Commitment schemes

Distributed Lab

August 20, 2024

Plan

Commitments Overview

Commitment Definition

Definition

A cryptographic commitment scheme allows one party to commit to a chosen statement without revealing the statement itself. The commitment can be revealed in full or in part at a later time, ensuring the integrity and secrecy of the original statement until the moment of disclosure.

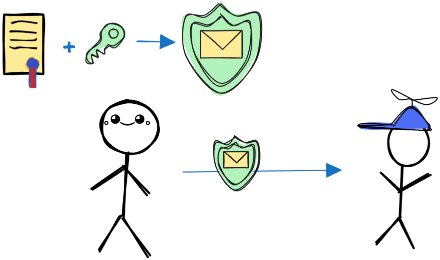


Figure: Overview of a commitment scheme

Commitment Definition

Definition

Commitment Scheme $\Pi_{\text{commitment}}$ is a tuple of three algorithms: $\Pi_{\text{commitment}} = (\text{Setup}, \text{Commit}, \text{Verify})$.

1. Setup (1^λ): returns public parameter *pp* for both comitter and verifier;
2. Commit (pp, m, r): returns a commitment c to the message m using public parameters *pp* and, optionally, a secret opening hit r;
3. Open (pp, c, m, r): verifies the opening of the commitment to the message m with an opening hit r.

Commitment Scheme Properties

Definition

1. *Hiding:* verifier should not learn any additional information about the message given only the commitment $\mathcal{C}$.
   1. *Perfect hiding*: adversary with any computation capability tries even forever cannot understand what you have hidden.
   2. *Computationally hiding*: we assume that the adversary have limited computational resources and cannot try forever to recover hidden value.
2. *Binding:* prover could not find another message m_1 and open the commitment $\mathcal{C}$ without revealing the commited message m.
   1. *Perfect binding*: adversary with any computation capability tries even forever cannot find another m_1 that would result to the same $\mathcal{C}$.
   2. *Computationally binding*: we assume that the adversary have limited computational resources and cannot try forever.

Note

Perfect hiding and perfect binding cannot be achived at the same time

Hash-based Commitments

Hash-based commitments

As the name implies, we are using a cryptographic hash function H in such scheme.

Definition

1. Prover selects a message m from a message space M which he wants to commit to: $m \leftarrow \mathbb{M}$
2. Prover samples random value r from a challange space C (usually called blinding factor) from $\mathbb{Z}$: $r \xleftarrow{R} \mathbb{C}$
3. Both values will be concatenated and hashed with the hash function H to produce the commitment: $\mathcal{C} = H(m \parallel r)$

Vector Commitments

Merkle Tree commitments

A naive approach for a vector commitment would be hash the whole vector. More sophisticated scheme uses divide-and-conquer approach by building a binary tree out of vector elements.

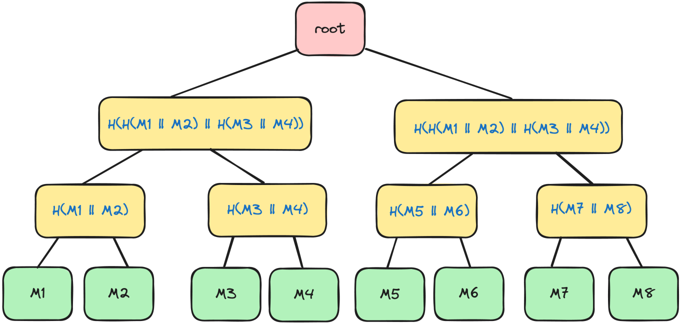


Figure: Merkle Tree structure

Merkle Tree Proof (MTP)

To prove the inclusion of element into the tree, a corresponding Merkle Branch is used. It allows to perform selective disclosure of the elements without revealing all of them at once.

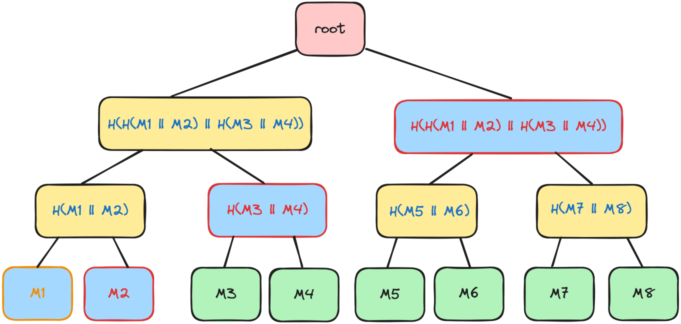


Figure: Merkle Tree inclusion proof branch

Pedersen Commitment

Pedersen commitments allow us to represent arbitrarily large vectors with a single elliptic curve point. Pedersen commitment uses a public group $\mathbb{G}$ of order q and two random public generators G and U: $U = [u]G$. Secret parameter u should be unknown to anyone, otherwise the *Binding* property of the commitment scheme will be violated.

Note: Transparent random points generation

User can pick the publicly known number (like x coordinate of group generator G), calculate $x_i = H(x \parallel i)$ and corresponding y_i. Check whether (x_i, y_i) is in the elliptic curve group. Repeat the process for sequential $i = 1, 2 \ldots$ until point (x_i, y_i) is in the elliptic curve group.

Pedersen Commitment

Definition

Pedersen commitment scheme algorithm:

1. Prover and Verifier agrees on G and U points in a elliptic curve point group $\mathbb{G}$, q is the order of the group.
2. Prover selects a value m to commit and a blinder factor r: $m \leftarrow \mathbb{Z}_q$, $r \xleftarrow{R} \mathbb{Z}_q$
3. Prover generates a commitment and sends it to the Verifier: $\mathcal{C} \leftarrow [m]G + [r]U$

During the opening stage, prover reveals (m, r) to the verifier.
To check the commitment, verifier computes: $\mathcal{C}_1 = [m]G + [r]U$.
If $\mathcal{C}_1 = \mathcal{C}$, prover has revealed the correct pair (m, r).

Pedersen Commitment

In case the discrete logarithm of U is leaked, the *binding* property can be violated by the *Prover*:

$$c = [m]G + [r]U = [m]G + [r \cdot u]G = [m + r \cdot u]G$$

For example, $(m + u, r - 1)$ will have the same commitment value:

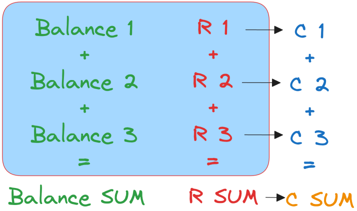
$$[m + u + (r - 1) \cdot u]G = [m + u - u + r \cdot u]G = [m + r \cdot u]G$$

Pedersen Commitment Aggregation

Pedersen commitment have some advantages compared to hash-based commitments. Additively homomorphic property allows to accumulate multiple commitments into one. Consider two pairs: $(m_1, r_1), (m_2, r_2)$.

$$\mathcal{C}_2 = [m_1]G + [r_1]U,$$
$$\mathcal{C}_2 = [m_2]G + [r_2]U,$$
$$\mathcal{C}_a = \mathcal{C}_1 + \mathcal{C}_2 = [m_1 + m_2]G + [r_1 + r_2]U$$

This works for any number of commitments, so we can encode as many points as we like in a single one.



Pedersen Vector Commitment

Suppose we have a set of random elliptic curve points $(G_1, \ldots, G_n)$ of cyclic group $\mathbb{G}$ (that nobody knows the discrete logarithm of), a vector $(m_1, m_2 \ldots m_n)$ and a random value r. We can do the following:

$$\mathcal{C} = m_1 \cdot [G_1] + m_2 \cdot [G_2] \ldots + m_n \cdot [G_n] + r \cdot [Q]$$

Since the *Prover* does not know the discrete logarithm of the generators, so he can only reveal $(v_1, \ldots, v_n)$ to produce $[C]$ later, they cannot produce another vector.
Prover can later open the commitment by revealing the vector $(m_1, m_2 \ldots m_n)$ and a blinding term r.

Thanks for your attention!