0.1 Commitments

Definition 0.1. A cryptographic commitment scheme allows one party to commit to a chosen statement (such as a value, vector, or polynomial) 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.

Before delving into the details, here is the intuition of cryptographic commitments.

Imagine putting a letter with some message into a box and locking it with your key. You then give that box to your friend, who cannot open it without the key. In this scenario, you have made a commitment to the message inside the box. You cannot change the content of the letter, as it is in your friend's possession. At the same time, your friend cannot access the letter since they do not have the key to unlock the box.

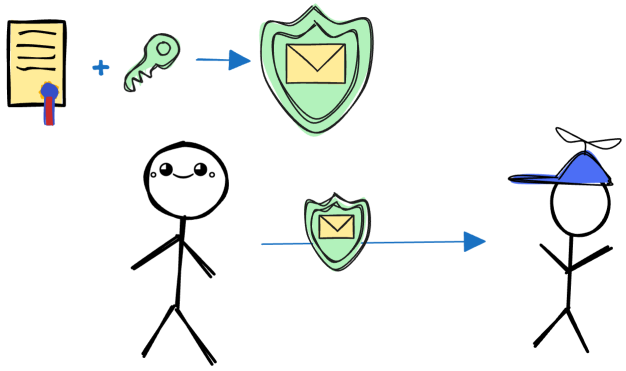

Figure 1: Commitment scheme

Definition 0.2 (Commitment Scheme)**.** Commitment Scheme $\Pi_{\text{commitment}}$ is a tuple of three algorithms: $\Pi_{\text{commitment}} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$.

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

Definition 0.3 (Commitment Scheme)**.** Properties of commitment schemes:

1. *Hiding:* verifier should not learn any information about the message given only the commitment c. To put it formally, we define a game:
   (a) Adversary chooses two messages m_1, m_2 and sends to the challenger.
   (b) Challenger chooses a random bit b, commits to both messages: $c_1 \leftarrow \mathsf{Commit}(pp, m_1)$, $c_2 \leftarrow \mathsf{Commit}(pp, m_2)$, and sends c_b to the adversary.
   (c) Adversary guesses a bit $\hat{b}$.

   We define the hiding advantage of a PPT adversary $\mathcal{A}$ as

$$\mathsf{HideAdv}[\mathcal{A}, \Pi_{\mathsf{commitment}}] := \left| \Pr[b = \hat{b}] - \frac{1}{2} \right| \tag{1}$$

   We say that the commitment scheme $\Pi_{\mathsf{commitment}}$ is *hiding* if for any adversary, the aforementioned advantage is negligible.
2. *Binding:* prover could not find another message m_1 and open the commitment c without revealing the commited message m. To put it formally, we define a game:
   (a) Adversary chooses five values: commitment c and two distinct pairs (m_0, r_0) and (m_1, r_1).
   (b) Adversary computes $b_j \leftarrow \mathsf{Open}(\mathsf{pp}, c, m_j, r_j)$.

   Define the advantage in the binding game as:

$$\mathsf{BindAdv}[\mathcal{A}, \Pi_{\mathsf{commitment}}] = \Pr[b_0 = b_1 \neq 0 \wedge m_0 \neq m_1] \tag{2}$$

   We say that the commitment scheme is binding if for any adversary, such advantage is negligible.

0.1.1 Hash-based commitments

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

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 (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: $c = H(m \parallel r)$

Commitment should be shared with a verifier. During the opening stage, prover reveals (m, r) to the *Verifier*. To check the commitment, verifier computes: $c_1 = H(m \parallel r)$.

If $c_1 = c$, prover has revealed the correct pair (m, r).

It should be noted that a cryptographic hash function aims to provide collision resistance, meaning that the probability two different messages will result in one output is negligible. Because the *Verifier* knows the hash function digest c before the *Prover* reveals m and r, the *Prover* would need to find a collision $H(m' \parallel r') = H(m \parallel r)$ to be able to convince the *Verifier* that m' value was committed.

However, due to the collision resistance, finding such m' and r' is computationally infeasible.

Which means the *Prover* won't be able to convince the *Verifier* that the commitment was done to another value providing a *binding* property.

A cryptographically secure hash function is a one-way function, which means that finding the hash preimage is almost as hard as bruteforcing all possibile input values. Given large challenge space, the probability of the *Verifier* of finding (m, r) such that $H(m, r) = c$ is negligible, which ensures *hiding* property of the commitment scheme.

0.1.2 Pedersen commitments

Pedersen commitments allow us to represent arbitrarily large vectors with a single elliptic curve point, while optionally hiding any information about the vector. 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. EC point U is chosen randomly using "Nothing-up-my-sleeve" to assure no one knows the discrete logarithm of a selected point.

Remark. Transparent random points generation
User can pick the publicly chosen random number (like a hash of project name, first numbers of π, etc), and hash that result to obtain another value. If that results in an x value that lies on the elliptic curve, use that as the random point and hash the (x, y) pair again (to obtain the next one, it needed). Otherwise, if the x-value does not land on the curve, increment x until it does. Because the committer is not generating the points, they don't know their discrete log.

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).

Remark. 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:

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

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: $(m_1, r_1), (m_2, r_2)$.

$$c_2 = [m_1]G + [r_1]U,$$

$$c_2 = [m_2]G + [r_2]U,$$
$$c_a = c_1 + c_2 = [m_1 + m_2]G + [r_1 + r_2]U$$

This works for any number of commitments, so we can encode as many points as we like in a single one. For example, if a set of balances is committed, the sum of any subset can be proven without revealing the exact value of each balance. This is achieved by disclosing the sum of the balances and the corresponding sum of the blinding factors.

0.1.3 Vector commitments

Vector commitment schemes allows to commit to a vector of values rather than a value and a blinding term.

Pedersen Vector Commitments

Suppose we have a set of random elliptic curve points $(G_1, \ldots, G_n)$ of cyclic group $\mathbb{G}$ (that we do not know the discrete logarithm of), a vector $(m_1, m_2 \ldots m_n)$ and a random value r. We can do the following:

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

Since the *Prover* does not know the discrete logarithm of the generators, they don't know the discrete logarithm of [C]. Hence, this scheme is binding: they 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.

Merkle Tree based Vector 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.

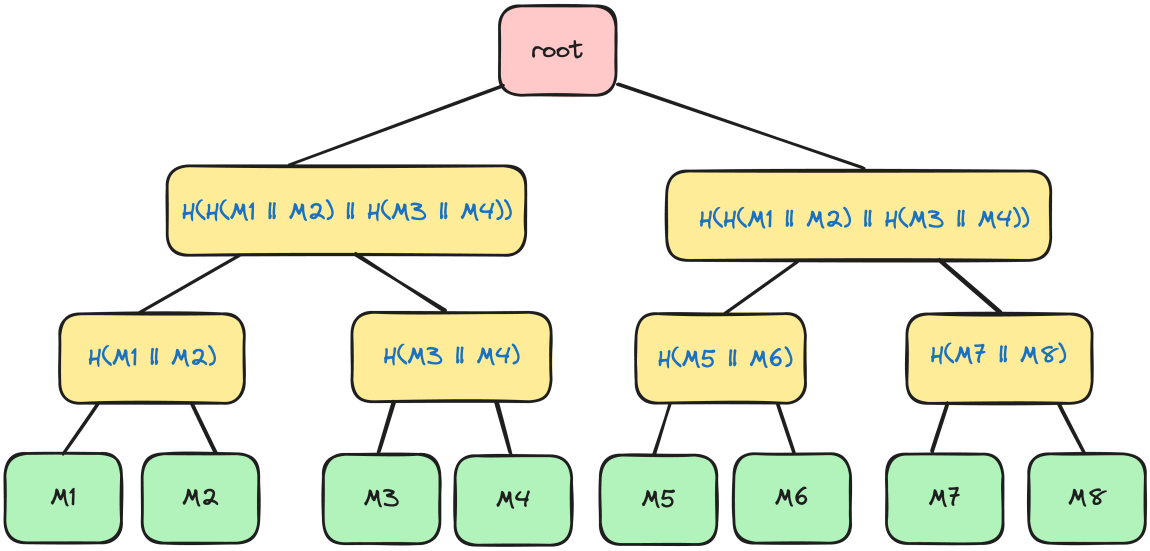

Figure 2: Merkle Tree structure

A Merkle Tree is a data structure to efficiently and securely verify the commiments to a vector of data. It is a binary tree where each leaf node represents a hash of a data block, and each non-leaf node is a hash of its child nodes' concatenated hashes. The top node, called the root hash or Merkle root, uniquely represents the entire data set. By comparing this root with a known valid root, one can quickly verify the authenticity and integrity of the data without needing to examine the entire dataset.

To prove the inclusion of element into the tree, a corresponding Merkle Branch is used. On the example below, M_1 inclusion is proved, and $(M_2, H(M_3 \parallel M_4), H(H(M_5 \parallel M_6) \parallel H(M_7 \parallel M_8)))$ is an inclusion branch vector.

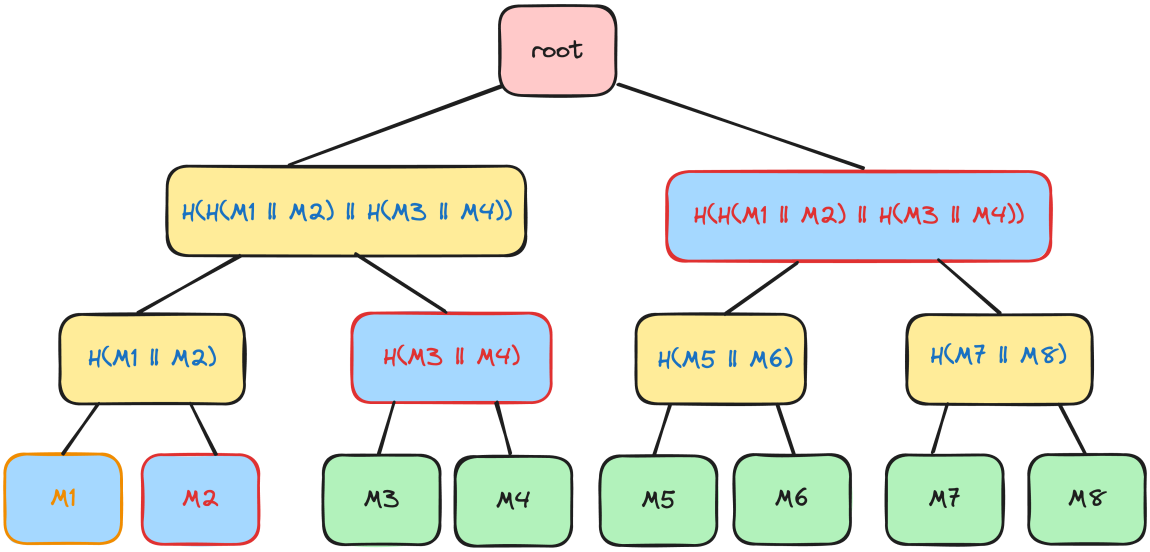

Figure 3: Merkle Tree inclusion proof branch

One of Merkle tree key advantages is that it allows for the selective disclosure of specific elements within the data set without revealing the rest.

0.1.4 Polynomial commitment

Polynomial commitment can be used to prove that the commited polynomial satisfies certain properties $P(x_1, x_2, \ldots, x_n) = y$, without revealing what the polynomial is. The commitment is generally succint, which means that it is much smaller than the polynomial it represents.

The KZG polynomial commitment scheme

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

1. *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. Public parameters of a trusted setup are calculated as:

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

Parameter τ should be deleted after the ceremony. If it is revealed, the *binding* property of the commitment scheme can be broken. This parameter is usually called the *toxic waste*.

2. *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 $p(\tau)$ directly since the value of τ is unknown, he can compute it using values $([\tau^0]G, [\tau^1]G, \ldots, [\tau^d]G)$:

$$[p(\tau)]G = [\textstyle\sum_{i=0}^{d} p_i \tau^i]G = \sum_{i=0}^{d} p_i [\tau^i]G$$

3. *Prove an evaluation*. To prove that at some point x_0 polynomial equals y_0 ($p(x_0) = y_0$), compute polynomial

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

   Polynomial $q(x)$ is called "quotient polynomial" and only exists if and only if $p(x_0) = y_0$:
   (a) If $p(x_0) = y_0$, we define $r(x) := p(x) - y_0$;
   (b) $r(x)$ has x_0 as a root, as $r(x_0) = 0$ by the definition. That is why there exists $q(x)$, such that $r(x) = q(x) \cdot (x - x_0)$;
   (c) Hence, the expression $q(x) = \frac{p(x)-y_0}{x-x_0}$ is a polynomial.

   The existance of this quotient polynomial serves as a proof of the evaluation. *Prover* calculates proof $\pi = [q(\tau)]G$ and sends it to the *Verifier*.

4. *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 τ using bilinear mapping:

$$e([q(\tau)]G_1, [\tau]G_2 - [x']G_2) = e([p(\tau)]G_1 - [y]G_1, G_2)$$

   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.