p(n)}) \cap \to C(x, \{0,1\}^{p(n)}) = \phi$.
Computational Hiding For every $x, x' \in \{0,1\}^n$, the random variables $C(x, U_n)$ and $C(x', U_n)$ are computationally indistinguishable.

A perfectly binding commitment scheme can be constructed under the assumption that one-way permutations exist [5] (using the generic hard-core bit of [13]). Another construction, under incomparable assumptions, was given by [4]. We can also use instead the two-round scheme of Naor [20], which can be based on any one-way function.

2.2.2 Hashing and Tree Hashing

Definition 2.2. An efficiently computable function ensemble $\{h_\alpha\}_{\alpha \in \{0,1\}^*}$, where $h_\alpha : \{0,1\}^* \rightarrow \{0,1\}^{|\alpha|}$ is called collision resistent if for every polynomial-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$,

$$\Pr_{\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n}[C_n(\alpha) = \langle x, y\rangle \text{ s.t. } x \neq y \text{ and } h_\alpha(x) = h_\alpha(y)] < \mathsf{neg}(n)$$

One can construct such functions based on several natural hardness assumptions, such as the hardness of factoring. It can be seen that hash functions were defined as mapping strings of arbitrary length to n-bit long strings. One can also define hash functions as mapping n-bit strings to $n/2$-bit strings, or n-bit strings to n^ε-bit strings, where $0 < \varepsilon < 1$ is some constant. It is not hard to see that a function ensemble satisfying one of these variants can be used to construct function ensembles satisfying the other variants. If h_α is a collision-resistent hash function ensemble, we will sometimes denote the set $\{h_\alpha | \alpha \in \{0,1\}^n\}$ by $\mathcal{H}_n$, and identify the hash function with its seed. For example, if we say that a party chooses a random $h \in \mathcal{H}_n$ and sends h, then we mean that this party chooses a random $\alpha \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends α.

Definition 2.3. (Random-access hashing) A random-access hashing collection is an ensemble $\{\langle h_\alpha, \mathsf{cert}_\alpha\rangle\}_{\alpha\in\{0,1\}^*}$ of a pairs of efficiently computable functions, where $h_\alpha : \{0,1\}^* \rightarrow \{0,1\}^{|\alpha|}$ and cert_α takes two inputs x, i, where $x \in \{0,1\}^*$ and $|i| = \log|x|$, and a polynomial-time algorithm V that satisfy the following properties:
Efficiency: $|\mathrm{cert}_\alpha(x,i)| = \mathrm{poly}(|\alpha|, \log|x|)$
Completeness: For every $\alpha, x, V_{\alpha,h_\alpha(x)}(i, x_i, \mathrm{cert}_\alpha(x,i)) = 1$.
Binding (Soundness): For every polynomial-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$,

$$\Pr_{\alpha\leftarrow_{\mathrm{R}}\{0,1\}^n}[C_n(\alpha) = \langle y, i, \sigma_0, \sigma_1\rangle \text{ s.t. } V_{\alpha,y}(i,0,\sigma_0) = 1 \text{ and } V_{\alpha,y}(i,1,\sigma_1) = 1] < \mathsf{neg}(n)$$

Constructing a random-access hashing scheme using hash trees. There is a well known construction due to Merkle of a random-access hash scheme based on any collision-resistent hash function ensemble [18].

2.2.3 Interactive Proofs and Arguments

An interactive proof [15] is a two-party protocol, where one party is called the prover and the other party is called the verifier. We use the following definition:

Definition 2.4. An interactive protocol (P, V) is called an interactive proof system for a language L if the following conditions hold:
Efficiency: The number and total length of messages exchanged between P and V are polynomially bounded and $V$is a probabilistic polynomial-time machine.
Perfect completeness: If $x \in L$, then V will always accept x.
Soundness: If $x \notin L$, then the probability that V accepts x is $\mathsf{neg}(n)$.

Let $L \in \mathbf{NP}$, an interactive argument for L [7] is the following variation on the definition of an interactive proof:
1. The soundness requirement is relaxed to quantify only over prover strategies P^* that can be implemented by a polynomial-sized circuit.
2. The system is required to have an efficient prover strategy.

2.2.4 Zero-Knowledge

Informally, a proof or argument system for L is zero-knowledge [15] if after seeing a proof that $x \in L$, the verifier does not learn anything about x that it didnt know before. Moreover this holds even if the verifier does not follow its prescribed strategy for the proof system, as long as its strategy can be implemented by an efficient algorithm. The formal definition is below:

Definition 2.5. Let $L = L(R)$ be some language and let (P, V) be an interactive proof or argument for L. We say (P, V) is zero-knowledge if there exists a probabilistic polynomial-time algorithm, called simulator, such that for every polynomial-sized circuit V^* and every $(x, w) \in R$, the following two probability variables are computationally indistinguishable:
1. The view of V^* in the real execution of $(P(w), V^*)(x)$.
2. The output of the simulator on input (x, V^*).

There are two classical constructions of 3-round zero-knowledge proofs for **NP** (without requiring negligible soundness error probability) which are Blum's proof for Directed Hamilton Circuits (DHC) [6] and Goldreich, Micali and Wigderson's proof for Graph 3-Coloring [11] (Sec. 4.4.2).

2.2.5 Witness Indistinguishability

Witness indistinguishability is a weaker property than zero-knowledge, introduced by [10]. In a witness indistinguishable proof system if both w_1 and w_2 are witnesses that $x \in L$, then it is infeasible for the verifier to distinguish whether the prover used w_1 or w_2 as auxiliary input. The formal definition is below:

Definition 2.6. Let $L = L(R)$ be some language and (P, V) be a proof or argument for L. We say that (P, V) is witness indistinguishable (WI) if for any polynomial-sized circuit family $\{V_n^*\}_{n\in\mathbb{N}}$, any x, w_1, w_2 where $(x, w_1) \in R$ and $(x, w_2) \in R$ such that the view of V^* in the interacting with $P(x, w_1)$ is computationally indistinguishable from the view of V^* in the interacting with $P(x, w_2)$.

[10] showed that WI property can be preserved in concurrent setting. Hence n parallel composition of Blum's proof for **NP** is a construction of WI proofs for **NP** with negligible soundness error probability.

2.2.6 Universal Arguments

Universal arguments, introduced by [3], are interactive arguments of knowledge for proving membership in **NEXP**. For sake of simplicity, we introduce the definition of universal arguments only for an universal language $L_{\mathcal{U}}$: the tuple $\langle M, x, t\rangle$ is in $L_{\mathcal{U}}$ if M is a non-deterministic machine that accepts x within t steps. Clearly, every language in **NE** is linear-time reducible to $L_{\mathcal{U}}$ and every language in **NEXP** is polynomial-time reducible to $L_{\mathcal{U}}$.

Definition 2.7. An universal argument system is a pair of strategies, denoted (P, V), 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 $p(|y|)$.
Completeness by a relatively-efficient prover: For every $(y = (M, x, t), w)$ in $R_{\mathcal{U}}$

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

Furthermore, there exists a polynomial p such that the total time spent by $P(w)$, on common input (M,x,t), is at most $p(T_M(x,w)) \leq p(t)$.

Computational soundness: For every polynomial-sized circuit family $\{\tilde{P}_n\}_{n\in\mathbb{N}}$, and every $(M,x,t) \in \{0,1\}^n \backslash L_{\mathcal{U}}$,

$$\Pr[\langle \tilde{P}_n, V\rangle(M,x,t)] = 1] < \mathsf{neg}(n)$$

A weak proof of knowledge property: For every positive polynomial p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds:

For every polynomial-sized circuit family $\{\tilde{P}_n\}_{n\in\mathbb{N}}$ and every sufficiently long $y=(M,x,t) \in \{0,1\}^*$ if $\Pr[\langle \tilde{P}, V(M,x,t)] = 1] > \frac{1}{p(|y|)}$ then

$$\Pr[E^{\tilde{P}*}(y) = C \text{ s.t. } [C] \in R_{\mathcal{U}}(y)] > \frac{1}{p'(|y|)}$$

(where $[C]$ denotes the function computed by the Boolean circuit C).

The oracle machine E is called a (knowledge) extractor.

[3] gave a construction of constant-round Arthur-Merlin (a.k.a, public-coin) universal arguments. Furthermore, [3] presented approaches that can make the universal arguments zero-knowledge or Arthur-Merlin witness-indistinguishable (still in constant rounds).

2.3 Precise Concurrent Zero-Knowledge

Counting steps. If M is a probabilistic machine, denote by M_r the deterministic one obtained by fixing the content of M's random tape to r, by $\text{STEPS}_{M_r(x)}$ the number of computational steps taken by M_r on input x.

Assume (P,V) uses u-round prover's messages. In q-times concurrent execution of (P,V), for any machine V^* with an auxiliary input a, denote by $v=(x_1,\cdots,x_t,a,(m_1,m_2,...,m_{uq}))$ the view of V^* coordinating t sessions (w.l.o.g, assume V^* is deterministic). Then denote by $\text{STEPS}_{V^*}(v)$ the number of computational steps taken by V^* running on input $x_1,\cdots,x_q$ and letting the j^{th} message received be m_j. (In counting steps, we assume that an algorithm A, given the code of a second algorithm B and an input x, can simulate the computation of B on input x with linear-time overhead [19].)

Definition 2.8. (Concurrent execution) Let (P,V) be a two-party protocol, V^* be any interactive machine, $\{(a_i,b_i)\}_{i=1}^q$ be a set of q inputs to the protocol (P,V). A t-time concurrent execution of (P,V) coordinated by V^* on inputs $\{(a_i,b_i)\}_{i=1}^q$ is the following experiment:

1. Run q independent copies of P with the i^{th} copy getting a_i as input;
2. Provide V^* with the $b_1,\cdots,b_q$;
3. On each step V^* outputs a message (k,m). The k^{th} copy of P is given with the message m. V^* is given the prover's response.

Definition 2.9. (Precise Concurrent Zero-Knowledge [23]) Let (P,V) be an interactive proof or argument system for a language $L=L(R)$, $p : N\times N \to N$ be a monotonically increasing 2-variate function. We say that (P,V) is a concurrent zero-knowledge proof or argument with precision p if there exists a probabilistic algorithm S such that for every polynomial-time V^* and every auxiliary input $a \in \{0,1\}^{poly(n)}$ for V^* and every polynomial $g(n)$ and every list $\{(x_i,w_i)\}_{i=1}^{g(n)}$, $(x_i,w_i) \in R$:

1. The following two ensembles are computationally indistinguishable:

a) The view of V^* in a $g(n)$-time concurrent execution of (P, V) with inputs $\{(x_i, w_i), x_i\}_{i=1}^{g(n)}$.
b) $S(x_1, \cdots, x_{g(n)}, V^*, a)$.

2. For every sufficiently long $r \in \{0,1\}^*$, let v be the view generated by $S_r(x_1, \cdots, x_{g(n)}, V^*, a)$. Then $\text{STEPS}_{S_r(x_1,\cdots,x_{g(n)},V^*,a)} \leq p(n, \text{STEPS}_{V^*}(v))$.
We refer to S as above as a precise simulator.

We give some comments on Definition 2.9 as follows.

Comment 2.10. In case of $g(n) = 1$, i.e., there is only one session, this is the definition of precise zero-knowledge (in stand-alone setting) given in [19].

Comment 2.11. In case that $g(n)$ is restricted to be n, i.e., there are n sessions executed concurrently and other conditions remain unchange, we say (P, V) is bounded-current zero-knowledge with precision p. Since the security parameter can be "scaled", this means that for every fixed polynomial $q(n)$, we can construct a protocol from (P, V) that remains zero-knowledge when executed $q(n)$ times concurrently. Further, if p is a polynomial in the second argument, the precisions of (P, V) and the "scaled" protocol have the same degree in the second argument.

Comment 2.12. Since V^* and S are usually required to run in polynomial-time, it is less meaningful if p is a super polynomial in n or the second argument, or else we say it is meaningful. In this paper we will focus on the question how to construct bounded-concurrent zero-knowledge protocols with meaningful (i.e. polynomial) precision.

3 Public-Coin Precise Zero-Knowledge in $\omega(1)$ Rounds

The goal of this section is devoted to the proof of the following theorem.

Theorem 3.1. *Assuming the existence of collision-resistent hash functions against $n^{\log\log n}$-sized circuits, there exist $\omega(1)$-round public-coin precise zero-knowledge arguments for* **NP** *with polynomial precision.*

Notice that the difference between Theorem 3.1 and Theorem 1.1 is the complexity assumption. In this and next sections if it is not mentioned explicitly the complexity assumption is the existence of collision-resistent hash functions against $n^{\log\log n}$-sized circuits.

In Section 3.1 we present the zero-knowledge protocol and show that it is an interactive argument for **NP**. In Section 3.2 we construct a precise simulator for this protocol. In Section 3.3 and Section 3.4 we show zero-knowledge property and polynomial precision can be achieved simultaneously via this simulator and thus complete the proof of Theorem 3.1.

3.1 The Zero-Knowledge Protocol

Our zero-knowledge protocol is a variation of Barak's non-black-box zero-knowledge argument [1]. We assume familiar with Barak's protocol. Recall that Barak's protocol consists of two phases: a generation protocol GenProt and a witness-indistinguishable universal argument, denoted WIUA. To prove the membership in a **NP** language L, the prover first interacts with the verifier in GenProt, which satisfies the computational soundness property. That is, the prover cannot output a witness that the transcript is in a pre-deterministic language $\Lambda \in \mathbf{Ntime}(n^{\log\log n})$, shown below, at the end of GenProt. Then the prover proves to the verifier that either $x \in L$ or the transcript in GenProt is in Λ via WIUA.

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$) **Additional parameter:** A super-constant function $\beta(n) = \omega(1)$	$\overset{w}{\downarrow}\ \overset{x}{\downarrow}$ $\boxed{P}\quad\quad\boxed{V}$
Step V0 (Choose hash-function): Verifier chooses a random hash function $h \leftarrow_{\mathrm{R}} \mathcal{H}_n$ and sends h to prover. For $k = 1, \cdots, \beta(n)$, do the following: **Step P1.k.1 (Commitment to hash of "junk"):** Prover computes $z_k \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$ and sends z_k to verifier. **Step V1.k.2 (Send random string):** Verifier selects a string $r_k \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it.	$\xleftarrow{h \leftarrow_{\mathrm{R}} \mathcal{H}_n}$ $\xrightarrow{z_k = \mathsf{Com}(h(0^n))}$ $\xleftarrow{r_k \leftarrow_{\mathrm{R}} \{0,1\}^n}$
End For	
Steps P,V2.X (WI Proof): Prover proves to verifier using its input w via the WIUA system that either $x \in L$ or there exists a k s.t. $(h, z_k, r_k) \in \Lambda$.	

Protocol 3.3. The precise zero-knowledge argument for **NP**.

Definition 3.2. (Language Λ [1]) Λ is defined as follows: $\tau = (h, z, r,)$ is in Λ iff there exists a program Π such that $z = \mathsf{Com}(h((\Pi)))$ and $\Pi(z, y)$ outputs r within $|r|^{\log\log|r|/5}$ steps. This can be verified in $\mathbf{Ntime}(n^{\log\log n/5})$. A witness that $(h, z, r) \in \Lambda$ is a tripe (Π, s) such that $z = \mathsf{Com}(\Pi; s)$ and $\Pi(z)$ outputs r within $|r|^{\log\log|r|/5}$ steps.

The actual description of our protocol is described as Protocol 3.3 shows. In Protocol 3.3, $\mathcal{H}_n$ denotes a hash function family and (for convenience of statement) Com denotes a non-interactive perfectly-binding commitment scheme (2-round constructions are also suitable).

It is easy to see that Protocol 3.3 differs from Barak's protocol in the construction of GenProt. In Protocol 3.3, GenProt consists of a verifier message of Step V0 and $\beta(n)$ slots, each of which consists of a prover message z_k of Step P1.k.1 and a verifier message r_k of Step V1.k.2. Here we require $\beta(n) = \omega(1)$ and thus our protocol uses $\omega(1)$ rounds. Clearly, our protocol is public-coin. In the case that $\beta(n)$ decreases to 1, it is actually Barak's protocol, which cannot achieve precision as shown in [26]. Although our modification slightly increases the round complexity, we will show at this expense Protocol 3.3 can achieve precision. But firstly we should guarantee that for Protocol 3.3 the completeness and soundness still hold, as the following theorem states.

Theorem 3.4. *Protocol 3.3 is an interactive argument for* **NP**.

Proof. As our protocol is a variation of Barak's protocol, we employ and extend the original proof in [1] to prove this theorem.
Completeness: Straightforward. If $(x, w) \in R_L$, the truthful prover can use w for $x \in L$ as the witness for the modified statement in the WI proof to make the verifier convinced.
Computational Soundness: We give the formal description of the computational soundness property of GenProt of our protocol as follows: when a $n^{\log\log n}$-sized cheating prover plays GenProt with the verifier, the probability that the prover outputs a witness for $(h, z_k, r_k) \in \Lambda$ for some k is negligible.

Now we show Protocol 3.3 is computationally sound. Suppose otherwise there exists a polynomial-sized cheating prover able of cheating the honest verifier to accept some $x \notin L$ with probability ε. Then from this cheating prover [1] presented a construction of a $n^{\log\log n}$-sized P^* which can violate the computational soundness of GenProt, i.e., P^* is able of outputting a witness that (h, z_k, r_k) is in Λ for some k at the end of GenProt with probability polynomially related to ε. Thus, to prove computational soundness of Protocol 3.3 we only need to show GenProt is computationally sound for any $n^{\log\log n}$-sized cheating prover.

Let P^* be a $n^{\log\log n}$-sized cheating prover for GenProt. We claim that the probability that P^* outputs a witness that (h, z_k, r_k) is in Λ for some k is negligible. Indeed, suppose otherwise that P^* outputs a witness with non-negligible probability ε. Then, for at least an $\varepsilon/2$ fraction of $h \in \mathcal{H}_n$, it holds that P^* outputs a witness with probability at least $\varepsilon/2$. This means there exists a $k = k(n)$ (k can be given as the non-uniform input to the finding-collision algorithm below) such that it is the witness for $(h, z_k, r_k) \in \Lambda$ with probability at least $\varepsilon/2\beta$. Fix such a $h \in \mathcal{H}_n$. Since P^* is non-uniform, w.l.o.g., assume P^* is deterministic. Further, for at least $\varepsilon/4\beta$ fraction of $r_1, \cdots, r_{k-1} \in \{0,1\}^n$, it holds that P^* outputs a witness for (h, z_k, r_k) with probability $\varepsilon/4\beta$. Also fix a choice of such $r_1, \cdots, r_{k-1}$. Thus, the message z_k is also fixed.

By our assumption, if we choose $r_k, \cdots, r_\beta \leftarrow_{\mathrm{R}} \{0,1\}^n$, then with probability $\varepsilon/4\beta$ P^* will be able to output a Π_k such (1) that $z_k = \mathsf{Com}(h(\Pi_k))$ and (2) that $\Pi_k(z_k) = r_k$. This means if we choose two different independent sequences $\{r_k, \cdots, r_\beta\}, \{r'_k, \cdots, r'_\beta\} \leftarrow_{\mathrm{R}} \{0,1\}^n$, then with probability $\varepsilon^2/16\beta^2$ we obtain two strings Π_k and Π'_k satisfying z_k is a commitment to both $h(\Pi_k)$ and $h(\Pi'_k)$ and $\Pi_k(z_k) = r_k$ and $\Pi'_k(z_k) = r'_k$. Since Com is a perfectly binding commitment scheme, it follows that $h_(\Pi_k) = h(\Pi'_k)$. Since we can assume $r_k \neq r'_k$ (as this holds with $1-2^n$ probability), it follows that $\Pi_k(z_k) \neq \Pi'_k(z_k)$ and so Π_k and Π'_k are two different programs. This means that Π_k and Π'_k are a collision for h. This means that we have a $n^{\log\log n}$-sized algorithm that for an $\varepsilon/2$ fraction of $h \in \mathcal{H}_n$ and an $\varepsilon/4\beta$ fraction of $r_1, \cdots, r_{k-1} \in \{0,1\}^n$, obtains a collision for h with probability $O(\varepsilon^2/\beta^2)$, which contradicts the collision-resistent against $n^{\log\log n}$-sized circuits of the family $\mathcal{H}_n$. Thus we complete the proof. □

Thus, to prove Theorem 3.1, we only need to prove the following theorem.

Theorem 3.5. *Protocol 3.3 is zero-knowledge with polynomial precision.*

To prove Theorem 3.5, we need to construct a simulator and analyze its running-time and output. In Section 3.2 we present the construction of the precise simulator. In Section 3.3 we show its output is computationally indistinguishable from the view of the verifier in a real interaction. In Section 3.4 we show its running-time is polynomial-time overhead of the verifier's running-time and thus complete the proof of Theorem 3.5.

3.2 The Precise Simulator

In this subsection we construct a precise simulator for Protocol 3.3 which can provide zero-knowledge property and precision simultaneously. This subsection gives the description of the simulator and the analysis of its output and running-time will be presented in the next two subsections. As mentioned earlier, [26] showed that the simulator of Barak's protocol in [1] is imprecise. Hence we first present the reason that results in the imprecision in the following. Then we propose an idea to overcome the imprecision. After that we show the overview and actual description of our simulator.

3.2.1 Imprecision of The Known Simulation Strategy

Recall that in the case of $\beta = 1$, Protocol 3.3 is actual Barak's protocol. Given access the verifier's code, Barak's protocol can be simulated without making use of rewinding: To perform simulation, the simulator commits to the hash of verifier's message function (instead of committing to zeros) in Step P1.1. The verifier'next message function is then a program whose output, on input z, is r. This provides the simulator with a valid witness to use in WIUA.

However, consider a verifier V^* that has a very long auxiliary input, but most of the time only accesses a small portion of it. The simulator will always commit to the hash of whole description of V^* (including the whole auxiliary input) and will thus always take long time, while V^* might run fast a large portion of the time. Hence the known simulation strategy is imprecise.

It can be seen that the known simulation strategy can be modified easily to output indistinguishable views for our protocol: The simulator computes the current verifier's next message function in every Step P1.k.1 and commits to the hash of it. This means that the simulator has β witnesses for the transcript in GenProt. So it can use anyone of these β witnesses as the witness for the combined statement in WIUA. But by the above analysis this simulation strategy is also imprecise, while our goal is to construct a precise simulator.

3.2.2 The Idea for Obtaining Precision

Notice that the reason that the known simulation strategy is imprecise is that the simulator should commit to V^* and its whole auxiliary input, while V^* might only access few portions of the auxiliary input in a specific computation. So an idea to overcome the imprecision is that the simulator doesn't compute V^*'s next message function using the *whole* auxiliary input in Step P1.k.1. Rather, it computes this function in such way: it chooses a *part* of the auxiliary input which contain all portions V^* accessed so far. Hence we have if given this part as the real auxiliary input, then V^*'s messages prior to Step P1.k.1 would be identical to those in this simulation (if simulator uses the same coins). Further, the simulator supposes that V^* will not access any portion of the auxiliary input outside this chosen part in Step V1.k.2. Then it computes V^*'s next message function using this part (rather than whole auxiliary input) and commits to this program.

(We remark that this idea doesn't mean reverse-engineering or understanding programs. Actually, the verifier is a polynomial time machine with an auxiliary input. W.l.o.g, assume V^* is deterministic. As we know, V^* and the auxiliary input are inputs to the simulator. In simulation all V^*'s computing is emulated by the simulator. The simulator puts each tape head of V^* at the first (i.e. leftmost) portion of the corresponding tape at the beginning of simulation and then emulates V^*'s computing. Hence specifically, in simulation the simulator of course knows which bits of the auxiliary input are accessed by V^* since it can record the rightmost portion in this tape V^*'s head has scanned so far. Also the simulator can compute V^*'s next message function using this part as the auxiliary input as if it is the real auxiliary input. So this idea doesn't mean reverse-engineering or understanding programs since we don't assume the simulator understands V^*'s computing, while instead, what the simulator does is nothing but to monitor V^*'s computing. The similar technique of monitoring verifier was used by Goldreich and Oren in proving Sequential Composition Lemma of zero-knowledge protocols [14].)

If V^* doesn't access any portion outside (i.e. at the right hand of) this part of the auxiliary input in Step V1.k.2, then the committed program in Step P1.k.1 is indeed a witness for the transcript (By saying "it is a witness for the transcript" we mean it is a witness for $(h, z_k, r_k) \in \Lambda$ for some k). In this case we call the simulator succeeds in obtaining a witness in this slot. But if V^* accesses the bits of the auxiliary input outside this part in Step V1.k.2, this means the program the simulator committed to is not the witness for the transcript since it cannot output r_k on input z_k. In this case we call the simulator fails in obtaining a witness in this slot. (It can be seen the simulator can check whether the head of V^*'s auxiliary input tape moves to the right hand of a_k in computing r_k. This is because the movement of V^*'s tape heads is emulated by the simulator.)

It is because of the possibility that the simulator fails that we cannot modify the simulator of Barak's protocol to make it precise since it only has one slot. But for our protocol the situation is different. Our protocol differs from Barak's in that there are many slots our simulator can make use of. When it fails in a slot, the simulator will still use this strategy but to choose a longer part of the auxiliary input in the next slot. As our protocol has $\beta(n)$ slots, we will show the simulator will succeed in a slot rather than fail in all slots. This means it can obtain a witness for the transcript and output indistinguishable view in WIUA. In the following subsections we will make this idea more precisely and present the detailed description of the simulator.

3.2.3 Overview

W.l.o.g. assume V^* is a deterministic polynomial-time verifier and a is the auxiliary input for V^*. We will always use a variable b to denote the rightmost portion in a V^* accesses so far. Assume the simulator failed in the first $k-1$ slots. Then in the k^{th} slot, the simulator chooses n^k-bit prefix of a as the "auxiliary input", denoted by a_k (if $n^k > |a|$, let a_k be a). It is easy to see that along with k's increasing, a_k becomes longer. If b is larger than n^k, this means a_k is not sufficiently long even for V^*'s computing prior to this step, in this case the simulator adopts the honest prover strategy to compute z_k. Otherwise, it computes V^*'s next message function as program Π_k. The key importance in computing Π_k is the simulator supposes a_k is the real auxiliary input for V^*. Then it commits to the hash of Π_k as z_k and sends z_k to V^*.

In emulating V^*'s computing of r_k, the simulator checks whether or not V^* accesses any portion of a outside a_k. If it doesn't, this means (Π_k, s_k) is the witness for $(h, z_k, r_k) \in \Lambda$ where s_k is the randomness for z_k and the simulator succeeds. Once it succeeds, the simulator will adopt the honest prover strategy in the residual interaction. But if V^* does access some bits of a outside a_k, this means (Π_k, s_k) isn't a witness for $(h, z_k, r_k) \in \Lambda$ and thus the simulator fails. Then the simulator proceeds to the next slot.

However, the simulator will not fail in all slots. We claim there must exist a constant $c < \beta(n)$ satisfying the simulator succeeds in the c^{th} slot. Actually, there exists a constant c satisfying either V^*'s running-time is less than n^c or $|a|$ is less than n^c. Hence it is easy to see the simulator will succeed in (or prior to) the c^{th} slot. This is because in the former case V^* has no time to use any bit outside n^c-bit prefix and in the latter case the simulator commits to the whole auxiliary input, as the known simulator [1] does. (Since V^*'s running-time is not a-priori bounded by any fixed polynomial, we cannot fix the number of slots as any constant. This is also why we let Protocol 3.3 have $\beta(n)$ slots.)

3.2.4 Actual Description

Our simulator's operation follows the above description. We now turn to formally describing the simulator as Algorithm 3.6 shows.

In Section 3.3 we will show that the simulator S can output an indistinguishable view. In Section 3.4 we will prove the polynomial precision can be achieved.

Input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$");
$V^* \in \{0,1\}^n$: the description of the verifier and its auxiliary input $a \in \{0,1\}^{poly(n)}$.

Parameters and Variables: A super-constant function $\beta(n) = \omega(1)$; A variable b: the rightmost portion in a V^* accessed so far; A variable f: the flag denoting whether S has succeed. Initally, set f false.

Step V0: Run $V^*(x;a)$ to output a hash function h from $\mathcal{H}_n$. Set b as the rightmost portion in a V^* accesses in this step.

For $k = 1, \cdots, \beta(n)$, do the following:
Step P1.k.1: S computes z_k in two strategies according to the values of f and n^k:

1. If $f = \mathsf{true}$ or $n^k < b$, S adopts the honest prover strategy, i.e., computes $z_k \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$.
2. If $f = \mathsf{false}$ and $n^k \geq b$, S computes V^*'s next message function using "auxiliary input" a_k, denoted Π_k, where a_k is the n^k-bit prefix of a (if $n^k > |a|$, then let $a_k = a$). Compute $z_k \leftarrow_{\mathrm{R}} \mathsf{Com}(h(\Pi_k))$.

Step V1.k.2: If Case 1 in Step P1.k.1 happens, simply run $V^*(x, z_1, \cdots, z_k; a)$ to output a string r_k. Otherwise, during running V^*, S checks whether V^* accesses any bit of a at the a_k's right hand. If V^* doesn't, set $f = \mathsf{true}$ and store (Π_k, s_k), where s_k is the randomness for the commitment z_k. In both cases set b as the rightmost portion in a V^* accesses so far.
End For

Steps P,V2.X: S proves to verifier using (Π_k, s_k) (gathered in simulated Step V1.k.2 for some k) as the witness via the WIUA system that either $x \in L$ or $(h, z_k, r_k) \in \Lambda$.

Algorithm 3.6. The precise simulator S.

3.3 The Simulator's Output Distribution

As mentioned earlier, there must exist a constant c such that the simulator succeeds in obtaining a witness for the transcript in the c^{th} slot. This means (Π_c, s_c) is the witness for $(h, z_c, r_c) \in \Lambda$. Accordingly, the simulator can certainly use this witness to convince the verifier in WIUA. Furthermore, the hiding property of Com and witness-indistinguishability property of WIUA ensure that the messages generated by S are computationally indistinguishable from the view of the verifier in a real interaction. [1] presented an formal argument for this indistinguishability in the case of $\beta = 1$. Since β is a super constant in our protocol, we still give a full argument for our protocol as follows, which augments the argument in [1] in proving the indistinguishability of messages in GenProt.

Claim 3.7. *S' output is computationally indistinguishable from the view of the verifier in an interaction of Protocol 3.3 with input $\{(x, w), x\}$, where w is a witness for $x \in L$.*

Proof. The proof follows from the hybrid argument. We construct a hybrid simulator $\widehat{S}$ which has w as the auxiliary input. $\widehat{S}$ adopts the same strategy as S does in GenProt, but uses w as the witness in WIUA. To prove this claim we only need to show both S' output and the real V^*'s view is computationally indistinguishable from $\widehat{S}$' output.

1. We first show S' output is indistinguishable from $\widehat{S}$' output. Basically, this fact follows from the WI property of WIUA. Indeed, if there exists a polynomial-time algorithm D that distinguishes two distributions with probability ϵ, then there must exist a particular view v for GenProt such that D distinguishes the following two distributions: v combined with S' output in WIUA and v combined with $\widehat{S}$' output in WIUA. Let D_v be the distinguisher D with v hardwired as its first input and let V_v^* be the residual verifier V^* with v hardwired. Then D_v can distinguish with probability ϵ between prover messages output by a honest prover using w as the witness and prover messages output by a honest prover using (Π_c, s_c) as the witness in WIUA when interacts with V_v^*. By the WI property of WIUA, we have ϵ is negligible.

2. We then show $\widehat{S}$' output is indistinguishable from the real view. Basically, this fact follows from the hiding property of Com. Notice that there are β Coms in GenProt. So we use the hybrid argument here again. We construct $\beta + 1$ hybrid simulators H_i, $0 \leq i \leq \beta$. H_i adopts $\widehat{S}$' strategy in the first i slots and adopts the honest prover strategy in the last $\beta - i$ slots (Of course, every H_i also has the same w as the auxiliary input and uses w as the witness in WIUA). Clearly, $\widehat{S}$' output is identical to H_β's output and the real view is identical to H_0's output. Thus to prove this claim we need to show H_i's output is indistinguishable from H_{i+1}'s output for $0 \leq i \leq \beta - 1$.

   Suppose there exists a polynomial-time algorithm D that distinguishes two outputs of H_i and H_{i+1} with probability ϵ. Suppose we fix a choice of coins used for the first i commitments such that D distinguishes between H_i's output and H_{i+1}'s output on this choice with probability ϵ. Then D can be converted to a distinguisher between the $i+1^{st}$ Com output by H_i and by H_{i+1} respectively by hardwiring all messages before Step P1.$i + 1$.1, and since the later messages are a function of the input and the previous messages and coins. By the hiding property of Com, we have ϵ is negligible. (Actually, H_{i+1} may adopt the honest prover strategy in slot $i + 1$ too. In this case H_i's output and H_{i+1}'s output are identical.)

□

3.4 The Simulator's Running Time

The goal of this subsection is devoted to the proof of the following claim, which ensures that the simulator S is precise.

Claim 3.8. *There exists a polynomial $p(n,t)$ such that for every $x \in \{0,1\}^n \cap L$ and every sufficiently long $r \in \{0,1\}^*$ and every polynomial time machine V^* and its auxiliary input $a \in \{0,1\}^{poly(n)}$ letting $v \leftarrow S_r(x,V^*,a)$, $\mathrm{STEPS}_{S_r(x,V^*,a)} \leq p(n, \mathrm{STEPS}_{V^*}(v))$.*

Proof. Fix a random tape $r \in \{0,1\}^*$ for S. Let v denote the view S generates. Denote V^*'s running-time on v by $\mathrm{STEPS}_{V^*}(v)$. For abbreviation let t denote $\mathrm{STEPS}_{V^*}(v)$.

Since V^*'s running-time and length of a are polynomial, there exists the first constant c such that in Step V1.c.2 (and all previous steps) V^* doesn't access any bit of a outside a_c. This means (Π_c, s_c) is the witness S can use to convince verifier in WIUA and S will thus adopt the honest prover strategy in the residual $\beta - c$ slots and WIUA.

Consider S' computation and running-time in the 1st, $\cdots$, cth slots. In any of these slots S may adopt either the honest or non-honest prover strategy according to Algorithm 3.6. So we divide these slots into two sets: S_1 and S_2, where for every $k \in S_1$ S adopts the honest prover strategy and for every $k \in S_2$ S adopts the non-honest prover strategy in Step P1.k.1.

We first omit to evaluate S' running-time in computing n^k in all slots explicitly as it is obvious a tiny polynomial quantity (We will take this quantity into account later). Assume $h(x)$ and $\mathsf{Com}(x)$ are computable in time $p_h(n)$ and $p_{\mathsf{com}}(n)$ respectively, where $n = |x|$. Then S' running-time in all slots in S_1 is $\sum_{k \in S_1} [p_h(n) + p_{\mathsf{com}}(n)]$. For every $k \in S_2$ S' running-time in the kth slot is the sum of time for generating Π_k, hashing and commitment. Let us focus on the time of generating Π_k. Actually, Π_k can be generated in linear-time in lengths of the code of V^*, x and a_k and $z_1, \cdots, z_{k-1}$. (Indeed, Π_k can be generated by integrating x, a_k and $z_1, \cdots, z_{k-1}$ with the code of V^*, which only needs linear time in lengths of these strings. Then on input z_k, Π_k might work simply as follows: It first emulates V^*'s computing on x and historical prover messages $z_1, \cdots, z_{k-1}$ from the beginning to the step of receiving z_k. This is an internal computing regardless of z_k. Then Π_k outputs r_k corresponding to this z_k. It is easy to see Π_k runs in polynomial-time.) Since every $|z_k|$ is less than n^2 [1], we have the length of these strings is $2n + n^k + kn^2$. So there is a linear function $l'(\cdot)$ such that generating Π_k needs $l'(n^k + kn^2)$ time. Then S' running-time in all slots in S_2 is less than $\sum_{k \in S_2} [l'(n^k + kn^2) + p_h(l'(n^k + kn^2)) + p_{\mathsf{com}}(n)]$.

Next, we have S' running-time in the following $\beta - c$ slots is $(\beta - c)(p_h(n) + p_{\mathsf{com}}(n))$. Denote by T_{UA} S' running-time in WIUA. By the relative efficient prover property of WIUA, we have $T_{\mathsf{UA}} \leq p_{\mathsf{UA}}(t)$ for a polynomial $p_{\mathsf{UA}}(\cdot)$. (Notice that given a witness for the combined statement of WIUA, verifying if it is a witness can be performed within a fixed polynomial-time in t.)

Thus, S' running-time as a prover is less than

$$\sum_{k \in S_1} [p_h(n) + p_{\mathsf{com}}(n)] + \sum_{k \in S_2} [l'(n^k + kn^2) + p_h(l'(n^k + kn^2)) + p_{\mathsf{com}}(n)] + (\beta - c)(p_h(n) + p_{\mathsf{com}}(n)) + p_{\mathsf{UA}}(t)$$

Clearing up this formula and take into account the time of computing n^k in all slots, we have S' running-time as a prover is less than $c[l'(n^c + cn^2) + p_h(l'(n^c + cn^2))] + A(n) + p_{\mathsf{UA}}(t)$ for some polynomial $A(n)$.

Further, we have $n^c < nt$. This is because c is the first constant such that S succeeds in the c^{th} slot and thus n^{c-1} cannot be larger than t (if $n^{c-1} \geq t$, then S succeeds in the $c - 1^{th}$ slot). Moreover, consider the steps by S in emulating V^*. Denote these steps by lt, where l is the linear time overhead. Hence we have the total S' running-time is less than

$$\text{STEPS}_{S_r(x,V^*,a)} < \beta[l'(tn+\beta n^2)+p_h(l'(tn+\beta n^2))]+A(n)+p_{\mathsf{UA}}(t)+lt$$

It can be seen the quantity at the right hand of the above inequality is a polynomial, denoted $p(n,t)$. Then $p(n,t)$ satisfies $\text{STEPS}_{S_r(x,V^*,a)} < p(n,t)$ (for every V^*, a and r).

Let us finally make certain the degree of $p(n,t)$ in t. Recall that $l'(t)$ is linear in t. Denote by c_h and c_{UA} the degrees of $p_h(\cdot)$ and $p_{\mathsf{UA}}(\cdot)$ in t respectively. Then it is obvious that the degree of $p(n,t)$ in t is d, where $d=\mathsf{max}(c_h,c_{\mathsf{UA}})$. □

By Claim 3.7 and Claim 3.8 we finish the proof of Theorem 3.5. Combining it with Theorem 3.4 we also finish the proof of Theorem 3.1. In the next section we will propose a precise bounded-concurrent zero-knowledge argument in $\omega(1)$ rounds based on the result obtained in this section.

4 Achieving Precise Bounded-Concurrent Zero-Knowledge

The goal of this section is devoted to the proof of the following theorem.

Theorem 4.1. *Assuming the existence of collision-resistent hash functions against $n^{\log\log n}$-sized circuits, there exist $\omega(1)$-round public-coin precise bounded-concurrent zero-knowledge arguments for* **NP** *with polynomial precision.*

Our approach for achieving Theorem 4.1 is to modify Protocol 3.3 (as [2] did) to achieve bounded-concurrent zero-knowledge and keep the precision obtained in the previous section. In Section 4.1 we present the modified protocol. In Section 4.2 we construct a precise simulator for this protocol. In Section 4.3 and Section 4.4 we show this simulator can provide bounded-concurrent zero-knowledge and precision simultaneously.

4.1 The Zero-Knowledge Protocol

The modified protocol is presented as Protocol 4.2 shows which differs from Protocol 3.3 in two aspects. One is the definition of language Λ, shown below, and the other is r_k is randomly chosen from $\{0,1\}^{n^4}$ rather than $\{0,1\}^n$ in Step V1.k.1 for all k.

Definition 4.3. (Language Λ [1]) Λ is defined as follows: $\tau=(h,z,r)$ is in Λ iff there exists a program Π such that $z=\mathsf{Com}(h((\Pi)))$ and there exists a string y such that $|y|\le|r|/2$ and $\Pi(z,y)$ outputs r within $|r|^{\log\log|r|/5}$ steps. This can be verified in $\mathbf{Ntime}(n^{\log\log n/5})$. A witness that $(h,z,r)\in\Lambda$ is a tripe (Π,s,y) such that $z=\mathsf{Com}(\Pi;s)$ and $\Pi(z,y)$ outputs r within $|r|^{\log\log|r|/5}$ steps.

First we should prove that Protocol 4.2 is still an interactive argument for **NP**.

Theorem 4.4. *Protocol 4.2 is an interactive argument for* **NP**.

Proof Sketch: This proof is almost identical to the previous section. So we only sketch this proof. Completeness is clearly satisfied. So we only need to show its soundness. By the proof of Theorem 3.4, to do this we only need to show GenProt is computationally sound.

Recall that in the proof of Theorem 3.4 any cheating prover can be converted into an algorithm to find a collision for the hash functions. In this proof, if there exists a cheating prover can output a witness for (h,z_k,r_k) for a $k=k(n)$ at the end of GenProt. This means Π_k is the program such that $\exists_{y\in\{0,1\}^{m/2}}\Pi_k(z_k,y)=r_k$ and we choose a random $r'_k,\cdots,r'_\beta\leftarrow_{\mathbb{R}}\{0,1\}^m$ and obtain

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$) **Additional parameter:** A super-constant function $\beta(n) = \omega(1)$	$\overset{w}{\downarrow}\ \ \overset{x}{\downarrow}$ $\boxed{P} \qquad \boxed{V}$
Step V0 (Choose hash-function): Verifier chooses a random hash function $h \leftarrow_{\mathrm{R}} \mathcal{H}_n$ and sends h to prover.	$\xleftarrow{h \leftarrow_{\mathrm{R}} \mathcal{H}_n}$
For $k = 1, \cdots, \beta(n)$, do the following:	
Step P1.k.1 (Commitment to hash of "junk"): Prover computes $z_k \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$ and sends z_k to verifier.	$\xrightarrow{z_k = \mathsf{Com}(h(0^n))}$
Step V1.k.2 (Send random string): Verifier selects a string $r_k \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}$ and sends it.	$\xleftarrow{r_k \leftarrow_{\mathrm{R}} \{0,1\}^{n^4}}$
End For	
Steps P,V2.X (WI Proof): Prover proves to verifier using its input w via the WIUA system that either $x \in L$ or there exists a k s.t. $(h, z_k, r_k) \in \Lambda$.	

Protocol 4.2. The precise bounded-concurrent zero-knowledge argument for **NP**.

with probability ϵ a program Π'_k such that $\exists_{y' \in \{0,1\}^{m/2}} \Pi'_k(z_k, y') = r'_k$, then Π_k will be different from Π'_k with probability at least $\epsilon - 2^{-m/2}$. This is because if $\Pi_k = \Pi'_k$ then it must hold that $r'_k \in \Pi_k(z_k, \{0,1\}^{m/2})$ which happens with probability at most $2^{-m/2}$. In our case $m = n^4$ and so $2^{-m/2}$ is a negligible quantity.

□

So we only need to prove the following theorem.

Theorem 4.5. *Protocol 4.2 is bounded-concurrent zero-knowledge with polynomial precision.*

In Section 4.2 we give a construction of a precise simulator for Protocol 4.2. In Section 4.3 we prove that its outputs are computationally indistinguishable from the views of the verifier in a real interaction. In Section 4.4 we show it can achieve precision and thus complete the proof of Theorem 4.5.

4.2 The Precise Simulator

We use and extend the idea presented in the previous section to construct a precise simulator for Protocol 4.2. Basically, the simulator combines this idea with the known simulation strategy [1] which ensures bounded-concurrent zero-knowledge can be obtained. So our emphases in this subsection is to describe how to obtain precision in concurrent simulation.

4.2.1 Overview

It is clear that in the case of $\beta = 1$ Protocol 4.2 is Barak's bounded-concurrent zero-knowledge argument. We first introduce his simulation strategy for the protocol. The simulator is almost identical to the one for stand-alone setting [1]. Its trick is still to commit to verifier's next message function in GenProt. In concurrent setting on receiving the simulator's commitment, the verifier may switch to other sessions. When it returns to this session and sends r. Then the simulator collects all prover messages sent in this interval as y. Denote the transcript in GenProt by (h, z, r). Then y and committed program and used randomness are a witness for $(h, z, r) \in \Lambda$.

Our simulator basically follows the above simulation strategy but to adopt the strategy of computing messages of Step P1.k.1 in GenProts shown in the previous section. Assume V^* is a polynomial-time verifier and a is the auxiliary input for V^* and there are n sessions executed concurrently. We still use b to denote the rightmost portion of a V^* accesses so far. Our emphases is to describe the simulation strategy in GenProts. For the βn slots in n sessions let us order them according to the scheduling of the prover messages in them. When it needs to compute the commitment z in the q^{th} slot, the simulator does the following, almost as the one described in the previous section: If it has succeed in this session or n^q is less than b then it adopts the honest prover strategy. Otherwise it computes V^*'s next message function using the n^q-bit prefix of a as the auxiliary input and computes the commitment z. Denote by (Π, s) the committed program and the randomness for the commitment.

After the simulator sends z to V^*, V^* may switch to other sessions. When it returns to this session and responses r, if the simulator hasn't succeed in this session then it checks whether or not b, the rightmost portion V^* accesses so far, is larger than the length of the prefix of the auxiliary input it used in computing z. If not, the simulator adopts the simulation strategy in [1] to gather information y and stores (Π, s, y) and thus succeeds in this session.

Similarly, we have V^*'s running-time and $|a|$ is polynomial. Since there are β slots for each session, it can be seen that the simulator will succeed in all sessions and output indistinguishable views. Further, by Claim 3.8 it is reasonable to take in faith that this simulation strategy will lead to polynomial precision. We will prove these facts after presenting the actual description of the simulator in the following.

4.2.2 Actual Description

Our simulator algorithm follows the above description. We first introduce notations and present its actual description in Algorithm 4.6.

Notations. Through this subsection we will use i to index a session (i.e., $1 \leq i \leq n$), superscript j to index an overall prover/verifier message (i.e., $1 \leq j \leq un$, u is a super-constant). We will use subscript p and v to denote that the message is a prover's or verifier's message (e.g., m_{v}^j denotes the j^{th} verifier's message) and use parenthesized superscript to denote the session that a message belongs to (e.g., $r_k^{(i)}$). We will sometimes drop the session number when it is clear from the context. We will sometime identify a prover or verifier message not by its overall index, but rather by its session number and step number. Thus we will say statements like "let $r = r_k^{(i)}$ denote the message of Step V1.k.2 of the i^{th} session".

Algorithm 4.6. The precise simulator S'
Input: $x_1, \cdots, x_n \in \{0,1\}^n$: the statement to be proved in the i^{th} session is that $x_i \in L$; $V^* \in \{0,1\}^n$: description of a polynomial-time verifier coordinating an n-times concurrent execution. W.l.o.g. assume V^* is deterministic; $a \in \{0,1\}^{poly(n)}$: the auxiliary input for V^*.
Initialization: The simulator constructs such tables and variables as follows.
A: a table of length n. Initially A is empty. $A[i]$ is used to contain the witness (Π, s, y) such that $z = \mathrm{Com}(h(\Pi), s)$ and $\Pi(z, y) = r$ within $n^{\log\log n}$ steps for the i^{th} session.
β: a super-constant function $\beta(n) = \omega(1)$.
b: a variable recording the rightmost portion in a V^* accessed so far;

f: a table of length n. $f[i]$ is the flag denoting whether S' has succeed in the i^{th} session. Initially, set $f[i]$ false for all i.

q: a variable denoting the order number of the current slot, where we order the βn slots according to the scheduling of the prover messages in them. Initially, set $q = 0$.

α: a table of length n. $\alpha[i]$ is used to store the order number of the most recently visited slot in the i^{th} session.

Simulating each step: For $j = 1, \cdots, un$ the simulator computes the j^{th} prover's message m_{p}^{j} in the following way (where $u = \beta + c'$, c' is a constant denoting the number of prover's rounds in WIUA): Feed the previously computed messages $(m_{\mathsf{p}}^{1}, \cdots, m_{\mathsf{p}}^{j-1})$ to V^* to obtain j^{th} verifier's message (k, m). Set b as the rightmost bit of a V^{**} accessed so far. Compute the message m_j according to the current step in the simulated proof of the i^{th} session:

Step P1.k.1: If V^*'s message is for Step V0 of the i^{th} session, let $h = h^{(i)}$ denote the verifier's message. S' does the following:

1. Let $q \leftarrow q + 1$ and store q into $\alpha[i]$.
2. If $f[i] = \mathsf{true}$ or $n^q < b$, S' adopts the honest prover strategy, i.e., computes $z_k \leftarrow_{\mathrm{R}} \mathsf{Com}(h(0^n))$. Otherwise, i.e. if $f[i] = \mathsf{false}$ and $n^q \geq b$, S' computes V^*'s next message function using auxiliary input a_q, denoted Π_k, where a_q is the n^q-bit prefix of a (if $n^q > |a|$, then let $a_q = a$). Compute $z_k \leftarrow_{\mathrm{R}} \mathsf{Com}(h(\Pi_k); s_k)$, where s_k is the randomness.

Receiving message of Step V1.k.2: If V^*'s message is for Step V1.k.2 of the i^{th} session, do as follows:

1. If $f[i] = \mathsf{false}$ and $n^{\alpha[i]} \geq b$, set $f[i] = \mathsf{true}$. Let j' denote the overall index of prover's message in Step P1.k.1 of this session. That is, $m_{j'}$ was the message z_k of the i^{th} session. Let $y = (y_1, \cdots, y_{j-j'-1})$ denote the sequence $(m_{j'+1}, \cdots, m_{j-1})$. Note that $|y| \leq O(\beta n^3) < n^4/2 = |r_k/2|$ for an appropriate β. Add (Π_k, s_k, y) to the cell $A[i]$.
2. If $k < \beta$, compute the prover's message for Step P1.$k+1$.1 using the strategy described in "Step P1.k.1". Otherwise, compute the first prover's message in the WIUA using the strategy described in "Steps in WI-UARGs" below.

Steps in WI-UARGs: S' adopts the honest prover strategy to convince V^* that either $x \in L$ or $(h, z_k, r_k) \in \Lambda$ using the witness (Π_k, s_k, y) in $A[i]$ for the i^{th} session.

In the next two subsections we will follow the approach for proving Theorem 3.5 to complete the proof of Theorem 4.5.

4.3 The Simulator's Output Distribution

In this subsection we show S' can output indistinguishable view for V^*. [1] has presented an argument of this indistinguishability in case of $\beta = 1$ in bounded-concurrent setting. By the analysis presented in the previous section we have S' will succeed in every session. This means it can store a tripe (Π_k, s_k, y) (for some k, k's are distinct in different sessions) for every session. To apply the argument in [1] in our case, we should ensure that (Π_k, s_k, y) is the witness for the transcript in every session. This can be reduced to ensure that $|y|$ is less than $r/2$. Actually, assume for a specific session the simulator succeeds in the k^{th} slot. That is, it stores (Π_k, s_k, y) as the witness for the transcript. Assume y is $m_{j'+1}, \cdots, m_{j-1}$. Hence it still holds $|m_{j'+1}| + \cdots + |m_{j-1}| \leq O(\beta n^3) < n^4/2 = |r_k/2|$ for an appropriate β. This means (Π_k, s_k, y) is indeed a witness for the transcript in every session. When this fact is ensured, what we need to prove is the following claim.

Claim 4.7. *S'' output is computationally indistinguishable from the view of V^* in an n-times concurrent interaction of Protocol 4.2 with inputs $\{(x_i, w_i), x_i\}_{i=1}^n$, where w_i is a witness for $x_i \in L$.*

Proof. The proof of this claim is similar to the previous section. We prove this claim still using the hybrid argument. That is, we construct a hybrid simulator $\widehat{S'}$ which has $w_1, \cdots, w_n$ as the auxiliary inputs. $\widehat{S'}$ adopts S'' strategy in GenProts and adopts the honest prover strategy in WIUA using w_i as the witness in the i^{th} session. Then proving this claim is reduced to proving two subclaims (1) that $\widehat{S'}$' output is indistinguishable from S'' output and (2) that $\widehat{S'}$' output is indistinguishable from the real view.

1. The first subclaim basically follows from the fact that WI property is closed under concurrent composition. We order n sessions according to the scheduling of the first step in WIUAs. We construct $n+1$ hybrid simulators G_i, $0 \leq i \leq n$. Every G_i also gets $w_1, \cdots, w_n$ as the auxiliary inputs. In GenProts G_i behaves as S'. In WIUAs G_i adopts honest prover strategy using w_i's as the witnesses in the first i sessions and adopts S'' strategy in the last $n-i$ sessions. Clearly, $G_0 = S'$ and $G_n = \widehat{S'}$. So we prove this subclaim by showing G_i's output and G_{i+1}'s output are computationally indistinguishable for $0 \leq i \leq n-1$.

   Suppose there exists a polynomial-time algorithm D that distinguishes between G_i's output and G_{i+1}'s output with probability ϵ. We fix a choice of coins used for all sessions except the $i+1^{st}$ and GenProt of the $i+1^{st}$ such that D distinguishes between G_i's output and G_{i+1}'s output on this choice with probability ϵ. Then D can be converted to a distinguisher between prover's messages in the $i+1^{st}$ WIUA output by G_i and by G_{i+1} respectively, by hardwiring all messages before the $i+1^{st}$ WIUA, and since the later messages are a function of the public inputs and these messages and coins. Then by the WI property of WIUA ϵ is negligible. This means $\widehat{S'}$' output is indistinguishable from S'' output.

2. The second subclaim basically follows from the multiple-sampling security of Com. When n Protocol 4.2s are executed concurrently, there are $n\beta$ Coms in the interaction. Let us order the $n\beta$ commitments according to their scheduling. We construct $n\beta+1$ hybrid simulators $H_0, \cdots, H_{n\beta}$, where H_i adopts the $\widehat{S'}$' strategy in computing first i Coms and adopts the honest prover strategy in computing last $n\beta - i$ Coms for $0 \leq i \leq n\beta$ (each has $w_1, \cdots, w_n$ as the auxiliary inputs and uses them in WIUAs). Clearly, H_0's output equals the real view and $H_{n\beta} = \widehat{S'}$. Thus we only need to show H_i's output and H_{i+1}'s output are computationally indistinguishable for $0 \leq i \leq n\beta - 1$.

   Suppose there exists a polynomial-time algorithm D that distinguishes between H_i's output and H_{i+1}'s output with probability ϵ. Assume the $i+1^{st}$ Com belongs to the j^{th} session. Then we fix a choice of coins used for all sessions except the j^{th} and the first i Coms in the j^{th} such that D distinguishes between H_i's output and H_{i+1}'s output on this choice with probability ϵ. Then D can be converted to a distinguisher between the $i+1^{st}$ Com output by H_i and by H_{i+1} respectively by hardwiring all messages before the $i+1^{st}$ Com, and since the later messages are a function of the public inputs and these messages and coins. Then by the hiding property of Com ϵ is negligible. (H_{i+1} may compute the $i+1^{st}$ Com using honest prover strategy too. In this case H_i's output and H_{i+1}'s output are actually identical.) This means $\widehat{S'}$' output is indistinguishable from the real view.

□

4.4 The Simulator's Running Time

The goal of this subsection is devoted to the proof of the following claim, which states precision is still obtained in a n-times concurrently execution of Protocol 4.2.

Claim 4.8. *There exists a polynomial $p(n,t)$ such that for every $\overrightarrow{x} = (x_1, \cdots, x_n), x_i \in \{0,1\}^n \cap L$ and every sufficiently long $r \in \{0,1\}^*$ and every polynomial time machine V^* and its auxiliary input $a \in \{0,1\}^{poly(n)}$ letting $v \leftarrow S'_r(\overrightarrow{x}, V^*, a)$, $\mathrm{STEPS}_{S'_r(\overrightarrow{x},V^*,a)} \leq p(n, \mathrm{STEPS}_{V^*}(v))$.*

Proof. Fix a random tape $r \in \{0,1\}^*$ for S'. Let v denote the view S' generates. Since V^*'s maximum running-time and length of a are polynomial, there exist n constants $c_1, \cdots, c_n$ such that S' succeeds in the c_i^{th} slot in the i^{th} session for $1 \leq i \leq n$.

Denote by T_i the simulator's running-time in the i^{th} session (including the time of emulating V^*). Denote by t_i the V^*'s running-time in the i^{th} session. Then $\mathrm{STEPS}_{V^*}(v) = \sum_{i=1}^{n} t_i$ and $\mathrm{STEPS}_{S'_r(x,V^*,a)} = \sum_{i=1}^{n} T_i$. For abbreviation let $t = \mathrm{STEPS}_{V^*}(v)$ and $T = \mathrm{STEPS}_{S'_r(x,V^*,a)}$.

By the almost same analysis in the proof of Claim 3.8 we have the following inequality holds for all i.

$$T_i < \beta[l'(t_i n + \beta n^2) + p_h(l'(t_i n + \beta n^2))] + A(n) + p_{\mathsf{UA}}(t) + l t_i$$

Notice that the third summand at the right hand is $p_{\mathsf{UA}}(t)$ rather than $p_{\mathsf{UA}}(t_i)$. Actually, assume S' succeeds in the k^{th} slot in the i^{th} session and S' generates Π_k only using V^*'s code and the public inputs $\overrightarrow{x}$, a prefix of a and all prover messages prior to this step. In verifying $\Pi_k(z_k) = r_k$, Π_k needs first to do an internal running from the beginning to this step of receiving z_k, which may involve many sessions. Hence the time of this verifying may be larger than t_i, but must be less than t.

Let i run from 1 to n and sum up the above inequalities, we have

$$T < \beta[l'(tn + \beta n^3)] + \sum_{1 \leq i \leq n} \beta[p_h(l'(t_i n + \beta n^2))] + nA(n) + np_{\mathsf{UA}}(t) + lt$$

As $t_i < t$, it is obvious there exists a polynomial $p(n,t)$ which has degree d in t, $d = \max(c_h, c_{\mathsf{UA}})$, such that $T < p(n,t)$ (for any V^*, a and r). Thus we complete the proof. $\square$

By Claim 4.7 and Claim 4.8 we complete the proof of Theorem 4.5. Combining it with Theorem 4.4, we also complete the proof of Theorem 4.1.

Remark 4.9. So far we have shown there exist $\omega(1)$-round zero-knowledge arguments which are precisely simulatable in case n sessions are executed concurrently. Since the security parameter can be "scaled", this means that for every fixed polynomial $q(n)$, we can construct a protocol that remains zero-knowledge when executed $q(n)$ times concurrently. Notice that Protocol 4.2 uses only $\omega(1)$ rounds. This means the "scaled" protocol still uses $\omega(1)$ rounds. (This is because for each $\delta(q(n)) = \omega(1)$, the desired round complexity of the "scaled" protocol, $\delta(n)$ is still $\omega(1)$ and Theorem 4.1 ensures the existence of the $\delta(n)$-round protocols.) Further, the "scaled" protocol still obtains polynomial precision.

Remark 4.10. As we achieve the results Theorem 3.1 and Theorem 4.1, by using the technique in [2][3] to reduce the complexity assumption, our results Theorem 1.1 and Theorem 1.2 follow. In the next section we will sketch this technique.

5 Reducing the Complexity Assumption

In this section we follow the technique in [2][3] to sketch how to obtain Theorem 1.1 and Theorem 1.2 by reducing the stronger assumption in previous sections to the more standard assumption that there exist hash functions that collision-resistent against polynomial-sized circuits. Since the applications of this technique in proving these two theorems are almost same, we only sketch it with respect to stand-alone setting to show how to obtain Theorem 1.1.

The main tools to reduce the complexity assumption are random-access hashing schemes (see [18] for a detailed construction) and error correcting codes.

Error-correcting codes: We use ECC to denote an error correcting code. That is, ECC is a polynomial-time computable function that satisfies the following:

1. **Polynomial expansion:** There exists some function $l(\cdot)$ such that $l(n) < n^{c_1}$ for some constant $c_1 > 0$, and every string $x \in \{0,1\}^*$, $|\mathsf{ECC}(x)| = l(|x|)$.
2. **Constant distance:** There exists some constant $\delta > 0$ such that for every $x \neq x' \in \{0,1\}^n$, $|\{i|\mathsf{ECC}(x)_i \neq \mathsf{ECC}(x')_i\}| \geq \delta \cdot l(n)$.

[2][3] showed that we can obtain a simple construction for binary codes with polynomial expansion by concatenating the Reed-Solomon code with the Hadamard code (with the Hadamard code applied individually to each symbol of the Reed-Solomon code).

To apply random-access hash schemes and ECC the definition of Λ should be modified slightly, as follows.

Definition 5.1. (Language Λ [2]) $\tau = (\alpha, z, r)$ is in Λ if there exists a circuit Π of size at most $T = |r|^{\log\log|r|/5c_1}$ such that $z = \mathsf{Com}(h(\Pi))$ and $\Pi(z) = r$. A *witness* that $\tau = (\alpha, z, r) \in \Lambda$ is a triple (F, v, s) with the following properties:

1. $z = \mathsf{Com}(v; s)$.

2. F is a *certified* version of $\mathsf{ECC}(\Pi)$ with respect to the hash value v. This means, if let $m = |\mathsf{ECC}(\Pi)|$, then F is a function on $[m]$ such that for every $i \in [m]$, $F(i) = (\mathsf{ECC}(\Pi)_i, \sigma)$, where σ is a certificate for $\mathsf{ECC}(\Pi)_i$ w.r.t. the random-access hashing scheme. More formally, F passes verification if for every $i \in [m]$, $V_{\alpha,v}(i, b, \sigma) = 1$ where $(b, \sigma) = F(i)$ and V is the verification algorithm of the random-access hashing scheme.

3. Using the decoding algorithm for the error-correcting code it is possible to recover $\Pi = \mathsf{ECC}^{-1}(F)$. To be a witness, it must hold that $\Pi(z) = r$.

It can be seen that this is indeed a valid witness relation for the language Λ and that $\Lambda \in$ $\mathbf{Ntime}(n^{\log\log n})$. With this modified Λ, we present the protocol, as Protocol 5.2 shows.

To prove Theorem 1.1 we need to show that Protocol 5.2 is a precise public-coin zero-knowledge argument. Indeed, the proofs of (concurrent) zero-knowledge property and precision are virtually unchanged, as Section 3 (and Section 4) shows, so we omit these details. (Since the time of computing tree hashing is more than the time of computing h, the degree of the precisions in t might change.) Thus, all that is left to prove is the following:

Theorem 5.3. *Protocol 5.2 with respect to the modified Λ is an interactive argument for* **NP** *(under the assumption that the existence of hash functions that are collision-resistent against polynomial-sized circuits).*

Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$") **Prover's auxiliary input:** w (a witness that $x \in L$) **Additional parameter:** A super-constant function $\beta(n) = \omega(1)$	$\overset{w}{\downarrow}$ $\overset{x}{\downarrow}$ $\boxed{P}$ $\boxed{V}$
Step V0 (Choose random-access hash function): Verifier chooses a random hash function $\alpha \leftarrow_{\mathrm{R}} \mathcal{H}_n$ and sends α to prover. For $k = 1, \cdots, \beta(n)$, do the following: **Step P1.k.1 (Commitment to hash of "junk"):** Prover computes $z_k \leftarrow_{\mathrm{R}} \mathsf{Com}(h_\alpha(0^n))$ and sends z_k to verifier. **Step V1.k.2 (Send random string):** Verifier selects a string $r_k \leftarrow_{\mathrm{R}} \{0,1\}^n$ and sends it. End For	$\xleftarrow{\alpha \leftarrow_{\mathrm{R}} \mathcal{H}_n}$ $\xrightarrow{z_k = \mathsf{Com}(h_\alpha(0^n))}$ $\xleftarrow{r_k \leftarrow_{\mathrm{R}} \{0,1\}^n}$
Steps P,V2.X (WI Proof): Prover proves to verifier using its input w via the WIUA system that either $x \in L$ or there exists a k s.t. $(\alpha, z_k, r_k) \in \Lambda$.	

Protocol 5.2. The precise zero-knowledge argument for **NP** w.r.t. the modified Λ

Proof. The proof of completeness is straightforward. We skip this part and only prove the computational soundness property.

First of all, we follow the relaxed computational soundness requirement of GenProt presented by [3]: the original requirement required that the probability that a $n^{\log\log n}$-sized cheating prover outputs a witness is negligible. The relaxation is that we only require the soundness condition holds against polynomial-sized provers. However, we require that such provers cannot even output an implicit representation of the witness (i.e., a circuit C such that $[C]$ is a witness). Note that in $n^{\log\log n}$ time it is possible to convert an implicit representation to an explicit representation.

Then suppose there exists a polynomial-sized cheating prover able of cheating the honest verifier to accept some $x \notin L$ with probability ε. Hence from this cheating prover [1] constructed a polynomial-sized P^* which can violate the computational soundness of GenProt with probability polynomially related to ε, i.e., P^* is able of computing an implicit representation of the witness at the end of GenProt. Thus, to prove computational soundness of Protocol 5.2 we only need to show GenProt is computationally sound for any polynomial-sized cheating prover.

In the same way as the computational soundness proof in Theorem 3.4, we can use P^* to obtain in polynomial-time two certified encoding F and F' of two different Π_k and Π'_k. Now, if we choose i at random from $[m]$, then with constant-probability $\mathsf{ECC}(\Pi_k)_i \neq \mathsf{ECC}(\Pi'_k)_i$. This means that with constant-probability $F(i)$ and $F'(i)$ yield two contradicting certificates for the same hash value, contradicting the security of the random-access hash value. □

References

[1] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In Proc. 42nd FOCS, IEEE, pages 106-115, 2001.

[2] B. Barak. Non-Black-Box Techniques in Cryptography. Dissertation for the Doctoral Degree, the Weizmann Institute of Science, 2004.

[3] B. Barak and O. Goldreich. Universal Arguments and Their Applications. In Porc. 17th CCC, IEEE, pages 194-203, 2002.

[4] B. Barak, S. J. Ong, and S. Vadhan. Derandomization in Cryptography, In Proc. Crypto'03, Springer-Verlag, LNCS 2729, pages 299-315, 2003.

[5] M. Blum. Coin Flipping by Phone. In Proc. 24th Computer Conference (Comp- Con), IEEE, pages 133-137, 1982.

[6] M. Blum. How to Prove a Theorem So No One Else Can Claim It. In Proc. the International Congress of Mathematicians, Berkeley, California, USA, pages 1444-1451, 1986.

[7] G. Brassard, D. Chaum, and C. Crépeau. Minimum Disclosure Proofs of Knowledge. J. Comput. Syst. Sci., 37(2) (1988):156-189.

[8] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. In Proc. 33rd STOC, ACM, pages 570-579, 2001.

[9] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero Knowledge. In Proc. 30th STOC, ACM, pages 409-418, 1998.

[10] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In Proc. 22nd STOC, ACM, pages 416-426, 1990.

[11] O. Goldreich. Foundations of Cryptography - Basic Tools. Cambridge University Press, Cambridge, 2001.

[12] O. Goldreich. Concurrent Zero-Knowledge with Timing, Revisited. In Proc. STOC'02, ACM, pages 332-340, 2002.

[13] O. Goldreich and L. A. Levin. A Hard-Core Predicate for All One-Way Functions. In Proc. 21st STOC, pages 25-32, ACM, 1989.

[14] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems, Journal of Cryptology, Vol. 7, No. 1(1994), pages 1-31.

[15] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In Proc. 17th STOC, ACM, pages 291-304, 1985.

[16] J. Kilian and E. Petrank. Concurrent Zero-Knowledge in Poly-logarithmic Rounds. In Proc. STOC'01, ACM, pages 560-569, 2001.

[17] Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In Proc. STOC'03, ACM, pages 683-692, 2003.

[18] R. C. Merkle. A Certified Digital Signature. In Proc. Crypto'89, Springer-Verlag, LNCS 435, pages 218-238, 1989.

[19] S. Micali and R. Pass. Local Zero Knowledge. In Proc. 38th STOC, ACM, pages 306-315, 2006.

[20] M. Naor. Bit Commitment Using Psedorandomness. J. Cryptology, 4(2)(1991), pages 151-158.

[21] M. Naor, R. Ostrovsky, R. Venkatesan and M. Yung. Zero-Knowledge Arguments for NP Can Be Based on General Assumptions. J. Cryptology, 11(1998), pages 87-108.

[22] M. Naor and M. Yung. Universal One-Way Hash Functions and Their Cryptographic Applications. In Proc. 21st STOC, ACM, pages 33-43, 1989.

[23] O. Pandey, R. Pass, A. Sahai, W. Tseng and M. Venkitasubramaniam. Precise Concurrent Zero-Knowledge. In Proc. Eurocrypt'08, Springer-Verlag, LNCS 4965, pages 397-414, 2008.

[24] R. Pass and A. Rosen, Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In Proc. FOCS'03, IEEE, pages 404-413, 2003.

[25] R. Pass, Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. In Proc. STOC'04, ACM, pages 232-241, 2004.

[26] R. Pass. A Precise Computational Approach to Knowledge, Dissertation for the Doctoral Degree, MIT, 2006.

[27] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent Zero Knowledge with Logarithmic Round-Complexity. In Proc. 43rd FOCS, IEEE, pages 366-375, 2002.

[28] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In Proc. Eurocrypt'99, Springer-Verlag, LNCS 1592, pages 415-431, 1991.