0.1 Threshold Signature Overview

Digital signature is the critical component 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 is 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 full private key is controlled by one party could be 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 spirit of decentralization.

Motivated to solve these problems of ECDSA signature, threshold signature(TSS) is proposed to enhance key security. Roughly speaking, a threshold signature differs with a digital signature in that the key generation and signature generation are 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 $t-1$ or fewer parties, which means that the attacker learns $t-1$ or fewer private shares, cannot learn any information about the full private key and is not able to forge a valid signature.

And any subset of t parties can jointly issue a valid signature in distributed manner without recoving the full private key. Hence, a threshold signature solves the problem of single point of failure of the 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 spirit 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 jointly. 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 jointly. 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 the public key pk, the algorithm outputs $Ture$ if the signature is valid, otherwise outputs $False$.

Two main advantages of threshold signatures.

1. **Increased Availability:**Can sign even through some keys are lost.

2. **Enhanced Security:**The attacker need to crrupt more players to forge a signature. The full private key is never stored in one location

However it has the drawbacks that threshold signature requires more space to store the signatures and keys and more computation time to verify.

The cryptographic techniques Shamir secret sharing and multi-signature have the similar functionality that distributes the signing power to multiple players as the TSS. However, there are several differences among them. In the following section, the secret sharing and BLS multi-signature will be introduced, and the differences with TSS will be discussed.

0.2 Threshold Signature vs Shamir Secret Sharing

Secret sharing is used to protect a secret key by splitting in into multiple fragments and distributing the fragments amongest a group of parties.The private key can be recovered only when a threshold of fragments are combined together. More specifically, a (t, n) Shamir secret sharing is cryptographic primitive that splits a private key into n shares among n players, and at least t shares are required to reconstruct the private key. The key idea of Shamir secret sharing is that it takes t points to define a polynomial of degree $t-1$ to recover the private key. A basic (t, n) Shamir secret sharing scheme has two algorithms:

- Share: It choose $a_1, ..., a_{t-1}$ as the coefficients of a polynomial of $f(z)$, where $f(z) = a_0 + a_1 \cdot z + a_2 \cdot z^2 + \cdots + a_{t-1} \cdot z^{t-1}$. Suppose private key is x, set $f(0) = a_0 = x$. The private key could be regarded as the intersection point between the polynomial curve with the Y axis. Then, the private share for the party P_i, where $i = 1, 2, ...n$, is $f(i)$. Finally, the dealer distributes $f(i)$ to every player P_i. Note that any group of which the size is less than t is not able to reconstruct the polynomial $f(z)$, thus is not be able to recover the private key $x = f(0)$

- Reconstruct: t shares can be combined to reconstruct the polynomial $f(z)$ using the efficient Lagrange Interpolation. Once the polynomial is reconstructed, the private key is recovered as $f(0)$.

Cpomaring to TSS, secret sharing approach has the following two main drawbacks.

- In the key generation phase of a secret sharing scheme, a trusted **dealer** is required to generate the single private key and the corresponding secret shares, and distribute the shares to a group of players. This means that a full private key is stores in a single location, which is a single point of failure. However, in a TSS scheme, no **dealer** is required to distribute secret shares, as the key generation phase is completed performed in the distributed manner. Hence, the scenario where a full private key is stored in a single location will never happen in TSS.

- In the signing phase of a secret sharing phase, multiple private shares must be combined to reconstruct the full private key to sign messages, which is also a single point of failure. However, in a TSS scheme, the distributed players jointly generate a signature on the message using their own secret shares without reconstructing a full private key.

0.3 Threshold Signature vs Multi-Signature

0.3.1 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.

0.3.2 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 $e(\sigma^{'}, g_1) = e(apk^{c\prime}, H_0(m))$.

0.3.3 Modified BLS Aggregated Signature

In order to resist public key attack, a modified BLS signature is proposed. 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.

0.3.4 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: $e(g_1, \sigma^{'}) = e(apk_1, H_0(m_1)) \cdot e(apk_2, H_0(m_2)) \cdots e(apk_b, H_0(m_b))$

0.3.5 Comparison

Multi-signature shares the similar functionality of distributing the signing power to multiple users as TSS. One difference is that in a multi-signature scheme, multiple signatures are generated by multiple users, whereas in TSS, only one signature is generated by multiple players interactively. Aonther critical diference between multi-signautre and TSS is that multi-signature is on-chain operation whereas most part of the TSS is performed off-chain. Due to this critical difference, multi-signature has the following drawbacks comparing with TSS.

- The threshold policy of multi-siggnature is not flexible. Suppose a new party wants to join the group, then the keys for the fund should be changed to set a new access policy, which menas that the fund must be moved to a new address on the blockchain. However, in TSS, the key generation and signature generation are all performed off-chain, updating the policy is convenient and does not incur transaction fee.

- The privacy of multi-signature scheme is weak, as all the operations are public on-chain. However, for a TSS scheme, only one signature is onchain for verification.

- Multi-signature is blockchain-specific, which means that a multi-signature scheme canno be used in different blockchains. Whereas, a TSS scheme can be utilized in all blockchains, as the key generation and signature generation are all performed off-chain. The only on-chain operation is the signature verification, which is the same as the ECDSA scheme and is compatible with many blockchains.

0.3.6 References

1. Boneh, Dan, Manu Drijvers, and Gregory Neven. "Compact multi-signatures for smaller blockchains." International Conference on the Theory and Application of

Cryptology and Information Security. Springer, Cham, 2018.

2. Gennaro, Rosario, and Steven Goldfeder. "One Round Threshold ECDSA with Identifiable Abort." IACR Cryptol. ePrint Arch. 2020 (2020): 540.