Adaptations from CGGMP21

Our implementation tries to follow the original specification provided in [CGGMP21], but we detail here the modifications we applied.

Broadcast considerations

Abort identification requires the use of a reliable broadcast channel.
Unfortunately, this condition is hard to meet in practice when the network is modeled as point-to-point.
The protocol generally requires the message from the first round to be reliably broadcast, and so for this step we use the "*echo broadcast*" from Goldwasser and Lindell.

In the keygen and refresh protocols, the broadcasted message is a commitment, so we need to add an extra round of communication to make sure the parties agree on the first set of messages.
For the signing protocol, the "echo round" can be done in parallel.

The main disadvantage of this approach is that an adversary can cause an abort by simply sending different messages to different parties.
It therefore makes little sense to implement the identifiability aspect of the signing protocol since an adversary could instead cause an annonymous abort beforehand.

On the flip side, the user of the library only needs to provide authenticity and integrity between the point-to-point connections between parties.

Threshold Keygen/Refresh Protocols

In the original paper, the TSS functionality is such that the keygen needs to run the refresh protocol right after.
Moreover, the scheme is not adapted to the threshold case.

We implement a combined keygen/refresh protocol, where we highlight the differences in blue and green respectively.

- Keygen performs the refresh at the same time, so there is no need to two protocols.
- Both protocols now use a threshold t for Shamir secret sharing ($t + 1$ secret key shares are required to reconstruct the full secret).

- Optionally, we can also include the extra *El Gamal* keys Y, but since we only care about the 4 round signing protocol, we ignore it.
- If we want to prove this scheme secure, then we would need to change the idea threshold signature functionality. Indeed, it currently assumes refresh is run after keygen. We would need to change the ideal functionality, and adapt the protocol to the t, n case.
- The main difference is that we add a final round of communication where the parties prove in zero-knowledge that they know their new secret share.

We define the SSID as $ssid = (sid, ...)$ where $sid = (q, G, t, n, \{P^{(j)}\}_{j=1}^{n})$ and where ... is the public information that all parties already know.
When the SSID is used as header for the message, it will most likely be its hash so that we don't send too much information in each message.
Moreover, the computation of this hash should be the same for all parties (i.e. deterministic).

At the start of the protocol, interpret $ssid = (sid, rid', \{pk'^{(j)}\}_{j=1}^{n}, \{N'^{(j)}\}_{j=1}^{n}, \{s_1'^{(j)}\}_{j=1}^{n}, \{s_2'^{(j)}\}_{j=1}^{n})$, where $sid = (q, G, t, n, \{P^{(j)}\}_{j=1}^{n})$, and retreive private input $sk'^{(i)}$.

Round 1

- Sample $x^{(i)} \in F_q$, set $X^{(i)} = x^{(i)} \cdot G$, and $sk'^{(i)} \leftarrow x^{(i)}$
- Sample 4κ-bit safe primes $p^{(i)}, q^{(i)}$
- Set $N^{(i)} = p^{(i)} \cdot q^{(i)}$
- Compute $s_1^{(i)} \leftarrow r^2 \bmod N^{(i)}, s_2^{(i)} \leftarrow r^{2\lambda} \bmod N^{(i)}$ where $\lambda^{(i)} \in \mathbb{Z}^*_{N^{(i)}}, r \in \mathbb{Z}^*_{\phi(N^{(i)})}$
- Sample $f^{(i)}(Z) \in \mathbb{F}_q[Z]$, such that $f^{(i)}(Z) = x^{(i)} + \sum_{l=1}^{t} f_l^{(i)} Z^l$
- Define VSS polynomial $\{F_l^{(i)} = f_l^{(i)} \cdot G\}_{l=1}^{t}$
- Sample $a^{(i)} \in \mathbb{F}_q$, define $A^{(i)} = a^{(i)} \cdot G$
- Sample $rid^{(i)}, u^{(i)} \in \{0,1\}^\kappa$
- Compute $V^{(i)} = H(ssid, i, rid^{(i)}, \{F_l^{(i)}\}_{l=1}^{t}, X^{(i)}, A^{(i)}, N^{(i)}, s_1^{(i)}, s_2^{(i)}, u^{(i)})$

Broadcast $(ssid, i, V^{(i)})$

Round 1 (Echo)

Upon reception of $(ssid, i, V^{(j)})$ from all $P^{(j)}$:

- Compute $E^{(i)} \leftarrow H(ssid, V^{(1)}, \dots, V^{(n)})$

Broadcast $(ssid, i, E^{(i)})$

Upon reception of $(ssid, i, E^{(j)})$ from all $P^{(j)}$:

- Check $E^{(i)} \equiv E^{(j)}$

Deliver $(ssid, i, V^{(j)})$ from all $P^{(j)}$ for the next round.

Round 2

Upon reception of $(ssid, i, V^{(j)})$ from all $P^{(j)}$:

Send $(ssid, i, rid^{(i)}, \{F_l^{(i)}\}_{l=1}^t, X^{(i)}, A^{(i)}, N^{(i)}, s_1^{(i)}, s_2^{(i)}, u^{(i)})$ to all

Round 3

Upon reception of $(ssid, j, rid^{(j)}, \{F_l^{(j)}\}_{l=1}^t, X^{(j)}, A^{(j)}, N^{(j)}, s_1^{(j)}, s_2^{(j)}, u^{(j)})$ from $P^{(j)}$:

- Verify $V^{(j)} \equiv H(ssid, j, rid^{(j)}, \{F_l^{(j)}\}_{l=1}^t, X^{(j)}, A^{(j)}, N^{(j)}, s_1^{(j)}, s_2^{(j)}, u^{(j)})$
- Check $\log_2 N^{(j)} \equiv 8\kappa$
- Set $F^{(j)}(Z) \leftarrow X^{(j)} + \sum_{l=1}^t Z^l \cdot F_l^{(j)}$

If all checks pass:

- Compute $rid = \bigoplus_j rid^{(j)}$
- Prove:
  - $mod^{(i)} = Prove(mod, (ssid, rid, i), N^{(i)}; (p^{(i)}, q^{(i)}))$
  - $prm^{(i)} = Prove(prm, (ssid, rid, i), (N^{(i)}, s_1^{(i)}, s_2^{(i)}); \lambda^{(i)})$
- Compute $C_j^{(i)} \leftarrow Enc_j(f^{(i)}(j))$ for $j = 1, \ldots, n$

Send $(ssid, i, mod^{(i)}, prm^{(i)}, C_j^{(i)})$ to each $P^{(j)}$

Round 4

Upon reception of $(ssid, j, mod^{(j)}, prm^{(j)}, C_j^{(j)})$ from $P^{(j)}$:

- Decrypt $x_i^{(j)} \leftarrow Dec_j(C_i^{(j)}) \mod q$
- Check $F^{(j)}(i) \equiv x_i^{(j)} \cdot G$
- Verify $mod^{(j)}((ssid, rid, j); N^{(j)})$
- Verify $prm^{(j)}((ssid, rid, j); N^{(j)}, s_1^{(j)}, s_2^{(j)})$

If all checks pass:

- Compute $sk^{(i)} \leftarrow sk'^{(i)} + \sum_{j=1}^n x_i^{(j)}$
- Set $F(X) = pk' + \sum_{j=1}^n F^{(j)}(X)$
- Compute $pk^{(j)} = F(j)$ for $j = 1, \ldots, n$

- Prove $sch^{(i)} = Prove(sch, (ssid, rid, i), (pk^{(i)}, A^{(i)}); (sk^{(i)}, a^{(i)}))$

Send $(ssid, i, sch^{(i)})$ to all

Output

Upon reception of $(ssid, i, sch^{(j)})$ from $P^{(j)}$:

- Verify $sch^{(j)}((ssid, rid, j); pk^{(j)}, A^{(j)})$

If all checks pass:

- Set $pk = F(0)$

Save:

- Secret $sk^{(i)}, p^{(i)}, q^{(i)}$
- Public $\{pk^{(j)}\}_{j=1}^{n}, \{s_1^{(j)}\}_{j=1}^{n}, \{s_2^{(j)}\}_{j=1}^{n}$
- $ssid' = (sid, rid, \{pk^{(j)}\}_{j=1}^{n}, \{N^{(j)}\}_{j=1}^{n}, \{s_1^{(j)}\}_{j=1}^{n}, \{s_2^{(j)}\}_{j=1}^{n})$

Signing

Interpret $ssid = (sid, rid, \{pk^{(j)}\}_{j=1}^{n}, \{N^{(j)}\}_{j=1}^{n}, \{s_1^{(j)}\}_{j=1}^{n}, \{s_2^{(j)}\}_{j=1}^{n}, \{P^{(j)}\}_{j \in S}, m)$, where S is a subset of $\{1, \ldots, n\}$ of size at least $t + 1$, and m is the message to be signed.

The protocol goes exactly as before, except that the `Session` will use S to determine Lagrange coefficients and apply them to the set of public keys, as well as the signer's secret share. Therefore, the resulting shares represent an additive sharing of the secret, and the original protocol can be used.