0.1 BLS signature

0.1.1 BLS Scheme

1. Params: e: $G_0 \times G_1 \to G_T$, g_0, g_1 are generators, hash function $H_0 : M \to G_0$
2. KegGen: choose a random α from Z_q as secret key sk, set h $\leftarrow g_1^{\alpha}$ as the corresponding public key pk.
3. Sign(sk, m): compute signature as $\sigma \leftarrow H_0(m)^{\alpha} \in G_0$
4. Verify(pk, m, σ): check if the following equation holds: $e(g_1, \sigma) = e(pk, H_0(m))$

Pros of BLS signature: Short, Aggregation, Multi-signature.

0.1.2 Signature Aggregation and Public Key Aggregation

If multiple signers sign on the same message m, all the public keys can be aggregated into one public key, which is used for verification.

1. Params: Given (pk_i, m, σ_i) for $i \in (1, 2, 3...n)$
2. Signature Aggregation: Compute aggregated signature as $\sigma \leftarrow \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdots \sigma_n = H_0(m)^{(\alpha_1+\alpha_2+...+\alpha_n)}$
3. Public Key Aggregation: Compute aggregated public key as $apk = pk_1 \cdot pk_2 \cdots pk_n = g_1^{(\alpha_1+\alpha_2+...+\alpha_n)}$
4. Verify:On receiving (apk, σ, m), verify the aggregated signature by checking the following equation : $e(g_1, \sigma) = e(apk, H_0(m))$.

   This means the aggregated public key can be computed before knowing the message and there is no need to provide the verifier with the individual public key to verify the aggregated signature.

- Public Key Attack: An attacker chooses a random number $\beta \leftarrow Z_q$, then cumputes a public key as $pk_{attacker} = g_1^{\beta} \cdot (pk_{Bob})^{-1}$, where pk_{Bob} is a public key of a suer Bob. Then the attacker can generate the aggregated signature as $\sigma^{'} = H_0(m)^{\beta}$, and claim that this is the aggregated signature signed by him and Bob. The signature can be verified using the aggretaed public key $apk^{'} = pk_{attacker} \cdot pk_{Bob} = g_1^{\beta}$, by checking the following equation $e(\sigma^{'}, g_1) = e(apk^{c\prime}, H_0(m))$.

0.1.3 Modified BLS Aggregated Signature

In order to resist public key attack, we use a modified BLS signature. The only modified part is the aggregation part.

1. Params: One more hash function H_1 is needed, the others are the same
2. KeyGen: Secret key $sk_i = \alpha_i \leftarrow Z_q, pk_i = g_1^{\alpha_i}$
3. Sign: $\sigma_i \leftarrow H_0(m)^{\alpha_i} \in G_0$
4. Signature Aggregation: Firstly, compute $t_i = H_1(pk_i)$ for $i = 1, 2..., n$, then aggregation signatures as $\sigma = \sigma_1^{t_1} \cdot \sigma_2^{t_2} \cdots \sigma_n^{t_n} \in G_0$
5. Public Key Aggregation: $apk = pk_1^{t_1} \cdot pk_2^{t_2} \cdots pk_n^{t_n} \in G_1$
6. Verify: On input (apk, σ, m), veryfy the aggregated signature by verifying the following equation $e(\sigma, g_1) = e(apk, H_0(m))$

Normally, all signers sign on the same message, so only two pairing operations are needed to verify the signature.

0.1.4 Batch Verification

On input multiple pairs of (m_i, apk_i, σ_i) for $i = 1, 2, ...b$, where apk_i is the aggregated public key for the aggregated signature σ_i on message m_i, the verifier checks the signatures as follows:

1. Aggregate signatures as $\sigma^{'} = \sigma_1 \cdot \sigma_2 \cdots \sigma_b \in G_0$;
2. check the equation: $e(g_1, \sigma^{'}) = e(apk_1, H_0(m_1)) \cdot e(apk_2, H_0(m_2)) \cdots e(apk_b, H_0(m_b))$

c

0.2 Threshold Signature Overview

Digital signature is the critical component that is used to ensure the validity of a cryptocurrency transaction. Basically, the transaction sender authorizes a transaction by generating a signature on the transaction using the private key and the signature can be verified using the corresponding public key to ensure that the transaction is indeed from a party that owns the corresponding private key and the transaction has not been modified during transmission. The most commonly used signature scheme in cryptocurrency is ECDSA, which is used in Bitcoin and some other cryptocurrencies.

In cryptocurrency space, anyone who controls the private key controls the money. For the ECDSA signature scheme, only the party who controls the one private key has the power the spend the cryptocurrency. However, this approach suffers from two main

drawbacks. The first drawback is that the fact that one private key is controlled by one party is actually a single point of failure, which makes it more vulnerable to key lost or theft. The second drawback is that the party that controlled the private key may abuse the power to spend the money, which contradicts the idea of decentralization.

Motivated to solve these problems of ECDSA signature, threshold signature is proposed to enhance key security. Roughly speaking, a threshold signature differs with a digital signature in that the key generation and signature generation is jointly performed by multiple players in distributed manner, whereas the signature verification remains the same. In another word, a threshold signature allows multiple parties to interactively issue a signature on a message, and the signature can be verified by a single public key. More specifically, in a typical $(t, n)-$threshold signature scheme, a private key is divided into n shares, which are shared by n parties individually. A threshold t is defined in the way that any attacker who compromises t or fewer parties, which means that the attacker learns t or fewer private shares, cannot learn information about the private key and is not able to forge a valid signature. And any subset of $t + 1$ parties can jointly issue a valid signature in the distributed manner without recoving the private key. Hence, a threshold signature solves the problem of single point of failure of a ECDSA signature, and allows honest parties to issue valid signatures even when some of the parties are compromised. Moreover, unlike the ECDSA signature scheme where a single party can spend the cryptocurrency, in a threshold signature scheme, multiple players jointly control the money, which conforms to the idea of decentralization.

A typical $(t, n)-$threshold signature scheme **S=(Thresh-KeyGen,Thresh-Sig, Veri)** involves the following three algorithms.

- Thresh-KeyGen:This is a distributed key generation protocol performed by n parties. It takes the security parameter as input. It outputs a single public key pk and n different private shares sk_i for every player P_i, where $i = 1, 2, ..., n$. The $sk_1, sk_2, ..., sk_n$ is a (t, n) threshold secret sharing of the private key sk.

- Thresh-Sig:This is a distributed signing protocol that performed by multiple parties. It takes a message m and private shares sk_i from the parties as input, and outputs a signature σ.

- Veri: This is the same verification process as the ECDSA signature scheme. It can be performed by any message recipient. On input a signature σ, message m and public key pk, the algorithm outputs true or false.

0.2.1 Threshold Signature VS Shamir Secret Sharing

The cryptographic techniques Shamir secret sharing and multi-signature has the similar functionality that distributes the signing power into multiple players as threshold signature. However, there are several differences among them. A (t, n) Shamir secret sharing is cryptographic primitive that splits a private key into n shares among n play-

ers, and at least $t + 1$ shares are required to reconstruct the private key. It only solves the problem

0.2.2 Threshold Signature VS Multi-Signature

Threshold signature does not enforce accountability, meaning that a t-out-of-n threshold signature does not tell the verifier which t of the n signers created the signature. However, a multisignature enforces accountability, which means that the verifier knows which t signers created the signature.

However, multisignature have several drawbacks. The first drawback is that multisignature is not flexible in terms of its policy. For instance, once we define a policy, which requires t out of n users to sign the transaction, if one or several parties lose connections and cannot generate the signatures, then such a multisignature will not be generated successfully. Another problem is that if a new party joins or leaves the group, the access structure should be changed, which means the public address of the fund should be changed. Moreover, the exact access policy is public on the blockchain, which undermines the privacy of the group.

Both threshold signature and multisignature aim to achieve the same function that t out of n members are required to generate a valid signature. However threshold signature have several advantages over multisignature.