CSE 410 Fall 2025
Privacy-Enhancing Technologies

Marina Blanton

Department of Computer Science and Engineering
University at Buffalo

Lecture 18: Zero-Knowledge Proofs of Knowledge

Zero-Knowledge Proof of Knowledge

A zero-knowledge proof of knowledge (ZKPK) is a cryptographic protocol between a prover and a verifier

- it allows the prover to prove that a certain statement is true

Informally, a ZKPK has to satisfy two properties

- proof of knowledge: the verifier is convinced that the statement of the proof holds
- zero knowledge: nothing beyond the validity of the statement is revealed

Zero-Knowledge Proof of Knowledge

A ZKPK uses one or more public inputs (known to the prover and the verifier)

It also has one or more private inputs (known to the prover)

A ZKPK is written as

$$\mathsf{ZKPK}\{(A_1, A_2, \ldots), (a_1, a_2, \ldots) : \mathsf{R}\}$$

- the first list $A_1, \ldots$ are public inputs
- the second list $a_1, \ldots$ are private inputs
- R is the statement (a function of all inputs) to be proved

Zero-Knowledge Proof of Knowledge

We start with a simple example

Suppose Alice wants to prove that she knows the discrete logarithm of h to the base g (in the setting where the discrete logarithm is hard)

- the public inputs are g, h
- Alice's private input is x
- Alice proves

$$\mathsf{ZKPK}\{(g,h),(x) : h = g^x\}$$

Zero-Knowledge Proof of Knowledge

A zero-knowledge proof is a challenge-response protocol

- the prover commits to some values
- the verifier sends a challenge
- the prover creates a response using the challenge

It is important (for security reasons) that the challenge is unpredictable to the prover

Discrete Logarithm Example

Setup:

- the prover and verifier share g, h; g is the generator of a group of order q
- the prover knows x such that $g^x = h$

ZK proof:

- the prover chooses random $v \in \mathbb{Z}_q$, computes $t = g^v$ and sends t to the verifier
- the verifier chooses $c \in \mathbb{Z}_q$ and sends it to the prover
- the prover computes response $r = (v - cx) \bmod q$ and sends it to the verifier
- the verifier computes $t' = g^r y^c$ and checks if it's equal to t

Discrete Logarithm Example

Here, t is the commitment, c is the challenge, and r is the response

$$t' = g^r y^c = g^{v-cx} y^c = g^v = t$$

It is possible to turn the protocol into a non-interactive proof using a Fiat-Shamir heuristic:

- the challenge is computed by the prover as a hash of all other values
- for the discrete logarithm ZKP, $c = H(g, h, t)$
- the verifier now additionally checks that $c = H(g, h, t')$

Discrete Logarithm Example

Full revised version of $\mathsf{ZKPK}\{(g,h),(x) : h = g^x\}$:

- the prover chooses random $v \in \mathbb{Z}_q$ and computes $t = g^v$
- the prover computes $c = H(g,h,t)$
- the prover computes $r = (v - cx) \bmod q$ and sends c and r to the verifier
- the verifier computes $t' = g^r h^c$ and checks that $c = H(g,h,t')$

This interaction does not disclose new information about x

It is also not feasible for an attacker without knowledge of x to pass the verification process

ZK Proof of Disjunction

Now let's do a ZK proof of disjunction: the prover knows the discrete logarithm of h_1 to the base g_1 or h_2 to the base g_2

$$\mathsf{ZKPK}\{(g_1, g_2, h_1, h_2), (x_1 \text{ or } x_2) : h_1 = g_1^{x_1} \vee h_2 = g_1^{x_2}\}$$

ZK Proof of Disjunction

Now let's do a ZK proof of disjunction: the prover knows the discrete logarithm of h_1 to the base g_1 or h_2 to the base g_2

$$\mathsf{ZKPK}\{(g_1, g_2, h_1, h_2), (x_1 \text{ or } x_2) : h_1 = g_1^{x_1} \vee h_2 = g_1^{x_2}\}$$

Suppose the prover knows x_2 and computes:

- choose random $v_1, v_2, w \in \mathbb{Z}_q$ and compute $t_1 = h_1^w g_1^{v_1}$ and $t_2 = g_2^{v_2}$
- compute $c = H(g_1, g_2, h_1, h_2, t_1, t_2) \bmod q$
- set $c_1 = w$, $c_2 = c - c_1 \bmod q$, $r_1 = v_1$, and $r_2 = v_2 - c_2 x_2 \bmod q$

The proof is (c_1, c_2, r_1, r_2)

ZK Proof of Disjunction

The verifier uses the proof (c_1, c_2, r_1, r_2) to compute

$$t_1' = g_1^{r_1} h_1^{c_1} \text{ and } t_2' = g_2^{r_2} h_2^{c_2}$$

and check that

$$c_1 + c_2 \stackrel{?}{=} H(g_1, g_2, h_1, h_2, t_1', t_2') \pmod{q}$$

ZK Proof of Linear Combination

One last example is that of a linear combination:

the discrete logarithms of $h_1 = g_1^{x_1}$ and $h_2 = g_2^{x_2}$ to the bases g_1 and g_2, respectively, satisfy $a_1 x_1 + a_2 x_2 = b \pmod{q}$

$$\mathsf{ZKPK}\{(g_1, g_2, h_1, h_2, a_1, a_2, b), (x_1, x_2) : h_1 = g^{x_1} \wedge h_2 = g^{x_2} \wedge a_1 x_1 + a_2 x_2 = b \pmod{q}\}$$

ZK Proof of Linear Combination

One last example is that of a linear combination:

the discrete logarithms of $h_1 = g_1^{x_1}$ and $h_2 = g_2^{x_2}$ to the bases g_1 and g_2, respectively, satisfy $a_1 x_1 + a_2 x_2 = b \pmod{q}$

$$\mathsf{ZKPK}\{(g_1, g_2, h_1, h_2, a_1, a_2, b), (x_1, x_2) : h_1 = g^{x_1} \wedge h_2 = g^{x_2} \wedge a_1 x_1 + a_2 x_2 = b \pmod{q}\}$$

The prover performs:

- choose random $v_1, v_2 \in \mathbb{Z}_q$ subject to $a_1 v_1 + a_2 v_2 = 0 \bmod q$ and compute $t_1 = g_1^{v_1}$ and $t_2 = g_2^{v_2}$
- compute $c = H(g_1, g_2, h_1, h_2, a_1, a_2, b, t_1, t_2) \bmod q$
- compute $r_1 = v_1 - cx_1 \bmod q$ and $r_2 = v_2 - cx_2 \bmod q$

ZKPKs

The resulting proof is (c, r_1, r_2) and the verifier computes

$$t_1' = g_1^{r_1} h_1^c \text{ and } t_2' = g_2^{r_2} h_2^c$$

and then check that

$$c \stackrel{?}{=} H(g_1, g_2, h_1, h_2, a_1, a_2, b, t_1', t_2')$$
$$\text{and } a_1 r_1 + a_2 r_2 \stackrel{?}{=} -cb \pmod{q}$$

ZKPKs

The resulting proof is (c, r_1, r_2) and the verifier computes

$$t_1' = g_1^{r_1} h_1^c \text{ and } t_2' = g_2^{r_2} h_2^c$$

and then check that

$$c \stackrel{?}{=} H(g_1, g_2, h_1, h_2, a_1, a_2, b, t_1', t_2')$$
$$\text{and } a_1 r_1 + a_2 r_2 \stackrel{?}{=} -cb \pmod{q}$$

We can prove more complex statements by combining these building blocks

- equality, conjunction, disjunction
- equations, comparisons, ...

Anonymous Credentials

Anonymous credentials is a cryptographic token on one or more attributes issued to a user by an authority

- it grants the user access or other privileges based on their attributes
- users prove their right or privilege without disclosing other information
- the service does not know anything about the user besides the access outcome
- multiple uses of the same credential cannot be linked together
- credentials cannot be forged by an attacker

Commitments

Before we continue with anonymous credentials, we need to introduce the concept of commitment

Given one or more messages m, one creates and announces commitment $\mathsf{com}(m)$

The commitment can later be opened to disclose m

Commitments

Before we continue with anonymous credentials, we need to introduce the concept of commitment

Given one or more messages m, one creates and announces commitment $\mathsf{com}(m)$

The commitment can later be opened to disclose m

A commitment $\mathsf{com}(m)$ comes with two properties:

- a commitment is hiding: $\mathsf{com}(m)$ does not disclose information about message m
- a commitment is binding: it is not feasible to open $\mathsf{com}(m)$ to a message different from m

Commitments

A popular type of commitment is called Pedersen commitment

The setup assumes generators g, h of a group of order q where the discrete logarithm is hard

To commit to message $m \in \mathbb{Z}_q$:

- generate a random $r \in \mathbb{Z}_q$
- compute $\mathsf{com}(m) = g^m h^r$
- we write $\mathsf{com}(m; r)$ to explicitly represent randomness

Commitments

To open Pedersen commitment $\mathsf{com}(m; r)$:

- disclose m and r
- the verifier checks that $g^m h^r$ matches $\mathsf{com}(m; r)$

The construction information-theoretically hides m

It is computationally infeasible to find different m', r' which will match the commitment $\mathsf{com}(m; r)$

To commit to k messages at once

- the setup consists of generators $g_1, g_2, \ldots, g_k, h$
- a commitment is $\mathsf{com}(m_1, \ldots, m_k; r) = g_1^{m_1} \cdots g_k^{m_k} h^r$

Commitments

Another popular type of commitment uses a hash function H

To commit to message $m \in \{0,1\}^*$:

- generate a random $r \in \{0,1\}^\kappa$, where κ is a security parameter
- compute $\mathsf{com}(m,r) = H(m;r)$

To open commitment $\mathsf{com}(m,r)$:

- disclose m and r
- the verifier checks that $H(m;r)$ matches $\mathsf{com}(m;r)$

The hiding and binding properties hold if H is modeled as a random one-way function (random oracle)

Anonymous Credentials

An anonymous credential is issued on a number of user attributes $a_1, a_2, \ldots$

Let a credential be encoded as follows

- it is a signature or a MAC issued by an authority
- the attributes are encorporated as $g_1^{a_1} g_2^{a_1} \cdots g_k^{m_k}$
- randomness is typically encorporated to protect the values
- credential is often randomizable to make its uses unlinkable

The binding property of this representation is important for achieving security

Anonymous Credentials

We now can use discrete logarithm based ZK proofs to convince a service that a credential matches their policy

Example 1: library access

- to access a digital library, one need to demonstrate possession of a valid membership
- i.e., the policy is that the membership expiration date is in the future
- the corresponding ZK proof can look as follows

$$\mathsf{ZKPK}\{(\mathsf{cred}, \mathsf{com}, g, h), (e, r) : com = g^e h^r \wedge e \geq \mathsf{currentdate} \\ \wedge\ e \text{ is part of } \mathsf{cred}\}$$

We also check the credential cred for validity

Anonymous Credentials

Example 2: scholarship application

- to qualify for a scholarship, a student must be 18 or older, full-time, and have GPA ≥ 3.0
- this can be shown using one or more credentials issued by different entities
- e.g., the date of birth is certified by one source
- student current full-time status and their current GPA is certified by a higher ed institution
- what is the full list of the attributes needed?

E-Cash

This mechanism allows us to create anonymous e-cash with detection of double spending

Consider the following design

- a bank issues a digital coin of certain denomination
- the coin is the bank's signature on (id, s, t)
- here, id is the user's identity
- s is the coin's serial number not known to the bank
- t is a randomizing value not known to the bank

E-Cash

This implies that the coin is issued using a ZK proof

- a user creates a commitment $\mathsf{com}(id, s, t; r) = g_1^{id} g_2^s g_3^t h^r$
- the user proves the knowledge of s and t and that the correct id was included
- the user pays the coin's denomination amount
- the bank issues a coin using $\mathsf{com}(id, s, t; r)$

E-Cash

When the user wants to spend the coin at a merchant:

- the user supplies a commitment to id, s, and t
- the user proves in ZK that she has a signature from the bank on those values
- the merchant chooses a random R and gives it to the user
- the user evaluates the double-spending equation $D = id + R \cdot t \pmod{q}$
- the user reveals the serial number s and D to the merchant
- the user proves in ZK correctness of s and D

Double-Spending in E-Cash

If a coin is used once, the revealed D does not disclose any information

If a coin is spent twice, the user's identity can be recovered

- the linear equation $D = id + R \cdot t$ is evaluated on two points
- that permits the recovery of the polynomial and thus id

If two merchants request money with the same s but different Rs, the bank recovers id and charges the user

If two requests use the same s and R, the merchant is at fault (and the request is denied)

Cryptocurrencies

Original cryptocurrencies designs were not fully anonymous

- transactions are associated with a (public) key
- one's public key acts as a pseudonym
- different transactions using the same key are linkable

Consequently, cryptocurrencies that provide anonymity guarantees were proposed

Zerocoin and Zerocash

Zerocoin was published in 2013

- similar to e-cash, it provides transaction unlinkability
- unlike e-cash, it does not rely on signatures to validate coins
- also unlike e-cash, it does not require a central bank to prevent double spending
- instead, it authenticates coins by proving in ZK that they belong to a public list of valid coins
- the list can be maintained on the block chain

Zerocoin and Zerocash

Zerocoin had two primary disadvantages

- performance: the cost of verifying cryptographic proofs was orders of magnitude higher than Bitcoin operations
- functionality: it lacked features required of full-fledged anonymous payments
  - this includes payments of exact values instead of fixed denominations
  - this also includes one user paying directly another user
  - lastly, Zerocoin unlinked payments from addresses, but did not hide other metadata

Zerocash followed in 2014 with improved features

Zerocoin and Zerocash

Zerocash eliminates the drawbacks of Zerocoin

- it relies on zk-SNARKs – a special type of ZK proofs
- zk-SNARKs are compact and non-interactive
- they are designed to make verification efficient
- their use reduces resource utilization by 98% compared to Zerocoin

Zerocoin and Zerocash

Zerocash also eliminates functionality drawbacks

- it allows for anonymous transactions of variable amounts
- it hides transaction amounts and the value of coins used by users
- it permits for payments to be made directly to a user's address without interaction

Zerocash was recognized as having a lasting impact (Test of Time Award) at S&P 2024

Zerocash

Zerocash uses the following ideas:

- a user deposits 1 money unit and its coin commitment is added to the ledger
- when proving in ZK that one's coin (serial number) is on the list of commitments, an efficient tree structure is used
- to enable transfers to a user, serial numbers are derived using the target address instead of being chosen at random
- coins are spent using pour operations: take a set of input coins and pour their value into a set of fresh output coins

Zerocash was implemented in Zcash

Summary

Zero-knowledge proofs of knowledge are cryptographic protocols with exciting properties

- they permit proving statements without disclosing information beyond the validity of the statement itself

ZKPKs can be utilized in a variety of domains to improve privacy protection of users