Framework of Threshold Signature Scheme

AMIS

November 17, 2022

Contents

1. ECDSA : CGGMP

We implement Key-generation (ref. subsection 1.1), Auxiliary Info. & Key Refresh in Three Rounds (ref. subsection 1.3), and signing (ref. subsection 1.2) in the CGGMP. According to the section 1.2.8 in the paper, it is possible to extend the original protocol: n-out-of-n multi-party signing to a general t-out-of-n threshold signing for $t < n$ by using Birkhoff interpolation(i.e. a generalization of Lagrange interpolation ref. subsection A.3). For the sake of avoiding possible issues described in the paper, our sign protocols are an interactive version. More precisely, we combine two stages: pre-signing and signing.

Notations. Throughout the document $\mathcal{G}$ denotes a group of prime order q (i.e. sometimes denoted by $N_{\mathcal{G}}$), and F_q the finite field with q elements. We let $\mathbb{Z}, \mathbb{N}$ denote the set of integers and natural number, respectively. For $\ell \in \mathbb{Z}$, we let $\pm\ell$ denote the interval of integers $\{-|\ell|, ..., 0, ..., |\ell|\}$. Finally, let $\gcd : \mathbb{N}^2 \to \mathbb{N}$ and $\phi : \mathbb{N} \to \mathbb{N}$ denote the gcd operation and Euler's phi function, respectively. We also let H_q be a function from $\{0,1\}^*$ to $\pm q$ and H a hash function.

1.1. Distributed Key Generation

Distributed key generation (Abbrev. DKG) is a cryptographic process in which multiple parties participate to the calculation of a shared public and a private key set. The main merit of distributed key generation is not rely necessarily on trusted third parties.

Let $\mathcal{G}$ be an elliptic curve group with order $N_{\mathcal{G}}$ and G be a fixed point of $\mathcal{G}$. For every $S \subset [1, ..., n]$ be the set of players participating in the signature protocol with $|S| \geq t$, we let $bk_i := (\text{xcoord}_i, n_i)$ be the Birkhoff parameter and $w_{i,S}$(abbrev. w_i) each player in S be the appropriate Birkhoff coefficients associated with $\{bk_i\}_{i \in S}$ (ref. subsection A.3). Here xcoord_i is the x-coordinate and n_i is the level of the participant $\mathcal{P}_i$. In particular, when the level n_i is zero, then Birkhoff interpolation is reduced to the Lagrange Interpolation.

In our implementation, the threshold t, the number of total participants n, and the level n_i of each participants $\mathcal{P}_i$ are determined before someone performs the following DKG protocol:

Remark 1.

1. *This version of DKG protocol is an analogous of Section 3.1 Key generation. We modify this algorithm to fit a threshold cryptography setting. In concrete, there exists two differences:*
   a. *Replace the generating method of secret key $s_1 + s_2 + ... + s_n = s$ with Birkhoff interpolation (ref. subsection A.3). Here s_i is a participant $\mathcal{P}_i$'s share and Birkhoff interpolation is a general type of Lagrange Interpolation. Furthermore, we use Feldman commitment to verify the correctness of shares, which is the same as GG18.*
   b. *In the last, we check that these shares can recover the public key because each participant has all $s_i \cdot G$.*

Algorithm 1 Distributed Key Generation

```
**Input: a sid, positive integers $t$, $n$ and a non-negative integer $n_i$.**
**Output: two sets of positive integers $\{n_i\}_{i=1}^n$ $\{x_i\}_{i=1}^n$, a share $s_i$, S $:= (s_1 \cdot G, ..., s_n \cdot G)$, rid, and the unique corresponding public Key $P$.**
1: Each participant $\mathscr{P}_i$
    a. randomly chooses $x$-coordinate $x_i \in [1, N_{\mathscr{G}} - 1]$, and $\{u_{i,j}\}_{j=0}^{t-1} \in [0, N_{\mathscr{G}} - 1]$ (i.e. $f_i(x) := \sum_{j=0}^{t-1} u_{i,j} x^j$ ).
    b. randomly chooses rid$_i$ with its bitLength which equals the bitLength of $N_{\mathscr{G}}$.
    c. randomly chooses $a_i \in [1, N_{\mathscr{G}} - 1]$ and computes Schnorr commitment $A_i := a_i \cdot G$.
    d. computes $u_{i,0} \cdot G$ and hash commitment $[\mathscr{C}_i, \mathscr{D}_i] = \mathrm{H}(\mathrm{sid}, (x_i, n_i), \mathrm{rid}_i, u_{i,0} \cdot G, A_i)$ (ref. subsection A.1).
    e. computes the associated Birkhoff coefficient $w_i$ (ref. Algorithm 17) and Feldman's commitment $F_{i,j} := u_{i,j} \cdot G$ (ref. subsection A.2).
    e. broadcasts $x$-coordinate $x_i$, the level $n_i$, Feldman commitment, and the hash commitment $\mathscr{C}_i$.
2: Each participant $\mathscr{P}_i$
    a. computes the values $f_i^{(n_k)}(x_k)$ for all $1 \le k \le n$.
    b. sends own decommitment $\mathscr{D}_i$ and $f_j^{(n_k)}(x_k)$ to $\mathscr{P}_j$.
3: Each participant $\mathscr{P}_i$
    a. verifies the decommitment $\mathscr{D}_i$ (ref. Algorithm 13).
    b. verifies the Feldman commitment for $f_k^{(n_i)}(x_i)$ for all $1 \le k \ne i \le n$ (ref. Algorithm 16).
    c. computes rid $:= \oplus_i \mathrm{rid}_i$. Here $\oplus$ is the xor operation.
    d. computes the public key $\sum_{i=1}^n u_{i,0} \cdot G$, the own share $s_i := \sum_k f_k^{(n_i)}(x_i)$, the point $s_i \cdot G$ and Schnorr's proof of $s_i \cdot G$ (ref. Algorithm 27).
    e. sends $s_i \cdot G$ and its Schnorr's proof of $\psi_i := \mathscr{M}(\mathrm{prove}, \Pi^{\mathrm{sch}}, (\mathrm{sid}, (x_i, n_i), \mathrm{rid}), si \cdot G; s_i, a_i)$.
4: Each participant $\mathscr{P}_i$
    a. verifies $A_j = \hat{a}_j \cdot G$, where $\hat{a}_j$ is $a_j$ of the Schnorr's proof $\psi_j = \mathscr{M}(....; s_i, a_i)$.
    b. verifies the Schnorr's proof $\psi_j$ for all $1 \le j \ne i \le n$ and the Public key is given by by $\sum_{k=1}^n w_k \cdot (s_k \cdot G) = \sum_{k=1}^n u_{k,0} \cdot G$.
```

1.2. HIERARCHICAL THRESHOLD SIGNATURE

A (t, n)-threshold signature scheme is a digital signature scheme where any t or more signers of a group of n signers can produce signatures. This implementation is the sign protocols in Figure 7, 8, 9, 10 CGGMP. Let $\mathcal{G}$ be an elliptic curve group with order $N_{\mathcal{G}}$ and G be a fixed point of $\mathcal{G}$. For every $S \subset [1, ..., n]$ be the set of players participating in the signature protocol with $|S| \geq t$, we let $bk_i := (\text{xcoord}_i, n_i)$ be the Birkhoff parameter and $w_{i,S}$(abbrev. w_i) each player in S be the appropriate Birkhoff coefficients associated with $\{bk_i\}_{i \in S}$ (ref. subsection A.3). Here xcoord_i is the x-coordinate and n_i is the level of the Participant $\mathcal{P}_i$. Set $x_i := w_i \cdot s_i$ called "modified share ", where s_i is the share of $\mathcal{P}_i$. Let $\bigodot$ be a homo-multiplication and $\bigoplus$ be a homo-addition. Our protocols and the protocols described in CGGMP are the same except for:

1. The input x_i in the paper is $w_i \cdot s_i$ (i.e. In fact, it does not change the protocols).
2. In round 2, we replace $-\beta_{i,j}, -\hat{\beta}_{i,j}$ with $\beta_{i,j}$ and $\hat{\beta}_{i,j}$. This change only effects the error handles (i.e. $D_{i,j} \to D_{i,j}^{-1}$ and $\hat{D}_{i,j} \to \hat{D}_{i,j}^{-1}$).

Algorithm 2 Sign 3 Round: Part 1

Input: a Hash message m, a threshold t, the modified share x_i, the public Key P, a Paillier secret key p_i, q_i, partial public keys (X_i), Pedersen parameters (N_i, s_i, t_i), a set of bk parameter bk_i.

Output: (r, s) **or** $\perp$.

```
1: Each participant $\mathcal{P}_i$
    a. randomly chooses $k_i$ and $\gamma_i \in [0, N_{\mathcal{G}} - 1]$ and sets $G_i := \text{enc}_i(\gamma_i; \nu_i)$, $K_i := \text{enc}_i(k_i; \rho_i)$.
    b. computes $\psi^0_{j,i} := \mathcal{M}(\text{prove}, \Pi^{\text{enc}}_j, (\text{ssid}, bk_i), (I_\epsilon, K_i); k_i, \rho_i)$ for every $j \neq i$.
    c. broadcasts $K_i, G_i$ and sends $\psi^0_{j,i}$ to the Participant $\mathcal{P}_j$.
2: Each participant $\mathcal{P}_i$
    a. verifies the proof $\psi^0_{i,j}$.
    b. randomly chooses $r_{i,j}, s_{i,j}, \hat{r}_{i,j}, \hat{s}_{i,j} \in \mathbb{Z}^*_{N_j}$, $\beta_{i,j}, \hat{\beta}_{i,j} \in \mathcal{J}$ for $j \neq i$.
    c. computes $D_{j,i} := (\gamma_i \bigodot K_j) \bigoplus \text{enc}_j(\beta_{i,j}, s_{i,j})$ and $F_{j,i} = \text{enc}_i(\beta_{i,j}, r_{i,j})$.
    d. computes $\hat{D}_{j,i} := (x_i \bigodot K_j) \bigoplus \text{enc}_j(\hat{\beta}_{i,j}, \hat{s}_{i,j})$ and $\hat{F}_{j,i} = \text{enc}_i(\hat{\beta}_{i,j}, \hat{r}_{i,j})$.
    e. computes $\psi_{j,i} := \mathcal{M}(\text{prove}, \Pi^{\text{aff-g}}_j, (\text{ssid}, bk_i), (I_\epsilon, \mathcal{J}_\epsilon, D_{j,i}, K_j, F_{j,i}, \Gamma_i); \gamma_i, \beta_{i,j}, s_{i,j}, r_{i,j})$.
       Here $\Gamma := \gamma \cdot G$.
    f. computes $\hat{\psi}_{j,i} := \mathcal{M}(\text{prove}, \Pi^{\text{aff-g}}_j, (\text{ssid}, bk_i), (I_\epsilon, \mathcal{J}_\epsilon, \hat{D}_{j,i}, K_j, \hat{F}_{j,i}, X_i); x_i, \hat{\beta}_{i,j}, \hat{s}_{i,j}, \hat{r}_{i,j})$.
    g. computes $\psi'_{j,i} := \mathcal{M}(\text{prove}, \Pi^{\log *}_j, (\text{ssid}, bk_i), (I_\epsilon, G_i, \Gamma_i, G); \gamma_i, \nu_i)$.
    h. sends $\Gamma_i$, $D_{j,i}, F_{j,i}, \hat{D}_{j,i}, \hat{F}_{j,i}, \psi_{j,i}, \hat{\psi}_{j,i}, \psi'_{j,i}$ to each $\mathcal{P}_j$.
```

Algorithm 3 Sign 3 Round: Part 2

```
3: Each participant $\mathscr{P}_i$
    a. verifies the proof $\psi_{i,j}$, $\hat{\psi}_{i,j}$, and $\psi'_{i,j}$.
    b. sets $\alpha_{i,j} := \mathrm{dec}_i(D_{i,j})$ and $\hat{\alpha}_{i,j} = \mathrm{dec}_i(\hat{D}_{i,j})$ and
        $$\begin{cases} \delta_i = \gamma_i k_i + \sum_{j \neq i}(\alpha_{i,j} + \beta_{i,j}) \mod N_{\mathscr{G}}, \\ \chi_i = x_i k_i + \sum_{j \neq i}(\hat{\alpha}_{i,j} + \hat{\beta}_{i,j}) \mod N_{\mathscr{G}} \end{cases}$$
       for every $j \neq i$. Here $\Gamma := \sum_j \Gamma_j$ and $\Delta_i := \Gamma^{k_i}$.
    c. computes $\psi"_{j,i} := \mathscr{M}(\mathrm{prove}, \Pi_j^{\log *}, (\mathrm{ssid}, bk_i), (I_\epsilon, K_i, \Delta_i, \Gamma); k_i, \rho_i)$ for all $j \neq i$.
    d. sends $\delta_i$ and sends $\Delta_i, \psi"_{j,i}$ to each $\mathscr{P}_j$.
4: Each participant $\mathscr{P}_i$
    a. verifies the proof $\psi"_{i,j}$.
    b. verifies $\delta \cdot G = \prod_j \Delta_j$, where $\delta := \sum_j \delta_j$. In case of failure do:
        – For each $j \neq i$, reproves that $\{D_{j,i}\}_{j \neq i}$ are well-formed according to $\Pi_\ell^{\text{aff-g}}$ for $\ell \neq j, i$.
        – computes $H_i = \mathrm{enc}_i(k_i \gamma_i)$ and proves in ZK that $H_i$ is well formed wrt $K_i$ and $G_i$ in $\Pi^{\mathrm{mul}}$.
        – proves in ZK that $\delta_i$ is the plaintext value mod $N_{\mathscr{G}}$ of the ciphertext obtained as $H_i \prod_{j \neq i} D_{i,j}^{-1} F_{i,j}$ according to $\Pi_\ell^{\mathrm{dec}}$ for $\ell \neq i$.
        – broadcasts the above proofs.
    c. computes $R := \delta^{-1} \cdot \Gamma$, $r := R|_{\text{x-axis}}$ and $\sigma_i := km + r\chi$.
    d. sends $\sigma_i$ to all.
5: Each participant $\mathscr{P}_i$
    a. computes $\sigma := \sum_j \sigma_j$.
    b. verifies $(r, \sigma)$ is a valid signature. If not, do:
        – For each $j \neq i$, reproves that $\{\hat{D}_{j,i}\}_{j \neq i}$ are well-formed according to $\Pi_\ell^{\text{aff-g}}$ for $\ell \neq j, i$.
        – computes $\hat{H}_i = \mathrm{enc}_i(k_i x_i)$ and proves in ZK that $\hat{H}_i$ is well formed wrt $K_i$ and $X_i$ in $\Pi^{\mathrm{mul}*}$.
        – proves in ZK that $\delta_i$ is the plaintext value mod $N_{\mathscr{G}}$ of the ciphertext obtained as $K_i^m \left(\hat{H}_i \prod_{j \neq i} \hat{D}_{i,j}^{-1} \hat{F}_{i,j}\right)^r$ according to $\Pi_\ell^{\mathrm{dec}}$ for $\ell \neq i$.
        – broadcasts the above proofs.
```

Remark 2.

1. *In the sign protocol, the parameters of Paillier homomorphism (ref. subsection A.4.1) are* $g = n+1$, $\lambda = \phi(n)$, $\mu = \phi(n)^{-1} \mod n$. *Here* $n = pq$ *is the product of two primes.*

2. *Compare Algorithm 23 and Algorithm 22, the computation of* $a \bigodot b := a^b \mod n^2$ *and* $a \bigoplus b := ab \mod n^2$ *without salt* r^n. *Because* $s_{i,j}$ *and* $\hat{s}_{i,j}$ *are randomly chosen,* $D_{j,i}$ *and* $\hat{D}_{j,i}$ *are still indistinguishable.*

Algorithm 4 Sign 6 Round: Part 1

```
    Input: a Hash message $m$, a threshold $t$, the modifying share $x_i$, the public Key $P$, $y_i$, a Paillier secret key $p_i, q_i$, partial public keys $(X_i)$, Pedersen parameters $(N_i, s_i, t_i)$, a set of bk paramater $bk_i$.

    Output: $(r, s)$ or $\bot$.
1: Each participant $\mathscr{P}_i$
    a. randomly chooses $k_i, \gamma_i, b_i \in [0, N_{\mathscr{G}} - 1]$, $\rho_i, \nu \in \mathbb{Z}^*_{N_i}$, $\nu \in \{0,1\}^\kappa$, and sets $Z_i := (b_i \cdot G, k_i \cdot G + b_i \cdot Y_i)$. Here $Y_i = y_i \cdot G$.
    b. computes $K_i := \mathrm{enc}_i(k_i; \rho_i)$, $G_i := \mathrm{enc}_i(\gamma_i; \nu_i)$, $\Gamma_i = \gamma_i \cdot G$
    c. computes $\psi^0_{j,i} := \mathscr{M}(\mathrm{prove}, \Pi^{\text{enc-elg}*}_j, (\mathrm{ssid}, bk_i), (I_\epsilon, K_i, Y_i, Z); k_i, \rho_i, b_i)$ for every $j \neq i$.
    c. broadcasts $K_i, G_i, Z_i$ and sends $\psi^0_{j,i}$ to the Participant $\mathscr{P}_j$.
2: Each participant $\mathscr{P}_i$
    a. verifies the proof $\psi^0_{i,j}$.
    b. randomly chooses $r_{i,j}, s_{i,j}, \hat{r}_{i,j}, \hat{s}_{i,j} \in \mathbb{Z}^*_{N_j}$, $\beta_{i,j}, \hat{\beta}_{i,j} \in \mathscr{J}$ for $j \neq i$.
    c. computes $D_{j,i} := (\gamma_i \bigodot K_j) \bigoplus \mathrm{enc}_j(\beta_{i,j}, s_{i,j})$ and $F_{j,i} = \mathrm{enc}_i(\beta_{i,j}, r_{i,j})$.
    d. computes $\hat{D}_{j,i} := (x_i \bigodot K_j) \bigoplus \mathrm{enc}_j(\hat{\beta}_{i,j}, \hat{s}_{i,j})$ and $\hat{F}_{j,i} = \mathrm{enc}_i(\hat{\beta}_{i,j}, \hat{r}_{i,j})$.
    e. computes $\psi_{j,i} := \mathscr{M}(\mathrm{prove}, \Pi^{\text{aff-p}}_j, (\mathrm{ssid}, bk_i), (I_\epsilon, \mathscr{J}_\epsilon, D_{j,i}, K_j, F_{j,i}, G_i); \gamma_i, \beta_{i,j}, s_{i,j}, \nu_i)$.
    f. computes $\hat{\psi}_{j,i} := \mathscr{M}(\mathrm{prove}, \Pi^{\text{aff-g}}_j, (\mathrm{ssid}, bk_i), (I_\epsilon, \mathscr{J}_\epsilon, \hat{D}_{j,i}, K_j, \hat{F}_{j,i}, X_i); x_i, \hat{\beta}_{i,j}, \hat{s}_{i,j}, \hat{r}_{i,j})$.
    g. sends $D_{j,i}, F_{j,i}, \hat{D}_{j,i}, \hat{F}_{j,i}, \psi_{j,i}, \hat{\psi}_{j,i}$ to each $\mathscr{P}_j$.
3: Each participant $\mathscr{P}_i$
    a. verifies the proof $\psi_{i,j}$, and $\hat{\psi}_{i,j}$.
    b. randomly chooses $\hat{b}_i \in [0, N_{\mathbb{G}} - 1]$.
    c. sets $\alpha_{i,j} := \mathrm{dec}_i(D_{i,j})$ and $\hat{\alpha}_{i,j} = \mathrm{dec}_i(\hat{D}_{i,j})$ and
        $$\begin{cases} \delta_i = \gamma_i k_i + \sum_{j \neq i}(\alpha_{i,j} + \beta_{i,j}) \mod N_{\mathscr{G}}, \\ \chi_i = x_i k_i + \sum_{j \neq i}(\hat{\alpha}_{i,j} + \hat{\beta}_{i,j}) \mod N_{\mathscr{G}}, \\ \hat{Z}_i = (\hat{b}_i \cdot G, \chi_i \cdot G + \hat{b}_i \cdot Y_i). \end{cases}$$
       for every $j \neq i$.
    d. sends $\delta_i, \hat{Z}_i$ to all.
4: Each participant $\mathscr{P}_i$
    a. computes $\psi'_{j,i} := \mathscr{M}(\mathrm{prove}, \Pi^{\log*}_j, (\mathrm{ssid}, bk_i), (I_\epsilon, G_i, \Gamma_i, G); \gamma_i, \nu_i)$ for $j \neq i$. Here $\Gamma_i := \gamma_i \cdot G$.
    b. sends $\Gamma_i$ and $\psi'_{j,i}$ to each $\mathscr{P}_j$.
```

Algorithm 5 Sign 6 Round: Part 2

```
5: Each participant $\mathscr{P}_i$
    a. verifies the proof $\psi'_{i,j}$.
    b. computes $\psi_i := \mathcal{M}(\text{prove}, \Pi_j^{\text{elog}}, (\text{ssid}, bk_i), (Y_i, z_i, \Delta_i, \Gamma); k_i, b_i)$ for $j \neq i$. Here $\Gamma := \sum_j \gamma_j$ and $\Delta_i := \Gamma^{k_i}$.
    c. sends $\Delta_i, \psi_i$ to all.
6: Each participant $\mathscr{P}_i$
    a. verifies the proof $\psi"_{i,j}$.
    b. verifies $\delta \cdot G = \prod_j \Delta_j$, where $\delta := \sum_j \delta_j$ and $R := \Gamma^{\delta^{-1}}$. (If fail then red alert ♯1).
    c. computes $S_i := R^{\chi_i}$ and $\{\bar{R}_j := \Delta_j^{\delta^{-1}}\}$ for all $1 \leq j \leq n$.
    d. computes $\pi_i := \mathcal{M}(\text{prove}, \Pi^{\text{elog}}, (\text{ssid}, i), (Y_i, \hat{Z}_i, S, R); \chi_i, \lambda_i)$.
    e. sends $S_i, \pi_i$ to all.
7: Each participant $\mathscr{P}_i$
    a. verifies the proof $\pi_j$ and $\sum_j S_j = X$ (if the latter part fail then red alert ♯2).
    b. computes $R := \delta^{-1} \cdot \Gamma$, $r := R|_{\text{x-axis}}$ and $\sigma_i := km + r\chi$.
    c. sends $\sigma_i$ to all.
8: Each participant $\mathscr{P}_i$
    a. verifies $(r, \sigma)$ is a valid signature, where $\sigma := \sum_j \sigma_j$.
    b. If the above fails, check $\sigma_j \cdot R = m \cdot \bar{R}_j + r \cdot S_j$ for all $j \neq i$.
```

Algorithm 6 Red Alert 1 : Nonce-Reveal Fail

```
1: Each participant $\mathscr{P}_i$
    a. computes $\psi_i := \mathcal{M}(\text{prove}, \Pi^{\text{Nth}}, (\text{ssid}, bk_i), (N_i, \rho_i^N); \rho_i)$.
    b. computes $\psi_{i,j} := \mathcal{M}(\text{prove}, \Pi^{\text{Nth}}, (\text{ssid}, bk_i), (N_i, \mu_{i,j}^N); \mu_{i,j})$. Here $\{\mu_{i,j} := ((1+N_i)^{-\alpha_{i,j}} D_{i,j})^{N_i^{-1}} \mod N_i^2\}_{j \neq i}$.
    c. broadcasts $k_i, \rho_i^{N_i}, \psi_i, \gamma_i, (\alpha_{i,j}, \mu_{i,j}^N, \psi_{i,j})_{i \neq j}$.
2: Each participant $\mathscr{P}_i$, upon receiving $k_j, \tilde{\rho}_j, \psi_j, \gamma_j, (\alpha_{j,k}, \mu_{j,k}^N, \psi_{j,k})_{k \neq j}$ from $\mathscr{P}_j$, do:
    a. verifies the proofs $\psi_j = \mathcal{M}(\text{verify}, \Pi^{\text{Nth}}, (\text{ssid}, bk_j), (N_j, \tilde{\rho}_j))$ and $K_j = (1+N_j)^{k_j} \tilde{\rho}_j \mod N_j^2$.
    b. verifies the proofs $\psi_{j,k} = \mathcal{M}(\text{verify}, \Pi^{\text{Nth}}, (\text{ssid}, bk_j), (N_j, \mu_{j,k}^N))$ for all $k \neq j$, and $D_{j,i} = (1+N_j)^{\alpha_{j,i}} \tilde{\mu}_{j,i} \mod N_j^2$.
    c. verifies $\gamma_j \cdot G = \Gamma_j$ and $\delta_j = k_j \gamma_j + \sum_{\ell \neq j} (\alpha_{j,\ell} + k_\ell \gamma_j - \alpha_{\ell,j}) \mod N_{\mathscr{G}}$.
    d. reports $\mathscr{P}_j$ as corrupt, in case of failure.
```

Algorithm 7 Red Alert 2 : Pseudo-Key Reveal Fail

```
1: Each participant $\mathscr{P}_i$
    a. computes $\psi'_i := \mathcal{M}(\text{prove}, \Pi^{\log}, (\text{ssid}, i), (\hat{b}_i \cdot G, Y_i, \tilde{Y}_i); \hat{b}_i, y_i)$. Here $\tilde{Y}_i = b_i \cdot Y_i$.
    b. computes $\psi_i := \mathcal{M}(\text{prove}, \Pi^{\text{Nth}}, (\text{ssid}, i), (N_i, \hat{\rho}_i^N); \rho_i)$.
    c. computes $\psi_{i,j} := \mathcal{M}(\text{prove}, \Pi^{\text{Nth}}, (\text{ssid}, i), (N_i, \mu_{i,j}^N); \mu_{i,j})$. Here $\{\mu_{i,j} := ((1+N_i)^{-\hat{\alpha}_{i,j}} \hat{D}_{i,j})^{N_i^{-1}} \mod N_i^2\}_{j \neq i}$.
    d. broadcasts $k_i, \rho_i^{N_i}, \psi_i, \tilde{Y}_i, \psi'_i, (\hat{\alpha}_{i,j}, \hat{\mu}_{i,j}^N, \psi_{i,j})_{i \neq j}$.
2: Each participant $\mathscr{P}_i$, when obtaining $k_j, \tilde{\rho}_j, \psi_j, \tilde{Y}_j, \psi'_j, (\hat{\alpha}_{j,k}, \hat{\mu}_{j,k}^N, \psi_{j,k})_{k \neq j}$, interpret $\hat{Z}_j = (B_j, M_j)$ and do:
    a. verifies the proofs $\psi'_j = \mathcal{M}(\text{verify}, \Pi^{\log}, (\text{ssid}, bk_j), (B_j, Y_j, \tilde{Y}_j))$.
    b. verifies the proofs $\psi_j = \mathcal{M}(\text{verify}, \Pi^{\text{Nth}}, (\text{ssid}, bk_j), (N_j, \tilde{\rho}_j))$ and $K_j = (1+N_j)^{k_j} \tilde{\rho}_j \mod N_j^2$.
    c. verifies the proofs $\psi_{j,k} = \mathcal{M}(\text{verify}, \Pi^{\text{Nth}}, (\text{ssid}, bk_j), (N_j, \mu_{j,k}^N))$ for all $k \neq j$ and $\hat{D}_{j,i} = (1+N_j)^{\hat{\alpha}_{j,i}} \tilde{\mu}_{j,i} \mod N_j^2$.
    d. verifies $M_j - \tilde{Y}_j = k_j \cdot X_j + \sum_{\ell \neq j} \left(\hat{\alpha}_{j,\ell} \cdot G + k_\ell \cdot X_j - \hat{\alpha}_{\ell,j} \cdot G\right)$.
    e. reports $\mathscr{P}_j$ as corrupt, in case of failure.
```

Remark 3.

1. The original paper checks $D_{j,k} = (1+N_j)^{\hat{\alpha}_{j,k}}\tilde{\mu}_{j,k} \mod N_j^2$ *for* $k \neq j$*. However, the participant* $\mathscr{P}_i$ *does not have* $D_{j,k}$ *for* $k \neq i$*. Thus, this verification should be* $D_{j,i} = (1+N_j)^{\hat{\alpha}_{j,i}}\tilde{\mu}_{j,i} \mod N_j^2$*. The condition* $D_{j,i} = (1+N_j)^{\alpha_{j,i}}\tilde{\mu}_{j,i} \mod N_j^2$ *is similar to modify.*

1.3. Auxiliary Info. & Key Refresh in Three Rounds

This implementation appears in Figure 6 in CGGMP. Let $\mathcal{G}$ be an elliptic curve group with order $N_{\mathcal{G}}$ and G be a fixed point of $\mathcal{G}$. Now we point out the difference between our implementation with the protocol in CGGMP:

a. Replace the generating method of secret key $x_1 + x_2 + ... + x_n = 0$ with Birkhoff interpolation (ref. subsection A.3). Here Birkhoff interpolation is a general type of Lagrange Interpolation. Furthermore, we use Feldman commitment to verify the correctness of shares, which is the same as Section 5.2.

The following implementation of the reshare is the standard algorithm:

Algorithm 8 Auxiliary Info. & Key Refresh in Three Rounds: Part 1

```
    Input: a threshold $t$, the public Key $P$, a set of levels $\{n_i\}_{i=1}^{n}$, a set of $x$-coordinate $\{x_i\}_{i=1}^{n}$ and a share $s_i$.
    Output: a new share $\tilde{s}_i$, a $y_i$, a Paillier secret key $p_i, q_i$, partial public keys $(\tilde{X}_i)$, Pedersen parameters $(N_i, s_i, t_i)$.
1: Each participant $\mathscr{P}_i$
    a. randomly chooses two $4\kappa$-bit long safe primes $p_i, q_i$. Set $N_i := p_i q_i$.
    b. randomly chooses $y_i \in \mathbb{F}_q$, sets $Y_i = y_i \cdot G$, and $(B_i, \tau) = \mathscr{M}(\text{com}, \Pi^{\text{sch}})$, where $B_i = \tau \cdot G$.
    c. randomly chooses $u_{i,0} = 0$, and $\{u_{i,j}\}_{j=1}^{t-1} \in [0, N_{\mathcal{G}} - 1]$ (i.e. $f_i(x) := \sum_{j=0}^{t-1} u_{i,j} x^j$ ).
    d. computes Birkhoff coefficient $w_i$ (ref. Algorithm 17) and Feldman's commitment $F_{i,j} := u_{i,j} \cdot G$ (ref. subsection A.2).
    e. randomly chooses $r \in \mathbb{Z}_{N_i}^*$, $\lambda \in \mathbb{Z}_{\phi(N_i)}$, set $t_i = r^2 \mod N_i$ and $s_i = t_i^{\lambda} \mod N_i$.
    f. computes $\hat{\psi}_i = \mathscr{M}(\text{prove}, \Pi^{\text{prm}}, (ssid, bk_i), (N_i, s_i, t_i); \lambda)$. Here $bk_i$ is the Birkhoff parameter of $\mathscr{P}_i$.
    g. randomly chooses $\tau_j$ with $A_i^j = \tau_j \cdot G$ for $1 \le j \ne i \le n$. Set $A_i := (A_i^j)_j$.
    h. randomly chooses $\rho_i, u_i \in \{0,1\}^{\kappa}$, and compute $[\mathscr{C}_i, \mathscr{D}_i] = H(\text{ssid}, bk_i,$ Feldman's commitment $, A_i, Y_i, B_i, N_i, s_i, t_i, \hat{\psi}_i, \rho_i, u_i)$.
    c. broadcasts the Hash commitments $C_i$.
2: Each participant $\mathscr{P}_i$
    a. when obtaining Hash commitment $C_j$ from all $P_j$, sends the decommitment $\mathscr{D}_i$ to all participants.
```

Algorithm 9 Auxiliary Info. & Key Refresh in Three Rounds: Part 2

```
3: Each participant $\mathscr{P}_i$
    a. verifies that the hash commitment $[\mathscr{C}_j, \mathscr{D}_j]$ is correct, the bit-length of $N_j \geq 8\kappa$, and $\mathscr{M}(\text{verify}, \Pi^{\text{prm}}, (\text{ssid}, bk_j), (N_j, s_j, t_j), \hat{\psi}_j)$ is true.
    b. computes $\rho := \oplus_j \rho_j$
    c. computes $\psi_i := \mathscr{M}(\text{prove}, \Pi^{\text{mod}}, (\text{ssid}, \rho, bk_i), N_i; (p_i, q_i))$, and $\phi_{j,i} := \mathscr{M}(\text{prove}, \Pi^{\text{fac}}, (\text{ssid}, \rho, bk_i), N_i, \kappa; (p_i, q_i))$ for all $j \neq i$.
    d computes $C_i^j := \text{enc}_j(f_i^{(n_j)}(x_j))$ and $\psi_i^j := \mathscr{M}(\text{prove}, \Pi^{\text{sch}}, (\text{ssid}, \rho, bk_i), X_i^j; ((f_i^{(n_j)}(x_j), \tau_j))$. Here $X_i^j := f_i^{(n_j)}(x_j) \cdot G$.
    e computes $\pi = \mathscr{M}(\text{prove}, \Pi^{\text{sch}}, (\text{ssid}, \rho, bk_i), Y_i; (y_i, \tau))$.
    f. sends $\psi_i, \phi_{j,i}, \pi_i, C_i^j, \psi_i^j$ to all.
4: Each participant $\mathscr{P}_i$
    a. decrypts ciphertext to get $f_j^{(n_i)}(x_i)$ for all $1 \leq j \neq i \leq n$.
    b. verifies the Feldman commitment for $f_j^{(n_i)}(x_i)$ for all $1 \leq j \neq i \leq n$ (ref. Algorithm 16).
       If "Dec Error", then computes $\mu := (C_j^i \cdot (1+N_i)^{-f_i^{(n_j)}(x_j)})^{1/N_i} \mod N_i^2$ and broadcast $C_j^i, f_j^{(n_i)}(x_i), \mu$.
    c. verifies the proofs $\Pi^{\text{mod}}$ and $\Pi^{\text{fac}}$.
    d. interprets $\pi_j = (\hat{B}_j, ....)$ and verifies $\hat{B}_j = B_j$ and the proof $\mathscr{M}(\text{verify}, \Pi^{\text{sch}}, (\text{ssid}, \rho, bk_j), Y_j; (y_i, \tau))$.
    e. interprets $\psi_j^k = (\hat{A}_j^k, ....)$, and verifies $\hat{A}_j^k = A_j^k$ and the proof $\mathscr{M}(\text{verify}, \Pi^{\text{sch}}, (\text{ssid}, \rho, bk_j), X_j^i; ((f_j^{(n_i)}(x_i), \tau_j))$.
    f. sets $\tilde{s}_i := s_i + \sum_{j=1}^{n} f_j^{(n_i)}(x_i) \mod N_{\mathscr{G}}$ and $\tilde{X}_j := X_j + \sum_{k=1}^{n} f_k^{(n_j)}(x_j)$ for $1 \leq j \neq i \leq n$.
    g. verifies the Public key is given by $P = \sum_{k=1}^{n} w_k \cdot \tilde{X}_k$.
```

2. Echo protocol (used in CGGMP)

The protocol appears in the Protocol 2 of the paper. The authors of CGGMP observe that the protocol instructs the parties to (verifiably) broadcast some of their messages (as opposed to messages which are “sent to all”, where equality verification is not required). For non-unanimous halting, this can be achieved in a point-to-point network using echo-broadcasting with one extra round of communication. For completion, we state the protocol as follows:

Algorithm 10 Echo Protocol

```
    Input: (broadcast, $x$).
1:  Each participant $\mathscr{P}_i$ sends $x$ to all participants.
2:  Upon receiving a value $x^j$ from $\mathscr{P}_i$ , party $\mathscr{P}_i$ sends the value $x^j$ that it received to all other parties.
3:  Party $\mathscr{P}_j$ waits to receive a message from every party (other than party $\mathscr{P}_i$ ). Denote the message received from $\mathscr{P}_k$ by $x^j_k$ . Then $\mathscr{P}_j$ outputs (broadcast, $\mathscr{P}_i, x^j$) if and only if for every $k$, it holds that $x^j_k = x_j$ (i.e., $x^j = x^j_1 = \ldots = x^j_n$). Otherwise, it outputs nothing.
```

Remark 4.

1. *We use the digest of a hash function to compare the equality of the message x^j_i for $1 \le i \le n$.*

3. EdDSA : FROST

We implement the paper Figure 3: FROST Single-Round Signing Protocol in FROST: Flexible Round-Optimized Schnorr Threshold Signatures without an aggregator. The authors in the paper indicate that FROST can be instantiated without a signature aggregator; each participant simply performs a broadcast in place of SA performing coordination. At last, we also remark that there exists one difference:

a. Replace Lagrange coefficients with Birkhoff coefficients (ref. subsection A.3).

Let $\mathscr{G}$ be an elliptic curve group with order $N_{\mathscr{G}}$ and G be a fixed point of $\mathscr{G}$. For every $S \subset [1, ..., n]$ be the set of players participating in the signature protocol with $|S| \geq t$, we let $bk_i := (\text{xcoord}_i, n_i)$ be the Birkhoff parameter and $w_{i,S}$(abbrev. w_i) each player in S be the appropriate Birkhoff coefficients associated with $\{bk_i\}_{i \in S}$ (ref. subsection A.3). Here xcoord_i is the x-coordinate and n_i is the level of the Participant $\mathscr{P}_i$. Set $x_i := w_i \cdot s_i$ called “modified share ”, where s_i is the share of $\mathscr{P}_i$. The algorithm is described as follows:

Algorithm 11 Sign

```
**Input: a threshold $t$, a message $m$, the public Key $P$, a set of levels $\{n_i\}_{i=1}^n$, a set of $x$-coordinate $\{x_i\}_{i=1}^n$ and a share $s_i$.**
**Output: a signature $(R, z)$.**
1: Each participant $\mathscr{P}_i$
    a. randomly chooses $d_i, e_i \in [0, N_{\mathscr{G}} - 1]$.
    b. computes a commitment $D_i := d_i \cdot G$ and $E_i := e_i \cdot G$.
    c. broadcasts $D_i$ and $E_i$.
2: Each participant $\mathscr{P}_i$
    a. verifies the message $m$ correct.
    b. computes $\rho_j := H(bk_j, m, B)$ for $1 \leq j \leq n$ and $R := \sum_{j=1}^{n} R_j$. Here $w_i$ is the corresponding Birkhoff coefficient of participants, $B :=< (bk_i, D_i, E_i) >_{i \in S}$ and $R_j := D_j + \rho_j \cdot E_j$.
    c. computes $c := H(R, Y, m)$.
    d. computes $z_i := d_i + (e_i \cdot \rho_i) + w_i \cdot s_i \cdot c$.
    e. broadcasts $z_i$.
3: Each participant $\mathscr{P}_i$
    a. verifies $z_j \cdot G = R_j + c w_j \cdot Y_j$ for $1 \leq j \neq i \leq n$. If the equality does not hold, first identify and report the misbehaving participant, and then abort. Otherwise, continue.
    b. computes $z = \sum_{j=1}^{n} z_j$.
    c. verifies the signature $(R, z)$.
```

Remark 5.

1. *We adopt the DKG protocol of GG18 (ref. subection 29) as the DKG protocol of EdDSA without generating Paillier's key. This part had already been audited.*

A. Appendix A

A.1. Hash commitments

We implement the hash commitment to a message $x \in \{0,1\}^n$ as $H(x,r)$ for a uniformly chosen $r \in \{0,1\}^n$, where H is a cryptography hash function. This commitment is computational binding and perfect hiding.

Remark 6.

1. *We use the cryptography hash function blake2b, which is the standard library in golang. So far, there does not exist effective collision attack for blake2b.*
2. *For length extension attack, we survey some literatures: BLAKE2: simpler, smaller, fast as MD5, blake2b length extension attack approach, Analysis of BLAKE2, and length extension attack. They indicate that Blake2 can resist length extension attack because its encoding bases on Dan Bernstein's ChaCha stream cipher. However, Algorithms like MD5, SHA-1, and SHA-2 that are based on the Merkle–Damgard construction are susceptible to this kind of attack.*
3. *Other analysis: The Boomerang Attacks on BLAKE and BLAKE2.*
4. *In this discussion, there exists another candidate SHA-2 512/256. Comparing to blake2b, is it more security?*
5. *For hash multiple inputs, we use method in this library to sequentialize inputs (ref. How to hash a list of multiple items?) such that we are able to avoid collision as much as possible.*

Algorithm 12 NewHashCommitmenter.

```
Input: a message $m$ and a positive integer $n$.
Output: the digest commitment.
1: Generate a random salt $r$ with length $n$.
2: Compute the commitment by $H(m,r)$.
```

Algorithm 13 Decommit.

```
Input: the commitment $\mathscr{C}$ and the decommitment: $m$ and $r$.
Output: True or False.
1: Compute the digest by $H(m,r)$.
2: If the digest $H(m,r)$ is equal to $\mathscr{C}$, then return True. Otherwise, return False.
```

A.2. Feldman commitments

Let $N_{\mathscr{G}}$ be the order of an elliptic curve group $\mathscr{G}$ and G be a fixed point of $\mathscr{G}$. Let $p(x) := \sum_{i=0}^{t-1} a_i x^i$ be a polynomial with degree $t-1$ over the ring $\mathbb{Z}_{N_{\mathscr{G}}}$ and $\sigma_i := p^{(n_i)}(x_i)$, where $p^{(n_i)}$ is the n-th derivative of p. Feldman's commitment (VSS) is an extension of Shamir secret sharing in which the dealer also publishes the points(i.e. commitments)

$$F_i := a_i \cdot G \text{ for all } i \in [0, t-1].$$

Using this auxiliary information, each player P_i with x- coordinate x_i and n_i-th differential can check his share σ_i for consistency by verifying:

$$\sigma_i \cdot G = \sum_{j=0}^{t-1} (x^j)^{(n_i)} |_{x=x_i} \cdot F_j.$$

If the check does not hold for any player, it raises a complaint and the protocol terminates.

Algorithm 14 buildFeldmanCommitMessage.

```
    Input: a polynomial $p(x) = \sum_i a_i x^i$ and a fixed point $G$.
    Output: Feldman commitment $F_i$, which is a list of point commitments.
1:  Compute $F_i = a_i \cdot G$ for all $0 \le i \le \deg(p)$. Here $\deg(p)$ is the degree of the polynomial $p(x)$.
```

Algorithm 15 GetVerifyMessage.

```
    Input: the polynomial $p(x)$, the $n_i$-th differential and a big integer $x_i$.
    Output: the value $p^{(n_i)}(x_i)$.
1:  Compute the value $p^{(n_i)}(x_i) \mod N_{\mathscr{G}}$.
```

Algorithm 16 Verify.

```
    Input: $n_i$-th differential, an integer $x_i$, point commitments $\{F_j\}$ and a challenge $\sigma$.
    Output: True or False
1:  Compute the expected point $\sigma \cdot G$.
2:  Compute $\left( \sum_{j=0}^{t-1} (x^j)^{(n_i)} |_{x=x_i} \right) \cdot F_j$.
3:  If the above two points are the same then the output is True. Otherwise, return False.
```

A.3. The Birkhoff Interpolation

Birkhoff interpolation [1] is the problem of finding a polynomial f of degree $t-1$ [2] such that certain derivatives have specified values at specified points.

$$f^{(n_i)}(\alpha_i) := \frac{d^{(n_i)} f(x)}{dx^{(n_i)}} = \beta_i, \text{ for } i = 0, 1, \ldots, k.$$

where the data points $(\alpha_i, \beta_i) \in \mathbb{F}_q \times \mathbb{F}_q$, no two α_i are the same and the non-negative integers n_i are given. In particular, if $n_i = 0$ for all $0 \le i \le k$, then Birkhoff interpolation is reduced to Lagrange interpolation. For simplicity, we assume that $q > t$ and $q > k$. Let $f(x) := \sum_{i=0}^{t-1} a_i x^i$, where $a_{t-1} \in \mathbb{F}_q^\times$ and $a_i \in \mathbb{F}_q$ for $0 \le i \le t-2$ and set

$$[x_n^{(i)}](j) := \begin{cases} \left(\prod_{m=0}^{i-1}(n-m)\right) \cdot j^{n-i}, & \text{if } n \ge i; \\ 0, & \text{otherwise.} \end{cases}$$

Rewrite the equations above in the matrix forms as follows:

$$\left[A(\{n_i\},\{\alpha_i\})\right]_{(k+1)\times t} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix},$$

where $[A(\{n_i\},\{\alpha_i\})]_{(k+1)\times t}$(abbrev. $[A(\{n_i\},\{\alpha_i\})]$) :=

$$\begin{bmatrix} [x_0^{(n_0)}](\alpha_0) & [x_1^{(n_0)}](\alpha_0) & \ldots & [x_{t-1}^{(n_0)}](\alpha_0) \\ [x_0^{(n_1)}](\alpha_1) & [x_1^{(n_1)}](\alpha_1) & \ldots & [x_{t-1}^{(n_1)}](\alpha_1) \\ \cdots & \cdots & \cdots & \cdots \\ [x_0^{(n_k)}](\alpha_k) & [x_1^{(n_k)}](\alpha_k) & \ldots & [x_{t-1}^{(n_k)}](\alpha_k) \end{bmatrix}.$$

If there exists a pseudoinverse of $[A(\{n_i\},\{\alpha_i\})]$, then we have:

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix} = \left[A(\{n_i\},\{\alpha_i\})\right]^\dagger \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}.$$

In particular,

$$a_0 = \sum_{j=0}^{k} [A(\{n_i\},\{\alpha_i\}]^\dagger_{1,j+1} \cdot \beta_j.$$

[1] https://en.wikipedia.org/wiki/Birkhoff_interpolation

[2] The degree here is $t-1$ for the notation consistency of the threshold.

Algorithm 17 ComputeBkCoefficient.

```
    Input: two sets of integers $\{\alpha_i\}$, and $\{n_i\}$.
    Output: The first row of the matrix $[A^\dagger(\{n_i\},\{\alpha_i\}]$.
1:  Establish $[A(\{n_i\},\{\alpha_i\})]$ (ref. Algorithm 18).
2:  Compute the pseudoinverse $[A^\dagger(\{n_i\},\{\alpha_i\}]$ of $[A(\{n_i\},\{\alpha_i\})]$(ref. Algorithm ??).
3:  Get the first row of the matrix $[A^\dagger(\{n_i\},\{\alpha_i\}]$.
```

Algorithm 18 GetLinearEquationCoefficient.

```
    Input: a positive integer $n$, two integral lists $\{\alpha_i\}$, and $\{n_i\}$.
    Output: The matrix $[A(\{n_i\},\{\alpha_i\}]$ associated with $\{\alpha_i\}$, and $\{n_i\}$ .
1:  Compute the matrix $[A(\{n_i\},\{\alpha_i\}]_{j,k} = [x_n^{(n_j)}](\alpha_k)$.
```

A.4. Homomorphism Encryptions

Homomorphic encryption is a form of encryption that allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext. In this section, we implement two standard libraries: Paillier and CL homomorphic encryptions.

A.4.1. Paillier homomorphic encryption

Paillier Cryptosystem provides an additive homomorphic encryption based on public key cryptography, hence encrypted secret shares can be added together and decrypted by the private key. Let N_{Pai} be a positive integer. We denote by $Z^\times_{N_{Pai}}$ the multiplicative group of integers modulo N_{Pai}. We fix notations as follows:

- Choose two large prime integers p and q randomly and independently of each other such that $\gcd(pq,(p-1)(q-1)) = 1$.
- Compute $N_{Pai} := pq$ and $\lambda = \text{lcm}(p-1,q-1)$. lcm means Least Common Multiple.
- $g \in \mathbb{Z}^\times_{N^2_{Pai}}$ is a random element.
- $\mu := \left(L(g^\lambda \mod N^2_{Pai})\right)^{-1} \mod N_{Pai}$. Here $L(x) := \frac{x-1}{N_{Pai}}$ and $\frac{a}{b}$ is the largest integer value $v \geq 0$ to satisfy the relation $a \geq vb$. In particular, if $a \geq 0$ and $b \geq 1$, then $\frac{a}{b} = \left\lfloor \frac{a}{b} \right\rfloor$.

Remark 7.

1 *Because we do not perform range proofs in MtA (ref. subsection **??**), the generating primes is not necessarily "safe prime" (i.e. A prime p is called safe prime if p is a prime and* $\frac{p-1}{2}$ *is also a prime). In our implementation, we use API, Prime, to generate **random primes** assuming that the bit-length of public key in RSA-module is bigger than* 2048 *(ref. Table B.2. NIST).*

2. *We can assume that the output of Prime is "true" prime because so far there does not exist any probabilistic-prime which can pass Rabin-Millier test and Lucas test simultaneously.*

The algorithms in this part are listed as follows:

Algorithm 19 NewPaillier

```
    Input: an integer $k$.
    Output: a private key $(\lambda, \mu)$ and the public key $(N_{Pai}, g)$.
1: Check $k \geq 2048$.
2: Compute $N_{Pai}$ and $\lambda$ (i.e. getNAndLambda: Algorithm 24).
3: Compute $N_{Pai}^2$.
4: Get a generator $g$ and the parameter $\mu$ (i.e. getGAndMu: Algorithm 26).
```

Algorithm 20 Encrypt

```
    Input: an integer $m$.
    Output: a ciphertext $c_m$.
1: Check $0 \leq m < N_{Pai}$.
2: Check $\gcd(r, N_{Pai}) = 1$ and $r \in [0, N_{Pai} - 1]$.
3: Compute $c_m := g^m \cdot r^{N_{Pai}} \mod N_{Pai}^2$.
```

Algorithm 21 Decrypt

```
    **Input: an integer $c$.**
    **Output: a plaintext $p_m$.**
1:  Check that $c$ is correct form( ref. isCorrectCiphertext: Algorithm 25).
2:  Compute $x = c^{\lambda} \mod N_{Pai}^2$.
3:  Compute $p_m := L(c^{\lambda} \mod N_{Pai}^2) \cdot \mu \mod N_{Pai}$.
```

Algorithm 22 EvalAdd

```
    **Input: two ciphertexts $c_1$, and $c_2$.**
    **Output: a new ciphertext $c_3$.**
1:  Check that $c_1$ and $c_2$ are both correct forms( ref. isCorrectCiphertext: Algorithm 25).
2:  Randomly choose $r \in [0, N_{Pai} - 1]$ with $\gcd(r, N_{Pai}) = 1$.
3:  Compute $c_3 := c_1 \cdot c_2 \cdot r_{Pai}^N \mod N_{Pai}^2$.
```

Algorithm 23 EvalMulConst

```
    **Input: a ciphertext $c$ and a plaintext $b$.**
    **Output: a new ciphertext $c'$.**
1:  Check that $c$ is the correct form( ref. isCorrectCiphertext: Algorithm 25).
2:  Compute $\bar{b} := b \mod N_{Pai}$.
3:  Randomly choose $r \in [0, N_{Pai} - 1]$ with $\gcd(r, N_{Pai}) = 1$.
4:  Compute $c' := c^{\bar{b}} \cdot r^{N_{Pai}} \mod N_{Pai}^2$.
```

Algorithm 24 getNAndLambda

```
    **Input: a positive integer keysize.**
    **Output:** $N_{Pai}$, $\lambda$.
1:  Set pqSize:= keysize/2.
2:  Generate p, q with bit-Length of $p$ = pqSize and bit-Length of $q$ = pqSize such that
    $\gcd(pq, (p-1)(q-1)) = 1$.
3:  Compute $\lambda := \text{lcm}(p-1, q-1)$.
```

A.5. Schnorr Non-Interactive proof

We implement a non-interactive zero knowledge proof of Schnorr protocol(i.e. more general protocol is called Sigma protocol) by applying the protocol appearing in An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. Comparing to the interactive version, the most important merit of non-interactive version is reducing the number of rounds(i.e. 3 rounds to 1 rounds). Let $\mathcal{G}$ be an elliptic curve group and

Algorithm 25 isCorrectCiphertext

```
**Input: a ciphertext $c$ and $N_{Pai}$.**
**Output: True or False.**
1: Check $0 < c < N_{Pai}^2$.
2: Check that $c$ and $N_{Pai}$ are coprime. If the above condition holds, then the output is true.
   Otherwise, the output is false.
```

Algorithm 26 getGAndMu

```
**Input: an integer $\lambda$ and an integer $N_{Pai}$.**
**Output: a generator $g$ and an integer $\mu$.**
1: Randomly generate $g \in [2, N_{Pai}^2)$ such that the inverse of $\mu' := \left\lfloor \frac{(g^{\lambda} \mod N_{Pai}^2)-1}{N_{Pai}} \right\rfloor$ exists.
2: If the inverse of $\mu'$ exists, set $\mu := (\mu')^{-1}$. Otherwise, go to 1.
```

$N_{\mathcal{G}}$ be the order of $\mathcal{G}$. Let G and R be two fixed points of the elliptic curve group $\mathcal{G}$. Set $V := a_1 \cdot G + a_2 \cdot R \in \mathcal{G}$. In order to prove the knowledge of a_1 and a_2, the prover interacts with the verifier as follows:

1. A prover:
   a. randomly chooses two numbers m, n in $[1, N_{\mathcal{G}} - 1]$ and sends $\alpha := m \cdot G + n \cdot R$ to the verifier.
   b. computes $c := H(G, V, R, \alpha)$. Here H is a cryptography hash function.
   c. computes $u := m + c \cdot a_1 \mod N_{\mathcal{G}}$ and $t := n + c \cdot a_2 \mod N_{\mathcal{G}}$. The resulting proof is the (u, t, α)
2. Anyone can check this proof by $t \cdot R + u \cdot G = \alpha + c \cdot V$, and $u, t \in [0, N_{\mathcal{G}} - 1]$.

Algorithm 27 NewSchorrMessage

```
**Input: two integers $a_1$ and $a_2$ and a point $R$.**
**Output: two points $V, \alpha$, and two integers $u$ and $t$ .**
1: Check $a_1, a_2 \in [0, N_{\mathcal{G}} - 1]$.
2: Compute $V = a_1 \cdot G + a_2 \cdot R$.
3: Randomly choose $m, n \in [0, N_{\mathcal{G}} - 1]$ and compute $\alpha = mG + nR$.
4: Compute $c = H(G, V, R, \alpha)$.
5: Compute $u := m + c \cdot a_1 \mod N_{\mathcal{G}}$ and $t := n + c \cdot a_2 \mod N_{\mathcal{G}}$.
```

Algorithm 28 Verify

```
    Input: Schnorr proof and the point $V$.
    Output: True of False.
1:  Check $u, t \in [0, N_{\mathscr{G}} - 1]$.
2:  Compute $c = H(G, V, R, \alpha)$.
3:  Check that $t \cdot R + u \cdot G = \alpha + c \cdot V$. If this equality holds, then return True. Otherwise, return
    False.
```

Remark 8.

1. *The interactive version of the above protocol is given in Section 4.3 The Zero-Knowledge Proofs.*
2. *When R is the identity element of the elliptic curve group $\mathscr{G}$, then this protocol reduces to the well known Schnorr non-interactive protocol.*
3. *This protocol is a honest-verifier zero knowledge proof.*

A.6. Distributed Key Generation

Distributed key generation (Abbrev. DKG) is a cryptographic process in which multiple parties participate to the calculation of a shared public and a private key set. The main merit of distributed key generation is not rely necessarily on trusted third parties. Let $\mathscr{G}$ be an elliptic curve group with order $N_{\mathscr{G}}$ and G be a fixed point of $\mathscr{G}$. In our implementation, the threshold t, the number of total participants n, and the level n_i of each participants $\mathscr{P}_i$ are determined before someone performs the following DKG protocol:

Algorithm 29 Distributed Key Generation

Input: positive integers t, n and a non-negative integer n_i.

Output: two sets of positive integers $\{n_i\}_{i=1}^{n}$ $\{x_i\}_{i=1}^{n}$, a share s_i, and the unique corresponding public Key P.

```
1: Each participant $\mathscr{P}_i$
    a. randomly chooses $x$-coordinate $x_i \in [1, N_{\mathscr{G}} - 1]$, and $\{u_{i,j}\}_{j=0}^{t-1} \in [0, N_{\mathscr{G}} - 1]$ (i.e. $f_i(x) := \sum_{j=0}^{t-1} u_{i,j} x^j$ ).
    b. computes $u_{i,0} \cdot G$ and hash commitment $[\mathscr{C}_i, \mathscr{D}_i] = \mathrm{H}(u_{i,0} \cdot G)$ (ref. subsection A.1).
    c. broadcasts $x$-coordinate $x_i$, the level $n_i$, and the hash commitment $\mathscr{C}_i$.
2: Each participant $\mathscr{P}_i$
    a. computes the associated Birkhoff coefficient $w_i$ (ref. Algorithm 17) and Feldman's commitment $F_{i,j} := u_{i,j} \cdot G$ (ref. subsection A.2).
    b. broadcasts own decommitment $\mathscr{D}_i$ and Feldman's commitment $F_{i,j}$.
3: Each participant $\mathscr{P}_i$
    a. verifies the decommitment $\mathscr{D}_i$ (ref. Algorithm 13).
    b. computes the values $f_i^{(n_k)}(x_k)$ for all $1 \le k \le n$.
    c. sends the value $f_i^{(n_k)}(x_k)$ to the participant $\mathscr{P}_k$.
4: Each participant $\mathscr{P}_i$
    a. verifies the Feldman commitment for $f_k^{(n_i)}(x_i)$ for all $1 \le k \ne i \le n$ (ref. Algorithm 16).
    b. computes the public key $\sum_{i=1}^{n} u_{i,0} \cdot G$, the own share $s_i := \sum_k f_k^{(n_i)}(x_i)$, the point $s_i \cdot G$ and Schnorr's proof of $s_i \cdot G$ (ref. Algorithm 27).
    c. broadcasts $s_i \cdot G$ and Schnorr's proof of $s_i \cdot G$.
5: Each participant $\mathscr{P}_i$
    a. verifies the Schnorr's proof of $s_k \cdot G$ for all $1 \le k \ne i \le n$ (ref. Algorithm 28) and the Public key is given by by $\sum_{k=1}^{n} w_k \cdot (s_k \cdot G) = \sum_{k=1}^{n} u_{k,0} \cdot G$.
```