# Commitment Schemes

*August 15, 2024*

**Distributed Lab**

zkdl-camp.github.io

github.com/ZKDL-Camp

# 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.

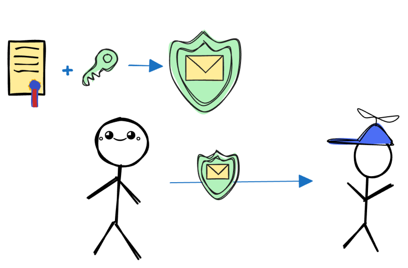



**Figure:** Overview of a commitment scheme

# Commitment Definition

***Definition***

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

1. $\text{Setup}(1^\lambda)$: returns public parameter pp for both comitter and verifier;
2. $\text{Commit}(\text{pp}, m)$: returns a commitment $c$ to the message $m$ using public parameters pp and, optionally, a secret opening hint $r$;
3. $\text{Open}(\text{pp}, c, m, r)$: verifies the opening of the commitment $c$ to the message $m$ with an opening hint $r$.

# Commitment Scheme Properties

**Definition**

1. *Hiding:* verifier should not learn any additional information about the message given only the commitment $c$.
   1.1. *Perfect hiding*: adversary with any computation capability tries even forever cannot understand what you have hidden.
   1.2. *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 $c$ without revealing the commited message $m$.
   2.1. *Perfect binding*: adversary with any computation capability tries even forever cannot find another $m_1$ that would result to the same $c$.
   2.2. *Computationally binding*: we assume that the adversary have limited computational resources and cannot try forever.

**Note:** Perfect hiding and perfect binding cannot be achived simultaneously.

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

# 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.

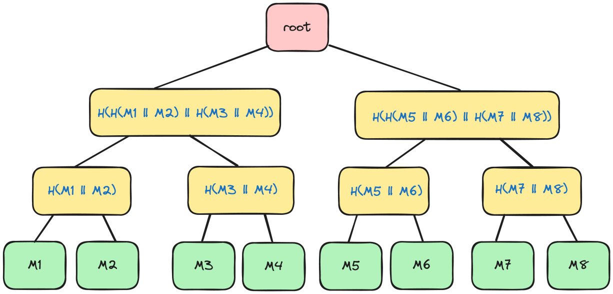



**Figure:** Merkle Tree structure

# 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.

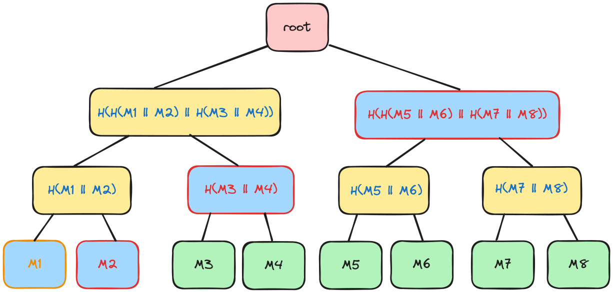



**Figure:** Merkle Tree inclusion proof branch

# 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: $c \leftarrow [m]G + [r]U$

During the opening stage, prover reveals $(m, r)$ to the verifier.

To check the commitment, verifier computes: $c_1 = [m]G + [r]U$.

If $c_1 = c$, 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:

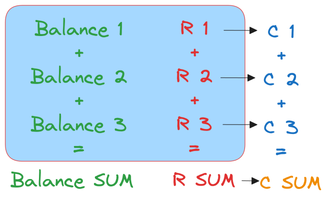
$$[m + u + (r - 1) \cdot u]G = [m + u - u + r \cdot u]G = [m + r \cdot u]G$$

# Pedersen Commitment Aggregation

Pedersen commitment has a significant advantage over hash-based commitments: the former are additively homomorphic and thus can be accumulated. Consider two pairs $(m_1, r_1), (m_2, r_2)$.

$$\begin{aligned} c_1 &= [m_1]G + [r_1]U, \\ c_2 &= [m_2]G + [r_2]U, \\ c_{\text{agg}} &= c_1 + c_2 = [m_1 + m_2]G + [r_1 + r_2]U. \end{aligned}$$

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:

$$c = m_1 \cdot [G_1] + m_2 \cdot [G_2] \ldots + m_n \cdot [G_n] + r \cdot [Q]$$

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 $r$.

# Polynomial commitment

# Polynomial Commitment

***Definition***

Polynomial commitment can be used to prove that the commited polynomial satisfies certain properties (passes through a certain point $(x, y)$), without revealing what the polynomial is. The commitment is generally succint, which means that it is much smaller than the polynomial it represents.

# KZG Commitment. Simplified example

Given the polynomial: $P(x) = x^3 - 15x^2 + 71x - 103$

Prove that $P(3) = 2$

$P(3) = 2 \rightarrow 3$ is a root of polynomial $P(x) - 2$

Proof:

$$Q(x) = \frac{P(x) - 2}{x - 3} = \frac{(x^3 - 15x^2 + 71x - 103) - 2}{x - 3} = x^2 - 12x + 35$$

Verify: $Q(x) \cdot (x - 3) = P(x) - 2$

# KZG Commitment

The KZG (Kate-Zaverucha-Goldberg) is a polynomial commitment scheme:

*One-time "Powers-of-tau" trusted setup stage*. During trusted setup a set of elliptic curve points is generated. Let $G$ be a generator point of some pairing-friendly elliptic curve group $\mathbb{G}$, $s$ some random value in the order of the $G$ point and $d$ be the maximum degree of the polynomials we want to commit to.

$$[\tau^0]G, [\tau^1]G, \ldots, [\tau^d]G$$

Parameter $\tau$ should be deleted after the ceremony. If it is revealed, the commitment scheme can be broken. This parameter is usually called the *toxic waste*.

# KZG Commitment

*Commit to polynomial*. Given the polynomial $p(x) = \sum_{i=0}^{d} p_i x^i$, compute the commitment $c = [p(\tau)]G$ using the trusted setup. Although the committer cannot compute $[p(\tau)]G$ directly since the value of $\tau$ is unknown, he can compute it using values $([\tau^0]G, [\tau^1]G, \ldots, [\tau^d]G)$.

# KZG Commitment

*Prove an evaluation.* Given evaluation $p(x_0) = y_0$ compute proof $q(\tau)$,

where $q(x) = \frac{p(x) - y_0}{x - x_0}$.

Polynomial $q$ is called "quotient polynomial" and only exists if and only if $p(x_0) = y_0$. The existance of this quotient polynomial serves as a proof of the evaluation.

# KZG Commitment

*Verify the proof*. Given a commitment $c = [p(\tau)]G$, an evaluation $p(x_0) = y_0$ and a proof $[q(\tau)G]$, we need to ensure that $q(\tau) \cdot (\tau - x_0) = p(\tau) - y_0$. This can be done using trusted setup without knowledge of $\tau$ using bilinear mapping:

$$e(q(\tau), [\tau]G_2 - [x_0]G_2) = e(c - [y_0]G_1, G_2)$$

Polynomial commiment schemes such as KZG are used in zero knowledge proof system to encode circuit constraints as a polynomial, so that verifier could check random points to ensure that the constraints are met.

# Thank you for your attention

♥

zkdl-camp.github.io

github.com/ZKDL-Camp