Lecture 22: Zero-Knowledge Proofs

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

November 25, 2021

Overview

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

Rate this course!

Please log in to

https://evaluate.uwaterloo.ca/

from *November 24th until December 7th* and provide us with your evaluation and feedback on the course!

- This would really help me figuring out what worked and what didn't for the course

Research Opportunities at UW!

Consider doing a URA, URF or USRA with a U Waterloo faculty!
See research openings at:

- Undergraduate Research Assistantship (URA):

  `https://cs.uwaterloo.ca/computer-science/current-undergraduate-students/research-opportunities/undergraduate-research-assistantship-ura-program`

- Undergraduate Research Fellowship (URF):

  `https://grec.cs.uwaterloo.ca/`

- Undergraduate Research Internship (URI):

  `https://cs.uwaterloo.ca/current-undergraduate-students/research-opportunities/undergraduate-research-internship-uri-program`

- For Canadians, please check out NSERC's USRA:

  `https://cs.uwaterloo.ca/usra`

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.

How do you prove that you know something without showing how you know 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?

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!)

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:
  - 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*!

C = graph G is Hamiltonian

P := give a Hamiltonian cycle

Example: Factoring

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

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 $N = p \cdot q$

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*)

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

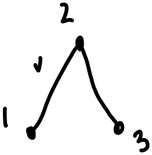
- Setup:
  - A claim $\mathcal{C} :=$ graphs G_0, G_1 are isomorphic

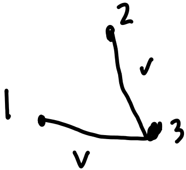
iff there is a permutation of vertices

$$\sigma : V_{G_0} \to V_{G_1}$$

s.t.

$$\{u, v\} \in E_{G_0} \iff \{\sigma(u), \sigma(v)\} \in E_{G_1}$$





$\sigma(1) = 1$

$\sigma(2) = 3$

$\sigma(3) = 2$

isomorphic

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.

G_0 G_1 isomorphic ($\underline{\rho}$ isomorphism)

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

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

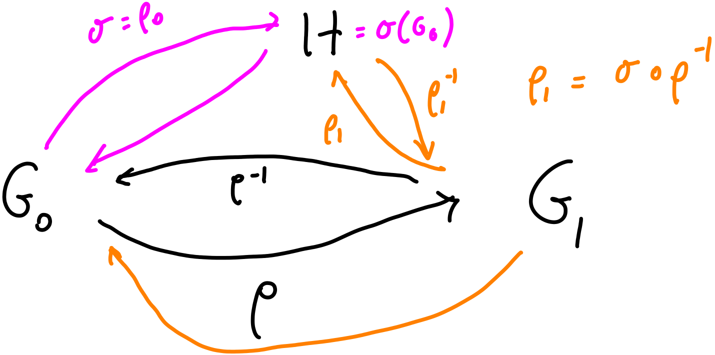
Setup:

- 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

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 $b \in \{0, 1\}$

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 $\rho_b(H) = G_b$

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!

$$\rho_b = \rho \circ \tilde{\pi}$$

random permutation!

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 $= 1/2$

$G_0 \leftrightarrow H$

$b = 1$ then prover's done

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!

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!

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!)

knowledge

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

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 $\leq 1/2$, $M^*(x) = \perp$ fails
2. Conditioned on $M^*(x) \neq \perp$, the following variables are identially distributed: not failing
   - $\langle P, V^* \rangle (x)$, 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

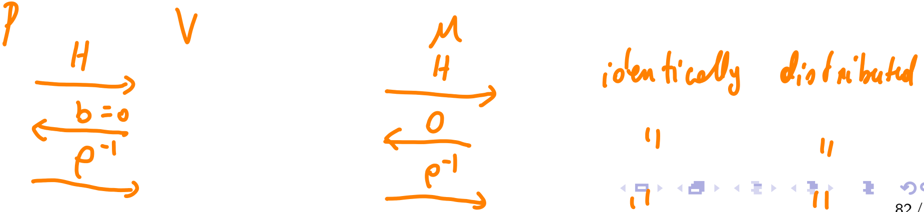
- Key idea: if claim is indeed true, then verifier's view of proof could have been simulated by the verifier alone!
- Simulated protocol:
- The simulator produces a random permutation ρ and outputs $H = \rho(G_0)$.
- Simulator then picks random bit b
- If $b \neq 0$ then output $\perp$

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 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!

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