Threshold Signature for Distributed Systems

18120019 – Nguyen Hoang Dung
18120130 – Vo Anh Khoa

Project Abstract

We build a digital multi-signature algorithm to be used in distributed systems such as blockchain. For a group with N users, we want any subset of at least k signers to be able to jointly produce the same digital signature under the identity of the whole group.
All operations on digital signatures are performed on the group of points of an elliptic curve $E(\mathbb{Z}_p)$, whose group has order n and generator G, satisfying the elliptic curve equation.

1 Extending MuSig with a Threshold of k Signers

Based on the **whitepaper** of MuSig Schnorr Signature from TariLabs University, we extend this construction to support a threshold of k signers out of N.

Phase 1: Setup Secret Polynomials

Each user i randomly initializes a secret polynomial of degree $k-1$:

$$f_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,k-1}x^{k-1} \mod n$$

The coefficient $a_{i,0}$ is the private key of user i. The public key of user i is

$$P_i = a_{i,0} \cdot G.$$

The group secret polynomial is the sum of the secret polynomials of all users:

$$f(x) = \sum_{i=1}^{N} f_i(x).$$

The constant coefficient $a_0 = f(0)$ of the polynomial $f(x)$ is the private key of the whole group. No user learns any information about the coefficients of the group secret polynomial $f(x)$.

Phase 2: Aggregated Public Key

Let L be the hash of all public keys of the group members: $L = \mathsf{H}(P_1, P_2, \ldots, P_N)$.
From its own public key, each user i computes its individual challenge value: $c_i = \mathsf{H}(L, P_i)$.
Each user then derives a *challenge private key* by multiplying its own private key with this challenge:

$$s_i = c_i \cdot a_{i,0} \pmod n.$$

The aggregated group public key is

$$P = \sum_{i=1}^{N} c_i \cdot P_i = \sum_{i=1}^{N} (c_i \cdot a_{i,0}) \cdot G.$$

Phase 3: Secret Shares

For each user i, define a *challenge polynomial* $f_i'(x)$:

$$f_i'(x) = s_i + a_{i,1}x + \cdots + a_{i,k-1}x^{k-1}$$

Broadcast Commitment | User i commits polynomial coefficients and broadcasts as:

$$F_i(x) = A_{i,0} + A_{i,1}x + \cdots + A_{i,k-1}x^{k-1},$$

where $A_{i,0} = s_i \cdot G = (c_i \cdot a_{i,0}) \cdot G$, and for $t \geq 1$, $A_{i,t} = a_{i,t} \cdot G$

Verify Secret Shares | Via a secure (private) channel, user i sends to user j a secret share $(j,\ f_i'(j))$, which can be verified by the equation:

$$f_i'(j) \cdot G = \sum_{t=0}^{k-1} A_{i,t}\, j^t = A_{i,0} + A_{i,1}j + \cdots + A_{i,k-1}j^{k-1}.$$

After the exchange of all secret information between the users, user j possesses the point (j, y_j') as the j-th secret share of the *aggregated polynomial* $f'(x)$

$$y_j' = f'(j) = \sum_{i=1}^{N} f_i'(j) \quad \text{with} \quad f'(x) = \sum_{i=1}^{N} f_i'(x).$$

On the other hand, $f'(0)$ is precisely the sum of all challenge private keys – the scalar corresponding to the aggregated group public key $P = f'(0) \cdot G$

$$f'(0) = \sum_{i=1}^{N} f_i'(0) = \sum_{i=1}^{N} s_i = \sum_{i=1}^{N} c_i \cdot a_{i,0}.$$

Lagrange Interpolation | The polynomial $f'(x)$ has degree $k-1$ with k coefficients. Each user i has share (i, y_i'). So any subset S of k users who participate can jointly reconstruct $f'(0)$

$$f'(0) = \sum_{i \in S} \left[y_i' \prod_{j \in S \setminus \{i\}} (-j)(i-j)^{-1} \right].$$

Equivalently, we can view each user $i \in S$ as holding a *group key share*

$$\pi_i = y_i' \cdot \prod_{j \in S \setminus \{i\}} (-j)(i-j)^{-1}$$

Phase 4: Threshold Signing

Now suppose a subset S of k users in the original group wish to jointly sign a message m. They use their indices and shares (i, y_i') to reconstruct the aggregated group public key

$$P = \sum_{i \in S} G \cdot \pi_i$$

Each of the k users chooses a random nonce r_i and computes $R_i = r_i \cdot G$, then broadcasts a commitment $t_i = \mathsf{H}(R_i)$. After receiving the $(k-1)$ other t_i values, all users simultaneously reveal their R_i values and check that each $\mathsf{H}(R_i)$ matches the received t_i. If all checks pass, they compute signature (R, s)

$$R = \sum_{i \in S} R_i$$

$$e = \mathsf{H}(R, P, m)$$

$$s = \sum_{i \in S} (r_i + e \cdot \pi_i)$$

It is known that if any commitment t_i cannot be verified, the protocol should abort, else a multi-session attack can be performed, where attackers open many parallel sessions, pick their nonces after seeing the honest nonces, and then choose messages so that a linear relation holds between the challenges. Summing partial signatures gives a forgery on a message honest parties never agreed to.

Signature Verification

To verify a signature (R, s) on message m under the aggregate public key P, the verifier recomputes

$$e = \mathsf{H}(R, P, m)$$

and checks the verification equation

$$G \cdot s = R + P \cdot e.$$

Expanding the definitions, we see that

$$\begin{aligned} s &= \sum_{i \in S} \big(r_i + e \cdot \pi_i\big) \\ G \cdot s &= \sum_{i \in S} \big(r_i \cdot G\big) + e \cdot \sum_{i \in S} \big(\pi_i \cdot G\big) \\ &= \sum_{i \in S} R_i + e \cdot f'(0) \cdot G \\ &= R + P \cdot e \end{aligned}$$

2 Implementation & Demo

To be translated later ...