Lecture 23: Zero-Knowledge Proofs

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

July 29, 2021

Overview

- Why Zero Knowledge?
- Zero-Knowledge Proofs
- Conclusion
- Acknowledgements

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.
- Or we may not trust the channel of communication
    - someone may *eavesdrop* our messages
    - messages could be *corrupted*
    - someone may try to *impersonate* us
    - it's a wild world out there

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.
- Or we may not trust the channel of communication
  - someone may *eavesdrop* our messages
  - messages could be *corrupted*
  - someone may try to *impersonate* us
  - it's a wild world out there
- Situation
  - Alice has all her files encrypted (in public database)
  - Bob requests from her the contents of one of her files

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.
- Or we may not trust the channel of communication
  - someone may *eavesdrop* our messages
  - messages could be *corrupted*
  - someone may try to *impersonate* us
  - it's a wild world out there
- Situation
  - Alice has all her files encrypted (in public database)
  - Bob requests from her the contents of one of her files
  - She could simply send the decrypted file to Bob

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.
- Or we may not trust the channel of communication
    - someone may *eavesdrop* our messages
    - messages could be *corrupted*
    - someone may try to *impersonate* us
    - it's a wild world out there
- Situation
    - Alice has all her files encrypted (in public database)
    - Bob requests from her the contents of one of her files
    - She could simply send the decrypted file to Bob
    - Bob has no way of knowing that this message comes from Alice (or that this is indeed the right file)

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.
- Or we may not trust the channel of communication
  - someone may *eavesdrop* our messages
  - messages could be *corrupted*
  - someone may try to *impersonate* us
  - it's a wild world out there
- Situation
  - Alice has all her files encrypted (in public database)
  - Bob requests from her the contents of one of her files
  - She could simply send the decrypted file to Bob
  - Bob has no way of knowing that this message comes from Alice (or that this is indeed the right file)
  - Alice could *prove* to Bob this is the correct file by sending her encryption key

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.
- Or we may not trust the channel of communication
  - someone may *eavesdrop* our messages
  - messages could be *corrupted*
  - someone may try to *impersonate* us
  - it's a wild world out there
- Situation
  - Alice has all her files encrypted (in public database)
  - Bob requests from her the contents of one of her files
  - She could simply send the decrypted file to Bob
  - Bob has no way of knowing that this message comes from Alice (or that this is indeed the right file)
  - Alice could *prove* to Bob this is the correct file by sending her encryption key
  - But then Bob has access to her entire database!

Cyptography

- In cryptography, want to communicate with other people/entities whom we may not trust.
- Or we may not trust the channel of communication
  - someone may *eavesdrop* our messages
  - messages could be *corrupted*
  - someone may try to *impersonate* us
  - it's a wild world out there
- Situation
  - Alice has all her files encrypted (in public database)
  - Bob requests from her the contents of one of her files
  - She could simply send the decrypted file to Bob
  - Bob has no way of knowing that this message comes from Alice (or that this is indeed the right file)
  - Alice could *prove* to Bob this is the correct file by sending her encryption key
  - But then Bob has access to her entire database!
  - Can Alice convince Bob that she gave right file without giving any more *knowledge* beyond that she gave right file?

Zero-Knowledge Proofs

Proofs in which the verifier gains *no knowledge* beyond the validity of the assertion.

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?
- First question is quite complex, so let's only talk about the second and third

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?
- First question is quite complex, so let's only talk about the second and third
- Knowledge has to do with your *computational ability*
  - If you could have found the answer (i.e. computed it) without help, then you *gained no knowledge*

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?
- First question is quite complex, so let's only talk about the second and third
- Knowledge has to do with your *computational ability*
  - If you could have found the answer (i.e. computed it) without help, then you *gained no knowledge*
- Example:
  - Bob asks Alice whether a graph G is Eulerian

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?
- First question is quite complex, so let's only talk about the second and third
- Knowledge has to do with your *computational ability*
  - If you could have found the answer (i.e. computed it) without help, then you *gained no knowledge*
- Example:
  - Bob asks Alice whether a graph G is Eulerian
  - Bob gains no knowledge in this interaction, since he could have computed it by himself (By Euler's theorem: check that all vertices have even degree)

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?
- First question is quite complex, so let's only talk about the second and third
- Knowledge has to do with your *computational ability*
  - If you could have found the answer (i.e. computed it) without help, then you *gained no knowledge*
- Example:
  - Bob asks Alice whether a graph G is Eulerian
  - Bob gains no knowledge in this interaction, since he could have computed it by himself (By Euler's theorem: check that all vertices have even degree)
  - Bob asks Alice if graph G has Hamiltonian cycle

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?
- First question is quite complex, so let's only talk about the second and third
- Knowledge has to do with your *computational ability*
  - If you could have found the answer (i.e. computed it) without help, then you *gained no knowledge*
- Example:
  - Bob asks Alice whether a graph G is Eulerian
  - Bob gains no knowledge in this interaction, since he could have computed it by himself (By Euler's theorem: check that all vertices have even degree)
  - Bob asks Alice if graph G has Hamiltonian cycle
  - Bob now gains knowledge ($P \neq NP \Rightarrow$ Bob could not compute it)

Knowledge vs Information

- What do you mean by knowledge?
- What does it mean to "learn something/gain knowledge"?
- What is difference between knowledge and information?
- First question is quite complex, so let's only talk about the second and third
- Knowledge has to do with your *computational ability*
  - If you could have found the answer (i.e. computed it) without help, then you *gained no knowledge*
- Example:
  - Bob asks Alice whether a graph G is Eulerian
  - Bob gains no knowledge in this interaction, since he could have computed it by himself (By Euler's theorem: check that all vertices have even degree)
  - Bob asks Alice if graph G has Hamiltonian cycle
  - Bob now gains knowledge ($P \neq NP \Rightarrow$ Bob could not compute it)
- In both cases Alice conveyed *information*!

Knowledge vs Information

- *Knowledge*:
  - related to computational difficulty
  - about publicly known objects
    - One gains knowledge when one obtains something one *could not compute* before!

Knowledge vs Information

- *Knowledge*:
  - related to computational difficulty
  - about publicly known objects
    - One gains knowledge when one obtains something one *could not compute* before!
- *Information*:
  - unrelated to computational difficulty
  - about partially known objects
    - One gains information when one obtains something one *could not access* before!

Classical Proofs

- Our usual notion of proof:
  - A claim $\mathcal{C}$ is given

Classical Proofs

- Our usual notion of proof:
  - A claim $\mathcal{C}$ is given
  - A prover P writes down a proof that $\mathcal{C}$ is correct

Classical Proofs

- Our usual notion of proof:
  - A claim $\mathcal{C}$ is given
  - A prover P writes down a proof that $\mathcal{C}$ is correct
  - Prover P sends this proof to a verifier V

Classical Proofs

- Our usual notion of proof:
  - A claim $\mathcal{C}$ is given
  - A prover P writes down a proof that $\mathcal{C}$ is correct
  - Prover P sends this proof to a verifier V
  - Verifier has procedure (axioms and derivation rules) to check validity of proof

Classical Proofs

- Our usual notion of proof:
  - A claim $\mathcal{C}$ is given
  - A prover P writes down a proof that $\mathcal{C}$ is correct
  - Prover P sends this proof to a verifier V
  - Verifier has procedure (axioms and derivation rules) to check validity of proof
  - Verifier accepts or rejects based on these rules

Classical Proofs

- Our usual notion of proof:
  - A claim $\mathcal{C}$ is given
  - A prover P writes down a proof that $\mathcal{C}$ is correct
  - Prover P sends this proof to a verifier V
  - Verifier has procedure (axioms and derivation rules) to check validity of proof
  - Verifier accepts or rejects based on these rules
- One-way communication (or, in other words, very little interaction!)

$\mathcal{C}$

Prover $\xrightarrow{\pi}$ Verifier

$V(\mathcal{C}, \pi)$ → valid / not valid

Classical Proofs

- Our usual notion of proof:
  - A claim $\mathcal{C}$ is given
  - A prover P writes down a proof that $\mathcal{C}$ is correct
  - Prover P sends this proof to a verifier V
  - Verifier has procedure (axioms and derivation rules) to check validity of proof
  - Verifier accepts or rejects based on these rules
- One-way communication (or, in other words, very little interaction!)
- Verifier *does not trust* prover. Otherwise no need to verify proof!

Example: NP (Efficient Verifiable Proofs)

- Setup: set of Claims with efficiently verifiable proofs
    - A claim $\mathcal{C} := x \in L$ is given

Example: NP (Efficient Verifiable Proofs)

- Setup:
  - A claim $\mathcal{C} := x \in L$ is given
  - A prover P writes down a proof (witness) w that $x \in L$

Example: NP (Efficient Verifiable Proofs)

- Setup:
  - A claim $\mathcal{C} := x \in L$ is given
  - A prover P writes down a proof (witness) w that $x \in L$
  - Prover P sends w to a verifier V

Example: NP (Efficient Verifiable Proofs)

- Setup:
  - A claim $\mathcal{C} := x \in L$ is given
  - A prover P writes down a proof (witness) w that $x \in L$
  - Prover P sends w to a verifier V
  - Verifier has procedure (axioms and derivation rules) to check validity of proof (*deterministic, polynomial time algorithm*)

Example: NP (Efficient Verifiable Proofs)

- Setup:
  - A claim $\mathcal{C} := x \in L$ is given
  - A prover P writes down a proof (witness) w that $x \in L$
  - Prover P sends w to a verifier V
  - Verifier has procedure (axioms and derivation rules) to check validity of proof (*deterministic, polynomial time algorithm*)
  - Verifier accepts iff $V(x, w) = 1$

Example: NP (Efficient Verifiable Proofs)

- Setup:
  - A claim $\mathcal{C} := x \in L$ is given
  - A prover P writes down a proof (witness) w that $x \in L$
  - Prover P sends w to a verifier V
  - Verifier has procedure (axioms and derivation rules) to check validity of proof (*deterministic, polynomial time algorithm*)
  - Verifier accepts iff $V(x, w) = 1$
- In this setting, verifier *learns the proof*!

because the prover sent the whole proof to the verifier!

$P \neq NP$ then verifier alone could not have given right answer on its own

Example: Factoring

- Setup:
  - A claim $\mathcal{C}$:= $\boxed{N \text{ is a product of two primes}}$ is given

claim

Example: Factoring

- Setup:
  - A claim $\mathcal{C} := N$ is a product of two primes is given
  - A prover P writes down a proof: two primes $\boxed{p, q}$ that $N = p \cdot q$

proof

Example: Factoring

- Setup:
  - A claim $\mathcal{C} := N$ is a product of two primes is given
  - A prover P writes down a proof: two primes p, q that $N = p \cdot q$
  - Prover P sends (p, q) to a verifier V

Example: Factoring

- Setup:
  - A claim $\mathcal{C} := N$ is a product of two primes is given
  - A prover P writes down a proof: two primes p, q that $N = p \cdot q$
  - Prover P sends (p, q) to a verifier V
  - Verifier computes $p \cdot q$ and checks that p, q are prime.
    Checking validity of proof (*deterministic, polynomial time algorithm*)

Can multiply in P

Can check that number is prime in P

Example: Factoring

- Setup:
  - A claim $\mathcal{C} := N$ is a product of two primes is given
  - A prover P writes down a proof: two primes p, q that $N = p \cdot q$
  - Prover P sends (p, q) to a verifier V
  - Verifier computes $p \cdot q$ and checks that p, q are prime.
    Checking validity of proof (*deterministic, polynomial time algorithm*)
  - Verifier accepts iff p, q prime, and $N = pq$

Example: Factoring

- Setup:
  - A claim $\mathcal{C} := N$ is a product of two primes is given
  - A prover P writes down a proof: two primes p, q that $N = p \cdot q$
  - Prover P sends (p, q) to a verifier V
  - Verifier computes $p \cdot q$ and checks that p, q are prime.
    Checking validity of proof (*deterministic, polynomial time algorithm*)
  - Verifier accepts iff p, q prime, and $N = pq$
- In this setting, verifier *learns the proof* (in this case factorization)!

Example: Graph Isomorphism

$V = [n]$ $G_0(V, E_0)$

$G_1(V, E_1)$

- Setup:
  - A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic

G_0 and G_1 are isomorphic iff there is a permutation of the vertices $\rho \in S_n$

s.t. $\{i, j\} \in E_0 \iff \{\rho(i), \rho(j)\} \in E_1$

Example: Graph Isomorphism

- Setup:
  - A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P writes down an isomorphism ρ such that $\rho(G_0) = G_1$

Example: Graph Isomorphism

- Setup:
  - A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P writes down an isomorphism ρ such that $\rho(G_0) = G_1$
  - Prover P sends ρ to a verifier V

Example: Graph Isomorphism

- Setup:
  - A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P writes down an isomorphism ρ such that $\rho(G_0) = G_1$
  - Prover P sends ρ to a verifier V
  - Verifier checks that ρ is a permutation of vertices, and that $\rho(G_0) = G_1$ (*deterministic, polynomial time algorithm*)

Example: Graph Isomorphism

- Setup:
  - A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P writes down an isomorphism ρ such that $\rho(G_0) = G_1$
  - Prover P sends ρ to a verifier V
  - Verifier checks that ρ is a permutation of vertices, and that $\rho(G_0) = G_1$ (*deterministic, polynomial time algorithm*)
  - Verifier accepts iff the above is correct.

Example: Graph Isomorphism

- Setup:
  - A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P writes down an isomorphism ρ such that $\rho(G_0) = G_1$
  - Prover P sends ρ to a verifier V
  - Verifier checks that ρ is a permutation of vertices, and that $\rho(G_0) = G_1$ (*deterministic, polynomial time algorithm*)
  - Verifier accepts iff the above is correct.
- In this setting, verifier *learns the isomorphism* (i.e., the proof)!

Can we convince people differently?

- Yes! But we need to modify the way proofs are checked.

Can we convince people differently?

- Yes! But we need to modify the way proofs are checked.
  - Make proofs *interactive*, instead of only one-way

Can we convince people differently?

- Yes! But we need to modify the way proofs are checked.
  - Make proofs *interactive*, instead of only one-way
  - Verifier is allowed *private randomness*

Can we convince people differently?

- Yes! But we need to modify the way proofs are checked.
  - Make proofs *interactive*, instead of only one-way
  - Verifier is allowed *private randomness*
- In the end, we will see a (zero-knowledge) proof for graph isomorphism as follows:

  Alice: I will not give you an isomorphism, but I will prove that I could give you one, if I wanted to.

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic

Example: Graph Isomorphism

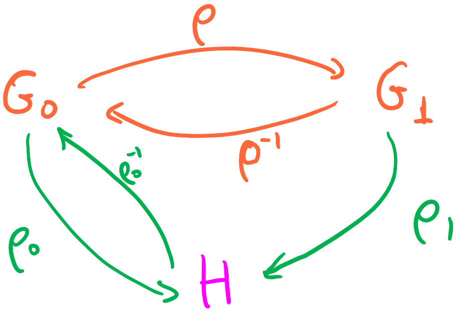
Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
    - A prover P produces a random graph H for which:

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P produces a random graph H for which:
    - It can give isomorphism ρ_0 from G_0 to H
    - It can give isomorphism ρ_1 from G_1 to H



H is a random permutation of G_0

$H = \sigma(G_0)$

$$\boxed{\begin{aligned} \sigma &= \rho_0 \\ \rho_1 &= \rho_0 \circ \rho \end{aligned}}$$

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
    - A prover P produces a random graph H for which:
        - It can give isomorphism ρ_0 from G_0 to H
        - It can give isomorphism ρ_1 from G_1 to H
    - *Above possible iff G_0 and G_1 isomorphic!*

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P produces a random graph H for which:
    - It can give isomorphism ρ_0 from G_0 to H
    - It can give isomorphism ρ_1 from G_1 to H
  - *Above possible iff G_0 and G_1 isomorphic!*
  - Verifier picks random bit $b \in \{0, 1\}$

prover sends graph H to verifier

verifier sends bit b to prover

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P produces a random graph H for which:
    - It can give isomorphism ρ_0 from G_0 to H
    - It can give isomorphism ρ_1 from G_1 to H
  - *Above possible iff G_0 and G_1 isomorphic!*
  - Verifier picks random bit $b \in \{0, 1\}$
  - Prover gives isomorphism ρ_b

Example: Graph Isomorphism

Setup:

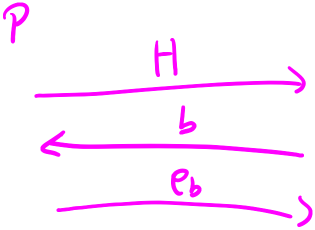
- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
    - A prover P produces a random graph H for which:
        - It can give isomorphism ρ_0 from G_0 to H
        - It can give isomorphism ρ_1 from G_1 to H
    - *Above possible iff G_0 and G_1 isomorphic!*
    - Verifier picks random bit $b \in \{0, 1\}$
    - Prover gives isomorphism ρ_b
    - Verifier checks that $\boxed{\rho_b(H) = G_b}$

$V(H, b, \rho_b)$

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P produces a random graph H for which:
    - It can give isomorphism ρ_0 from G_0 to H
    - It can give isomorphism ρ_1 from G_1 to H
  - *Above possible iff G_0 and G_1 isomorphic!*
  - Verifier picks random bit $b \in \{0, 1\}$
  - Prover gives isomorphism ρ_b
  - Verifier checks that $\rho_b(H) = G_b$
- Note that verifier will not learn isomorphism between G_0 and G_1!



Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P produces a random graph H for which:
    - It can give isomorphism ρ_0 from G_0 to H
    - It can give isomorphism ρ_1 from G_1 to H
  - *Above possible iff G_0 and G_1 isomorphic!*
  - Verifier picks random bit $b \in \{0, 1\}$
  - Prover gives isomorphism ρ_b
  - Verifier checks that $\rho_b(H) = G_b$
- Note that verifier will not learn isomorphism between G_0 and G_1!
- Note that:
  - Claim is *true* $\Rightarrow$ prover can always give isomorphism!
  - Claim is *false* $\Rightarrow$ can catch bad proof with probability $= 1/2$

P V

$H \sim G_0 \xrightarrow{H}$

$\xleftarrow{1}$

ρ_1 doesn't exist

because G_0, G_1 are not isomorphic

Verification always succeeds

$G_0 \xrightarrow{\sigma} H$, $H \xrightarrow{\sigma^{-1}} G_0$ $\quad G_1$

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P produces a random graph H for which:
    - It can give isomorphism ρ_0 from G_0 to H
    - It can give isomorphism ρ_1 from G_1 to H
  - *Above possible iff G_0 and G_1 isomorphic!*
  - Verifier picks random bit $b \in \{0, 1\}$
  - Prover gives isomorphism ρ_b
  - Verifier checks that $\rho_b(H) = G_b$
- Note that verifier will not learn isomorphism between G_0 and G_1!
- Note that:
  - Claim is *true* $\Rightarrow$ prover can always give isomorphism!
  - Claim is *false* $\Rightarrow$ can catch bad proof with probability $= 1/2$
  - Can amplify probability of catching bad proof by repeating protocol above!

Example: Graph Isomorphism

Setup:

- A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic
  - A prover P produces a random graph H for which:
    - It can give isomorphism ρ_0 from G_0 to H
    - It can give isomorphism ρ_1 from G_1 to H
  - *Above possible iff G_0 and G_1 isomorphic!*
  - Verifier picks random bit $b \in \{0, 1\}$
  - Prover gives isomorphism ρ_b
  - Verifier checks that $\rho_b(H) = G_b$
- Note that verifier will not learn isomorphism between G_0 and G_1!
- Note that:
  - Claim is *true* $\Rightarrow$ prover can always give isomorphism!
  - Claim is *false* $\Rightarrow$ can catch bad proof with probability $= 1/2$
  - Can amplify probability of catching bad proof by repeating protocol above!
- How can we model the fact that verifier does not gain knowledge?!

Simulation!

Can do it by itself

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The verifier (privately) produces a random permutation ρ and a bit b and outputs $H = \rho(G_b)$.

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The verifier (privately) produces a random permutation ρ and a bit b and outputs $H = \rho(G_b)$.
- Verifier then picks bit b from previous step

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The verifier (privately) produces a random permutation ρ and a bit b and outputs $H = \rho(G_b)$.
- Verifier then picks bit b from previous step
- Verifier gives isomorphism ρ^{-1}

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The verifier (privately) produces a random permutation ρ and a bit b and outputs $H = \rho(G_b)$.
- Verifier then picks bit b from previous step
- Verifier gives isomorphism ρ^{-1}
- Verifier checks that $\rho^{-1}(H) = G_b$

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The verifier (privately) produces a random permutation ρ and a bit b and outputs $H = \rho(G_b)$.
- Verifier then picks bit b from previous step
- Verifier gives isomorphism ρ^{-1}
- Verifier checks that $\rho^{-1}(H) = G_b$
- Simulation $\Rightarrow$ V gained no new information!

$$\xrightarrow{\quad H \quad}$$
$$\xleftarrow{\quad b \quad}$$
$$\xrightarrow{\quad \rho \quad}$$

(H, b, ρ)
random variable

Perfect Zero Knowledge Proof

Note that we usually talked about not trusting provers so far, but for Zero-Knowledge, we will *not trust verifiers* (as they may try to obtain information about the proof!)

Perfect Zero Knowledge Proof

Note that we usually talked about not trusting provers so far, but for Zero-Knowledge, we will *not trust verifiers* (as they may try to obtain information about the proof!)

Definition (Perfect Zero Knowledge)

A prover P is *perfect zero-knowledge* for language L if for every polynomial time, randomized verifier V^*, there is a randomized algorithm M^* such that for every $x \in L$ the following random variables are identically distributed:

- $\langle P, V^* \rangle(x)$, that is, output of interaction between prover P and verifier V^* on input x random variable of proof
- $M^*(x)$, that is, output of algorithm M^* (simulation) on input x

random variable of simulation

Perfect Zero Knowledge Proof

Note that we usually talked about not trusting provers so far, but for Zero-Knowledge, we will *not trust verifiers* (as they may try to obtain information about the proof!)

Definition (Perfect Zero Knowledge)

A prover P is *perfect zero-knowledge* for language L if for every polynomial time, randomized verifier V^*, there is a randomized algorithm M^* such that for every $x \in L$ the following random variables are identically distributed:

- $\langle P, V^* \rangle(x)$, that is, output of interaction between prover P and verifier V^* on input x
- $M^*(x)$, that is, output of algorithm M^* (simulation) on input x

- The above captures the idea that V^* is not gaining any extra computational power by interacting with P, since same output could have been generated by M^*

Perfect Zero Knowledge Proof[2]

- Previous definition is a bit too strict to be useful, so we relax it.[1]
- We will allow simulator to fail with small probability (denoted by outputting $\perp$)

[1] Very common phenomenon in crypto, that statistical indistinguishability too strict.
[2] For applications in cryptography, one can even relax this definition further, to include computational zero-knowledge

Perfect Zero Knowledge Proof[2]

- Previous definition is a bit too strict to be useful, so we relax it.[1]
- We will allow simulator to fail with small probability (denoted by outputting $\perp$)

Definition (Perfect Zero Knowledge)

A prover P is *perfect zero-knowledge* for language L if for every polynomial time, randomized verifier V^*, there is a randomized algorithm M^* such that for every $x \in L$ the following holds:

1. With probability $\leq 1/2$, $M^*(x) = \perp$ simulator fails $\leq \frac{1}{2}$ of time
2. Conditioned on $M^*(x) \neq \perp$, the following variables are identially distributed:
   - $\langle P, V^* \rangle(x)$, that is, output of interaction between prover P and verifier V^* on input x
   - $M^*(x)$, that is, output of algorithm M^* (simulation) on input x

[1] Very common phenomenon in crypto, that statistical indistinguishability too strict.

[2] For applications in cryptography, one can even relax this definition further, to include computational zero-knowledge

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The simulator produces a random permutation ρ and outputs $H = \rho(G_0)$.

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The simulator produces a random permutation ρ and outputs $H = \rho(G_0)$.
- Simulator then picks random bit b

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The simulator produces a random permutation ρ and outputs $H = \rho(G_0)$.
- Simulator then picks random bit b
- If $b \neq 0$ then output $\perp$ ← output failure

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The simulator produces a random permutation $\boxed{\rho}$ and outputs $H = \rho(G_0)$.
- Simulator then picks random bit b
- If $b \neq 0$ then output $\perp$
- Otherwise simulator gives isomorphism ρ^{-1}
- Simulator checks that $\rho^{-1}(H) = G_0$ always passes

if $b=0$

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The simulator produces a random permutation ρ and outputs $H = \rho(G_0)$.
- Simulator then picks random bit b
- If $b \neq 0$ then output $\perp$
- Otherwise simulator gives isomorphism ρ^{-1}
- Simulator checks that $\rho^{-1}(H) = G_0$
- Simulation $\Rightarrow$ perfect zero knowledge for our prover P!

Simulation of Protocol

- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The simulator produces a random permutation ρ and outputs $H = \rho(G_0)$.
- Simulator then picks random bit b
- If $b \neq 0$ then output $\perp$
- Otherwise simulator gives isomorphism ρ^{-1}
- Simulator checks that $\rho^{-1}(H) = G_0$
- Simulation $\Rightarrow$ perfect zero knowledge for our prover P!
- Note that whenever we don't fail, we output same distribution as the original protocol!

$P, V \quad (H, 0, \rho_0)$ $\qquad$ $M \quad (H, 0, \rho_0)$

$H = \rho_0(G_0)$ $\qquad$ $H = \rho_0(G)$

random variable ρ_0 $\longleftrightarrow$ picking ρ.

Conclusion

- We saw today how the power of interaction can be used to verify validity of “proofs” without conveying information about it

Conclusion

- We saw today how the power of interaction can be used to verify validity of "proofs" without conveying information about it
- Has applications in
  - Modern cryptography
  - Credit Cards
  - Passwords
  - Complexity Theory (can use zero-knowledge to construct complexity classes)
  - Used in cryptocurrencies (validate transactions without giving details about transactions)

Acknowledgement

- Lecture based largely on:
  - Oded Goldreich's Foundations of Cryptography book, Chapter 6
  - Berkeley & MIT's 6.875 Lecture 14

    `https://inst.eecs.berkeley.edu/~cs276/fa20/slides/lec14.pdf`