A Review of Threshold Digital Signature Schemes

Yongge Wang
UNC Charlotte

January 26, 2020

Abstract

In recent years, threshold cryptography has been used in cryptography currencies. For example, they have been used in secure random beacon protocols such as Randao and for the purpose of wallet key protection and full cryptocurrency custody solutions to large customers. Threshold ECDSA schemes has recently been used in the Swingby project for the purpose of moving assets between blockchains. This paper reviews a few threshold digital signature schemes that are of interest for the blockchain community.

1 Introduction and Digital Signature Algorithms DSA and ECDSA

Efficient threshold cryptography protocols have been designed for many cryptographic schemes such as RSA signing and decryption, ElGamal and ECIES encryption, Schnorr signatures, Cramer-Shoup, and many others. However, full-threshold DSA/ECDSA schemes with practical distributed key generation and signing are unknown until recent work by Gennaro and Goldfeder [6] and Lindell, Nof, and Ranellucci [9]. The reader is referred to [9] for a complete review of the efforts for threshold DSA/ECDSA.

In this paper, we review a few threshold digital signature schemes that are of interest for the blockchain community. We first review the DSA and ECDSA schemes. A group G is cyclic if there is an element $g \in G$ such that for each $y \in G$ thre is an integer x with $y = g^x$. Such an element g is called a generator of G. For an element $g \in G$, the order of g is defined to be the least positive integer x such that $g^x = 1$. As an example, for a given prime number p, the multiplicative group $\mathbb{Z}_p^* = \{1, \cdots, p-1\}$ is a cyclic group of order $p-1$. A discrete logarithm problem (DLP) is defined as follows. Give a prime p, a generator g of $\mathbb{Z}_p^*$, and an element $y \in \mathbb{Z}_p^*$, find the integer $x, 0 \le x \le p-2$, such that $g^x = y \mod p$.

The most recent version of the Digital Signature Scheme (DSS) is published in FIPS PUB 186-4 [8]. FIPS PUB 186-4 includes description of DSA and ECDSA algorithms. The DSA algorithm works as follows. Let p be a prime of L bits and q be an N-bits prime divisor of $p-1$. Let $1 < g < p$ be a generator of a subgroup of order q in the multiplicative group $\mathbb{Z}_p^*$. For various security strength, DSA recommends the following parameter pairs

1. $L = 1024, N = 160$
2. $L = 2048, N = 224$
3. $L = 2048, N = 256$
4. $L = 3072, N = 256$

The DSA signature mechanism requires a hash function $H : \{0,1\} \to \mathbb{Z}_q^*$. Let $\min(N, outlen)$ be the minimum of the positive integers N and $outlen$, where $outlen$ is the bit length of the hash function output block. For simplification, we abuse our notation by using $H(M)$ to denote the leftmost $\min(N, outlen)$-bits of $H(M)$. The DSA algorithm includes key generation, signature generation, and signature verification algorithms.

DSA Key generation algorithm. For the security parameter (L, N), each entity creates a public key y and corresponding private key x as follows:

1. Obtain a string c of $N + 64$ bits using a random bit generator
2. Let $x = (c \mod (q-1)) + 1$ and $y = g^x \mod p$.

DSA signature generation algorithm. An entity with private key x generates a digital signature (r, s) for a binary message M of arbitrary length as follows

1. Selects a random integer k and let $r = (g^k \mod p) \mod q$
2. Let $s = (k^{-1}(H(M) + xr)) \mod q$.

DSA signature verification algorithm. Let M', r', and s' be the received versions of M, r, and s, respectively. Let y be the public key of the claimed signatory. The signature verification process is as follows:

1. Let $w = (s')^{-1} \mod q$
2. Let $u_1 = wH(M') \mod q$ and $u_2 = r'w \mod q$
3. Let $v = (g^{u_1}y^{u_2} \mod p) \mod q$
4. The signature is valid if and only if $v = r'$

FIPS PUB 186-4 refers ECDSA to ANSI X9.62. Domain parameters for ECDSA are of the form (q, FR, a, b, G, n, h), where q is the field size; FR is an indication of the basis used; a and b are two field elements that define the equation of the curve (that is, $y^2 = x^3 + ax + b$); G is a base point of prime order on the curve (i.e., $G = (x_G, y_G)$), n is the order of the point G, and h is the cofactor (which is equal to the order of the curve divided by n). ECDSA is defined for any one of the arithmetic fields $GF(p) = \mathbb{Z}_p^*$ or $GF(2^m)$. The ECDSA algorithm includes key generation, signature generation, and signature verification algorithms.
ECDSA Key generation algorithm. An ECDSA key pair (x, Q) for a set of domain parameters (q, FR, a, b, G, n, h) is generated as follows:

1. Obtain a string c of $\mathbf{len}(n) + 64$ bits using a random bit generator
2. Let $x = (c \bmod (n-1)) + 1$ and $Q = xG$.

ECDSA signature generation algorithm. An entity with private key d generates a digital signature (r, s) for a binary message M of arbitrary length as follows

1. Select a random integer k, $1 \le k \le n-1$.
2. Compute $kG = (x_1, y_1)$ and convert x_1 to an integer $\overline{x}_1$.
3. Compute $r = \overline{x}_1 \bmod n$. If $r = 0$ then go to step 1.
4. Compute $s = k^{-1}(H(M) + xr) \bmod n$. If $s = 0$ then go to step 1.
5. The signature for the message m is (r, s).

ECDSA signature verification algorithm. For a signature (r, s) on a message M, Assume that the public key is Q. The signature verification process is as follows:

1. Compute $w = s^{-1} \bmod n$.
2. Compute $u_1 = wH(M) \bmod n$ and $u_2 = rw \bmod n$.
3. Compute $X = u_1G + u_2Q$.
4. Convert the x-coordinate x_1 of X to an integer $\overline{x}_1$, and compute $v = \overline{x}_1 \bmod n$.
5. Accept the signature if and only if $v = r$.

2 Threshold DSA by Gennaro and Goldfeder

There are several approaches to develop threshold DSA schemes. For example, one may share the secret keys x among participants (e.g., using Shamir secret share schemes) and then jointly compute both r and s. This could be achieved by designing mechanisms to compute two multiplications over secret values that are shared among the participants. Alternatively, one may encrypt the secret key x using an additively homomorphic encryption scheme E (e.g., Paillier's scheme [10]) and share the secret key of E among the participants. But this approach may require the distributed generation of the additively homomorphic encryption scheme E. Recently, Gennaro and Goldfeder [6] designed an efficient threshold ECDSA scheme using the SPDZ approach [3]. For a high level description, Gennaro and Goldfeder's approach [6] works as follows. The participants start with a (t, N) Shamir secret sharing of the secret key x. When $t + 1$ participants want to sign a message, they generate an additive sharing of two random values $k = \sum_i k_i$ and $\gamma = \sum_i \gamma_i$. Then the participants compute additive sharings of the products $\delta = k\gamma$ and $\sigma = kx \sum_i w_i$. Note that for two secrets $a = a_1 + \cdots + a_N$, $b = b_1 + \cdots + b_N$ additively shared among N participants where P_i holds a_i and b_i. One can compute the additive sharing of $c = ab$ by computing an additive sharing of each individual term $a_i b_j$ since $ab = \sum_{i,j} a_i b_j$. The details of the Gennaro and Goldfeder's approach is described in the following paragraphs.

We first describe Paillier's additive encryption scheme [10]. The public key $\mathtt{pk} = (n, g)$ for a Paillier's scheme consists of two integers where $n = pq$ divides the order of $g \in \mathbb{Z}^*_{n^2}$ and p, q are two prime numbers. The private key $\mathtt{sk} = (\lambda, \mu)$ is a pair of integers where $\lambda = \text{lcm}(p-1, q-1)$ and $\mu = \left(\frac{(g^\lambda \mod n^2)-1}{n}\right)^{-1} \mod n$. A message m is encrypted to $c = \mathtt{Enc}_{\mathtt{pk}}(m) = g^m \cdot r^n \mod n^2$ for a randomly selected $r \in \mathbb{Z}^*_n$. A ciphertext c is decrypted to the message $m = \mathtt{Dec}_{\mathtt{sk}}(c) = \frac{\mu((c^\lambda \mod n^2)-1)}{n} \mod n$.

In a Shamir's secret sharing scheme, the dealer generates a random degree t polynomial $p(x) = s + a_1 x + \cdots + a_t x^t$ over $\mathbb{Z}_q$ where the secret is $s = p(0)$. A participant P_i receives the share $s_i = p(i) \mod q$. Feldman's verifiable Secret Sharing Scheme (VSS) is an extension of Shamir secret sharing scheme where the dealer also publishes $v_0 = g^s$ and $v_i = g^{a_i}$ for all $i = 1, \cdots, t$. Using this auxiliary information, each participant P_i can check its share s_i for consistency by verifying whether $g^{s_i} = \prod_{j=0}^{t} v_j^{i^j}$.

We next describe a share conversion protocol from multiplicative shares to additive shares. Assume that two participants P_1 and P_2 multiplicatively share a secret $x = ab \in \mathbb{Z}_q$ where P_1 holds a and P_2 holds b. P_1 and P_2 would like to additively share the secret $x = \alpha + \beta \in \mathbb{Z}_q$ where P_1 holds α and P_2 holds β. This could be achieved using the Paillier's additively homomorphic scheme as follows

1. P_1 creates his Paillier's public key $\mathtt{pk}$ and his private key $\mathtt{sk}$.
2. P_1 initiates the protocol by sending $c_A = \mathtt{Enc}_{\mathtt{pk}}(a)$ to P_2 and proving in ZK that $a < K$ via a range proof where K is a constant bound that we will discuss later[1].
3. P_2 computes the ciphertext $c_B = bc_A - \mathtt{Enc}_{\mathtt{pk}}(\beta) = \mathtt{Enc}_{\mathtt{pk}}(ab - \beta)$ where β is chosen uniformly at random. P_2 sets his share to β and responds to P_1 by sending c_b and proving in ZK that $b < K$. Furthermore, if g^b is public, P_2 proves he knows b and β in ZK.[2]
4. P_1 decrypts c_b to obtain his share $\alpha = ab - \beta$.

Now we are ready to describe Gennaro and Goldfeder's threshold DSA scheme GG-DSA. We assume that each participant P_i is associated with a public key $\mathtt{pk}_i$ for an additively homomorphic encryption scheme.
GG-DSA Key generation algorithm.

1. Each participant P_i selects a random $u_i \in \mathbb{Z}_q$, computes $y_i = g^{u_i}$, $h_i = H(y_i, r_i)$ for a random r_i, and broadcasts h_i.
2. Each participant P_i broadcasts y_i and r_i. The participant P_i performs a (t, N) Feldman-VSS of the value u_i. The public key is set to $y = \prod_i y_i$. Each participant adds the private shares received during the N Feldman VSS protocols. The resulting values x_i are a (t, N) Shamir's secret sharing of the secret key $x = \sum_i u_i$. Note that the values $X_i = g^{x_i}$ are public.

[1]Note that [6] conjectured that this ZK proof could be removed.
[2]Note that [6] conjectured that the first ZK proof could be removed and the second ZK proof could be simplified.

3. Let $N_i = p_i q_i$ be the RSA modulus associated with $\mathtt{pk}_i$. Each participant P_i proves in ZK that he knows x_i using Schnorr's protocol [11] and that he knows p_i, q_i using any proof of knowledge of integer factorization.

GG-DSA signature generation algorithm. Let $S = \{i_1, \cdots, i_t\} \subset \{P_1, \cdots, P_N\}$ where $\mathbb{P} = \{P_{i_1}, \cdots, P_{i_t}\}$ is a set of t participants that participate in the signature protocol. For the signing protocol we can share any ephemeral secrets using a (t,t) secret sharing scheme. Note that using the appropriate Lagrangian coefficients $\lambda_{i,S}$, each participant in $\mathbb{P}$ can locally map its own (t,N) share x_i of x into a (t,t) share w_i of x: $w_i = x_i \lambda_{i,S}$. That is, $x = \sum_{i \in S} w_i$. Since $X_i = g^{x_i}$ and $\lambda_{i,S}$ are public values, all participants can compute $W_i = g^{w_i} = X_i^{\lambda_{i,S}}$. The participants in $\mathbb{P}$ jointly generate a DSA signature (r,s) on the message M as follows where $m = H(M)$.

1. Each participant $P_i \in \mathbb{P}$ selects $k_i, \gamma_i \in_R \mathbb{Z}_q$, computes $h_i = H(g^{\gamma_i}, \theta_i)$ for a random θ_i, and broadcasts h_i. Define $k^{-1} = \sum_{i \in S} k_i$, $\gamma = \sum_{i \in S} \gamma_i$. Note that $k^{-1}\gamma = \sum_{i,j \in S} k_i \gamma_j \mod q$ and $k^{-1}x = \sum_{i,j \in S} k_i w_j \mod q$.
2. Every pair of players $P_i, P_j \in \mathbb{P}$ engages in two multiplicative-to-additive (MtA) share conversion sub-protocols
   (a) P_i, P_j run MtA with shares k_i, γ_j respectively. Let α_{ij} (respectively, β_{ij}) be the share received by participant P_i (respectively P_j) at the end of this protocol. That is, $k_i \gamma_j = \alpha_{ij} + \beta_{ij}$. Participant P_i sets $\delta_i = k_i \gamma_i + \sum_{j \neq i} \alpha_{ij} + \sum_{i \neq j} \beta_{ij}$. Note that the δ_i are a (t,t) additive sharing $k^{-1}\gamma = \sum_{i \in S} \delta_i$.
   (b) P_i, P_j run MtA with shares k_i, w_j respectively. Let μ_{ij} (respectively, ν_{ij}) be the share received by participant P_i (respectively P_j) at the end of this protocol. That is, $k_i w_j = \mu_{ij} + \nu_{ij}$. Participant P_i sets $\sigma_i = k_i w_i + \sum_{j \neq i} \mu_{ij} + \sum_{i \neq j} \nu_{ij}$. Note that the σ_i are a (t,t) additive sharing $k^{-1}x = \sum_{i \in S} \sigma_i$.
3. Every participant $P_i \in \mathbb{P}$ broadcasts δ_i and the participants reconstruct $\delta = \sum_{i \in S} \delta_i = k^{-1}\gamma$.
4. Every participant $P_i \in \mathbb{P}$ broadcasts g^{γ_i} and θ_i and proves in ZK that he knows γ_i using Schnorr's protocol [11]. The participants compute

$$r = \left(\prod_{i \in S} g^{\gamma_i} \right)^{\delta^{-1}} = g^{\sum_{i \in S} \gamma_i k \gamma^{-1}} = g^k.$$

5. Each participant $P_i \in \mathbb{P}$ sets $s_i = m k_i + r \sigma_i$. Note that

$$\sum_{i \in S} s_i = m \sum_{i \in S} k_i + r \sum_{i \in S} \sigma_i = mk^{-1} + rk^{-1}x = k^{-1}(m + xr) = s.$$

   That is, the s_i is a (t,t) sharing of s.
   (a) Each participant $P_i \in \mathbb{P}$ selects $l_i, \rho_i \in_R \mathbb{Z}_q$, computes $V_i = r^{s_i} g^{l_i}, A_i = g^{\rho_i}$, $h'_i = H(V_i, A_i, \theta'_i)$ for a random θ'_i, and broadcasts h'_i. Let $l = \sum_i l_i$ and $\rho = \sum_i \rho_i$.
   (b) Each participant $P_i \in \mathbb{P}$ broadcasts (V_i, A_i, θ'_i) and proves in ZK that he knows s_i, l_i, ρ_i such that $V_i = r^{s_i} g^{l_i}$ and $A_i^{\rho_i}$. Let $V = g^{-m} y^{-r} \prod_{i \in S} V_i = g^l$ and $A = \prod_{i \in S} A_i$.
   (c) Each participant $P_i \in \mathbb{P}$ computes $U_i = V^{\rho_i}$, $T_i = A^{l_i}$, $h''_i = H(U_i, T_i, \theta''_i)$, and broadcasts h''_i.
   (d) Each participant $P_i \in \mathbb{P}$ broadcasts (U_i, T_i, θ''_i).
   (e) For a participant $P_i \in \mathbb{P}$, if the received (de-committed) values satisfies $\prod_{i \in S} T_i = \prod_{i \in S} U_i$, then P_i broadcasts s_i. All participants compute $s = \sum_{i \in S} s_i$. (r,s) is the signature on M.

GG-DSA signature verification algorithm. This is the same as the DSA signature verification algorithm.

3 Threshold ECDSA by Lindell-Nof-Ranellucci

Lindell, Nof, and Ranellucci [9] proposes a threshold ECDSA by replacing the Paillier additively homomorphic encryption with ElGamal additively homomorphic encryption in-the-exponent. We use LNR-ECDSA to denote the threshold protocol in [9]. Let $\mathbb{G}$ be a group of order n where the Decisional Diffie-Hellman problem is hard, and let G be a generator of the group. We will introduce Lindell, Nof, and Ranellucci's scheme using elliptic curve notations. For ElGamal additively homomorphic ecryption in-the-exponent, an encryption of a value $m \in \mathbb{Z}_n$ with public key $P \in \mathbb{G}$ is defined as

$$\texttt{EGexpEnc}_P(m) = (A, B) = (rG, rP + mG)$$

where $r \in_R \mathbb{Z}_n$. It is noted that, for ElGamal encryption in-the-exponent, decryption requires solving the discrete logarithm problem. That is, given the private key d where $P = dG$ and $\texttt{EGexpEnc}_P(m)$, one can compute mG.

LNR-ECDSA recommends the use of Fiat-Shamir's paragidm [5] for ZK proofs. LNR-ECDSA leverages a private multiplication protocol that computes additive shares of the multiplication of two previously shared secrets with privacy but not correctness. That is, assume that each participant P_i holds the additive shares (x_i, y_i) and wants to compute the multiplicative shares z_i such that

$$\sum_{i=1}^{N} z_i = \left(\sum_{i=1}^{N} x_i\right) \cdot \left(\sum_{i=1}^{N} y_i\right) \mod n.$$

This kind of protocol could be constructed using Oblivious-Transfer as in [7] which has low computation cost but higher bandwidth or using Paillier encryption scheme as follows which has higher computation cost but much lower bandwidth. The following protocol is based on Paillier encryption scheme.

1. Each participant P_i generates his Paillier key pair $(\texttt{sk}_\texttt{i}, \texttt{pk}_\texttt{i})$.

2. Each participant P_i computes $c_i = \texttt{Enc}_{\texttt{pk}_\texttt{i}}(x_i)$. In addition, for every j, P_i prepares a non-interactive ZK range proof $\pi^1_{i\to j}$ proving that $x_i \in \mathbb{Z}_n$. For every $j \neq i$, P_i sends $(c_i, \pi^1_{i\to j})$ to P_j.

3. For every $j \neq i$, P_i receives $c_j = \texttt{Enc}_{\texttt{pk}_\texttt{j}}(x_j)$ and $\pi^1_{j\to i}$. P_i uses the homomorphic operations to generate the value $c_{i\to j}$ where $\circ$ is the homomorphic scalar multiplication:

   $$c_{i\to j} = (c_j + 2^{t+l}n) \circ y_i + \delta_{i\to j} = \texttt{Enc}_{\texttt{pk}_\texttt{j}}(x_j y_i + 2^{t+l} n y_i + \delta_{i\to j})$$

   for a randomly chosen $\delta_{i\to j} \in \mathbb{Z}_{n^2 2^{t+l+s}}$. In addition, P_i generates a non-interactive ZK proof $\pi^2_{i\to j}$ of knowledge $(y_i, \delta_{j\to i})$ within $c_{i\to j}$. P_i sends $(c_{i\to j}, \pi^2_{i\to j})$ to P_j.

4. P_i receives $(c_{j\to i}, \pi^2_{j\to i})$ from P_j for every $j \neq i$. P_i verifies the ZK proof, decrypts $x_i y_j + 2^{t+l} n y_j + \delta_{j\to i}$ and adds $(x_i + 2^{t+l}n)2^{t+l}n + 2^{t+l}n^2 2^{t+l+s}$. Finally, P_i sums the results, adds $x_i y_i$ and subtracts all its $\delta_{i\to j}$ values, and reduces the result modulo n. The result is P_i's output z_i.

Next we describe the secure multiplication protocol $\texttt{F}_\texttt{mult}$ that will be used in LNR-ECDSA. The protocol $\texttt{F}_\texttt{mult}$ is initiated as follows:

1. Each participant P_i selects a random $d_i \in_R \mathbb{Z}_n$, computes $P_i = d_i G$, prepares a ZK proof π_i for the knowledge of d_i, and broadcasts $h_i = H(P_i, \pi_i, \theta_i)$ for a random string θ_i.

2. Each participant P_i broadcasts (P_i, π_i, θ_i).

3. Each participant P_i computes the ElGamal (in-the-exponent) public key $P = \sum_{j=1}^{N} P_i$ corresponding to the private key $d = \sum_{j=1}^{N} d_i$. P_i stores $(d_i, P, P_1, \cdots, P_N)$.

The `input-enc` procedure generates an ElGamal in-the-exponent encryption of $a = \sum_{i=1}^{N} a_i \bmod n$ where each P_i holds a_i and P is the ElGamal public key.

1. P_i chooses a random $s_i \in_R \mathbb{Z}_n$ and computes $(U_i, V_i) = \texttt{EGexpEnc}_P(a_i; s_i) = (s_i G, s_i P + a_i G)$. P_i broadcasts $(U_i, V_i; \pi_i)$ where π_i is the ZK proof of the knowledge a_i.

2. P_i computes $U = \sum_{l=1}^{N} U_l, V = \sum_{l=1}^{N} V_l$, and stores $(U, V, a_i, s_i, (U_1, V_1), \cdots, (U_N, V_N))$.

The element-out procedure is used by the participants to obtain $A = aG$ for an input value a that was input. The element-out procedure requires that the input-enc procedure has been complete. Note that a tuple (A, B, C, D) is called a Diffie-Hellman tuple if there is an integer z such that $C = zA$ and $D = zB$.

1. Each participant P_i broadcasts $A_i = a_iG$ and a ZK proof that $(G, P, U_i, V_i - A_i)$ is a Diffie-Hellman tuple.

2. Each participant P_i computes $A = \sum_{l=1}^{N} A_l$.

The secure-mult procedure is used to compute the multiplication $c = ab$ of two shared secrets a and b. Assume that the participants share the secret $a = \sum_{i=1}^{N} a_i$ and $b = \sum_{i=1}^{N} b_i$. In the following steps, each participate computes and outputs the value ab.

1. The participants carry out the input-enc procedure for each of the shares $a = \sum_{i=1}^{N} a_i$ and $b = \sum_{i=1}^{N} b_i$.

2. The participants carry out a private multiplication protocol for $a = \sum_{i=1}^{N} a_i$ and $b = \sum_{i=1}^{N} b_i$. Participant P_i computes the share c_i of $c = ab$ by the end of this protocol.

3. P_i chooses a random s'_i and computes $(E_i, F_i) = (a_iX + s'_iG, a_iY + s'_iP)$. P_i broadcasts (E_i, F_i) and a ZK proof of knowledge (s_i, s'_i, a_i) to all participants.

4. P_i sets $(E, F) = \left(\sum_{l=1}^{N} E_l, \sum_{l=1}^{N} F_l\right)$. P_i chooses a random $\hat{s}_i$ and computes $(A_i, B_i) = \texttt{EGexpEnc}_P(c_i; \hat{s}_i)$. P_i then broadcasts (A_i, B_i) and a ZK proof of knowledge $(c_i, \hat{s}_i)$ to all participants.

5. P_i computes $A = E - \sum_{l=1}^{N} A_l$ and $B = F - \sum_{l=1}^{N} B_l$. All participants run a DH check protocol to check that $B = dA$ where $d = \sum_{l=1}^{N} d_i$.

6. P_i broadcasts c_i and a ZK proof for $((P, A_i, B_i - c_iG), \hat{s}_i)$ to all participants.

7. P_i computes $c = \sum_{l=1}^{N} c_l$.

Now we are ready to describe the scheme LNR-ECDSA. Assume that all participants have the public parameter $(\mathbb{G}, G, n)$ for the ECDSA and know the number N of participants.
LNR-ECDSA Key generation algorithm. The participants jointly generate an ECDSA signature (r, s) on the message M as follows where $m = H(M)$.

1. The participants initialize the secure multiplication protocol $\texttt{F}_{\texttt{mult}}$.

2. P_i selects a random $x_i \in \mathbb{Z}_n$. The ECDSA private key is $x = \sum_{i=1}^{N} x_i$.

3. All participants carry out input-enc procedure. At the end of the input-enc procedure, each participant P_i stores (U, V, x_i, s_i) where (U, V) is an ElGamal encryption of the private key x.

4. All participants carry out element-out procedure to obtain the ECDSA public key xG.

LNR-ECDSA signature generation algorithm.

1. The participants initialize the secure multiplication protocol $\texttt{F}_{\texttt{mult}}$.

2. P_i selects random $k_i, \rho_i \in_R \mathbb{Z}_n$. Let $k = \sum_{i=1}^{N} k_i$ and $\rho = \sum_{i=1}^{N} \rho_i$.

3. All participants carry out the secure-mult procedure to compute the value $\tau = k\rho$.

4. All participants carry out element-out procedure to obtain the value of $kG = \sum_{j=1}^{N} k_jG = (x_1, y_1)$.

5. Each P_i converts x_1 to an integer $\overline{x}_1$ and computes $r = \overline{x}_1 \bmod n$.

6. All participants compute the shares of $m' + xr$ locally.

7. All participants carry out the `secure-mult` procedure to compute the value $\beta = \rho(m' + xr)$.
8. Each P_i computes $s' = \tau^{-1}\beta = k^{-1}(m' + xr) \bmod n$ and $s = \min\{s, n - s\}$.
9. P_i outputs the signature (r, s).

LNR-ECDSA signature verification algorithm. This is the same as the ECDSA signature verification algorithm.

4 Threshold signature scheme BLS

Boneh, Lynn, and Shacham's signature scheme (BLS) [2] has been used in blockchain environments. For example, it is used in the random beacon protocol Randao [4]. BLS scheme is based on gap groups in which the computational Diffie-Hellman problem is hard but the decisional Diffie-Hellman problem can be efficiently solved. Typically the BLS scheme is instantiated in a gap group constructed from non-degenerate, efficiently computable, and bilinear pairings. We first briefly describe the bilinear maps and bilinear map groups. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order n and g_1, g_2 be a generators of $\mathbb{G}_1$, $\mathbb{G}_2$ respectively. A bilinear map is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1. bilinear: for all $g_i, g_j \in \mathbb{G}$, and $x, y \in Z$, we have $\hat{e}(g_i^x, g_j^y) = \hat{e}(g_i, g_j)^{xy}$.
2. non-degenerate: $\hat{e}(g_1, g_2) \neq 1$.

Assume that $\mathbb{G}$ is a multiplicative cyclic group with a generator g of prime order p. We say that the decisional Diffie-Hellman problem is hard if it is difficult to decide whether $c = ab$ given the tuple (g, g^a, g^b, g^c). We say that the computational Diffie-Hellman problem is hard if it is difficult to compute g^{ab} given the tuple (g, g^a, g^b). A group is called Gap Diffie-Hellman (GDH) group if the the computational Diffie-Hellman problem is hard though the decisional Diffie-Hellman problem is easy. The GDH Signature Scheme [2] consists three algorithms.
GDH Key generation algorithm: Pick a random x and compute $v = g_2^x$. The public key is $v \in \mathbb{G}_2$ and the private key is x.
GDH signature generation algorithm: Given a secret key x and a message M, compute $h = H(M) \in \mathbb{G}_1$ and $\sigma = h^x$. The signature on M is $\sigma \in \mathbb{G}_1$.
GDH signature verification algorithm: Given a public key v, a message M, and a signature σ, compute $h = H(M)$, and verify that $\hat{e}(\sigma, g_2) = \hat{e}(h, v)$.

It is shown in [1] that the GDH signature scheme has the aggregation property: Given N signatures on N distinct messages from N distinct users, it is possible to aggregate all these signatures into a single short signature. This aggregated signature and the N original messages can be used to convince the verifier that the N users did indeed sign the N original messages. Specifically, the aggregation of the signatures works as follows.
Aggregation: For the aggregating subset of users U, assign to each user an index i, ranging from 1 to $k = |U|$. Each user $u_i \in U$ provides a signature σ_i on a message M_i of his choice. The messages M_i must all be distinct. Compute the aggregate signature $\sigma = \prod_{i=1}^{k} \sigma_i$.
Aggregate Verification: Given an aggregate signature σ for an aggregating subset of users U, indexed as before, and given the original messages M_i and public keys v_i for all users u_i. To verify the aggregate signature σ

1. ensure that the messages M_i are all distinct, and reject otherwise; and
2. compute $h_i = H(M_i)$ for $1 \leq i \leq k = |U|$, and accept if $\hat{e}(\sigma, g_2) = \prod_{i=1}^{k} \hat{e}(h_i, v_i)$.

These properties can be used to design threshold signature schemes.

References

[1] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Eurocrypt*, pages 416–432. Springer, 2003.

[2] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of cryptology*, 17(4):297–319, 2004.

[3] I. Damgård, M. Keller, E. Larraia, C. Miles, and N.P. Smart. Implementing aes via an actively/covertly secure dishonest-majority mpc protocol. In *International Conference on Security and Cryptography for Networks*, pages 241–263. Springer, 2012.

[4] Ethereum. Randao: A DAO working as RNG of ethereum, 2017. `https://github.com/randao/randao` and `https://www.randao.org/whitepaper/Randao_v0.85_en.pdf`.

[5] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194. Springer, 1986.

[6] R. Gennaro and S. Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proc, 2018 ACM CCS*, pages 1179–1194, 2018.

[7] N. Gilboa. Two party rsa key generation. In *Annual International Cryptology Conference*, pages 116–129. Springer, 1999.

[8] C.F. Kerry and C.R. Director. FIPS PUB 186-4 federal information processing standards publication digital signature standard (dss). 2013.

[9] Y. Lindell and A. Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proc. 2018 ACM CCS*, pages 1837–1854, 2018.

[10] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238. Springer, 1999.

[11] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.