Lecture 11: *PCP Theorem & Zero-knowledge Proofs*

Advanced Algorithms

Sid Chi-Kin Chau

Australian National University

✉ sid.chau@anu.edu.au

October 26, 2022

Can you prove that you know a secret, without revealing the secret?

Review: NP Problems

Definition (Class NP)

- Let $|\mathcal{I}| = \#$bits to represent $\mathcal{I}$ and $|w| = \#$bits to represent w
- Define a class of problems called NP: For all $\mathcal{L} \in$ NP, there exist a polynomial-time bound Turing machine $\mathcal{M}$ and a polynomial function $p(\cdot)$, such that
  - If $\mathcal{I} \in \mathcal{L}$, then there exists a witness w where $|w| \leq p(|\mathcal{I}|)$, such that $\mathcal{M}(\mathcal{I}, w)$ returns TRUE, and
  - If $\mathcal{I} \notin \mathcal{L}$, then for any witness w where $|w| \leq p(|\mathcal{I}|)$, $\mathcal{M}(\mathcal{I}, w)$ returns FALSE

- NP stands for Non-deterministic Polynomial-time
- NP are the problems that can be verified efficiently when given a proof
- Questions:
  - Does verification really take polynomial time? Can it be faster?
  - Do we really take the witness (i.e., solution) for verification? Can the proof be smaller?

Interactive Proof System

Definition (Interactive Proof System)

- **Prover** claims to know a witness w to $\mathcal{I} \in \mathcal{L}$
  - ⚠ Prover may be dishonest
- **Verifier** wants to verify Prover's claim and decide to accept/reject $\mathcal{I} \in \mathcal{L}$
- **Protocol** is the conversation between Prover and Verifier

- Naïve protocol: Just send the witness w from Prover to Verifier
- Efficient protocol: Smaller than $|w|$ and faster running time

Example (Applications of Interactive Proof Systems)

- Prover claims to have a valid credential for authentication without revealing the credential
- Prover wants to convince the truth of some private data (e.g., income, date of birth) for privileged applications (e.g., mortgage, restricted access) without revealing the data

Interactive Proof System

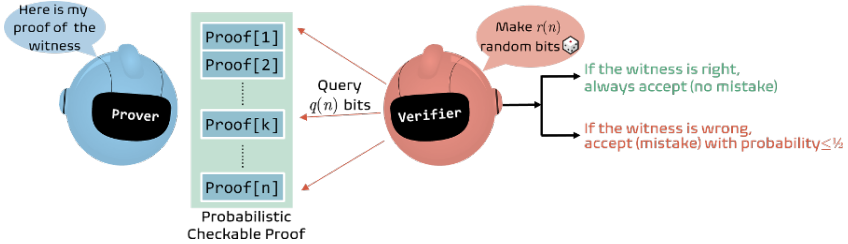
Definition (Properties of Interactive Proof System)

- **Completeness**: If $\mathcal{I} \in \mathcal{L}$, then Prover will eventually convince Verifier
- **Soundness**: If $\mathcal{I} \notin \mathcal{L}$, then Prover will never convince Verifier (except with a small probability to make mistakes)
  - Possible to improve soundness by extra computational hardness assumptions: Prover cannot do factorization or discrete logarithm fast (not without a quantum computer)
- **Zero-Knowledge**: Verifier learns nothing about the witness w, but only the fact $\mathcal{I} \in \mathcal{L}$
  - E.g., Verifier knows that Prover has a valid password, but does not know the password

Probabilistic Checkable Proof

Definition (Probabilistic Checkable Proof (PCP))

- Prover prepares a special type of proof, called *Probabilistic Checkable Proof* (PCP), based on the witness w
- Verifier makes probabilistic checking decisions with $r(n)$ random bits
- Verifier need to access at most $q(n)$ bits on the PCP to decide to accept/reject
  - If $\mathcal{I} \in \mathcal{L}$, Verifier always accepts (i.e., no mistake)
  - If $\mathcal{I} \notin \mathcal{L}$, Verifier accepts (i.e., makes mistake) with probability $< \frac{1}{2}$



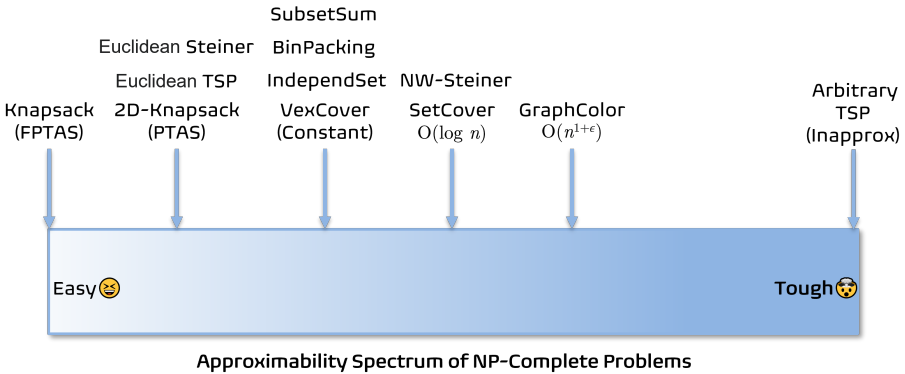
PCP Theorem

Theorem (PCP Theorem)

Any NP *problem admits a probabilistic checkable proof (PCP) that can be checked with* $r(n) = \mathsf{O}(\log n)$ *random bits to access* $q(n) = \mathsf{O}(1)$ *bits on the PCP*

- Conventional (logical) proofs must be accessed sequentially from the first line to the last line to determine if the proof is correct or not
  - Random access to conventional proofs cannot make sense of the correctness
- Probabilistic checkable proofs use *error correction code* in encoding the proofs, such that random access to the proof can give an idea whether the proof is correct or not
  - The size of a PCP can be larger than the size of a witness $|w|$, but only a constant number of bits in the probabilistic checkable proof are accessed by Verifier
  - A PCP may take significant running time to construct, but only constant running time to verify
- PCP Theorem can characterize the hardness of approximation algorithms of NP problems

Spectrum of Approximability

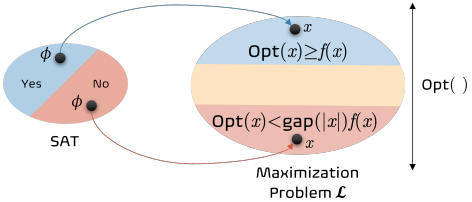


- Why do NP-complete problems have different approximability?
- Are these approximation ratios tight? What are the lower bounds of approximation ratios?

Gap-introducing Reductions

Definition (Gap-introducing Reduction)

- **Reduction to Maximization Problem**: Reducing from SAT to maximization problem $\mathcal{L}$, given an instance ϕ of SAT, it outputs in polynomial time an instance x of $\mathcal{L}$, such that
  - ϕ is satisfiable $\Rightarrow$ $\mathsf{Opt}(x) \geq f(x)$, and
  - ϕ is not satisfiable $\Rightarrow$ $\mathsf{Opt}(x) < \mathsf{gap}(|x|) \cdot f(x)$
- **Reduction to Minimization Problem**: Similarly, ϕ is satisfiable $\Rightarrow$ $\mathsf{Opt}(x) \leq f(x)$, and ϕ is not satisfiable $\Rightarrow$ $\mathsf{Opt}(x) > \mathsf{gap}(|x|) \cdot f(x)$
- We write SAT $\preceq_{\mathsf{gap}} \mathcal{L}$



Inapproximability of NP Problems

Corollary

If there is a gap-introducing reduction from SAT *to maximization problem* $\mathcal{L}$ *(*$\mathsf{SAT} \preceq_{\mathsf{gap}} \mathcal{L}$*), then the approximation ratio of* $\mathcal{L}$ *is upper bounded by* $\alpha(n) \leq \mathsf{gap}(n)$*, unless* $\mathsf{P} = \mathsf{NP}$

Proof:

- Suppose approximation algorithm $\mathcal{A}(x)$ of instance x with approximation ratio $\alpha(|x|)$:

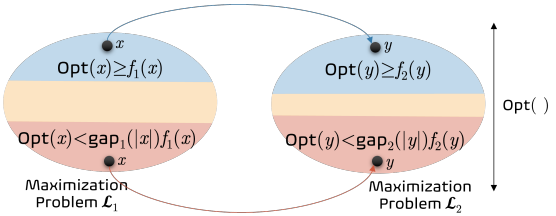
$$\alpha(|x|) \cdot \mathsf{Opt}(x) \leq \mathcal{A}(x) \leq \mathsf{Opt}(x)$$

- Since $\mathsf{SAT} \preceq_{\mathsf{gap}} \mathcal{L}$, given an instance ϕ of SAT, there is an instance x of $\mathcal{L}$, such that
  - ϕ is satisfiable $\Rightarrow$ $\frac{1}{\alpha(|x|)}\mathcal{A}(x) \geq \mathsf{Opt}(x) \geq f(x)$, and
  - ϕ is not satisfiable $\Rightarrow$ $\mathcal{A}(x) \leq \mathsf{Opt}(x) < \mathsf{gap}(|x|) \cdot f(x)$
- Suppose $\alpha(|x|) > \mathsf{gap}(|x|)$. Then, we can use $\mathcal{A}(x)$ to solve SAT in polynomial time
  - ϕ is satisfiable $\Rightarrow$ $\mathcal{A}(x) \geq \mathsf{gap}(|x|) \cdot f(x)$, and
  - ϕ is not satisfiable $\Rightarrow$ $\mathcal{A}(x) < \mathsf{gap}(|x|) \cdot f(x)$

Gap-preserving Reductions

Definition (Gap-preserving Reductions)

- **Gap-preserving Reduction**: Reducing from maximization problem $\mathcal{L}_1$ to maximization problem $\mathcal{L}_2$, given an instance x of $\mathcal{L}_1$, it outputs in polynomial time an instance y of $\mathcal{L}_2$
  - $\mathsf{Opt}(x) \geq f_1(x) \;\Rightarrow\; \mathsf{Opt}(y) \geq f_2(y)$,
  - $\mathsf{Opt}(x) > \mathsf{gap}_1(|x|) \cdot f_1(x) \;\Rightarrow\; \mathsf{Opt}(y) < \mathsf{gap}_2(|y|) \cdot f_2(x)$
- Similarly, gap-preserving reductions from maximization/minimization problem $\mathcal{L}_1$ to maximization/minimization problem $\mathcal{L}_2$
- We write $\mathsf{SAT} \preceq_{\mathsf{gap}} \mathcal{L}_1 \preceq_{\mathsf{gap}} \mathcal{L}_2 \;\Rightarrow\; \mathsf{SAT} \preceq_{\mathsf{gap}} \mathcal{L}_2$



Max3SAT and Max$k$FuncSAT

Definition (Max3SAT)

- 3CNF formula ϕ, with n variables and m clauses. Each clause has 3 literals:
  - $\phi = (x_1 \vee \bar{x}_2 \vee x_4) \wedge ... \wedge (x_2 \vee \bar{x}_3 \vee \bar{x}_n)$
- Find an assignment that satisfies as many clauses as possible

Definition (Max$k$FuncSAT)

- Given n boolean variables $(x_1, x_2, ..., x_n)$ and m boolean functions $(f_1, f_2, ..., f_m)$
- Each $f_j : \{0,1\}^k \mapsto \{0,1\}$ from a constant k number of the literals of $(x_1, x_2, ..., x_n)$
- Find an assignment that satisfies as many boolean functions as possible

Inapproximability of Max3SAT

Theorem

SAT $\preceq_{\text{gap}}$ Max$k$FuncSAT $\preceq_{\text{gap}}$ Max3SAT. *Hence, there is no PTAS for* Max3SAT

Proof:

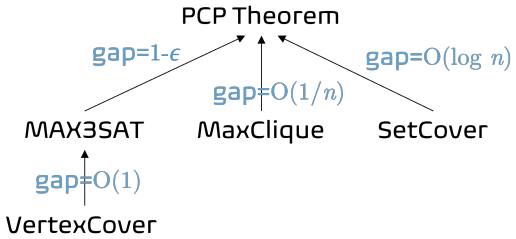
- We first show SAT $\preceq_{\text{gap}}$ Max$k$FuncSAT
- Verifier reads PCP of boolean formula ϕ for SAT with $c \log n$ random bits and q query bits
  - The size of PCP is at most $q2^{c \log n} = qn^c$ bits
  - For each $r \in \{0,1\}^{c \log n}$, Verifier reads q bits out of a total of qn^c bits in the PCP
- Define an instance of Max$k$FuncSAT based on the PCP of ϕ
  - Instance x is the qn^c-bit string of the PCP of ϕ
  - Boolean functions $(f_1, f_2, ...f_{n^c})$, each $f_j : \{0,1\}^q \mapsto \{0,1\}$ is the decision of Verifier on x of each random bit
- We have SAT $\preceq_{\text{gap}}$ Max$k$FuncSAT, because
  - ϕ is satisfiable $\Rightarrow f_j = 1$ for all j (i.e. total num of satisfied boolean functions $= n^c$)
  - ϕ is not satisfiable $\Rightarrow \mathbb{P}(\text{Verifier accepts}) < \frac{1}{2}$ (i.e. total num of satisfied boolean funcs $< \frac{n^c}{2}$)

Inapproximability of Max3SAT

Proof (Cont.):

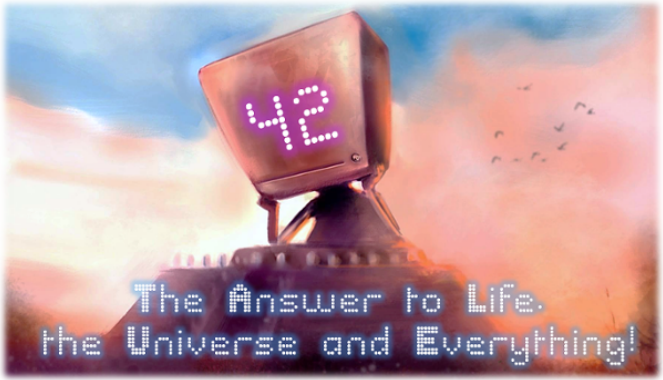
- We next show Max$k$FuncSAT $\preceq_{\text{gap}}$ Max3SAT
- Each f_j can be represented as a SAT formula, ψ_j with at most 2^q clauses each having at most q literals
- Define $\psi = \bigwedge_{1 \leq j \leq n^c} \psi_j$. We obtain
  - ϕ is satisfiable $\Rightarrow$ ψ is satisfiable (i.e., max num of satisfied clauses in $\psi = n^c 2^q$)
  - ϕ is not satisfiable $\Rightarrow$ at least 1 clause in each f_j is not satisfiable (i.e., max num of satisfied clauses in $\psi < n^c(2^q - \frac{1}{2})$)
- Every instance ψ of SAT with $n^c 2^q$ clauses can be converted to an instance ψ' of 3SAT with $n^c 2^q (q-2)$ clauses
  - E.g., $l_1 \vee ... \vee l_n \rightarrow (l_1 \vee l_2 \vee x_2) \wedge (\bar{x}_2 \vee l_3 \vee x_3) \wedge (\bar{x}_3 \vee l_4 \vee x_4) \wedge ... \wedge (\bar{x}_{n-2} \vee l_{n-1} \vee l_n)$
- There exists a gap-introducing reduction from an instance ϕ of SAT to an instance ψ' of Max3SAT such that
  - ϕ is satisfiable $\Rightarrow$ Max num. of satisfied clauses in $\psi' = n^c 2^q (q-2)$
  - ϕ is not satisfiable $\Rightarrow$ Max num. of satisfied clauses in $\psi' < n^c\big(2^q(q-2) - \frac{1}{2}\big)$

Inapproximability by PCP Theorem



- Gap-preserving reductions can prove inapproximability of NP-hard problems
- Use PCP Theorem to make gap-introducing reductions to various NP-hard problems
- Advanced techniques are needed to amplify the gap to get tighter bounds for approximation ratios

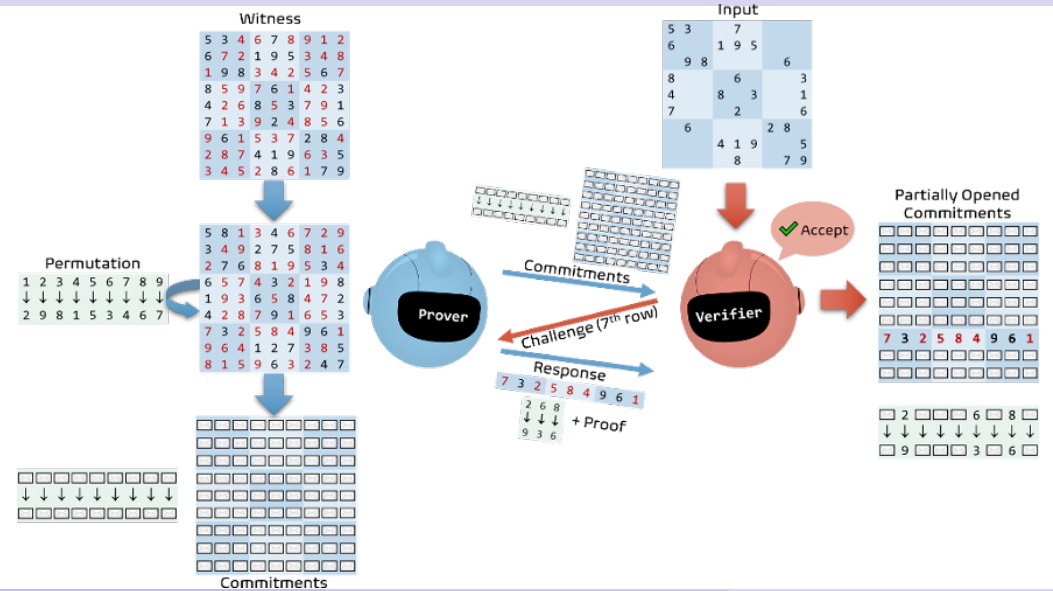
A Short Answer to Everything!?



The Ultimate Answer to Life, The Universe and Everything is...42!

– The Hitchhiker's Guide to the Galaxy

Zero-Knowledge Proof: Prove You Know the Solution Without Showing it



Zero-Knowledge Proofs

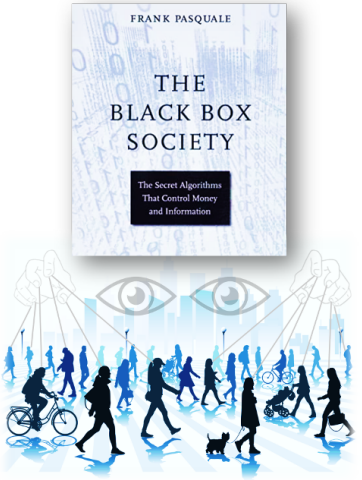
- PCP Theorem is a revolutionary idea beyond for proving inapproximability
- PCP Theorem opens up the possibility of zero-knowledge proofs in many applications:
  - Outsourcing computations in cloud computing
  - Secure authentication and decentralized identities
  - Privacy-preserving databases
  - Cryptocurrencies, blockchain and decentralized finance

Definition (zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge))

- **Zero-Knowledge**: Prove the knowledge of a secret, without revealing the secret
- **Succinctness**: Compact in the proof size for communication and efficient in verification
- **Non-Interactiveness**: It can be self-proving, without the presence of a verifier to issue a challenge

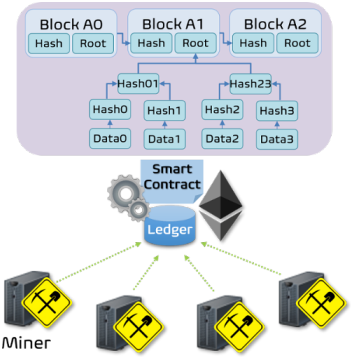
Black Box Society

- We are living in a **black box** society
  - Controlled by a lot of opaque authorities & entities (e.g., tech companies, governments & regulators)
  - Entrust private data & key decisions to third-parties without awareness of how the data is used & decisions are made
  - Rely too much on algorithms for decision-making and operation management
  - Over-abundance of AI & data mining in every aspect of lives
  - We need more transparency & accountability in our society
- Every system or algorithm should prove themselves for correctness, fairness, inclusiveness, etc.
- **How can systems & algorithms be properly checked and verified in a transparent, accountable & efficient manner?**

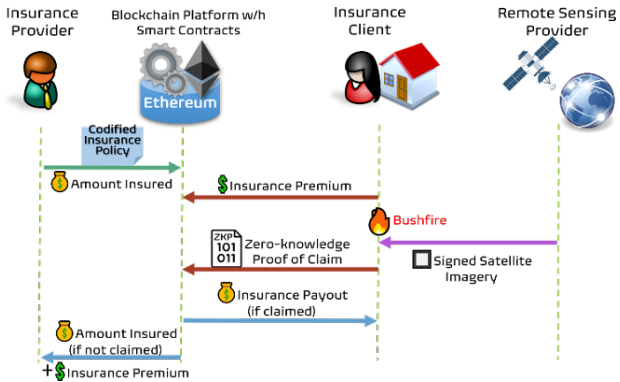


Blockchain Platform

- Permissionless blockchain is a *decentralized* programmable ledger platform with public ownership
  - Smart contracts are programs executed on blockchain platform, which require cryptocurrencies to pay for (gas) costs
  - Smart contracts are executed independently by miners, who are rewarded by cryptocurrencies, using a consensus protocol
  - Smart contract executions are recorded on a pubic immutable ledger (based on hash pointers & Merkle trees)
- Blockchain can be a public transparent platform of verification to mitigate a black box society
  - Submit publicly verifiable zero-knowledge proofs on blockchain
  - Blockchain automatically verifies the zero-knowledge proofs and records the verification results
  - Public transparent verification can be oblivious of the verified subjects and reveals no private data



Example: Blockchain-enabled Insurance



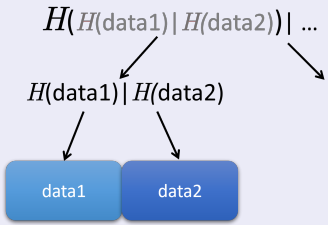
- Blockchain-enabled insurance
  - Automate underwriting: Increase speed and cost efficiency with smart contracts
  - Automate claims settlement: Cut costs with automated claim and data verification
  - Reduce fraud and abuse: Prevent misuse with improved traceability and accountability

Review: Merkle Tree

- Hash pointers:
  - Unforgeability of Collision-resistant Hash Function:
    - ★ If we know $H(x) = H(y)$, it's safe to assume $x = y$
  - Use $H(\text{data})$ as a fingerprint for a given data block
  - Hash pointer is message digest of data
  - Useful to build more sophisticated data structures

Definition (Merkle Tree)

- A binary tree with hash pointers:
  - The leaves are the data blocks
  - Each data block is pointed by a hash pointer
  - A parent hash pointer can point to two child hash pointers
  - The root pointer indirectly points to all children
- Merkle tree can detect the differences of data blocks efficiently by comparing the hash pointers at each level, which takes only $\log(n)$ in time/space



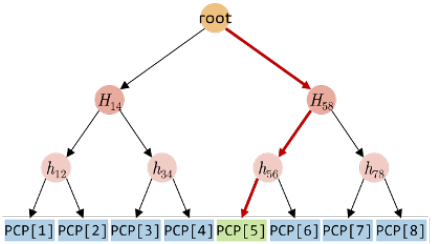
Zero-knowledge Proofs by PCP

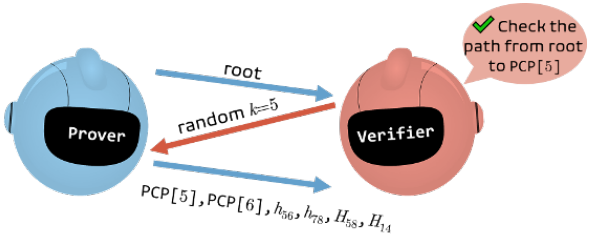
- Can Prover just hand over the requested q query bits to Verifier at his request? This take only constant communication
  - ⚠ Prover may be dishonest, who forges incorrect PCP based on Verifier's random choices

PCP-based Zero-Knowledge Proof [Input: $x \in \mathcal{L}$]

1. Both Prover and Verifier agree on a collision-resistant hash function $H(\cdot)$
2. Prover uses witness w of input x to construct a PCP π. Prover builds a Merkle tree with the leaf values as $\{\pi[k]\}$ (each has $q(n)$-bit) using $H(\cdot)$ and send the root value to Verifier
3. Verifier generates $r(n)$ random bits and sends to Prover
4. Prover relies with the $q(n)$-bit leaf value in the Merkle tree, specified by the $r(n)$ random bits, together with an authentication path to prove that the answer is consistent with the root of the Merkle tree
5. Verifier accepts if the $q(n)$-bit PCP $\pi[k]$ is consistent with the root of the Merkle tree, and is a correct PCP of input x

Zero-knowledge Proofs by PCP

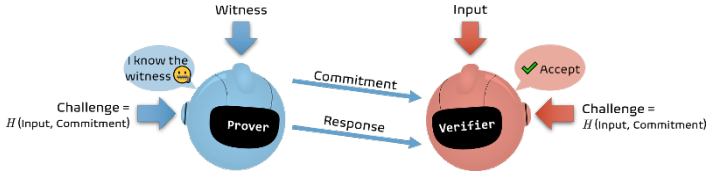




- The PCP protocol is succinct, because the communication size is small $O(\log n)$, which is size of the authentication path of PCP $\pi[k]$
- Merkle tree root ensures the unforgeability property of PCP π
  - Prover cannot alter PCP π after receiving the $r(n)$ random bits from Verifier
- Assumption: Presence of collision-resistant hash function
  - ⚠ No hash function has been theoretically proved yet

Non-interactive Zero-Knowledge Proofs

- PCP-based protocol is interactive, requiring a challenge from a honest Verifier
- Can Prover generate a succinct proof without the presence of a honest Verifier?
- Yes, using Fiat-Shamir heuristic:
  - Prover sets random $k = H(\text{Input}, \text{Commitment})$
  - Since $H(\cdot)$ is collision-resistant hash function, it is difficult for Prover to manipulate the root to influence the choice of k to be a specific value
  - Any public Verifier can verify the proof by setting random $k = H(\text{Input}, \text{Commitment})$



- PCP-based zero-knowledge proofs are succinct and non-interactive, but require too much computational overhead to generate PCP, and are not practical for real-world applications

Cryptographic Zero-Knowledge Proofs

- Practical zero-knowledge proofs use cryptography
  - E.g., factoring, discrete logarithm of large integers
  - Assume Prover is computationally limited, who cannot solve certain cryptographic problems

Definition (Cryptographic Commitment)

- Cryptographic commitment emulates a sealed envelope of a secret committed value
- $\text{Commit}(m, r)$, where m is the committed value, r is randomness to improve hiding
- $\text{Verify}[C, m', r'] = \begin{cases} \text{accept}, & \text{if } C = \text{Commit}(m', r') \\ \text{reject}, & \text{otherwise} \end{cases}$
- **Binding**: Cannot find the two valid openings efficiently for a cryptographic commitment
  - Cannot find $m'_1 \neq m'_2$ such that $\text{Verify}[C, m'_1, r'_1] = \text{Verify}[C, m'_2, r'_2] = \text{accept}$
- **Hiding**: Cannot reveal anything about the committed value
  - If $C = \text{Commit}(m, r)$ and r is uniformly sampled, then C and m are statistically independent

Cryptographic Commitments: Examples

Example (Hash Based Cryptographic Commitment)

- Collision-resistant hash function $H(\cdot)$
- Commit$(\mathsf{m}, \mathsf{r}) \triangleq H(\mathsf{m}|\mathsf{r})$
- Binding follows from collision-resistance

Example (Perdersen Commitment)

- Finite cyclic group $\mathbb{G} = \{1, g, g^2, ..., g^{q-1}\}$
  - $g^i \cdot g^j = g^{i+j \mod q}$, and q is a prime number
- Uniformly pick $g, h \in \mathbb{G}$ at random, and let m, r be integers $\in \{0, ..., q-1\}$
- Commit$(\mathsf{m}, \mathsf{r}) \triangleq g^{\mathsf{m}} \cdot h^{\mathsf{r}} \in \mathbb{G}$
- Binding follows from the computational hardness of discrete logarithm $\mathsf{m} \stackrel{?}{=} \log(g^{\mathsf{m}})$
- Perdersen commitment is computationally binding & statistically hiding

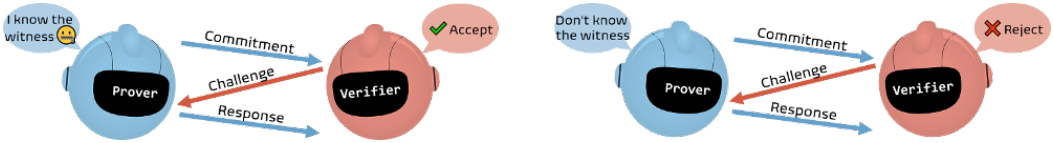
Homomorphic Property of Perdersen Commitment

- Let $C_1 = \mathsf{Commit}(\mathsf{m}_1, \mathsf{r}_1) \triangleq g^{\mathsf{m}_1} \cdot h^{\mathsf{r}_1}$ and $C_2 = \mathsf{Commit}(\mathsf{m}_2, \mathsf{r}_2) \triangleq g^{\mathsf{m}_2} \cdot h^{\mathsf{r}_2}$
- Homomorphic property:

$$C_1 \cdot C_2 = g^{(\mathsf{m}_1+\mathsf{m}_2)} \cdot h^{(\mathsf{r}_1+\mathsf{r}_2)} = \mathsf{Commit}(\mathsf{m}_1 + \mathsf{m}_2, \mathsf{r}_1 + \mathsf{r}_2)$$

- Hence, anyone can obtain the commitment of a sum of committed values by multiplication on Perdersen commitments
- Homomorphic property is very useful to build zero-knowledge proofs for Perdersen commitments

Σ-Protocol



Definition (Σ-Protocol)

1. Prover knows a witness w to a statement x
   - E.g., w is a password, x is the statement that "do you have a valid password?"
2. Prover sends some commitment C (e.g., Perdersen commitments) to Verifier that prevent Prover from altering the witness w he wants to prove
3. Verifier sends a random challenge β to test Prover
4. Prover responds with a proof $\pi(w, \beta)$
5. Verifier accepts if π is sufficient to prove that Prover knows a witness w to x, or rejects

Σ-Protocol Properties

Definition (Σ-Protocol Properties)

- **Completeness**: If Prover indeed knows a witness w to x and he strictly follows the Σ-protocol, then Verifier should accept
- **Soundness**: If Prover does not know a witness w to x and no matter what he does, then Verifier should reject with high probability
  - Equivalently, if Prover strictly follows the Σ-protocol and Verifier accepts, then Prover should know a witness w. So, it is possible to *extract* w from π (by some *extraordinary* techniques)
- **Zero-Knowledge**: π should not reveal the witness w, if Verifier honestly follows the Σ-protocol
  - Equivalently, if Verifier cannot distinguish between Prover who knows the witness w, and an *extraordinary* Simulator who can simulate Prover's response but does not know the witness w

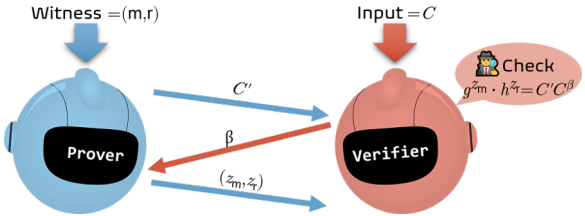
- Soundness and zero-knowledge seems contradictory to each other at first glance, with an emphasis on some *extraordinary* scenarios

Σ-Protocol

Example (Σ-Protocol for Knowledge of Perdersen Commitment)

Verifier publicly knows commitment C; Prover wants to prove that he privately knows (m, r) such that $g^{\mathsf{m}} \cdot h^{\mathsf{r}} = C$

1. Prover randomly generates $(\mathsf{m}', \mathsf{r}')$ and sends $C' = g^{\mathsf{m}'} \cdot h^{\mathsf{r}'}$ to Verifier
2. Verifier sends a random challenge β to Prover
3. Prover responds with $z_{\mathsf{m}} = \mathsf{m}' + \beta \cdot \mathsf{m}$ and $z_{\mathsf{r}} = \mathsf{r}' + \beta \cdot \mathsf{r}$
4. Verifier accepts if $g^{z_{\mathsf{m}}} \cdot h^{z_{\mathsf{r}}} \stackrel{?}{=} C' \cdot C^{\beta}$



Σ-Protocol

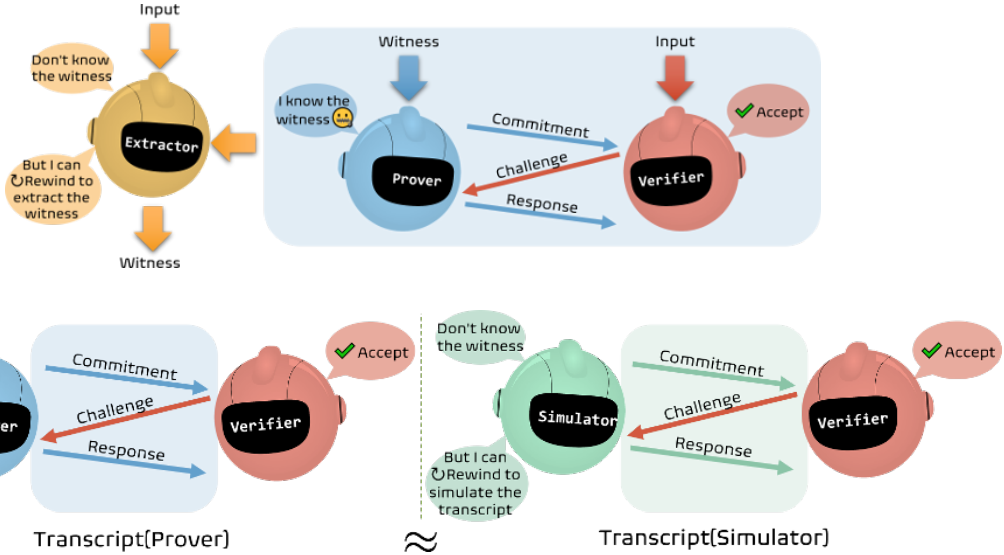
Theorem

Σ-Protocol for knowledge of Perdersen commitment satisfies completeness, soundness and zero-knowledge

Proof:

- Note that Σ-protocol for knowledge of commitment is used in Elliptic Curve Digital Signature Algorithm (ECDSA) in most cryptocurrencies to authenticate transactions
- Completeness: $g^{\mathsf{m}} \cdot h^{\mathsf{r}} = C \;\Rightarrow\; g^{\mathsf{m}'+\beta\cdot\mathsf{m}} \cdot h^{\mathsf{r}'+\beta\cdot\mathsf{r}} = g^{\mathsf{m}'} \cdot h^{\mathsf{r}'} \cdot (g^{\mathsf{m}} \cdot h^{\mathsf{r}})^{\beta} = C' \cdot C^{\beta}$
- Soundness: We show that for every possible Prover, an Extractor exists, who can extract the witness (m, r) by *rewinding* the Σ-Protocol and replaying it with a different challenge
- Zero-Knowledge: We create a Simulator, who can simulate the response z_{m} and z_{r} without knowing the witness (m, r) by *rewinding* the Σ-Protocol

Proof Ideas


Input
Don't know the witness
Extractor
But I can ↻Rewind to extract the witness
Witness
Witness
Input
I know the witness
Commitment
Challenge
Response
Prover
Verifier
Accept
I know the witness
Prover
Commitment
Challenge
Response
Verifier
Accept
Transcript(Prover)
≈
Don't know the witness
Simulator
But I can ↻Rewind to simulate the transcript
Commitment
Challenge
Response
Verifier
Accept
Transcript(Simulator)

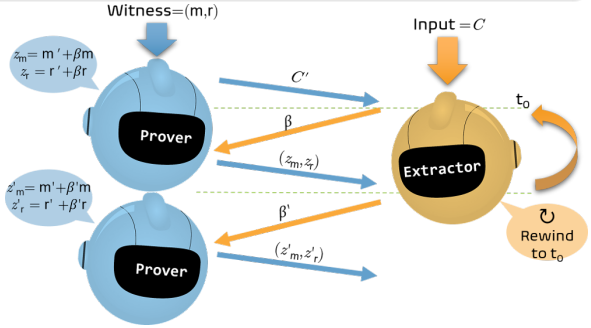
Extractor

Definition (Extractor)

- Extractor is a special Verifier, and if the Prover succeeds in completing the proof, then the Extractor should be able to extract the Prover's original witness
- Extractor is not required to exist during a normal run of the protocol. We simply show that it exists if we can *rewind* to Prover's execution and allow us to extract the witness

- After Prover responding $(z_{\mathsf{m}}, z_{\mathsf{r}})$ to challenge β, Extractor rewinds to send Prover a different challenge β'
- Prover is not aware of being rewound and responds $(z'_{\mathsf{m}}, z'_{\mathsf{r}})$ to challenge β'
- We can extract the witness by:

$$\mathsf{m} = \frac{z_{\mathsf{m}} - z'_{\mathsf{m}}}{\beta - \beta'}, \quad \mathsf{r} = \frac{z_{\mathsf{r}} - z'_{\mathsf{r}}}{\beta - \beta'}$$

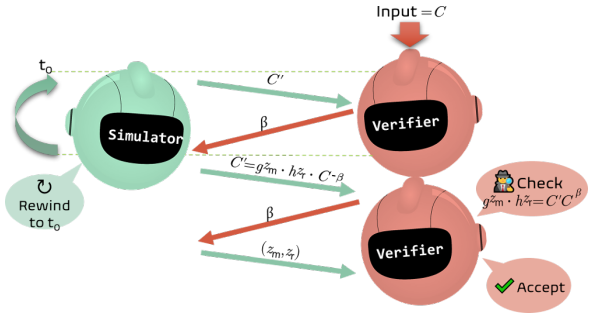


Simulator

Definition (Simulator)

- Simulator is like a special kind of Prover, but, unlike a real Prover (which knows the witness), it does not know the witness
- Simulator can "fool" honest Verifier by *rewinding*, and produce an accepting response that is statistically indistinguishable from the response of a real Prover

- After Verifier revealing challenge β, Simulator rewinds to instead send $C' = g^{z_m} \cdot h^{z_r} \cdot C^{-\beta}$ to Verifier, where (z_m, z_r) are just random numbers
- Verifier is not aware of being rewound and accepts because $g^{z_m} \cdot h^{z_r} = C' \cdot C^{\beta}$
- Response (z_m, z_r) are statistically indistinguishable from real Prover's one



Σ-Protocol

Example (Σ-Protocol for Knowledge of Membership)

Verifier publicly knows a set $\mathcal{X} = \{\mathsf{m}_1, ..., \mathsf{m}_n\}$ and a commitment C; Prover wants to prove that he privately knows $(\mathsf{m}_i, \mathsf{r})$ such that $\mathsf{m}_i \in \mathcal{X}$ and $g^{\mathsf{m}_i} \cdot h^{\mathsf{r}} = C$

1. Prover first randomly generates $(\mathsf{m}'_j, \mathsf{r}'_j)$ and computes $g^{\mathsf{m}'_j} h^{\mathsf{r}'_j}$ for all $j \in \{1, ..., n\}$. Then, Prover randomly generates β_j for each $j \in \{1, ..., n\} \backslash \{i\}$, and computes

$$z_{\mathsf{m}_j} = \begin{cases} \mathsf{m}'_j + (\mathsf{m}_i - \mathsf{m}_j)\beta_j, & \text{if } j \in \{1, ..., n\} \backslash \{i\} \\ \mathsf{m}'_i, & \text{if } j = i \end{cases}$$

   Prover sends $(g^{\mathsf{m}'_j} h^{\mathsf{r}'_j}, z_{\mathsf{m}_j})_{j=1}^n$ to Verifier
2. Verifier sends a random challenge β to Prover

Example (Σ-Protocol for Knowledge of Membership)

(Cont.)

4. Prover sets $\beta_i = \beta - \sum_{j \neq i} \beta_j$, then computes $z_{\mathsf{r}_j} = \mathsf{r}'_j + \mathsf{r} \cdot \beta_j$ for all $j \in \{1, ..., n\}$, and sends $(\beta_j, z_{\mathsf{r}_j})_{j=1}^n$ to Verifier
5. Verifier checks whether $\beta \stackrel{?}{=} \sum_{i=1}^n \beta_j$ and

$$g^{z_{\mathsf{m}_j}} \cdot h^{z_{\mathsf{r}_j}} \stackrel{?}{=} g^{\mathsf{m}'_j} h^{\mathsf{r}'_j} \cdot \left(\frac{C}{g^{\mathsf{m}_j}}\right)^{\beta_j} \text{for all } j \in \{1, ..., n\}$$

- Completeness: $g^{\mathsf{m}_i} \cdot h^{\mathsf{r}} = C \;\Rightarrow\; g^{\mathsf{m}'_j + (\mathsf{m}_i - \mathsf{m}_j)\beta_j} \cdot h^{\mathsf{r}'_j + \mathsf{r} \cdot \beta_j} = g^{\mathsf{m}'_j} h^{\mathsf{r}'_j} \cdot \left(\frac{C}{g^{\mathsf{m}_j}}\right)^{\beta_j}$
- Soundness: Can be proven by creating a suitable extractor
- Zero-Knowledge: Can be proven by creating a suitable simulator

References

Reference Materials

- Approximation Algorithms (V. Vazirani), Springer
  - Chapter 29: Hardness of Approximation

Recommended Materials

- *Proofs, Arguments, and Zero-Knowledge* (Justin Thaler), 2022