1 The non-concurrent multi-signature scheme

Typically, multi-signature schemes aggregate multiple signatures into one, which need at least two-round interactions for schnorr-based schemes. Generally, the first round is used to determine and exchange the commitments among all signers, and the second round is used to generate and exchange the partial signatures among all signers. Therefore, the process is concurrent, requiring all signers online all the time.

In some scenario, it is difficult for signers to keep online during the whole process. Thus, the non-concurrent multi-signature scheme is proposed, considering the signatures for the same message are aggregated one-by-one. When the signer attends to aggregate his/her own signature with others, it is not necessary for other signers online, by introducing smart contract as a bulletin board, which also contributes to defense k-sum attacks.

Different with the classic multi-signature schemes, the presigning process is introduced, which we assume only two signers participate in, in order to demonstrate one-by-one aggregation. Actually, this presigning process is exactly the signing process in classic multi-signature, which can be executed among any number of singers. Particularly, only current signer involves in the signing process, with smart contract as a bulletin board.

Parameter Generation (**ParamGen**(λ)). Given security parameter λ, this algorithm generates a group $\mathbb{G}$ with prime p order and a generator $g \in \mathbb{G}$. In the end, it outputs $par = (\mathbb{G}, p, g)$.

Key Generation (**KeyGen**$(par) \rightarrow (pk_i, sk_i)$). Each signer $\mathcal{S}_i$ randomly generates a secret private key $sk_i : x_i \leftarrow_r \mathbb{Z}_p$ and calculates the corresponding public key $pk_i : X_i = g^{x_i}$.

PreSigning (**PreSign**(par, sk_1, sk_2, m)): This algorithm is called by the specified two signers $\mathcal{S}_1$ and $\mathcal{S}_2$, with the input of public parameters par, private key sk_1, sk_2 and the message m. $\mathcal{S}_1$ and $\mathcal{S}_2$ will conduct two-round interactions to generate a pre-signature $\sigma_{1,2}$.

- ***Round 1:***
  First, the signer $\mathcal{S}_1$ generates a random number $r_1 \leftarrow_r \mathbb{Z}_p$, computes his/her commitment $R_1 = g^{r_1}$ and sends R_1 to smart contract. Similarly, the signer $\mathcal{S}_2$ computes R_2 and sends it to smart contract.

  Next, smart contract waits to receive commitments $\{R_1, R_2\}$ and forwards them to the two signers.

  Last, smart contract records the timestamp t_2 of current time, makes a hash $w_2 = H_1(t_2)$ and computes $W_2 = g^{w_2}$. Smart contract returns w_2 and W_2 to the two signers.

  As smart contract is automatic and public, the commitments $\{R_1, R_2\}$ and timestamp t_2 are public and immutable.

- ***Round 2:***
  After receiving $\{R_1, R_2\}$, w_2, W_2 from smart contract, $\mathcal{S}_2$ continues to

calculate the aggregated commitment value

$$R_{1,2} = {W_2}^2 R_1 R_2$$

and the partial signature

$$s_2 = w_2 + r_2 + c_2 sk_2 (modp)$$

where $c_2 = H_2(X_2, R_{1,2}, m)$.

Similarly, $\mathcal{S}_1$ computes s_1. After collecting s_1, $\mathcal{S}_2$ computes $s_{1,2} = s_1 + s_2(modp)$ as the joint signature.

The output of this algorithm is $\sigma_{1,2} = (R_{1,2}, s_{1,2})$.

Signing (**Sign**$(par, sk_{i+1}, m, \sigma_{i-1,i})$): Based on the pre-signature from signers $\mathcal{S}_1$ and $\mathcal{S}_2$, this algorithm is iteratively output by the signer $\mathcal{S}_{i+1}(i \geq 2)$ to output the signature $\sigma_{i,i+1}$.

- ***Round 1:***
  First, the signer $\mathcal{S}_{i+1}$ generates a random number $r_{i+1} \leftarrow_r \mathbb{Z}_p$, computes his/her commitment $R_{i+1} = g^{r_{i+1}}$ and sends R_{i+1} to smart contract.

  Next, smart contract waits to receive R_{i+1} and forwards pre-commitment $(R_1, \cdots, R_i)$ and $(W_2, \cdots, W_i)$ to $\mathcal{S}_{i+1}$.

  Last, smart contract records the timestamp t_{i+1} of current time, makes a hash $w_{i+1} = H_1(t_{i+1})$ and computes $W_{i+1} = g^{w_{i+1}}$. Smart contract returns w_{i+1} and W_{i+1} to $\mathcal{S}_{i+1}$.

- ***Round 2:***
  $\mathcal{S}_{i+1}$ continues to calculate the aggregated commitment value

$$R_{i,i+1} = \prod_{j=1}^{i+1} W_j R_j$$

  and the partial signature

$$s_{i+1} = w_{i+1} + r_{i+1} + c_{i+1} sk_{i+1} (modp)$$

  where $W_1 = W_2$ and $c_{i+1} = H_2(X_{i+1}, R_{i,i+1}, m)$.

  Finally, $\mathcal{S}_{i+1}$ can compute $s_{i,i+1} = s_{i-1,i} + s_{i+1}(modp)$ as the joint signature.

  The output of this algorithm is $\sigma_{i,i+1} = (R_{i,i+1}, s_{i,i+1})$.

Verification (**Verify**$(par, L_{pk}, L_{com}, L_{time}, m, \sigma) \rightarrow 1/0$). Assume there are n signers totally. Given the public key list $L_{pk} = \{pk_1, ..., pk_n\}$, the commitment value list $L_{com} = \{R_1, ..., R_n\}$, the timestamp list $L_{time} = \{W_1, ..., W_n\}$, a

message m, an aggregated signature $\sigma = (R, s)$ under public list L, the verifier calculates

$$R_{i-1,i} = \prod_{j=1}^{i} W_j R_j$$

and

$$c_i = H_2(pk_i, R_{i-1,i}, m)$$

where $i = 2, \cdots, n$.

Finally, output 1 if

$$R = R_{n-1,n}$$

and

$$g^s = R \cdot \prod_{i=1}^{n} pk_i^{c_i}$$