Zero-Knowledge Proofs and Sets

Overview

Zero-Knowledge Proofs: intro

Zero-Knowledge Sets: intro

Implementation of a ZK set

Extensions/Open Questions

Abstractly, a ZK proof involves:

- a prover and a verifier
- the prover wants to convince the verifier of statement X (with high probability)
- the prover does not want to reveal how to actually prove statement X

Huh?

Where is this useful:

- anonymity
- remote maintenance of information
- authentication
- verification over net

Note: precisely ZK proofs don't involve perfect proof.

We can thus prove with whatever confidence verifier wants: • after k distinct tests, probability of success for imposter is • e.g. have a real person pass with probability • basic idea is to prove X with arbitrarily high probability

Example: finding square roots of numbers mod p

Take some number r. Say n = r^2 mod p.

E.g. r = 5; p = 6; r^2 = 25; n = r^2 mod p = 1.

r is the square root of n mod p.
Zero-Knowledge Proofs

Interesting feature of \( r \) and \( n = r^2 \mod p \):

- \( r \) is easy to compute given \( r \), \( n \) is easy to compute
- \( n \) is non-trivial to compute given only \( n \), \( r \) is non-trivial to compute
- If \( p, r \) are very large, finding \( r \) from \( n \) is practically impossible

This is an example of a one-way function:

If \( r \) are very large, finding \( r \) from \( n \) is practically impossible
- no efficient (poly time) algorithm known

Using the square root mod problem for fun and profit:

Zero-Knowledge Proofs

Using the square roots mod \( p \) problem for fun and profit:

We can use this for basic authentication of identity

\( \text{prover} P \) wants to prove he is \( P \) to \( \text{verifier} V \)

- initially, \( P \) gives \( V \) a large number \( n \)
  - \( n \) has a square root mod \( p \), \( r \) that only \( P \) knows

- by receiving \( r \), \( V \) can check \( n = r^2 \mod p \)
- \( V \) knows \( P \) is \( P \) - impostor couldn't have guessed \( r \)

However, we'd prefer not to transmit \( r \) publicly ...

\( P \) can prove with high probability that he knows \( r \) - without giving it away!

First \( P \) gives \( V \) the value \( n \).

Then:

- \( P \) chooses random number \( m \), sends \( V \) the value \( x = m^2 \mod p \)
- \( V \) sends \( P \) random bit \( b \in \{ 0, 1 \} \)
- \( P \) sends \( V \) the value \( y = mr^b \)
- \( V \) tests if \( y^2 = x^{bn} \mod p \)
  - since \( y^2 = m^2 (r^b)^2 = xn^b \mod p \)

We would like our proof system to be:

1. complete - \( P \) can give correct answer every time
2. sound - \( P \) cannot lie, or an imposter can be caught
3. zero-knowledge - an eavesdropper can't find "secret" info from public info

**Completeness:**

- \( P \) can successfully complete this for both \( b = 0 \) and \( 1 \)

- \( P \) and \( V \) need to know both \( m \) and \( n \) - thus know \( r \)

**Soundness:**

- if \( b = 0 \) \( P \) can succeed - just choose \( m \), send \( x \), \( y = m \)

- if \( b = 1 \) \( P \) chooses \( m \), sends \( x = m^2 n \) and \( y = m \)

- correct answer is reliable with probability \( \frac{1}{2} \)

**Zero-knowledge:**

- can eavesdropper figure out \( r \)?

Answer: no

- eavesdropper sees either \( m \) and \( x \), or \( mr \) and \( x \)

- neither is enough to find out \( r \)

- by repeating test \( n \) times, chance of imposter's success becomes about \( \frac{1}{2^n} \).
Overview

"Zero Knowledge Sets" - S. Micali, M. Rabin, J. Killian, FOCS 2003

Goal: an efficient representation of zero-knowledge (ZK) sets

What characterizes ZK sets?

- membership can be proven/disproven without revealing other evidence about set:
  - cardinality,
  - other members/nonmembers of set, etc.

Ideally, we'd like to do this efficiently (poly time)

Note: seems tough to prove non-membership without giving this away ...

---

EDB Verifi cation Phase

P provides commitment

Basic procedure:

P receives D and public random string T

P computes two keys:
- \( PK \) (public)
- \( SK \) (private)

Example:

\( D = \{(0,1), (10,0), \ldots\} \)

T: 010101111...

\( PK: 010101101101... \)

\( SK: 010... \)

---

Note: seems tough to prove non-membership without giving this away

- directly: we'd have to do this secretly (by trial)
- other members/nonmembers of set, etc.
- cardinality
- other evidence

Memberships can be proven/disproven without revealing other evidence

---

Zero Knowledge Sets - S. Micali
M. Rabin
FOCS 2003

---
EDB Proof Phase

Say $V$ gives $P$ a string $x$. $P$ finds $y = D(x)$ (may be π).

Using $SK$, $P$ produces a proof $x$ of $D(x) = y$.

$V$ runs algorithm on $x$, $PK$, and $T$.

$V$ concludes proof is valid or invalid.

$P V: 010101111...$.
Our Proof System Should Be:

Complete - for any EDB, any value of \( x \) can be proven.

Sound - PK commits prover to partial function \( D \), no one can, in polytime, find \( x, y, z \) where \( y \neq z \) and prove \( D(x) = y \) and \( D(x) = z \). This ensures prover cannot lie, or imposter cannot forge result.

Zero-knowledge - say \( V \) sees a commitment to EDB \( D \) and a sequence of proofs for \( x_1; x_2; \ldots \). Then \( V \) queries trusted party about \( x_1; x_2; \ldots \) and only receives values in response. Knowledge obtained by both processes should be identical.

---

**Pedersen's Commitment Scheme and hash function**

We will now go over these...

Pedersen's scheme is a commitment scheme and hash function.

**Pedersen's Hash Function**

Pedersen's commitment scheme yields a good hash function \( H \): 

\[
H(a, b) = g^a h^b \mod p
\]

It is very difficult to find two \((a, b)\) that hash to the same value with this function.

Pedersen's commitment scheme relies on "Discrete Logarithm Assumption".

---

**Trees**

A good scheme: parties \( P \) and \( V \) share random string \( T \). For \( P \) to commit:

- \( P \) picks random \( r \), outputs \( c = g^m h^r \mod p \).
- \( V \) checks \( c, r \) using \( m, T \). For verification:

- \( P \) publicizes input \( c, r \).
- \( V \) checks if \( c = g^m h^r \mod p \). It is very very difficult to find two \( m \) that produce same \( c \) relies on "Discrete Logarithm Assumption".

Common approach to commitment: \( T \) is a public quadruple \((p, q, g, h)\):

- \( p, q \) prime, \( q \mid p - 1 \)
- \( Z_p \) is group of integers mod \( p \)
- \( Z_q \) is a cyclic subgroup of \( Z_p \) with \( q \) elements
- \( g, h \) are generators of \( Z_q \)

To commit, \( P \) picks random \( r \), outputs \( c = g^m h^r \mod p \).

To verify, \( V \) gets \( c, r \) checks if \( c = g^m h^r \mod p \).

We can thus trust that given \((c, r)\) it will be tough to find another set of \( m \) that Hash to the same value with this function.

---

**Common approach to commitment:**

- \( P \) is given input \( m \).
- \( P \) returns commitment string \( c \) and keeps secret proof \( r \).
- \( V \) checks \( c, r \) using \( m \) and \( T \).

We will try to calculate a commitment for our ZK set.

We will now go over these...

We can trust that given \((c, r)\) it will be tough to find another set of \( m \) that Hash to the same value with this function.

---

**Commitment Schemes**

We will now go over these...

We can trust that given \((c, r)\) it will be tough to find another set of \( m \) that Hash to the same value with this function.
We will use Merkle trees to store our ZK set.

How does a Merkle tree work?

leaves may store data items (values in database).

find hash function \( H \) mapping two items \( x, y \) to value \( z \).

for node with children \( a \) and \( b \), store \( H(a, b) \).

Value at root node is dependent on values in leaves.

To prove that node \( x \) stores \( a \):

look at ancestors of \( x \) all the way up to root.

using values stored in nodes' siblings, calculate ancestors.

compare result at root to commitment.

\[ f \]
\[ b \]
\[ c = H(a, b) \]
\[ d \]
\[ e = H(c, d) \]
\[ g = H(e, f) \]
\[ a \]

Verified!
Merkle Trees

Path from root to $x$ with siblings represents authentication path

Given $M$'s root value, can we compute two different authentication paths to prove $y$ and $z$ both stored in $x$?

Choose a good hash function (e.g. Pedersen's), and this is infeasible.

ZK EDB - Commitment

What we want to do:

- create a commitment using Merkle trees, as described above

Do this in recursive, bottom-up fashion through the tree:

- calculate commitments for each node using Pedersen's commitment scheme
- for parent of nodes $a$ and $b$, store $H(a; b)$ in $p$
- store 0 in empty siblings as needed
- calculate commitment for $p$ using Pedersen's commitment scheme

Start with hash function $H$ (Pedersen's hash function)

When we do:

ZK EDB - Commitment

Final commitment given to verifier by prover is commitment of root.

ZK EDB - Verification

Now, say we want to prove $(x; y)$ is NOT a key in database.

- give values and commitments for each node from leaf to root, along with its sibling

Verifier checks that root commitment matches original commitment

Verifier confirms, using hash function $H$, that prover did not "cheat"

If we want to prove $x$ is NOT a key in database, its index

ZK EDB - Commitment

Choose a good hash function (e.g. Pedersen's), and this is infeasible.

Given $y$'s root value, can we compute two different authentication paths from root to node $y$?

Part from root to $y$ with siblings represents authentication path
We apply clever technique to "fake" nodes in Merkle tree to set up commitment so that we can change it for empty leaves.

To prove \( D(x) = ? \) create a fake subtree containing node \( G(x) \) containing 0, fill in values of parents as needed, and give path from \( G(x) \) to root.

The construction described can be enhanced so:

- one can not show whether \( x \in D \) and exactly what \( D(x) \) is (anonymity)
- prove portions of info in \( D(x) \), only certain people may read portions of \( D(x) \), or portions of \( D(x) \) can be distributed in nature
- database can be updated at low cost

Can we consider other ZK operations/data structures as well?

Can we handle multiple provers?

Can a ZK set be updated at low cost?

Open Questions

Additional Notes

Can a ZK set be updated at low cost?
Can we handle multiple provers?
Can we consider other ZK operations/data structures as well?

Additional Notes

Can a ZK set be updated at low cost?
Can we handle multiple provers?
Can we consider other ZK operations/data structures as well?