# COMP6712 Lecture Note 8

Privacy-Enhancing Technologies 2: Zero Knowledge Proof

Bin XIE, Bowen CUI and Xuyuan CAI

April 2, 2024

## Summary

In this lecture, we delve into identification protocols, demonstrating how to construct a robust signature scheme grounded in such protocols. Our focus begins with exploring Schnorr's identification protocol, which is renowned for its elegance and efficiency, from which the Schnorr signature scheme emerges. This scheme boasts provable security under the Discrete Logarithm (DL) assumption, leveraging a hash function treated as a random oracle. Expanding our discussion, we generalize these protocols, introducing the concept of Sigma protocols while offering insights into various implementations. Lastly, we delve into non-interactive zero-knowledge proof systems, exploring their application in higher dimensions. Additionally, we provide an overview of SNARK techniques, showcasing their ability to verify arbitrary NP relations using arithmetic circuits.

## 1 Identification Protocols and Signatures

### 1.1 Identification/Authentication Paradigm

An identification protocol is a cryptographic protocol that allows one party (the prover) to prove its identity to another party (the verifier) in a secure and verifiable manner. The goal of an identification protocol is to establish trust and confidence between the parties involved in a communication or transaction.

Abstractly, the identification problem involves two parties, a prover and a verifier.The prover has a secret key $sk$ that it uses to convince the verifier of its identity,while the verifier has a verification key $vk$ to confirm the prover's claim.

**Definition 1.1 (Identification Protocol).** *An identification protocol is a triple $\mathcal{I} = (G, P, V)$.*

- *$G$ is a probabilistic, key generation algorithm, that takes no input, and outputs a pair $(vk, sk)$, where $vk$ is called the verification key and$sk$ is called the secret key.*
- *$P$ is an interactive protocol algorithm called the prover, which takes as input a secret key $sk$ , as output by $G$ .*
- *$V$ an interactive protocol algorithm called the verifier, which takes as input a verification key $vk$ , as output by $G$, and which outputs* accept*or* reject*.*

*In the protocol, $P$ and $V$ interact with each other. For all possible outputs $(vk, sk)$ of $G$, if $P$ is initialized by $sk$ and $v$ is initialized by $vk$, $V$ output* accept *with probability of 1 at the end of the interaction. between $P$ and $V$.*

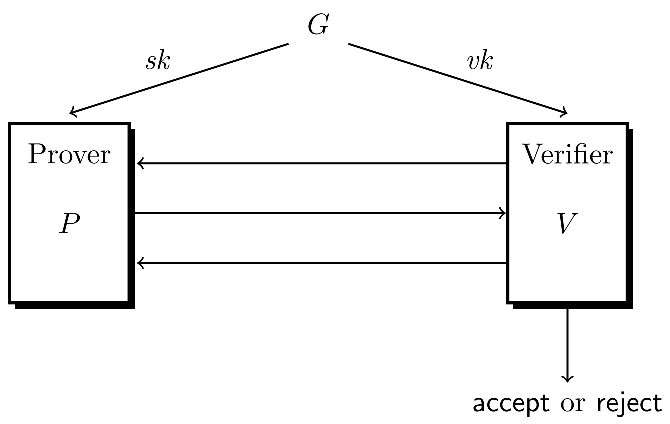


Figure 1: Identification Protocol.

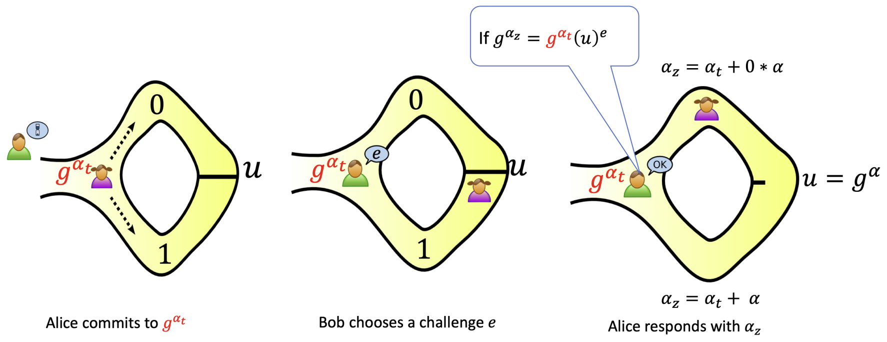

If $g^{\alpha_z} = g^{\alpha_t}(u)^e$
$\alpha_z = \alpha_t + 0 * \alpha$
0
1
$u$
$g^{\alpha_t}$
$e$
OK
$u = g^{\alpha}$
$\alpha_z = \alpha_t + \alpha$
Alice commits to $g^{\alpha_t}$
Bob chooses a challenge $e$
Alice responds with $\alpha_z$

Figure 2: A toy example: Ali Baba Cave

## 1.2 A toy example: Ali Baba Cave

There is a well-known story presenting the fundamental ideas of zero-knowledge proofs, first published in 1990 by Jean-Jacques Quisquater and others in their paper "How to Explain Zero-Knowledge Protocols to Your Children". As shown in fig,the two parties in the zero-knowledge proof story are Alice as the prover of the statement, and Bob, the verifier of the statement[QQQ+89].

In this story, Alice has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a ring, with the entrance on one side and the magic door blocking the opposite side. Bob wants to know whether Alice knows the secret word; but Alice, being a very private person, does not want to reveal her knowledge (the secret word) to Bob or to reveal the fact of her knowledge to the world in general.

They label the left and right paths from the entrance 1 and 0. First, Bob waits outside the cave as Alice goes in. Alice takes either path 1 or 0; Bob is not allowed to see which path she takes. Then, Bob enters the cave and shouts the name of the path he wants her to use to return, either 1 or 0, chosen at random. Providing she really does know the magic word, this is easy: she opens the door, if necessary, and returns along the desired path.

Suppose Alice did not know the magic world, she will randomly select one path 1 or 0. Then, when bob shouts the name of the path he want, Alice has a $1/2$ chance to return from the desired path because she can not open the door. If we repeat this process $n$ times, the probability of Alice returning from the desired path is $1/2^n$.

This process can be concretely formalized as a Schooner identification protocol, assuming the

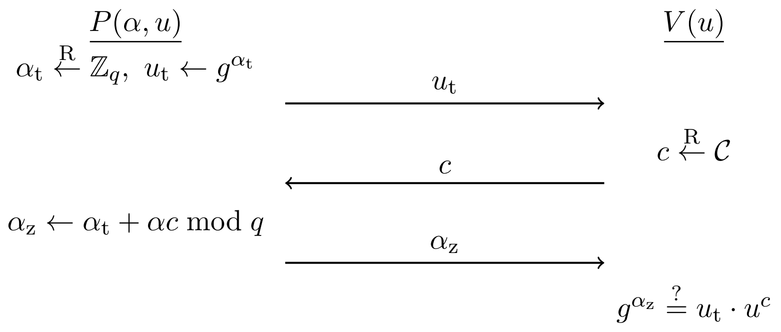

$P(\alpha, u)$
$V(u)$
$\alpha_{\mathrm{t}} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q,\ u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{t}}}$
$u_{\mathrm{t}}$
$c \overset{\mathrm{R}}{\leftarrow} \mathcal{C}$
$c$
$\alpha_{\mathrm{z}} \leftarrow \alpha_{\mathrm{t}} + \alpha c \bmod q$
$\alpha_{\mathrm{z}}$
$g^{\alpha_{\mathrm{z}}} \overset{?}{=} u_{\mathrm{t}} \cdot u^c$

Figure 3: Schnorr Identification Protocol.

magic word is $u = g^{\alpha}$. In this protocol, the first step is for Alice to commit to $g^{\alpha_t}$. Then Bob randomly chooses a challenge $e$. Finally, Alice responds with $\alpha_z = \alpha_t + e \cdot \alpha$. If the equation $g^{\mathrm{z}} = g^{\alpha_{\mathrm{t}}} \cdot u^e$ holds, Bob will accept that Alice know the magic word.

If Alice doesn't know the key, the proof was accepted with $1/n$. If repeat this process $n$ times, Bob will accpet with $1/2^n$

## 1.3 Schnorr's Identification Protocol

Let $\mathbb{G}$ be a cyclic group of prime order $q$ with generator $g \in \mathbb{G}$. Suppose the prover $P$ has a secret key $sk = \alpha \in \mathbb{Z}_q$, and the corresponding public verification key is $vk = u = g^{\alpha} \in \mathbb{G}$. To prove his identity to a verifier $V$, $P$ wants to convince $V$ that he knows $\alpha$ The simplest way to do this would be for $P$ simply send $\alpha$ to $V$. Instead, Schnorr's protocol is a clearly designed interective protocol that allows $P$ to convince $V$ that he knows the discrete logarithm of u to the base $g$, without actually sending this value to $V$ [Sch90].

**Definition 1.2 (Schnorr's Identification Protocol).** *Let $C$ be the subset of $\mathbb{Z}_q$, Schnorr's identification protocol is $\mathcal{I}_{\text{sch}} = (G, P, V)$, where:*

- *The key generation algorithm $G$ runs as follows:*

$$\alpha \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha}.$$

  *The verification key is $vk := u$, while the secret key is $sk := \alpha$.*

- *$P$ is a prover algorithm which takes a secret key $sk = \alpha$ as input.*

- *$V$ is a verifier algorithm which takes a verification key $vk = u$ as input.*

- *The protocol between $P$ and $V$ runs as follows, where the prover $P$ is initialized with $sk = \alpha$, and the verifier $V$ is initialized with $vk = u$.*

  1. *$P$ computes $\alpha_{\mathrm{t}} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$, $u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{t}}}$, and sends $u_{\mathrm{t}}$ to $V$;*
  2. *$V$ computes challenge $c \overset{\mathrm{R}}{\leftarrow} \mathcal{C}$ then sends $c$ to $P$;*
  3. *$P$ computes $\alpha_{\mathrm{z}} \leftarrow \alpha_{\mathrm{t}} + \alpha c \in \mathbb{Z}_q$ , and sends $\alpha_{\mathrm{z}}$ to $V$;*
  4. *$V$ checks if $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$ holds, if so, $V$ outputs* accept, *otherwise outputs* reject.

An interaction between $P(\alpha)$ and $V(u)$ generates a conversation$(ut, c, \alpha_{\mathrm{z}}) \in \mathbb{G} \times C \times \mathbb{Z}_q$. We call such a conversation **an accepting conversation** for $u$ if $V$ 's check passes, i.e., if $g^{g_{\mathrm{z}}^{\alpha}}$. It is easy to see that an interaction between $P$ and $V$ always generates an accepting conversation, since if $u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{t}}}$ and $\alpha_{\mathrm{z}} = \alpha_{\mathrm{t}} + \alpha c$ then

$$g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$$

Therefore, Schnorr's protocol satisfies the basic correctness requirement that any identification protocol must satisfy.

**Theorem 1.1 (Security against Direct Attacks).** *The Schnorr's protocol is secure against direct attacks. In proving this, we can see that any efficient adversary that can succeed in a direct impersonation attack with non-negligible probability can be turned into an algorithm that efficiently recovers the secret key $\alpha$ from the verification key $u$. For this reason, Schnorr's protocol is sometimes called a "proof of knowledge" of a discrete logarithm.*

*Proof idea.* Suppose $\mathcal{A}$ has advantage $\epsilon$ in attacking $\mathcal{I}_{\mathrm{sch}}$. The challenger generates the verification key $u = g^{\alpha}$. In his impersonation attempt, $\mathcal{A}$ generates the first transcript $u_{\mathrm{t}}$ arbitrarily. To succeed, $\mathcal{A}$ must respond to the random challenge $c$ with a valid response $\alpha_{\mathrm{z}}$ which satisfies $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$. Actually, if $\mathcal{A}$ can generate a valid response to such a challenge with probability $\epsilon$, it should be able to generate a valid response to 2 such challenges with probability $\epsilon^2$.

Then, we can take advantage of $\mathcal{A}$ to compute the discrete logarithm of a random $u \in \mathbb{G}$. Use $u$ as the verification key in $\mathcal{I}_{\mathrm{sch}}$, and let $\mathcal{A}$ generate the first transcript $u_{\mathrm{t}}$. Then, we feed a random challenge $c$ to $\mathcal{A}$ and hope it can generate a valid response $\alpha_{\mathrm{z}}$. If this happens, we can *rewind* $\mathcal{A}$'s internal state back to the point when it just finished generating $u_{\mathrm{t}}$, and feed it with another challenge $c'$, and hope it to generates another valid response $u_{\mathrm{t}}'$.

If all these happens (with probability $\approx \epsilon^2$), then we obtain 2 valid transcripts $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ and $(u_{\mathrm{t}}, c', \alpha_{\mathrm{z}}')$ for a given verification key $u$ and the with first flows $u_{\mathrm{t}}$. Moreover, with overwhelming probability, we have $c' \neq c$. Then, since either transcript is valid, we have the following equations:

$$g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c, \quad g^{\alpha_{\mathrm{z}}'} = u_{\mathrm{t}} \cdot u^{c'}.$$

Dividing the first equation by the second, the $u_t$'s cancel, and we have

$$g^{\alpha_{\mathrm{z}} - \alpha_{\mathrm{z}}'} = u^{c - c'}.$$

Since $c \neq c'$ and the group order $q$ is prime, $1/(c - c')$ must exists in $\mathbb{Z}_q$. So we can get:

$$g^{(\alpha_{\mathrm{z}} - \alpha_{\mathrm{z}}')/(c - c')} = u.$$

Therefore, the security of Schnorr is equivalent to the possibility of constructing two valid transcripts with probability $\approx \epsilon^2$, which is still a discrete logarithm problem. While assume the dicrete logarithm of $\mathbb{G}$ is hard. If someone passes the verification of schnorr Identification, we must have he knows the secrete $\alpha$ □

We showed that any adversary which can successfully perform a direct attack with non-negligible probability can be converted into an algorithm that efficiently recovers the secret key $\alpha$ from the verification key $u$. For this reason, Schnorr's identification protocol is sometimes referred to as a *proof of knowledge* of discrete logarithms.

**Theorem 1.2 (Security against Eavesdropping Attacks).** *We have shown that Schnorr's identification protocol is secure against direct attacks, under the DL assumption. In fact, under the same assumption, we can show that Schnorr's identification protocol is secure against eavesdropping attacks as well.*

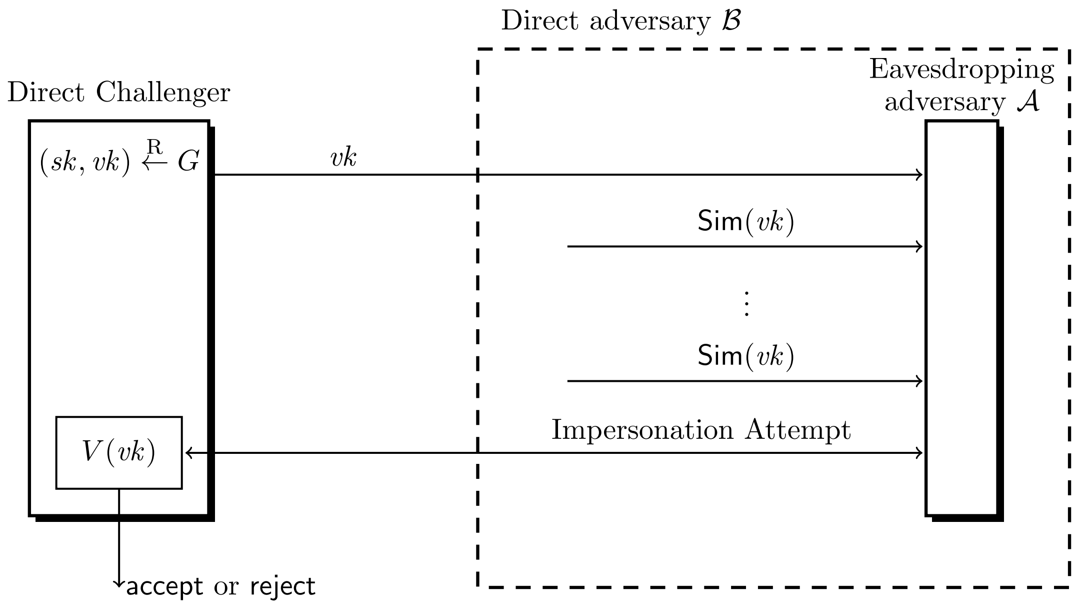


Figure 4: Proof of Theorem 1.2.

*Proof idea.* Now, in an eavesdropping attack, the adversary obtains $vk$ and a list of transcripts — conversations between $P$ (on input $sk$ ) and $V$ (on input $vk$ ). The idea is to show that these conversations do not help the adversary, because the adversary could have efficiently generated these conversations by himself, given $vk$ (but not $sk$ ). If we can show this, then we are done.

Indeed, suppose$\mathcal{A}$ is an adversary whose advantage in carrying out a successful impersonation via an eavesdropping attack is non-negligible. Then, we have another $\mathcal{B}$ in direct attacks. $\mathcal{B}$ can generates the valid respond $\alpha_z, c, u_{\mathrm{t}}$ by themselves, with probability $\epsilon$. Then he feed them to $\mathcal{A}$. So, it is easy to prove that adversary $\mathcal{A}$ has the same probability to extract $\alpha$ as adversary $\mathcal{B}$.

□

**Definition 1.3 (Honest Verifier Zero-Knowledge, HVZK).** *Let $\mathcal{I} = (G, P, V)$ be an identification protocol. We say that $\mathcal{I}$ is honest verifier zero-knowledge (HVZK) if there is an efficient probabilistic simulation algorithm* $\mathsf{Sim}$ *such that for all possible $(vk, sk)$ generated by $G$, the output of* $\mathsf{Sim}(vk)$ *is indistinguishable with the transcript between $P(sk)$ and $V(vk)$.*

*The term "zero knowledge" is meant to suggest that an adversary learns nothing from $P$ , because an adversary can simulate conversations on his own (using the algorithm Sim), without knowing sk . The term "honest verifier" conveys the fact this simulation only works for conversations between $P$ and the actual, "honest" verifier $V$ , and not some arbitrary, "dishonest" verifier,*

**Theorem 1.3.** *Schnorr's identification protocol is HVZK.*

*Proof.* The idea is that in generating a simulated conversation

$$(ut, c, \alpha_{\mathrm{z}})$$

, we do not need to generate the messages of the conversation in the given order, as in a real conversation between $P$ and $V$ . Indeed, our simulator Sim generates the messages in reverse order. On input $vk = u$, the simulator *Sim* computes

$$\alpha_{\mathrm{z}} \xleftarrow{R} \mathbb{Z}_q, c \xleftarrow{R} \mathcal{C}, u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{z}}}/u^c$$

Now we argue that the output of Sim on input $\mathrm{vk} = \mathrm{u}$ has the right distribution. Because $\alpha_z$ and $c$ are independent and the value $u_\mathrm{t}$ is uniquely determined by $u_\mathrm{t} = g^{\alpha_\mathrm{z}}/u^c$.It should be clear that this is the same as the output distribution of the simulator.

□

**Theorem 1.4 (Schnorr's Security).** *If Schnorr's identification protocol is secure against direct attacks, then it is also secure against eavesdropping attacks.*

*Proof.* This theorem comes directly from Theorem 1.3 and Theorem 1.2. □

To summarize, Schnorr's identification protocol has the following three important properties:

1. **Completeness**: if $P$ and $V$ execute the protocol honestly, the proof is accept.
2. **Soundness**: If $V$ outputs accept, we can extract a valid witness $\alpha$ effectively.
3. **Honest Verifier Zero-Knowledge**: we can efficiently simulate valid transcripts even if we do not know the witness $\alpha$.

## 1.4 Schnorr Signature

The Schnorr's identification protocol can be convert into a signature scheme. The signature scheme can be proven secure in the random oracle model under the DL assumption. Later in this chapter, we will see that this construction is actually a specific instance of a more general construction.

We start with Schnorr's identification protocol $\mathcal{I}_\mathrm{sch}$, which is defined in terms of a cyclic group $\mathbb{G}$ of prime order $q$ with generator $g \in \mathbb{G}$, along with a challenge space $\mathcal{C} \subseteq \mathbb{Z}_q$. Now, we need a hash function $H : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$, which will be modeled as a random oracle, where $\mathcal{M}$ is the message space in the signature scheme [Sch91].

**Definition 1.4 (Schnorr Signature).** *Schnorr signature scheme is*$\mathcal{S}_\mathrm{sch} = (G, S, V)$*, where:*

- *$G$ is a probabilistic key generation algorithm that runs as follows:*

$$\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad u \leftarrow g^\alpha.$$

*The public key is $pk := u$, while the secret key is $sk := \alpha$.*

- *$S$ is a signing algorithm which signs a message $m \in \mathcal{M}$ using a secret key $sk = \alpha$. $S$ runs as follows:*

$$\alpha_\mathrm{t} \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad u_\mathrm{t} \leftarrow g^{\alpha_\mathrm{t}}, \quad c \leftarrow H(m, u_\mathrm{t}), \quad \alpha_\mathrm{z} \leftarrow \alpha_\mathrm{t} + \alpha c.$$

*$S$ outputs $\sigma := (u_\mathrm{t}, \alpha_\mathrm{z})$ as the signature on the message $m$.*

- *$V$ is a verification algorithm which verifies a signature $\sigma = (u_\mathrm{t}, \alpha_\mathrm{z})$ on a message $m \in \mathcal{M}$ using a public key $pk = u$. To this end, $S$ computes $c \leftarrow H(m, u_\mathrm{t})$, and output* accept *if and only if $g^{\alpha_\mathrm{z}} = u_\mathrm{t} \cdot u^c$.*

## 1.5 The Identification for Decisional Diffie-Hellman

The Identification for Decisional Diffie-Hellman (ID-DDH) algorithm is a cryptographic protocol that allows a prover to prove its knowledge of a Diffie-Hellman private key to a verifier in a secure and verifiable manner. The goal of the protocol is to provide identity authentication based on the Decisional Diffie-Hellman (DDH) assumption.

**Definition 1.5 (The Identification for Decisional Diffie-Hellman).** *Schnorr signature scheme is* $\mathcal{S}_{\mathrm{ID}_{\mathrm{DDH}}} = (G, P, V)$*, where:*

- *The key generation algorithm $G$ runs as follows:*

  $$\beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad v \leftarrow g^{\beta}, \quad w \leftarrow u^{\beta}.$$

  *The verified key is $vk := (u, v, w)$, while the secret key is $sk := \beta$.*

- *$P$ is a prover algorithm which takes a secret key $sk = \beta$ as input.*

- *$V$ is a verifier algorithm which takes a verification key $vk = (u, v, w)$ as input.*

- *The protocol between $P$ and $V$ runs as follows, where the prover $P$ is initialized with $sk = \beta$, and the verifier $V$ is initialized with $vk = (u, v, w)$.*

  1. *$P$ computes $\beta_{\mathrm{t}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{t}}}$,and $w_{\mathrm{t}} \leftarrow u^{\beta_{\mathrm{t}}}$, then sends $u_{\mathrm{t}}, w_{\mathrm{t}}$ to $V$;*
  2. *$V$ computes challenge $c \xleftarrow{\mathrm{R}} \mathcal{C}$ then sends $c$ to $P$, where $\mathcal{C}$ is the subset of $\mathbb{Z}_q$;*
  3. *$P$ computes $\beta_{\mathrm{z}} \leftarrow \beta_{\mathrm{t}} + \beta c \in \mathbb{Z}_q$ , and sends $\beta_{\mathrm{z}}$ to $V$;*
  4. *$V$ checks if $g^{\beta_{\mathrm{z}}} = v_{\mathrm{t}} \cdot v^c$ and $u^{\beta_{\mathrm{z}}} = w_{\mathrm{t}} \cdot w^c$ holds, if so, $V$ outputs* accept*, otherwise outputs* reject.

The Sigma protocol satisfied the following properties:

1. **Completeness**: If $P$ and $V$ execute the protocol honestly, $g^{\beta_{\mathrm{z}}} = g^{\beta_{\mathrm{t}} + \beta c} = v_{\mathrm{t}} \cdot v^c$ and $u^{\beta_{\mathrm{z}}} = u^{\beta_{\mathrm{t}} + \beta c} = w_{\mathrm{t}} \cdot w^c$ must hold, the proof is accepted.

2. **Soundness**: Given two valid transaction $(u_{\mathrm{t}_1}, w_{\mathrm{t}_1}, c_1, \beta z_1)$ and $(u_{\mathrm{t}_2}, w_{\mathrm{t}_2}, c_2, \beta z_2)$ with $z_1 \neq z_2$, the verifier can extract the witness $\beta$.

3. **Honest Verifier Zero-Knowledge**: Witout knonwing the witness, simulator always output an accepting conversation for input challenge $c$.

$$\beta_{\mathrm{z}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q, c \xleftarrow{\mathrm{R}} \mathcal{C}, v_{\mathrm{t}} = \frac{g^{\beta_{\mathrm{z}}}}{v^c}, u_{\mathrm{t}} = \frac{u^{\beta_{\mathrm{z}}}}{u^c}$$

# 2 Sigma Protocol

## 2.1 Definition of Sigma Protocol

The introduced Schnorr's protocol above is a special case of a series of protocols named *Sigma Protocols*. To better understand Sigma Protocol, we introduce some basic concepts and then concisely describe them.

**Definition 2.1 (Binary Relation).** *$R \subseteq \mathcal{X} \times \mathcal{Y}$ is a binary relation, where $\mathcal{X}, \mathcal{Y}$ and $\mathcal{R}$ are finite sets.*

**Definition 2.2 (Language).** *Let $R \subseteq \mathcal{X} \times \mathcal{Y}$ be a binary relation. We call $y \in \mathcal{Y}$ a* ***statement****, and $x$ is a* ***witness*** *if $(x, y) \in \mathcal{R}$ holds. The language $\mathcal{L}_{\mathcal{R}}$ is the set of all true statements in $\mathcal{R}$.*

$$\mathcal{R}_{\mathcal{L}} = \{\, y \,|\, \exists x \in \mathcal{X} \ s.t. \ (x, y) \in \mathcal{R}\}.$$

For example, $\mathcal{R}_{\mathcal{DDH}}$ is relation defined over $\mathbb{Z}_p \times \mathbb{G}_p^3$ where $\mathcal{R}_{\mathcal{DDH}} = \{(\beta, (u, v, w)) \in \mathbb{Z}_p \times \mathbb{G}_p^3 : v = g^\beta \wedge w = u^\beta\}$. $\mathcal{L}_{\mathcal{DDH}} = \{\, (u, v, w) \,|\, \exists \beta \in \mathbb{Z}_p \ s.t. \ (\beta, (u, v, w)) \in \mathcal{R}_{\mathcal{DDH}}\}$.

Here, we give the definition of a sigma protocol.

**Definition 2.3 (Sigma Protocol).** *Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be a binary relation. A Sigma protocol over $\mathcal{R}$ is a four-round interactions between two parties ($P$ and $V$), where:*

- *$P$ is the prover and initially holds $(x, y) \in \mathcal{R}$ as the input.*
- *$V$ is the verifier who initially holds the statement $y$ as the input and outputs* accept *or* reject *at the end of the protocol.*
- *$P$ and $V$ conduct the following interactions:*
  - *$P$ generates a commitment $t$ and sends it to $V$;*
  - *On receiving $t$ from $P$, $V$ generates a challenge $c$ and returns it back to $P$;*
  - *On receiving $c$ from $V$, $P$ generates a response $z$ based on $c$ and $(x, y)$ and sends $z$ to $V$;*
  - *Upon receiving $z$ from $P$, $V$ outputs either* accept *or* reject *based on the statement $y$ and the transcript $(t, c, z)$.*

The Sigma protocol also has the same properties as Schnorr's identification protocols described as the following:

1. **Completeness**: If $P$ and $V$ execute the protocol honestly, the verifier always outputs accept.
2. **Special Soundness**: Given two valid transcripts $(t, c_1, z_1)$ and $(t, c_2, z_2)$, the verifier can extract the witness $x$.
3. **Honest Verifier Zero-Knowledge**: The verifier can efficiently generate valid transcripts for $y \in \mathcal{Y}$ without knowing witness $x \in \mathcal{X}$.

## 2.2 Cases of Sigma Protocol

Sigma protocol can be applied to a lot of applications to prove binary relations, such as DDH relation $\mathcal{R}_{\mathcal{DDH}} = \{(\beta, (u, v, w)) \in \mathbb{Z}_p \times \mathbb{G}_p^3 : v = g^\beta \wedge w = u^\beta\}$, Pederson commitment [Ped91] relation $\mathcal{R}_{com} = \{((\alpha, \beta), u) \in \mathbb{Z}_p^2 \times \mathbb{G}_p : u = g^\alpha \cdot h^\beta\}$. For a specific application, the construction of the Schnorr identification protocol relies on the proof of a discrete logarithm relation, which is described as the following.

$$\mathcal{R}_{sch} = \{(\alpha, u) \in \mathbb{Z}_p \times \mathbb{G}_p : \ u = g^\alpha\}$$

We propose a concrete Sigma protocol construction for Pederson's commitment relation in the following: assume that $\mathbb{G}_p$ is a cyclic group with a prime order $p$ and $g, h \in \mathbb{G}_p$ are generators. The prover holds $u \in \mathbb{G}_p$ (statement) and two secret values $\alpha, \beta$ (witness) that $u = g^\alpha \cdot h^\beta$. Okamoto's protocol gives a specific construction of Sigma protocol for $\mathcal{R}_{com}$, allowing the prover to show that it knows the opening of the commitment but does not expose the secret values to the verifier. The relation that the Okamoto protocol proves is as follows:

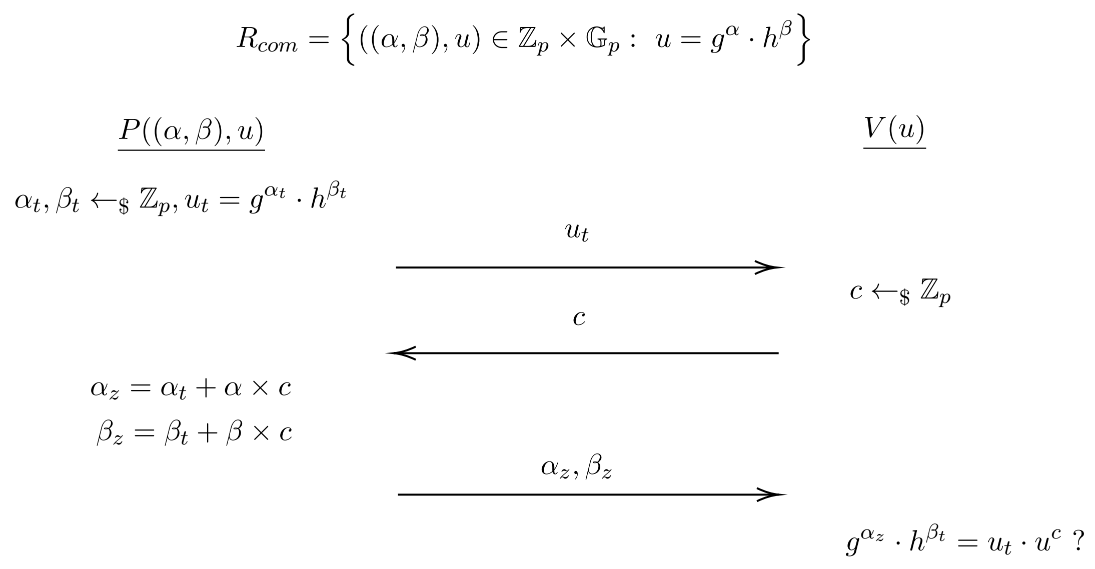

$R_{com} = \left\{((\alpha,\beta),u) \in \mathbb{Z}_p \times \mathbb{G}_p : \ u = g^{\alpha} \cdot h^{\beta}\right\}$
$P((\alpha,\beta),u)$
$V(u)$
$\alpha_t, \beta_t \leftarrow_{\$} \mathbb{Z}_p, u_t = g^{\alpha_t} \cdot h^{\beta_t}$
$u_t$
$c \leftarrow_{\$} \mathbb{Z}_p$
$c$
$\alpha_z = \alpha_t + \alpha \times c$
$\beta_z = \beta_t + \beta \times c$
$\alpha_z, \beta_z$
$g^{\alpha_z} \cdot h^{\beta_t} = u_t \cdot u^c$ ?

Figure 5: Okamoto's Protocol.

$$\mathcal{R}_{com} = \{((\alpha,\beta),u) \in \mathbb{Z}_p^2 \times \mathbb{G}_p : u = g^{\alpha} \cdot h^{\beta}\}$$

Here, we give a description of Okamoto's Protocol, which is illustrated in Figure 2.2.

**Definition 2.4 (Okamoto's Protocol).** *The prover and the verifier interact as the following operations to prove relation $\mathcal{R}_{com}$ with the statement $u$.*

- *$P$ randomly chooses two values $\alpha_t, \beta_t \leftarrow_{\$} \mathbb{Z}_p$, computes $u_t = g^{\alpha_t} \cdot h^{\beta_t}$, and then sends $u_t$ to the verifier.*
- *On receiving $u_u$, $V$ randomly chooses a challenge $c \leftarrow_{\$} \mathbb{Z}_p$ and returns it to $P$.*
- *On receiving $c$, $P$ computes $\alpha_z = \alpha_t + \alpha \cdot c, \beta_z = \beta_t + \beta \cdot c$, and return $\alpha_z$ and $\beta_z$ to $V$.*
- *$V$ checks whether $g^{\alpha_z} \cdot h^{\beta_z} = u_t \cdot u^c$. If true, $V$ outputs* accept*. Otherwise, $V$ outputs* reject*.*

Okamoto's protocol satisfies the three properties described in the Sigma protocol.

- **Completeness**: The completeness is trivial, because $g^{\alpha_z} \cdot h^{\beta_z} = g^{\alpha_t + \alpha \cdot c} \cdot h^{\beta_t + \beta \cdot c} = (g^{\alpha_t} \cdot h^{\beta_t}) \cdot (g^{\alpha} \cdot h^{\beta})^c = u_t \cdot u^c$. The above equation holds if $P$ and $V$ honestly execute the protocol.
- **Special Soundness**: If the verifier obtains two valid transcripts $(u_t, c, \alpha_z, \beta_z)$ and $(u_t, c', \alpha'_z, \beta'_z)$ for the same statement $u$, it can extract the witness by calculating $\alpha = (\alpha_z - \alpha'_z)/(c - c'), \beta = (\beta_z - \beta'_z)/(c - c')$.
- **Honest Verifier Zero-Knowledge**: The simulator $\mathsf{Sim}(u)$ can generate the message in a reverse manner: (1) Randomly choose $\alpha_z, \beta_z \leftarrow_{\$} \mathbb{Z}_p$; (2) Randomly choose $c \leftarrow_{\$} \mathbb{Z}_p$; (3) Computes $u_t = g^{\alpha_z} \cdot h^{\beta_z}/u^c$. The above-simulated transcript can also pass the verification.

## 2.3 Combination of Sigma Protocols

Sometimes, we need to prove complicated relations for specific scenarios that consist of a series of basic relations. The combination of relations can be divided into two categories. (1) AND composition requires the prover to prove that it knows the witnesses for all involved statements. While in the (2) OR composition, the prover only needs to prove that it knows a certain witness for one of the relations.

### 2.3.1 AND Composition

Assume that there is a discrete log relation $\mathcal{R}_{DL} = \{(\alpha, u) \in \mathbb{Z}_p \times \mathbb{G}_p : u = g^{\alpha}\}$. Then the combination of two discrete log relation can be described by $\mathcal{R}_{DL}^1 \wedge \mathcal{R}_{DL}^2 = \{(\alpha_1, \alpha_2; u_1, u_2) \in \mathbb{Z}_p^2 \times \mathbb{G}_p^2 : u_1 = g^{\alpha_1} \wedge u_2 = g^{\alpha_2}\}$. Here, we give a definition of AND composition for binary relations.

**Definition 2.5 (AND Composition).** *Given a set of binary relations $\{\mathcal{R}_i \subseteq \mathcal{X}_i \times \mathcal{Y}_i\}_{i\in[n]}$. Then a new relation $\mathcal{R}_{AND}$ can be described as follows:*

$$\mathcal{R}_{AND} = \{((x_1, \ldots, x_n), (y_1, \ldots, y_n)) \in (\mathcal{X}_1 \times \cdots \times \mathcal{X}_n) \times (\mathcal{Y}_1 \times \cdots \times \mathcal{Y}_n) : (x_1, y_1) \in \mathcal{R}_1 \wedge \cdots \wedge (x_n, y_n) \in \mathcal{R}_n\}$$

A naive method is used to construct the Sigma protocol for the AND composition relation for each atomic relation. The following method summarizes the naive construction of Sigma protocol for $\mathcal{R}_{DL}^1 \wedge \mathcal{R}_{DL}^2 = \{(x_1, x_2; h_1, h_2) \in \mathbb{Z}_p^2 \times \mathbb{G}_p^2 : h_1 = g^{x_1} \wedge h_2 = g^{x_2}\}$.

- For $i \in \{1, 2\}$, $P$ and $V$ parallelly or sequentially execute the following protocol.
  - $P$ randomly chooses $u_i \leftarrow_\$ \mathbb{Z}_p$, computes $a_i = g^{u_i}$, and sends $a_i$ to $V$;
  - $V$ randomly chooses $c_i \leftarrow_\$ \mathbb{Z}_p$ and returns it to $P$;
  - $P$ computes $r_i = u_i + c_i \times x_i$, and returns $r_i$ to $P$;
  - $V$ checks whether $g^{r_i} = a_i \cdot h_i^{c_i}$. It outputs accept if it holds. Otherwise, it outputs reject.

The above method requires several challenges from $V$ for each relation. A more efficient construction, such as the following, only needs one challenge for the composite relation, illustrated in Figure 6.

- $P$ randomly chooses $u_1, u_2 \leftarrow_\$ \mathbb{Z}_p$, computes $a_1 = g^{u_1}, a_2 = g^{u_2}$, and sends $a_1, a_2$ to $V$;
- $V$ randomly chooses $c \leftarrow_\$ \mathbb{Z}_p$ and returns it to $P$;
- $P$ computes $r_1 = u_1 + c \times x_1, r_2 = u_2 + c \times x_2$, and returns $r_1, r_2$ to $P$;
- $V$ checks whether $g^{r_1} = a_1 \cdot h_1^c$ and $g^{r_2} = a_2 \cdot h_2^c$. It outputs accept if it holds. Otherwise, it outputs reject.

### 2.3.2 OR Composition

**Definition 2.6 (OR Composition).** *Given a set of binary relations $\{\mathcal{R}_i \subseteq \mathcal{X}_i \times \mathcal{Y}_i\}_{i\in[n]}$. Then a new relation $\mathcal{R}_{OR}$ can be described as follows:*

$$\mathcal{R}_{OR} = \{((x_1, \ldots, x_n), (y_1, \ldots, y_n)) \in (\mathcal{X}_1 \times \cdots \times \mathcal{X}_n) \times (\mathcal{Y}_1 \times \cdots \times \mathcal{Y}_n) : (x_1, y_1) \in \mathcal{R}_1 \vee \cdots \vee (x_n, y_n) \in \mathcal{R}_n\}$$

In the case where both $\mathcal{R}_{DL}^1$ and $\mathcal{R}_{DL}^2$ are DL relation, then the OR composition relation is $\mathcal{R}_{DL}^1 \vee \mathcal{R}_{DL}^2 = \{(x_1, x_2; h_1, h_2) \in \mathbb{Z}_p^2 \times \mathbb{G}_p^2 : h_1 = g^{x_1} \vee h_2 = g^{x_2}\}$.

The specific Sigma protocol is constructed to prove the above relation, illustrated in Figure 7.

$$R^1_{DL} \wedge R^2_{DL} = \{((x_1, x_2; h_1, h_2) \in \mathbb{Z}_p^2 \times \mathbb{G}_p^2 :\ h_1 = g^{x_1} \wedge h_2 = g^{x_2}$$

$$\begin{array}{lcr}
\underline{P(x_1, x_2; h_1, h_2)} & & \underline{V(h_1, h_2)} \\
u_1, u_2 \leftarrow_{\$} \mathbb{Z}_p & & \\
a_1 = g^{u_1}, a_2 = g^{u_2} & & \\
 & \xrightarrow{\qquad\qquad a_1, a_2 \qquad\qquad} & \\
 & & c \leftarrow_{\$} \mathbb{Z}_p \\
 & \xleftarrow{\qquad\qquad c \qquad\qquad} & \\
r_1 = u_1 + x_1 \times c & & \\
r_2 = u_2 + x_2 \times c & & \\
 & \xrightarrow{\qquad\qquad r_1, r_2 \qquad\qquad} & \\
 & & g^{r_1} = a_1 \cdot h_1^c\ ? \\
 & & g^{r_2} = a_2 \cdot h_2^c\ ?
\end{array}$$

Figure 6: Protocol for AND Composition (for DL Relations).

$$R^1_{DL} \vee R^2_{DL} = \{((x_1, x_2; h_1, h_2) \in \mathbb{Z}_p^2 \times \mathbb{G}_p^2 :\ h_1 = g^{x_1} \vee h_2 = g^{x_2}$$

$$\begin{array}{lcr}
\underline{P(x_2 = \mathsf{Log}_g h_2; h_1, h_2)} & & \underline{V(h_1, h_2)} \\
c_1, r_1, u_2 \leftarrow_{\$} \mathbb{Z}_p & & \\
a_1 = g^{r_1} \cdot h_1^{-c_1}, a_2 = g^{u_2} & & \\
 & \xrightarrow{\qquad\qquad a_1, a_2 \qquad\qquad} & \\
 & & c \leftarrow_{\$} \mathbb{Z}_p \\
 & \xleftarrow{\qquad\qquad c \qquad\qquad} & \\
c_2 = c - c_1 & & \\
r_2 = u_2 + x_2 \times c_2 & & \\
 & \xrightarrow{\qquad\qquad c_1, c_2, r_1, r_2 \qquad\qquad} & \\
 & & c = c_1 + c_2\ ? \\
 & & g^{r_1} = a_1 \cdot h_1^{c_1}\ ? \\
 & & g^{r_2} = a_2 \cdot h_2^{c_2}\ ?
\end{array}$$

Figure 7: Protocol for OR Composition (for DL Relations).

- $P$ randomly chooses $c_1, r_1, u_2 \leftarrow_{\$} \mathbb{Z}_p$, computes $a_1 = g^{r_1} \cdot h_1^{-c_1}, a_2 = g^{u_2}$, and sends $a_1, a_2$ to $V$;
- $V$ randomly chooses $c \leftarrow_{\$} \mathbb{Z}_p$ and returns it to $P$;
- $P$ computes $c_2 = c - c_1, r_2 = u_2 + c_2 \times x_2$, and sends $c_1, c_2, r_1, r_2$ to $V$;
- $V$ checks whether $c_1 + c_2 = c, g^{r_1} = a_1 \cdot h_1^{c_1}, g^{r_2} = a_2 \cdot h_2^{c_2}$. If all equations hold, $V$ outputs accept. Otherwise, it outputs reject.

The trick for constructing the Sigma protocol of OR composition contains two points. Firstly, for statements without knowing witnesses, the prover acts like a simulator to simulate the transcripts for corresponding sub-relations. Secondly, $P$ generates a real transcript for the statement with the known witness.

For AND-OR composition of relations such as $(\mathcal{R}_1 \vee \mathcal{R}_2) \wedge (\mathcal{R}_3 \vee \mathcal{R}_4)$ and $(\mathcal{R}_1 \wedge \mathcal{R}_2) \vee (\mathcal{R}_3 \wedge \mathcal{R}_4)$, the Sigma protocol design can be seen as a recursive progress. We could regard the composition relation as the AND or OR composition of two sub-relations and design an abstract protocol for these two sub-relations. Then, we could divide the sub-relations into smaller ones and design Sigma protocols. Combining these allows us to obtain the final protocol to prove the complex composition relation.

#### 2.3.3 Electronic Voting

In the scenario of e-voting, each voter $v_i \in V$ provides an ElGamal cipher text $c_i = \mathcal{E}.enc(x_i, pk), x_i \in \{0, 1\}$ to hide the voting information. The final result can be obtained by computing $x_{sum} = \mathcal{E}.dec(\prod_{i \in [|V|]} c_i, sk)$. To prevent the malicious voter that encrypts messages not in $\{0, 1\}$, the voter also needs to provide proof to show that the cipher text is constructed correctly.

Let $pk = u$ and the cipher text of $v_i$ is $v = g^{\beta}, e = u^{\beta} \cdot g^{b}$. The relation for the statement that $(v, e)$ is a cipher text of 0 or 1 can be as the following:

$$\mathcal{R}_V = \{((b, \beta), (u, v, e)) : v = g^{\beta} \wedge e = u^{\beta} \cdot g^{b} \wedge b \in \{0, 1\}\}.$$

The above relation can be described differently: $(g, u, v, e)$ or $(g, u, v, e/g)$ is a DDH tuple. The above relation can be transformed into the OR composition relation of two DDH relations:

$$\mathcal{R}^1_{DDH} \vee \mathcal{R}^2_{DDH} = \{(\beta; u, v, e) \in \mathbb{Z}_p \times \mathbb{G}_p^3 : v = g^{\beta} \wedge (e = v^{\beta} \vee e/g = v^{\beta})\}.$$

Constructing a Sigma protocol for the above OR composition relation can ensure the cipher text is correctly encrypted. The protocol is illustrated in Figure 8.

## 3 Zero-Knowledge Proofs

### 3.1 Zero-Knowledge Proof System

Zero-knowledge proof is an extension of Sigma protocols which should be held for any verifier. The interactive of zero-knowledge proof is not necessary of 3-pass and soundness is not necessary of proof-of-knowledge. For language $L$, zero-knowledge proof systems have the following three properties:

- Correctness(Completeness): If $y \in L$, both $P$ and $V$ execute the protocol honestly, $V$ must output accept at the end of the interaction.

$$R^1_{DDH} \vee R^2_{DDH} = \{((\beta; u, v, e) \in \mathbb{Z}_p \times \mathbb{G}_p^3 : v = g^\beta \wedge \left(e = u^\beta \vee e/g = u^\beta\right)$$

$\underline{P\left(\beta; u, v, e = u^\beta \cdot g\right)}$ $\underline{V(u, v, e)}$

$c_1, r_1, r_2 \leftarrow_\$ \mathbb{Z}_p, a_1 = g^{r_1}$

$z_1 = c_1 \times \beta + r_1, a_2 = u^{z_1}/e^{c_1}$

$a_3 = g^{r_2}, a_4 = u^{r_2}$

$$\xrightarrow{a_1, a_2, a_3, a_4}$$

$c \leftarrow_\$ \mathbb{Z}_p$

$$\xleftarrow{c}$$

$c_2 = c - c_1$

$z_2 = c_2 \times \beta + r_2$

$$\xrightarrow{c_1, c_2, z_1, z_2}$$

$c = c_1 + c_2 \ ? \quad g^{z_1} = a_1 \cdot v^{c_1} \ ?$

$u^{z_1} = a_2 \cdot e^{c_1} \ ? \quad g^{z_2} = a_3 \cdot v^{c_2} \ ?$

$u^{z_2} = a_4 \cdot (e/g)^{c_2} \ ?$

Figure 8: Protocol for e-voting.

- Soundness: If $y \notin L$, for any computational-bounded prover $P$, $V$ outputs accept with negligible probability.
- Zero-knowledge: For any $V$, without knowing the witness $x$, we can simulate(generate) the valid transaction efficiently for any statement $y \in L$.

Given an NP language $L$, a prover $P$ with input $(x, y) \in relationR$ wants to prove $y \in L$. The above three properties of zero-knowledge proof imply the following propositions for NP language:

- if $y \in L$, verifier $V$ will output accept with overwhelming probability.
- if $y \notin L$, for any probabilistic polynomial-time(PPT) prover, verifier $V$ will output accept with negligible probability, which means reject it.
- Zero-knowledge: Any verifier $V$ could learn nothing about the witness $x \in \mathcal{X}$ during the interaction.

According to O Goldreich, S Micali and A Wigderson [GMW86], Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. we have the following theorem:

**Theorem 3.1.** *If there exists a secure probabilistic encryption, then every language in NP has a zero-knowledge interactive proof system in which the prover is a probabilistic polynomial-time machine that gets an NP proof as an auxiliary input.*

## 3.2 ZKP for 3-colorable Graphs

To prove that $\exists$ input $x$ such that $C(x) = y$, where $C$ is any polynomial size circuit. The circuit can be a polynomial function or an algorithm. There is a famous example of ZKP for 3-colorable Graphs to help understanding:

Let $G = (V, E)$ be graphs on n vertices and define $V = \{v_1, \ldots, v_n\}$ be the set of vertices, and $E = \{e_{i,j} : \exists edgee_{i,j}betweenv_i, v_j\}$ be the set of edges. We say that a graph G is 3-colorable(or $G \in 3COL$) if there is a function $c : V \to \{R, G, B\}$ such that for every edge $(v_i, v_j) \in E, c(v_i) \neq c(v_j)$.

The protocol for 3COL actually implies a protocol for all languages in NP, since 3COL is NP-complete, which can be used for the prover to convert their proof for any NP into a proof for the 3COL protocol. Therefore, we could design a commitment to generate a proof of 3COL problem.

We should define the commitment first. A commitment Com is a 3-tuple algorithm(Setup, Commit, Verify):

- **Setup:** Generate public parameters pp.
- **Commit(m):** Compute a commitment $c$ to $m$ with its opening $d$, and output $c$.
- **Verify(c,m,d):** indicate the validation of $(m, d)$ with respect to commitment $c$

A commitment could be statistical hiding and computational binding, or computational hiding and statistical hiding.

- **Hiding:** For any $m, m' \in \mathcal{M}_{com}$, their commitments are statistical indistinguisgable.
- **Binding:** No probability polynomial time(PPT) adversary could open a commitment $c$ on two different messages.

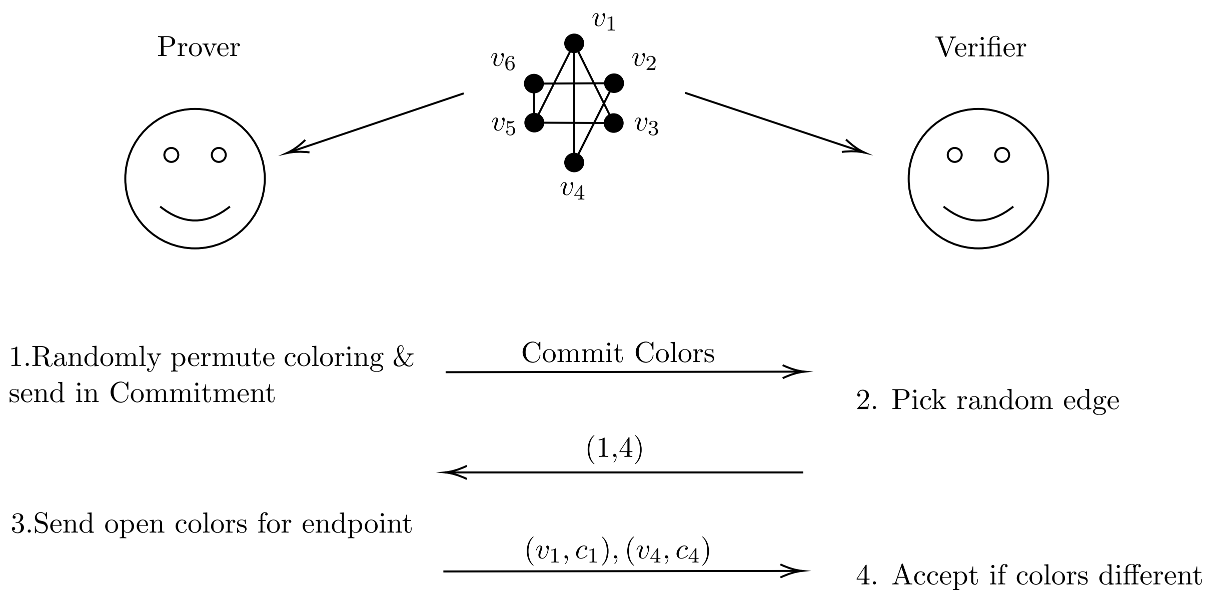
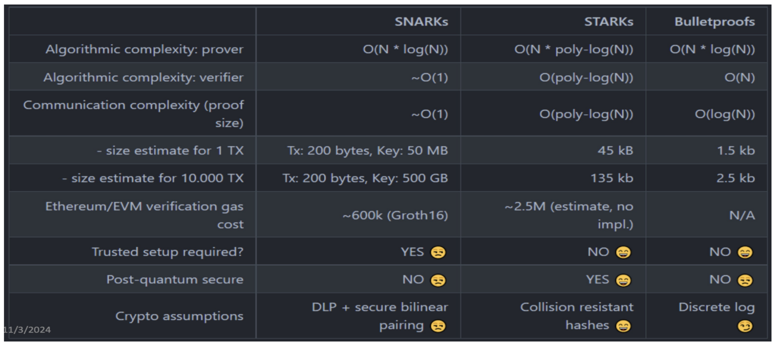

Prover
Verifier
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
1.Randomly permute coloring & send in Commitment
Commit Colors
2. Pick random edge
(1,4)
3.Send open colors for endpoint
$(v_1, c_1), (v_4, c_4)$
4. Accept if colors different

Figure 9: The ZKP commitment for 3-colorable Graphs.



| | SNARKs | STARKs | Bulletproofs |
|---|---|---|---|
| Algorithmic complexity: prover | O(N * log(N)) | O(N * poly-log(N)) | O(N * log(N)) |
| Algorithmic complexity: verifier | ~O(1) | O(poly-log(N)) | O(N) |
| Communication complexity (proof size) | ~O(1) | O(poly-log(N)) | O(log(N)) |
| - size estimate for 1 TX | Tx: 200 bytes, Key: 50 MB | 45 kB | 1.5 kb |
| - size estimate for 10.000 TX | Tx: 200 bytes, Key: 500 GB | 135 kb | 2.5 kb |
| Ethereum/EVM verification gas cost | ~600k (Groth16) | ~2.5M (estimate, no impl.) | N/A |
| Trusted setup required? | YES 😒 | NO 😁 | NO 😁 |
| Post-quantum secure | NO 😒 | YES 😁 | NO 😒 |
| Crypto assumptions | DLP + secure bilinear pairing 😒 | Collision resistant hashes 😁 | Discrete log 😏 |

11/3/2024

Figure 10: The relevant comparison

The ZKP commitment for 3-colorable Graphs like Figure 9 It is easy to check the completeness and soundness, so we just discuss the zero-knowledge properties for the honest verifier. If the prover colors every point of the graph for the verifier to check for many times, the verifier can learn the knowledge about 3COL graph. Thus, the zero-knowledge properties are implied by the hiding of the commit. The prover only selects two points to color with different colors.

## 3.3 Non-Interactive Zero-Knowledge (NIZK)

Non-interactive is better than interactive(latency) and usable for signature or e-voting situations. Actually, NIZK only exists for L in BPP, which is not interesting than NP. And any zero-knowledge systems based on the Sigma protocol could be converted into a non-interactive proof system using the Fiat-Shamir transform. The basic idea is not relying on the verifier to randomly generate the challenges while taking advantage of a hash function $H$ as a random oracle. The following introduced Fiat-Shamir transform is a technique that can convert a Sigma protocol into a non-interactive proof protocol [FS86].

## 3.4 Succinct Non-Interactive Arguments of Knowledge (SNARKs)

It is better if we have a very small (Succinct) proof and the verification of the proof is efficient. SNARK is such a zero-knowledge technique that can be used in arbitrary logic and circuits, and yield succinct proofs. The SNARKs can be used in verifiable outsourcing computation and blockchain. Actually, there are lots kinds of SNARK like zk-STARK. The relevant comparison is as Figure 10, shows the differences of SNARKs(groth16), STARKs and Bulletproofs.

## References

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto*, volume 86, pages 186–194. Springer, 1986.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *27th Annual Symposium on Foundations of Computer Science (FCS 1986)*, pages 174–187. IEEE Computer Society, 1986.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[QQQ$^+$89] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. How to explain zero-knowledge protocols to your children. In *Conference on the Theory and Application of Cryptology*, pages 628–631. Springer, 1989.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*, pages 239–252. Springer, 1990.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4:161–174, 1991.