Introduction to Zero Knowledge Proofs

Yuncong Zhang

October 16, 2020

Outline

NP Language

NP Language

What is NP language?

NP Language

What is NP language?

- Relation $\mathcal{R} = \{(x, w)\}$ is a set of instance-witness pairs

NP Language

What is NP language?

- Relation $\mathcal{R} = \{(x, w)\}$ is a set of instance-witness pairs
- Language $\mathcal{L}_{\mathcal{R}} = \{x : \exists w \text{ s.t.} (x, w) \in \mathcal{R}\}$ is induced from relation $\mathcal{R}$

NP Language

What is NP language?

- Relation $\mathcal{R} = \{(x, w)\}$ is a set of instance-witness pairs
- Language $\mathcal{L}_{\mathcal{R}} = \{x : \exists w \text{ s.t.} (x, w) \in \mathcal{R}\}$ is induced from relation $\mathcal{R}$
- $\mathcal{L}_{\mathcal{R}}$ is in NP iff there is a deterministic polynomial time algorithm f such that $f(x, w) = 1 \Leftrightarrow (x, w) \in \mathcal{R}$ (we say f decides $\mathcal{R}$)

Proof Systems

Proof Systems

A proof system is an interactive protocol where the prover $\mathcal{P}$ tries to convince the verifier $\mathcal{V}$ a statement x is true

Proof Systems

A proof system is an interactive protocol where the prover $\mathcal{P}$ tries to convince the verifier $\mathcal{V}$ a statement x is true

- **Completeness:** if statement x is true, then the verifier $\mathcal{V}$ accepts (outputs 1) with probability at least $1 - \eta$

Proof Systems

A proof system is an interactive protocol where the prover $\mathcal{P}$ tries to convince the verifier $\mathcal{V}$ a statement x is true

- **Completeness:** if statement x is true, then the verifier $\mathcal{V}$ accepts (outputs 1) with probability at least $1 - \eta$
- **Soundness:** if statement x is false, then for any prover $\mathcal{P}$, the verifier $\mathcal{V}$ accepts with probability less than ε (called soundness error)

Proof Systems

A proof system is an interactive protocol where the prover $\mathcal{P}$ tries to convince the verifier $\mathcal{V}$ a statement x is true

- **Completeness:** if statement x is true, then the verifier $\mathcal{V}$ accepts (outputs 1) with probability at least $1 - \eta$
- **Soundness:** if statement x is false, then for any prover $\mathcal{P}$, the verifier $\mathcal{V}$ accepts with probability less than ε (called soundness error)

Denote an execution of this protocol by $\langle \mathcal{P}, \mathcal{V} \rangle(x)$, and $\mathrm{tr}\langle \mathcal{P}, \mathcal{V} \rangle(x)$ is the transcript of the execution, which is the collection of all interaction messages

Proof Systems

Examples

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:
  1. The prover sends w to the verifier

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:
  1. The prover sends w to the verifier
  2. The verifier checks $f(x, w) = 1$

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:
  1. The prover sends w to the verifier
  2. The verifier checks $f(x, w) = 1$
  3. Perfect completeness and perfect soundness

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:
    1. The prover sends w to the verifier
    2. The verifier checks $f(x, w) = 1$
    3. Perfect completeness and perfect soundness
- Given an ECDSA public key pk, the prover proves that it learns the secret key sk

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:
  1. The prover sends w to the verifier
  2. The verifier checks $f(x, w) = 1$
  3. Perfect completeness and perfect soundness
- Given an ECDSA public key pk, the prover proves that it learns the secret key sk
  1. The verifier samples a message m and sends to prover

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:
  1. The prover sends w to the verifier
  2. The verifier checks $f(x, w) = 1$
  3. Perfect completeness and perfect soundness
- Given an ECDSA public key pk, the prover proves that it learns the secret key sk
  1. The verifier samples a message m and sends to prover
  2. The prover generates a signature σ and sends to verifier

Proof Systems

Examples

- For every NP language $\mathcal{L}$, the statement $x \in \mathcal{L}$ has a trivial proof system:
  1. The prover sends w to the verifier
  2. The verifier checks $f(x, w) = 1$
  3. Perfect completeness and perfect soundness
- Given an ECDSA public key pk, the prover proves that it learns the secret key sk
  1. The verifier samples a message m and sends to prover
  2. The prover generates a signature σ and sends to verifier
  3. The verifier checks $\mathrm{ECDSAVerify}(\sigma, \mathrm{pk}, \mathrm{m}) = 1$

Soundness

Soundness has several variations

Soundness

Soundness has several variations

- **Computational**: holds only for bounded adversary

Soundness

Soundness has several variations

- **Computational**: holds only for bounded adversary
- **Knowledge**: requires that the prover knows the witness

Soundness

Soundness has several variations

- **Computational**: holds only for bounded adversary
- **Knowledge**: requires that the prover knows the witness

Proof systems with different soundness have different names

Soundness

Soundness has several variations

- **Computational**: holds only for bounded adversary
- **Knowledge**: requires that the prover knows the witness

Proof systems with different soundness have different names

	Standard	**Knowledge**
Statistical	Proof	Proof of Knowledge
Computational	Argument	Argument of Knowledge

Soundness

Soundness has several variations

- **Computational**: holds only for bounded adversary
- **Knowledge**: requires that the prover knows the witness

Proof systems with different soundness have different names

	Standard	**Knowledge**
Statistical	Proof	Proof of Knowledge
Computational	Argument	Argument of Knowledge

Remark

Soundness is a property of the verifier.

Soundness

As cryptographer, when we construct a proof system and say it has knowledge soundness, we must prove it.

Soundness

As cryptographer, when we construct a proof system and say it has knowledge soundness, we must prove it.

- But how to prove that an adversary $\mathcal{A}$ knows something?

Soundness

As cryptographer, when we construct a proof system and say it has knowledge soundness, we must prove it.

- But how to prove that an adversary $\mathcal{A}$ knows something?
- Generally, we cannot prove this using game-based strategy, that means solving a hard problem when $\mathcal{A}$ doesn't know

Soundness

As cryptographer, when we construct a proof system and say it has knowledge soundness, we must prove it.

- But how to prove that an adversary $\mathcal{A}$ knows something?
- Generally, we cannot prove this using game-based strategy, that means solving a hard problem when $\mathcal{A}$ doesn't know

Instead, we use Extractor to formalize the notion of knowing

Soundness

As cryptographer, when we construct a proof system and say it has knowledge soundness, we must prove it.

- But how to prove that an adversary $\mathcal{A}$ knows something?
- Generally, we cannot prove this using game-based strategy, that means solving a hard problem when $\mathcal{A}$ doesn't know

Instead, we use Extractor to formalize the notion of knowing

- Assume $\mathcal{A}$ and $\mathcal{E}$ are two algorithms, let $\mathcal{A}||\mathcal{E}$ denote the algorithm where $\mathcal{A}$ and $\mathcal{E}$ execute simultaneously and $\mathcal{E}$ has white-box access to the internal state of $\mathcal{A}$

Soundness

As cryptographer, when we construct a proof system and say it has knowledge soundness, we must prove it.

- But how to prove that an adversary $\mathcal{A}$ knows something?
- Generally, we cannot prove this using game-based strategy, that means solving a hard problem when $\mathcal{A}$ doesn't know

Instead, we use Extractor to formalize the notion of knowing

- Assume $\mathcal{A}$ and $\mathcal{E}$ are two algorithms, let $\mathcal{A}\|\mathcal{E}$ denote the algorithm where $\mathcal{A}$ and $\mathcal{E}$ execute simultaneously and $\mathcal{E}$ has white-box access to the internal state of $\mathcal{A}$
- Denote by $\langle \mathcal{A}\|\mathcal{E}, \mathcal{V}\rangle(x) \to (w, b)$ the protocol where $\mathcal{A}$ interacts with $\mathcal{V}$ and in the meantime $\mathcal{E}$ has access to the internal state of $\mathcal{A}$. At the end, $\mathcal{E}$ outputs w and the verifier outputs b

Soundness

As cryptographer, when we construct a proof system and say it has knowledge soundness, we must prove it.

- But how to prove that an adversary $\mathcal{A}$ knows something?
- Generally, we cannot prove this using game-based strategy, that means solving a hard problem when $\mathcal{A}$ doesn't know

Instead, we use Extractor to formalize the notion of knowing

- Assume $\mathcal{A}$ and $\mathcal{E}$ are two algorithms, let $\mathcal{A}\|\mathcal{E}$ denote the algorithm where $\mathcal{A}$ and $\mathcal{E}$ execute simultaneously and $\mathcal{E}$ has white-box access to the internal state of $\mathcal{A}$
- Denote by $\langle \mathcal{A}\|\mathcal{E}, \mathcal{V}\rangle(x) \to (w, b)$ the protocol where $\mathcal{A}$ interacts with $\mathcal{V}$ and in the meantime $\mathcal{E}$ has access to the internal state of $\mathcal{A}$. At the end, $\mathcal{E}$ outputs w and the verifier outputs b

Knowledge Soundness

For every adversary $\mathcal{A}$, there exists an extractor $\mathcal{E}_{\mathcal{A}}$, such that $\Pr[\langle \mathcal{A}\|\mathcal{E}_{\mathcal{A}}, \mathcal{V}\rangle(x) \to (w, b) : b = 1 \wedge (x, w) \notin \mathcal{R}] < \varepsilon$

Soundness

Another way to define the extractor

Soundness

Another way to define the extractor

- Denote by $\mathcal{E}^{\langle \mathcal{A}, \mathcal{V} \rangle (x)}$ an algorithm which has black-box access to the protocol $\langle \mathcal{A}, \mathcal{V} \rangle (x)$, which means:
  - $\mathcal{E}$ can read all the messages during interaction
  - $\mathcal{E}$ can rewind the protocol back to any point during the execution, and reexecute the protocol from that point

Soundness

Another way to define the extractor

- Denote by $\mathcal{E}^{\langle \mathcal{A}, \mathcal{V} \rangle(x)}$ an algorithm which has black-box access to the protocol $\langle \mathcal{A}, \mathcal{V} \rangle(x)$, which means:
  - $\mathcal{E}$ can read all the messages during interaction
  - $\mathcal{E}$ can rewind the protocol back to any point during the execution, and reexecute the protocol from that point

Knowledge Soundness (Witness-extended emulation)

For every adversary $\mathcal{A}$, there exists an extractor $\mathcal{E}_{\mathcal{A}}$, such that
$\Pr[\mathcal{E}_{\mathcal{A}}^{\langle \mathcal{A}, \mathcal{V} \rangle(x)} \rightarrow w : \langle \mathcal{A}, \mathcal{V} \rangle(x) \rightarrow 1 \wedge (x, w) \notin \mathcal{R}] < \varepsilon$

Zero Knowledge

Zero Knowledge

Zero-Knowledge Proofs are proof systems that also have

Zero Knowledge

Zero-Knowledge Proofs are proof systems that also have

- **Zero-Knowledgeness:** if statement x is true, then the verifier cannot get any information from its view (except the correctness of x)

Zero Knowledge

Zero-Knowledge Proofs are proof systems that also have

- **Zero-Knowledgeness:** if statement x is true, then the verifier cannot get any information from its view (except the correctness of x)
- The view of the verifier consists of: randomness r and the transcript $\mathrm{tr}\langle\mathcal{P},\mathcal{V}\rangle(x)$

Zero Knowledge

Zero-Knowledge Proofs are proof systems that also have

- **Zero-Knowledgeness:** if statement x is true, then the verifier cannot get any information from its view (except the correctness of x)
- The view of the verifier consists of: randomness r and the transcript $\mathrm{tr}\langle\mathcal{P},\mathcal{V}\rangle(x)$

Formally: for any verifier $\mathcal{V}$, there exists a simulator $\mathcal{S}$, which on input a valid statement x, can sample the verifier view, i.e. the distribution of $\mathcal{S}(x)$ is indifferentiable from that of $(r, \mathrm{tr}\langle\mathcal{P},\mathcal{V}\rangle(x))$

Zero Knowledge

ZK has several variations

Zero Knowledge

ZK has several variations

- **Statistical**: statistical difference $\mathrm{SD}(\mathcal{S}(x), (r, \mathrm{tr}\langle \mathcal{P}, \mathcal{V} \rangle(x)))$ is negligible (zero for perfect ZK)

Zero Knowledge

ZK has several variations

- **Statistical**: statistical difference $\mathrm{SD}(\mathcal{S}(x), (r, \mathrm{tr}\langle\mathcal{P}, \mathcal{V}\rangle(x)))$ is negligible (zero for perfect ZK)
- **Computational**: for any P.P.T. differentiator $\mathcal{D}$, $|\Pr[D(\mathcal{S}(x)) = 1] - \Pr[D((r, \mathrm{tr}\langle\mathcal{P}, \mathcal{V}\rangle(x))) = 1]|$ is negligible

Zero Knowledge

ZK has several variations

- **Statistical**: statistical difference $\mathrm{SD}(\mathcal{S}(x), (r, \mathrm{tr}\langle\mathcal{P}, \mathcal{V}\rangle(x)))$ is negligible (zero for perfect ZK)
- **Computational**: for any P.P.T. differentiator $\mathcal{D}$, $|\Pr[D(\mathcal{S}(x)) = 1] - \Pr[D((r, \mathrm{tr}\langle\mathcal{P}, \mathcal{V}\rangle(x))) = 1]|$ is negligible
- **Honest Verifier**: assumes that the verifier follows the protocol (but may be curious, i.e. try to learn some information from the view)

Zero Knowledge

ZK has several variations

- **Statistical**: statistical difference $\mathsf{SD}(\mathcal{S}(x), (r, \mathsf{tr}\langle\mathcal{P}, \mathcal{V}\rangle(x)))$ is negligible (zero for perfect ZK)
- **Computational**: for any P.P.T. differentiator $\mathcal{D}$, $|\Pr[D(\mathcal{S}(x)) = 1] - \Pr[D((r, \mathsf{tr}\langle\mathcal{P}, \mathcal{V}\rangle(x))) = 1]|$ is negligible
- **Honest Verifier**: assumes that the verifier follows the protocol (but may be curious, i.e. try to learn some information from the view)

Remark

ZK is a property of the prover.

Example: Color Balls

Prove to a blindfold verifier that two balls have different colors without revealing the colors

Example: Color Balls

Prove to a blindfold verifier that two balls have different colors without revealing the colors

1. The verifier takes each ball in one hand, and shows the prover

Example: Color Balls

Prove to a blindfold verifier that two balls have different colors without revealing the colors

1. The verifier takes each ball in one hand, and shows the prover
2. The verifier puts the hands behind its back, samples a bit $b \in \{0, 1\}$ in its mind. If $b = 1$, the verifier switches the balls, otherwise it does nothing

Example: Color Balls

Prove to a blindfold verifier that two balls have different colors without revealing the colors

1. The verifier takes each ball in one hand, and shows the prover
2. The verifier puts the hands behind its back, samples a bit $b \in \{0, 1\}$ in its mind. If $b = 1$, the verifier switches the balls, otherwise it does nothing
3. The verifier shows the balls to the prover, and the prover guesses b'

Example: Color Balls

Prove to a blindfold verifier that two balls have different colors without revealing the colors

1. The verifier takes each ball in one hand, and shows the prover
2. The verifier puts the hands behind its back, samples a bit $b \in \{0, 1\}$ in its mind. If $b = 1$, the verifier switches the balls, otherwise it does nothing
3. The verifier shows the balls to the prover, and the prover guesses b'
4. the verifier accepts iff $b' = b$

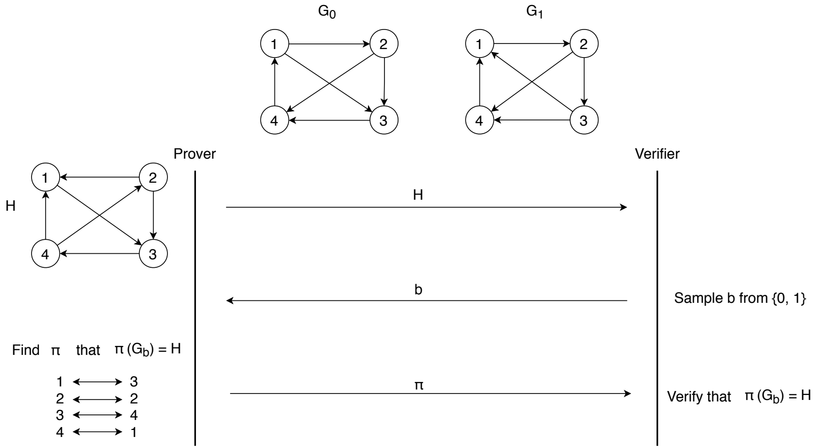
Example: Color Balls

Prove to a blindfold verifier that two balls have different colors without revealing the colors

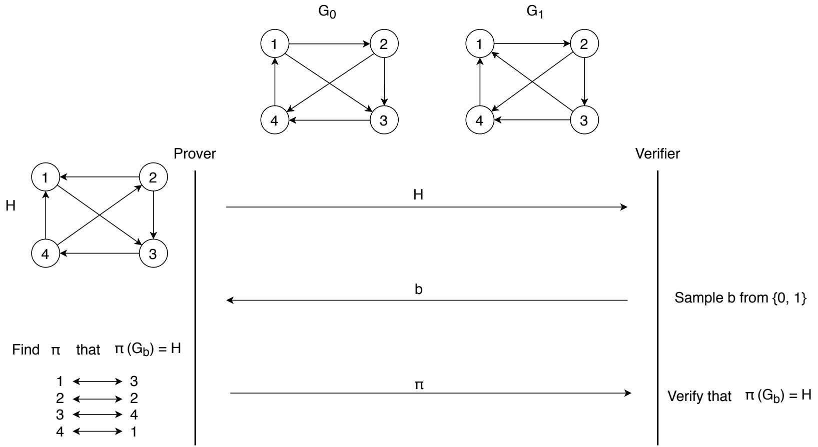
1. The verifier takes each ball in one hand, and shows the prover
2. The verifier puts the hands behind its back, samples a bit $b \in \{0, 1\}$ in its mind. If $b = 1$, the verifier switches the balls, otherwise it does nothing
3. The verifier shows the balls to the prover, and the prover guesses b'
4. the verifier accepts iff $b' = b$

Zero-knowledge: for any verifier $\mathcal{V}$, the simulator $\mathcal{S}$ does whatever $\mathcal{V}$ does, and in the last step directly sets $b' = b$

Example: Graph Isomorphism



Example: Graph Isomorphism



Zero-Knowledge: $\mathcal{S}$ samples the view (H, b, π) as follows:

1. Uniformly sample permutation π and bit b
2. Compute $H = \pi(G_b)$, output (H, b, π)

Example: Graph Isomorphism

Knowledge Soundness: construct the following extractor $\mathcal{E}$, which has black-box control of the protocol execution, and can read the transcript

Example: Graph Isomorphism

Knowledge Soundness: construct the following extractor $\mathcal{E}$, which has black-box control of the protocol execution, and can read the transcript

1. $\mathcal{E}$ observes a full execution and records the transcript (H, b, π_b)

Example: Graph Isomorphism

Knowledge Soundness: construct the following extractor $\mathcal{E}$, which has black-box control of the protocol execution, and can read the transcript

1. $\mathcal{E}$ observes a full execution and records the transcript (H, b, π_b)
2. $\mathcal{E}$ keeps rewinding the execution back to the point exactly before the verifier samples b, until it observes the verifier sampling $b' \neq b$

Example: Graph Isomorphism

Knowledge Soundness: construct the following extractor $\mathcal{E}$, which has black-box control of the protocol execution, and can read the transcript

1. $\mathcal{E}$ observes a full execution and records the transcript (H, b, π_b)
2. $\mathcal{E}$ keeps rewinding the execution back to the point exactly before the verifier samples b, until it observes the verifier sampling $b' \neq b$
3. $\mathcal{E}$ lets the execution proceed and obtains $(b', \pi_{b'})$

Example: Graph Isomorphism

Knowledge Soundness: construct the following extractor $\mathcal{E}$, which has black-box control of the protocol execution, and can read the transcript

1. $\mathcal{E}$ observes a full execution and records the transcript (H, b, π_b)
2. $\mathcal{E}$ keeps rewinding the execution back to the point exactly before the verifier samples b, until it observes the verifier sampling $b' \neq b$
3. $\mathcal{E}$ lets the execution proceed and obtains $(b', \pi_{b'})$
4. $\mathcal{E}$ outputs $\pi = \pi_1^{-1} \circ \pi_0$

NIZK

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

- $\pi \leftarrow \mathcal{P}(x, w)$

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

- $\pi \leftarrow \mathcal{P}(x, w)$
- $0/1 \leftarrow \mathcal{V}(x, \pi)$

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

- $\pi \leftarrow \mathcal{P}(x, w)$
- $0/1 \leftarrow \mathcal{V}(x, \pi)$

However, the above scenario implies

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

- $\pi \leftarrow \mathcal{P}(x, w)$
- $0/1 \leftarrow \mathcal{V}(x, \pi)$

However, the above scenario implies

- The view of the verifier is just π

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

- $\pi \leftarrow \mathcal{P}(x, w)$
- $0/1 \leftarrow \mathcal{V}(x, \pi)$

However, the above scenario implies

- The view of the verifier is just π
- If x is true: zero-knowledgeness implies a simulator $\mathcal{S}(x) \to \pi$ indifferentiable from valid π; in particular, the simulated π should be accepted with probability $1 - \eta$

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

- $\pi \leftarrow \mathcal{P}(x, w)$
- $0/1 \leftarrow \mathcal{V}(x, \pi)$

However, the above scenario implies

- The view of the verifier is just π
- If x is true: zero-knowledgeness implies a simulator $\mathcal{S}(x) \to \pi$ indifferentiable from valid π; in particular, the simulated π should be accepted with probability $1 - \eta$
- If x is false: soundness implies that any π should be accepted with probability less than ε

Non-Interactive Zero Knowledge

Non-interactive proof system consists of a single proof π from prover to verifier

You may imagine NIZK works as follows

- $\pi \leftarrow \mathcal{P}(x, w)$
- $0/1 \leftarrow \mathcal{V}(x, \pi)$

However, the above scenario implies

- The view of the verifier is just π
- If x is true: zero-knowledgeness implies a simulator $\mathcal{S}(x) \to \pi$ indifferentiable from valid π; in particular, the simulated π should be accepted with probability $1 - \eta$
- If x is false: soundness implies that any π should be accepted with probability less than ε
- Therefore, NIZK implies a P.P.T. algorithm that decides $\mathcal{L}$: $\mathcal{V}(x, \mathcal{S}(x))$

Non-Interactive Zero Knowledge

To obtain NIZK for general NP languages, we have to introduce non-standard models

Non-Interactive Zero Knowledge

To obtain NIZK for general NP languages, we have to introduce non-standard models

- Random Oracle (RO) model: both the prover and the verifier have access to a random oracle

Non-Interactive Zero Knowledge

To obtain NIZK for general NP languages, we have to introduce non-standard models

- Random Oracle (RO) model: both the prover and the verifier have access to a random oracle
- Common Reference String (CRS) model: the prover and the verifier have access to a common string

Q/A