Zero-Knowledge

Oleg Koshik
University of Tartu
oleg.koshik@ut.ee

Motivation behind zero-knowledge

- Take any reasonably complex protocol
- What happens if the participants misbehave?
  - ⋆ Chaos and havoc! :-(
  - ⋆ Think of an electronic payment protocol . . .
- Need to enforce correct behavior
- How?

Idea how to solve

- Participants prove that they behave correctly
- After every message, verify the proof
- Privacy: the proof must not reveal any extra knowledge on the secrets of a participant to another one

Traditional proofs

- ... can actually reveal much more than just validity of the assertion
- ... at least leaves the verifier with the ability to present the same proof to others and convince them of the assertion
- How to avoid this? Use zero-knowledge proofs!

General problem statement

- Let L be some language (set of words), let x be an (encrypted) value

- How to prove that $x \in L$ without giving out any additional knowledge?
  $\star$ x is positive? x is a full square? x prime? x is a private key, corresponding to your public key g?

- Generally: How to prove that I know an x such that $x \in L$

- *Bad solution*: Send x to verifier. Verifier sees x and can test that $x \in L$; but this gives away more knowledge than is necessary.

Preliminaries: complexity class $\mathcal{P}$

L is in $\mathcal{P}$ iff $\exists$ Turing Machine A such that for each x

- A accepts iff $x \in L$
- A runs in time polynomial in $|x|$

Preliminaries: complexity class $\mathcal{BPP}$

- Randomizing: algorithm can flip points upon request

- Probabilistic polynomial time algorithm: can flip coins and runs in polynomial time

- L is in $\mathcal{BPP}$ iff $\exists$ Turing Machine A such that for each x
  $\star$ if $x \in L$ A accepts with prob. $\geq \frac{2}{3}$
  $\star$ if $x \notin L$ A accepts with prob. $\leq \frac{1}{3}$
  $\star$ A is PPT

Preliminaries: complexity class $\mathcal{NP}$

- Contains problems with ”classical” proofs

- L is in $\mathcal{NP}$ iff $\exists$ efficient (polynomial or probabilistic polynomial time) proof-verification algorithm (called verifier):
  $\star$ *Completeness*: For every valid assertion, $\exists$ a proof (*NP*-witness) that the verifier will accept.
  $\star$ *Soundness*: For every invalid assertion, no ”proof” can make the verifier accept.

- It is known that $\mathcal{P} \subseteq \mathcal{NP}$ and $\mathcal{P} \subseteq \mathcal{BPP}$. Containments are believed to be strict.

Interactive proofs (I)

- Serve the same purpose as classical proofs – to convince a verifier with limited computational power that some assertion is true

- We assume that prover is computationally unbounded, verifier's computation time must be polynomial (in interactive *arguments*, prover is also bounded)

- After the parties exchange messages for some number of rounds, the verifier decides whether to accept or reject

- Both prover and verifier may be randomized

Interactive proofs (II)

- *Completeness*: For every valid assertion, there is a prover strategy that will make the verifier accept with high probability.

- *Soundness*: For every invalid assertion, the verifier will reject with high probability, no matter what strategy the prover follows.

  $\star$ Probabilities are taken over the coin tosses of P, V

- Let $\mathcal{IP}$ be the set of languages that have interactive proofs

- $\mathcal{IP}$ is much larger than $\mathcal{NP}$

Example: Graph Non-isomorphism

- Graph Isomorphism is in $\mathcal{NP}$. Efficient proof that two graphs are isomorphic is an isomorphism between them.

- It is not known whether $GNI \in \mathcal{NP}$

- We will show that $GNI \in IP$

Protocol 1: Interactive proof for GNI

Input: Graphs $G_0 = (V_0; E_0)$ and $G_1 = (V_1; E_1)$

1. V : Uniformly select $b \in \{0; 1\}$. Uniformly select a permutation π on V_b. Let $H = \pi(G_b)$. Send H to P.

2. P : If $G_0 \cong H$, let $c = 0$. Else let $c = 1$. Send c to V.

3. V : If $c = b$, accept. Otherwise, reject.

Correctness of IP system for GNI

- When $(G_0, G_1) \in GNI$):
  $\star$ P can distinguish isomorphic copies of graph G_0 from isomorphic copies of G_1; then V accepts with probability 1

- When $(G_0, G_1) \notin GNI$):
  $\star$ An isomorphic copy of G_0 is always an isomorphic copy of G_1. Thus the best strategy for P is to toss a coin, and hence the cheating probability is $(1/2)^k$.

Zero-knowledge proofs

- ZK proof is an interactive proof with a zero-knowledge property
- Verifier learns nothing from the interaction with the prover, other than the fact that the assertion being proven is true
- Intuition: whatever the verifier sees in the interaction with the prover is something it could have efficiently generated on its own

Simulator

- A probabilistic polynomial-time algorithm that ”simulates” the verifier’s view of the interaction with the prover

  ⋆ View is a concatenation of all the messages exchanged between the two parties, prefixed with all random coin tosses of verifier

- Simulator generates an output distribution that is ”close” to what the verifier sees when interacting with the prover (when the assertion being proven is true)

Interpretations of ”close”

- *Perfect zero-knowledge*: Requires that the distributions are identical.

- *Statistical zero-knowledge*: Requires that the distributions are statistically close, i.e. statistical distance between two distributions is negligible. Even omnipotent verifier cannot distinguish them.

- *Computational zero-knowledge*: Requires that the distributions cannot be distinguished by any PPT algorithm.

Complexity classification

The classes of languages that have computational/statistical/perfect zero-knowledge proofs:

$$\mathcal{BPP} \subseteq_{\text{Believed that } \neq} \mathcal{PZK} \subseteq \mathcal{SZK} \subseteq_{\text{Believed that } \neq} \mathcal{CZK} = \mathcal{IP}$$

$\mathcal{BPP} \subseteq \mathcal{PZK}$: Trivial, uses no interaction. Verifier can verify by himself whether $x \in L$.

Honest Verifier ZK

- A party is honest/nonmalicious when he follows the protocol (though tries to deduce new information from it)

- (P, V) is honest verifier ZK if it is ZK with respect to honest V.

- No cheating strategies are considered

Example: Protocol 1 is HVZK

- Protocol 1 is not ZK: V can submit an arbitrary graph H not necessarily isomorphic to G_0 or to G_1 and thus get to know additional information

- What can V learn if he follows the protocol?

- Intuition: The only message from P to V is c. If graphs are non-isomorphic, then always c equals to b, which V already knows (since he chooses b himself).

Simulator for GNI Proof System

Input: Graphs $G_0 = (V_0; E_0)$ and $G_1 = (V_1; E_1)$

1. Uniformly select $b \in \{0; 1\}$. Uniformly select a permutation π on V_b. Let $H = \pi(G_b)$.

2. Let $c = b$.

3. Output $(b; H; c; \pi)$

- Output distribution of the simulator is identical to the verifier's view of the interaction. Thus Protocol 1 is perfect HVZK.

ZK vs HVZK

- For every language in $\mathcal{IP}$ there exists constant-round ZK protocol

- ZK protocols require more than three rounds unless the underlying language is trivial (in $\mathcal{BPP}$).

- 2-round HVZK protocol exists for every language in $\mathcal{SZK}$. HVZK is sufficient in many applications.

- There exist efficient transformation methods for turning certain classes of HVZK protocols into ZK ones.

$\mathcal{NP} \in \mathcal{CZK}$

- To show that there are CZK proofs for every $\mathcal{NP}$-language, it is sufficient to show a proof for one concrete $\mathcal{NP}$-complete language

- A graph G is 3-colorable when there exists an coloring of the vertices of G with 3 colors so that for no edge, the vertices connected to this edge are colored with the same color

- 3COL: the set all 3-colorable graphs. Language 3COL is $\mathcal{NP}$-complete.

CZK proof for Graph 3-Colorability

Common Input: A graph G(V; E). Suppose that $V \equiv \{1, \ldots, n\}$ for $n := |V|$. P knows a 3-coloring ϕ: $V \rightarrow \{1, 2, 3\}$. The following 4 steps are repeated $|E|^2$ times

1. P : Select uniformly a permutation π over $\{1, 2, 3\}$. For $i = 1$ to n, send V an encrypted (using a probabilistic public-key cryptosystem) value $\pi(\phi(i))$. For each vertex use different public key.

2. V : Select uniformly an edge $e = (i, j) \in E$ and send it to P.

3. P : Send to V the decryption keys to the i-th and j-th values.

4. V : Check whether or not the decrypted values are different elements of $\{1, 2, 3\}$ and whether or not they match the encryptions received in Step 1.

Correctness of the protocol for 3COL

- If P knows the corresponding 3-coloring, V will never detect an incorrectly colored edge. Thus, V will accept with probability 1

- If G is not 3-colorable then $\pi(\phi(i)) = \pi(\phi(j))$ in all steps with probability $\geq |E|^{-1}$. After $|E|^2$ steps the probability that V will accept is exponentially small

Theorem

Every language L in **NP** has a computational zero-knowledge interactive proof. Furthermore, the prescribed prover strategy can be implemented in probabilistic polynomial-time, provided it is given as auxiliary-input an **NP**-witness for membership of the common input in L.

Application: forcing proper behaviour

- U has a secret and is supposed to take some action depending on its secret
- U's legal action is determined as a polynomial-time function of its secret and the public information
- U's claim to having taken the correct action is an $\mathcal{NP}$-assertion. U's secret is an $\mathcal{NP}$-witness to its validity
- Theorem implies that U is able to give a zero-knowledge proof of his correct behaviour

Application: identification (I)

- Alice wonts to be able to identify herself repeatedly to Bob
- Common solution: Alice generates a password, Bob stores it. If Alice wants to identify herself, she sends password to Bob who checks wether it is correct
- A problem: Eve can impersonate Bob and obtain Alice’s Password

Application: identification (II)

- Solution: use zero-knowledge
- Alice generates a true statement S, for which only she knows the proof. Bob stores the statement.
- To identify herself, Alice gives Bob a zero-knowledge proof of S.
- Eve is not be able to learn the proof for S.