Threshold Multi-Signature with an Offline Recovery Party

Riccardo Longo, Alessio Meneghetti, and Massimiliano Sala

Department of Mathematics, University of Trento, via Sommarive, 14 - 38123 Povo (Trento), Italy

January 7, 2020

Abstract

Key custody is a sensitive aspect of cryptocurrencies. The employment of a custodian service together with threshold-multi-party signatures helps to manage secret keys more safely and effectively, e.g. allowing the recovery of crypto-assets when users lose their own keys. Advancing from a protocol by Gennaro et al. we propose a protocol with two main properties. First it allows the recovery party to remain offline during the enrollment of any user, solving a real-life problem of maintaining online only one trusted third party. Second our multi-party signature is compatible with a deterministic derivation of public and private keys.

1 Introduction

In the cryptocurrency world digital signatures determine ownership rights and control over assets, meaning that protection and custody of private keys is of paramount importance. A particularly sensitive issue is the resiliency against key loss, since there is no central authority that can restore ownership of a digital token once the private key of the wallet is lost. This problem is even more crucial for the average user that usually lacks the competence or resources necessary to adequately manage and protect those keys.

A common solution is to rely on a trusted third-party custodian that takes responsibility of key management, but this also means that users must relinquish control over their assets, in complete opposition of the spirit of cryptocurrencies. Moreover this kind of centralization may form single points of failure and juicy targets for criminal takeovers, and there have already been plenty of examples of said events in the past [7].

More interesting is the approach that distributes the control over the wallet through multi-signature schemes, in particular with threshold-like policies, where k signers out of n are required in order to produce a complete and valid signature. The simplest approach to this solution are multi-sig wallets (available for some cryptocurrencies, like Bitcoin [25]) where the signatures are normal ones, but funds may be moved out of that wallet only with a sufficient number of signatures corresponding to a prescribed set of public keys. The main disadvantage to this approach is that it is not supported by every cryptocurrency (e.g. Ethereum [6]), moreover such wallets are very easily identifiable.

Another approach is to use Secure Multi-Party Computation in order to compute a single signature involving multiple parties, none of which has access to the secret key: each has a partial secret that can be combined with the others to create a signature. This method has the advantage that the resulting signature is indistinguishable from a *normal one*, so it can be used very discretely and, more importantly, with any wallet. On the other hand the signing procedure is much more complicated and requires online collaboration of the participating parties.

Another practical problem is that often the recovery party (that allows to recover the funds in case of key loss) is not willing to sustain the cost of frequent online collaboration. For example a bank may safely guard a *piece of the secret key* but it is inconvenient and quite costly to make it participate in the enrollment of every user. Therefore we propose a new protocol in which the recovery party is involved only once (in a preliminary set-up) and afterwards it isn't involved until a lost account must be recovered.

Yet another practical problem is to derive many keys from a single secret, for example to efficiently manage multiple wallets. Our protocol is capable of solving this problem while maintaining full compatibility with the offline recovery party.

Our signature scheme is presented in this paper in two versions: an ECDSA version and an EdDSA version. Its adaption to any DSA-like signature does not present any difficulty.

As use-case, we consider a custodian offering as service for its clients the possibility of relying on the Threshold Multi-Scheme Schemes here described. A client willing to sign a transaction relies on a Secure Multi-Party protocol involving the custodian to obtain a valid signature. The scheme is designed to be:

- safe: when either the client or the custodian are unable to participate to the protocol (e.g. the client's key is lost), a recovery transaction can still be signed with the aid of a recovery server (e.g. the custodian and the recovery server can move the client's funds into an alternative address whose key is still owned by the client);

- secure: signatures cannot be forged and only the rightful participants can sign their own transactions. In particular, no one has a "master" private key which allows to sign transactions: the custodian cannot sign without the help of the client, and the client's key is never shared with anyone.

- sound: a signature obtained with this scheme is indistinguishable from other signatures (e.g. it is infeasible to decide whether an ECDSA signature was obtained by using the ECDSA standard implementation or by the ECDSA-compatible Threshold Multi-Signature Scheme)

1.1 Related Work

The first Threshold Multi-Party Signature Scheme was a protocol for ECDSA signatures proposed by Gennaro et al. [14] where $t+1$ parties out of $2t+1$ were required to sign a message. Later MacKenzie proposed and then improved another scheme [22, 23], which has later been furthermore enhanced [8, 20, 9]. The first scheme supporting a general t, n threshold was proposed again Gennaro [13], improved in [5] and finally in [12] (that has been our starting point). A parallel approach has been taken by Lindell et al. in [21].

1.2 Layout

After a high-level presentation of the protocol and some properties inSection 2, we will present our protocol for the ECDSA digital signature in Section 3, then in Section 4 we will present the variant for the EdDSA digital signature, and finally we will state our security claims and draw some conclusions in Section 5. For the sake of completeness we provide some appendixes on various techniques and cryptographic primitives used in the protocol.

2 Protocol Overview

In this presentation we suppose a simple but realistic scenario with three actors following a 2-out-of-3 policy, but it can easily be generalized to more actors and a more general thresholds. The three actors that we consider are:

- A: the *Custodian*, which is assumed to be always online.
- B: the *Recovery Server*, which is assumed to be online once for the setup, then only to perform signatures in case of key loss.
- C: the *Client*, which is not known and online at the beginning but only later on. Note that a pair Custodian - Recovery Server can easily support multiple clients that may enroll at different times.

The scheme comprises four phases:

- a *Preliminary Phase*, during which the Recovery Server performs its setup and communicates its parameters to the Custodian;
- an *Enrollment Phase* when the Client comes online and sets up its own parameters synchronizing with the Custodian;
- an *Signature Phase* where Client and Custodian collaborate online to produce a new signature;
- a *Recovery Phase* that takes place after the Client or the Custodian has lost its private parameters and is therefore unable to perform the Ordinary Signature Phase: the Recovery Server is brought back online, it synchronizes itself with the surviving party and thereafter can substitute the missing party in the Signature Phase.

2.1 Keys and Derivation

In the Enrollment Phase it is created a public key that corresponds to the signatures that will be created in the Signature Phase using the secret parameters created by Client and Custodian. Implicitly there is a corresponding private key that generates equivalent signatures, but it is never created and cannot be created unless at least two parties collude sharing their secrets.

The secret parameters obtained in the Enrollment Phase can be utilized directly to sign messages and transactions, or they can be used to derive deterministically other key pairs. For example in Bitcoin it is good practice to always use fresh addresses, that correspond to different keys (e.g. with BIP32 [30]).

For this purpose it is sufficient that A and C agree on a (public) derivation index i, then, using a common secret d computed during the Enrollment Phase, they can independently derive other secret parameters that can be used in the Signature Phases. Note that the derived secrets correspond to a new public, and that the derivation can also be compound, that is, more keys can be derived from a derived key.

2.2 Mnemonic

The Recovery Phase requires to bring the Recovery Server back online, and afterwards to create a new wallet with a fresh enrollment and then move all the assets to this new wallet, therefore it can become quite inconvenient and/or expensive to perform. So, in order to minimize the occasions in which this drastic measure is necessary, alongside the protocol we propose a sub-protocol to generate a mnemonic (a la BIP39 [27]) from the secret parameters of a user (usually the Client) and to restore these parameters using the mnemonic.

In order to reduce the length of the mnemonic some shared secrets and public parameters are omitted in favor of only few checksum bits. When restoring private information from the mnemonic the other active user (usually the Custodian) will send these common information and its correctness will be verified using the checksum.

3 Threshold Multi-Party ECDSA

This version of the protocol derives directly from the scheme of Gennaro and Goldfeder [12], and produces signatures fully compatible with the ECDSA standard [1, 18], that is the Elliptic-Curve version of DSA [19]. ECDSA is used in many cryptocurrencies, including the most widely known and widespread: Bitcoin [25] and Ethereum [6].

The scheme comprises four phases: Preliminary Phase, Enrollment Phase, Signature Phase and Recovery Phase; for each of them we report the actors involved as well as the input/output values, specifying which values are to be kept private, which one are private and included in the mnemonic, and which ones are publicly known and can therefore be stored without particular precautions.

The protocol employs internal checks to protect against errors, attackers and malicious actors. Therefore it is possible that the procedure fails before completion and the actors must abort (and possibly start over).

3.1 ECDSA - Assumptions

The group G is the group of points of an elliptic curve, and the generator g is the base point $\mathcal{B}$. The curve has prime order q, so the scalars are in $\mathbb{Z}_q$.

3.2 ECDSA - Preliminary Phase

The Preliminary Phase occurs just once, at the beginning of the protocol. The actors involved in this phase are the custodian A and the recovery server B. This is the workflow of this phase:

1. B generates a non-ephemeral private/public key pair $(\mathsf{sk}_B, \mathsf{pk}_B)$ associated to some encryption scheme with plaintext space $\mathbb{Z}_q$.
2. B sends the public key pk_B to A.

Note 1. The encryption algorithm which generates the key pair $(\mathsf{sk}_B, \mathsf{pk}_B)$ is unrelated to the signature algorithm.

Note 2. B keeps the private key sk_B secret and never reveals it to other actors.

3.3 ECDSA - Enrollment Phase

This phase occurs whenever a new client C contacts A. The actors involved in this phase are A and C.

Player A
Input: pk_B
Output: BIP39: w_A; Private: $p_A, q_A, \mathsf{rec}_{BA}, \mathsf{rec}_{CB}, d$; Public: $\Gamma_A, N_C, \Gamma_C, y$.

Player C
Input: $-$
Output: BIP39: w_C; Private: $p_C, q_C, \mathsf{rec}_{BA}, \mathsf{rec}_{CB}, d$; Public: $\Gamma_C, N_A, \Gamma_A, y$.

The workflow of this phase is:

1. Recovery public key communication:
   a) A sends pk_B to C.
2. Secrets generation:
   a) A generates Paillier public key (N_A, Γ_A) and secret key (p_A, q_A).
   C generates Paillier public key (N_C, Γ_C) and secret key (p_C, q_C)
   b) A picks randomly $u_A \in \mathbb{Z}_q$.
   C picks randomly $u_C \in \mathbb{Z}_q$.
   c) A computes $[\mathsf{KGC}_A, \mathsf{KGD}_A] := \text{Com}(u_A\mathcal{B})$.
   C computes $[\mathsf{KGC}_C, \mathsf{KGD}_C] := \text{Com}(u_C\mathcal{B})$.
   d) A sends KGC_A to C.
   C sends KGC_C to A.
   e) A sends (N_A, Γ_A) to C.
   C sends (N_C, Γ_C) to A.
   f) A sends KGD_A to C.
   C sends KGD_C to A.
   g) A gets $y_C := \text{Ver}(\mathsf{KGC}_C, \mathsf{KGD}_C)$.
   C gets $y_A := \text{Ver}(\mathsf{KGC}_A, \mathsf{KGD}_A)$.
   h) A picks randomly $m_A \in \mathbb{Z}_q$.
   C picks randomly $m_C \in \mathbb{Z}_q$.
   i) A sets $f_A(x) := u_A + m_A x \mod q$, and $\sigma_{AB} := f_A(1)$, $\sigma_{AA} := f_A(2)$, $\sigma_{AC} := f_A(3)$.
   C sets $f_C(x) := u_C + m_C x \mod q$, and $\sigma_{CB} := f_C(1)$, $\sigma_{CA} := f_C(2)$, $\sigma_{CC} := f_C(3)$.
   j) A picks randomly $\sigma_{BA} \in \mathbb{Z}_q$.
   C picks randomly $\sigma_{BC} \in \mathbb{Z}_q$.
   k) A encrypts σ_{AB} and σ_{BA} with the public key pk_B, getting rec_{AB}.
   C encrypts σ_{CB} and σ_{BC} with the public key pk_B, getting rec_{CB}.

3. Shards communication:

   a) A sends σ_{AC}, $m_A\mathcal{B}$, $\sigma_{BA}\mathcal{B}$, $\sigma_{AB}\mathcal{B}$, rec_{AB} to C.
   C sends σ_{CA}, $m_C\mathcal{B}$, $\sigma_{BC}\mathcal{B}$, $\sigma_{CB}\mathcal{B}$, rec_{CB} to A.

4. Private key generation:

   a) A computes $x_A := \sigma_{AA} + \sigma_{BA} + \sigma_{CA}$.
   C computes $x_C := \sigma_{AC} + \sigma_{BC} + \sigma_{CC}$.

5. ZK proofs:

   a) A proves in ZK that A knows x_A using Schnorr's protocol.
   C proves in ZK that C knows x_C using Schnorr's protocol.

   b) A proves to C that he knows p_A, q_A such that $N_A = p_A q_A$ using integer factorization ZK Proof.
   C proves to A that he knows p_C, q_C such that $N_C = p_C q_C$ using integer factorization ZK Proof.

6. Public key generation and shares conversion:

   a) A and C compute the public key $y := y_A + y_B + y_C$ where $y_B := 3(\sigma_{BA}\mathcal{B}) - 2(\sigma_{BC}\mathcal{B})$ and $y_{rec} := (\sigma_{AB}\mathcal{B}) + 2(\sigma_{BA}\mathcal{B}) - (\sigma_{BC}\mathcal{B}) + (\sigma_{CB}\mathcal{B})$.

   b) A computes $w_A := 3x_A$.
   C computes $w_C := -2x_C$.

   c) A and C compute the common secret $d := \sigma_{BA}\sigma_{BC}\mathcal{B}$.

Note 3. $\sigma_{BA} = u_B + 2m_B, \sigma_{BC} = u_B + 3m_B \implies u_B = 3\sigma_{BA} - 2\sigma_{BC}$.

Note 4. $X_A := x_A\mathcal{B}$ and $X_C := x_C\mathcal{B}$ are public.

Note 5. Defining $x := u_A + u_B + u_C$, we have that $w_A + w_C = x$ and $y = x\mathcal{B}$.

3.4 ECDSA - Signature Phase

Two actors P_1 and P_2 want to sign a message M.

Each actor P_i, for $i \in \{1, 2\}$, is supposed to know a secret $w_{P_i} \in \mathbb{Z}_q$ and the public key y.

In the case of the usual signature between the custodian P_1 = A and the client P_2 = C, everything needed has been computed during the "Enrollment Phase".

Player P_1

Input: $M, w_{P_1}, \Gamma_{P_1}, p_{P_1}, q_{P_1}, N_{P_2}, \Gamma_{P_2}, y$.

Output: BIP39: $-$; Private: $-$; Public: (r, s).

Player P_2
Input: $M, w_{P_2}, \Gamma_{P_2}, p_{P_2}, q_{P_2}, N_{P_1}, \Gamma_{P_1}, y$.
Output: BIP39: $-$; Private: $-$; Public: (r, s).

The workflow of this phase is:

1. Commitment:
   a) Each P_i picks randomly $k_i, \gamma_i \in \mathbb{Z}_q$, computes $\Gamma_i = \gamma_i \mathcal{B}$ and $[\Delta_i, D_i] := \mathrm{Com}(\Gamma_i)$.
   b) P_1 sends Δ_1 to P_2
   P_2 sends Δ_2 to P_1.
2. Multiplicative to Additive Share Conversion:
   a) Each P_i computes $N_{P_i}, \lambda_{P_i}, \mu_{P_i}$ from $\Gamma_{P_i}, p_{P_i}, q_{P_i}$, as shown in Note 18 of Appendix B.
   b) P_1 and P_2 run $\mathrm{MtA}(k_1, \gamma_2)$: P_1 gets α_{12} and P_2 gets β_{12}
   (such that $k_1 \cdot \gamma_2 = \alpha_{12} + \beta_{12}$).
   P_2 and P_1 run $\mathrm{MtA}(k_2, \gamma_1)$: P_2 gets α_{21} and P_1 gets β_{21}
   (such that $k_2 \cdot \gamma_1 = \alpha_{21} + \beta_{21}$).
   c) P_1 sets $\delta_1 := k_1 \cdot \gamma_1 + \alpha_{12} + \beta_{21}$.
   P_2 sets $\delta_2 := k_2 \cdot \gamma_2 + \alpha_{21} + \beta_{12}$.
   d) P_1 and P_2 run $\mathrm{MtAwc}(k_1, w_{P_2})$: P_1 gets μ_{12} and P_2 gets ν_{12}
   (such that $k_1 \cdot w_{P_2} = \mu_{12} + \nu_{12}$).
   P_2 and P_1 run $\mathrm{MtAwc}(k_2, w_{P_1})$: P_2 gets μ_{21} and P_1 gets ν_{21}
   (such that $k_2 \cdot w_{P_1} = \mu_{21} + \nu_{21}$).
   e) P_1 sets $\sigma_1 := k_1 \cdot w_{P_1} + \mu_{12} + \nu_{21}$.
   P_2 sets $\sigma_2 := k_2 \cdot w_{P_2} + \mu_{21} + \nu_{12}$.
3. Communication:
   a) P_1 sends δ_1 to P_2.
   P_2 sends δ_2 to P_1.
   b) P_1 and P_2 compute $\delta := \delta_1 + \delta_2$ and $\delta^{-1} \mod q$.
4. Decommitment:
   a) P_1 sends D_1 to P_2.
   P_2 sends D_2 to P_1.
   b) P_1 gets $\Gamma_2 := \mathrm{Ver}(\Delta_2, D_2)$.
   P_2 gets $\Gamma_1 := \mathrm{Ver}(\Delta_1, D_1)$.
   c) Each P_i proves in ZK that P_i knows γ_i such that $\gamma_i \mathcal{B} = \Gamma_i$, using Schnorr's protocol.
   d) P_1 and P_2 compute $R := \delta^{-1}(\Gamma_1 + \Gamma_2)$ and $r := R_x$, where R_x is the first component of the point $R = (R_x, R_y)$.

Note 6. $R = k^{-1}\mathcal{B}$, where $k := k_1 + k_2$.

5. Signature and verification:

   a) Each P_i computes $m := H(M)$.

   b) Each P_i computes $s_i := mk_i + r\sigma_i$.

   c) Each P_i randomly picks $l_i, \rho_i \in \mathbb{Z}_q$, computes $V_i := s_iR + l_i\mathcal{B}$, $A_i := \rho_i\mathcal{B}$ and $[\hat{\Delta}_i, \hat{D}_i] = \mathrm{Com}(V_i, A_i)$.

   d) P_1 sends $\hat{\Delta}_1$ to P_2.
   P_2 sends $\hat{\Delta}_2$ to P_1.

   e) P_1 sends $\hat{D}_1$ to P_2.
   P_2 sends $\hat{D}_2$ to P_1.

   f) P_1 gets $[V_2, A_2] := \mathrm{Ver}(\hat{\Delta}_2, \hat{D}_2)$.
   P_2 gets $[V_1, A_1] := \mathrm{Ver}(\hat{\Delta}_1, \hat{D}_1)$.

   g) Each P_i proves in ZK that P_i knows s_i, l_i, ρ_i such that $V_i = s_iR + l_i\mathcal{B}$ and $A_i = \rho_i\mathcal{B}$ (if a ZK proof fails, the protocol aborts).

   h) P_1 and P_2 compute $V := -m\mathcal{B} - ry + V_1 + V_2$ and $A := A_1 + A_2$.

   i) Each P_i computes $U_i := \rho_iV$ and $T_i := l_iA$, and $[\tilde{\Delta}_i, \tilde{D}_i] := \mathrm{Com}(U_i, T_i)$.

   j) P_1 sends $\tilde{\Delta}_1$ to P_2.
   P_2 sends $\tilde{\Delta}_2$ to P_1.

   k) P_1 sends $\tilde{D}_1$ to P_2.
   P_2 sends $\tilde{D}_2$ to P_1.

   l) P_1 gets $[U_2, T_2] := \mathrm{Ver}(\tilde{\Delta}_2, \tilde{D}_2)$.
   P_2 gets $[U_1, T_1] := \mathrm{Ver}(\tilde{\Delta}_1, \tilde{D}_1)$.

   m) If $T_1 + T_2 \neq U_1 + U_2$ the protocol aborts.

   n) P_1 sends s_1 to P_2.
   P_2 sends s_2 to P_1.

   o) P_1 and P_2 compute $s := s_1 + s_2$.

   p) If (r, s) is not a valid signature, the players abort, otherwise they accept and end the protocol.

3.5 ECDSA - Key Derivation

In order to perform the key derivation, we need a derivation index i and the common secret d. We use the following substitutions:

- A and C perform key derivation:

$$\begin{aligned} w_A &\mapsto w_A^i := w_A + 3H(d\|i) \\ w_C &\mapsto w_C^i := w_C - 2H(d\|i) \\ y &\mapsto y^i := y + H(d\|i)\mathcal{B}. \end{aligned}$$

- A and B perform key derivation:

$$\begin{aligned} w_A &\mapsto w_A^i := w_A - H(d\|i) \\ w_B &\mapsto w_B^i := w_B + 2H(d\|i) \\ y &\mapsto y^i := y + H(d\|i)\mathcal{B}. \end{aligned}$$

- C and B perform key derivation:

$$\begin{aligned} w_C &\mapsto w_C^i := w_C - \frac{1}{2}H(d\|i) \\ w_B &\mapsto w_B^i := w_B + \frac{3}{2}H(d\|i) \\ y &\mapsto y^i := y + H(d\|i)\mathcal{B}. \end{aligned}$$

Note 7. $x^i := w_A^i + w_C^i = x + H(d\|i)$, see Note 5.

3.6 ECDSA - Mnemonic

We use a BIP39-like mnemonic encoding (see Appendix A) to save the secret values. In particular the mnemonic for the client C (which is the party most susceptible to key loss) is computed at the end of the Enrollment Phase as:

$$\mathsf{mnemo} = \text{BIP39\_encode}(w_C, (d\|y\|\mathsf{rec}_{AB}\|\mathsf{rec}_{CB}))$$

Later on, C can re-initialize itself from its mnemonic mnemo with the help of A:

- First A sends C the values $d, y, \mathsf{rec}_{AB}, \mathsf{rec}_{CB}$
- then C retrieves its secret:

$$w_C = \text{BIP39\_decode}(\mathsf{mnemo}, 256, d, y, \mathsf{rec}_{AB}, \mathsf{rec}_{CB})$$

- If the recovery is successful C generates another pair of Paillier keys performing the steps 2.a, 2.e and 5.b of the Enrollment Phase (A may generate new keys or re-use the old ones).

Once this steps are successfully completed C can resume its normal operation.

3.7 ECDSA - Recovery Phase

Player B
Input: M, sk_B
Output: BIP39: $-$; Private: $-$; Public: (r, s).

We consider the case in which one of the following actors is not able to sign.

Player A
Input: $M, y, \mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \Gamma_A, p_A, q_A$
Output: <u>BIP39</u>: –; <u>Private</u>: –; <u>Public</u>: (r, s).

Player C
Input: $M, y, \mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \Gamma_C, p_C, q_C$
Output: <u>BIP39</u>: –; <u>Private</u>: –; <u>Public</u>: (r, s).

We consider now the case in which the client C is not able to sign, while A and B want to sign.

The workflow of this case is:

1. Communication:
   a) A contacts B, which comes back online.
   b) A sends y and $(\mathsf{rec}_{AB}, \mathsf{rec}_{CB})$ to B.
2. Paillier keys generation and exchange:
   a) A computes $N_A := p_A q_A$.
   B generates Paillier public key (N_B, Γ_B) and secret key (p_B, q_B).
   b) A sends (N_A, Γ_A) to B.
   B sends (N_B, Γ_B) to A.
   c) A proves to B that he knows p_A, q_A such that $N_A = p_A q_A$ using integer factorization ZK Proof.
   B proves to A that he knows p_B, q_B such that $N_B = p_B q_B$ using integer factorization ZK Proof.
3. B's secrets generation:
   a) B decrypts rec_{AB} and rec_{CB} with its private key sk_B, getting $\sigma_{AB}, \sigma_{BA}, \sigma_{CB}, \sigma_{BC}$.
   b) B computes $x_B := \sigma_{AB} + 2\sigma_{BA} - \sigma_{BC} + \sigma_{CB}$.
   c) B proves in ZK that B knows x_B using Schnorr's protocol.
4. Signature:
   a) A computes $\widetilde{w}_A := -\frac{1}{3} w_A$.
   b) B computes $w_B := 2x_B$.
   c) A and B perform the "Signature Phase" (see Section 3.4) as P_1 and P_2 respectively, where the A uses $\widetilde{w}_A$ in place of w_A.

We consider now the case in which the custodian A is not able to sign, while C and B want to sign.

The workflow of this case is:

1. Communication:

a) C contacts B, which comes back online.

b) C sends y and $(\mathsf{rec}_{AB}, \mathsf{rec}_{CB})$ to B.

2. Paillier keys generation and exchange:

   a) C computes $N_C := p_C q_C$.
   B generates Paillier public key (N_B, Γ_B) and secret key (p_B, q_B).

   b) C sends (N_C, Γ_C) to B.
   B sends (N_B, Γ_B) to C.

   c) C proves to B that he knows p_C, q_C such that $N_C = p_C q_C$ using integer factorization ZK Proof.
   B proves to C that he knows p_B, q_B such that $N_B = p_B q_B$ using integer factorization ZK Proof.

3. B's secrets generation:

   a) B decrypts rec_{AB} and rec_{CB} with its private key sk_B, getting $\sigma_{AB}, \sigma_{BA}, \sigma_{CB}, \sigma_{BC}$.

   b) B computes $x_B := \sigma_{AB} + 2\sigma_{BA} - \sigma_{BC} + \sigma_{CB}$.

   c) B proves in ZK that B knows x_B using Schnorr's protocol.

4. Signature:

   a) C computes $\widetilde{w}_C := \frac{1}{4} w_C$.

   b) B computes $w_B := \frac{3}{2} x_B$.

   c) C and B perform the "Signature Phase" (see Section 3.4) as P_1 and P_2 respectively, where the C uses $\widetilde{w}_C$ in place of w_C.

Note 8. Notation: the fractions represent an element of $\mathbb{Z}_q$, that may be computed as the multiplication of the numerator by the inverse of the denominator modulo q. That is let D^{-1} be the unique element of $\mathbb{Z}_q$ such that $D \cdot D^{-1} \mod q = 1$, then $\frac{N}{D} := N \cdot D^{-1} \mod q$. Similarly the minus sign indicates the opposite element in $\mathbb{Z}_q$: $-x := q - x \mod q$

Note 9. The recovery server B can calculate the common secret $d = \sigma_{BA}\sigma_{BC}\mathcal{B}$ decrypting, using its private key, the values rec_{AB} and rec_{CB} from which it obtains σ_{BA} and σ_{BC}. Since the generator $\mathcal{B}$ of the group is publicly known, B can compute the value d and perform the signature also for any derived key.

4 EdDSA version

This is a variant of the protocol explained in the previous section where the same structure is adapted in order to produce signatures indistinguishable from EdDSA signatures [4] (instead of ECDSA signatures). EdDSA is a variant of the Schnorr signature scheme [29, ?] using Twisted Edwards curves [3, 17]. The main cryptocurrencies that use EdDSA are Stellar [24], Libra [2], and Tezos [16], and notably none of them use a multi-sig protocol.

The scheme still comprises four phases: Preliminary Phase, Enrollment Phase, Signature Phase and Recovery Phase; for each of them we report the actors involved as well as the input/output values, specifying which values are to be kept private, which one are private and included in the mnemonic, and which ones are publicly known and can therefore be stored without particular precautions.

The protocol employs internal checks to protect against errors, attackers and malicious actors. Therefore it is possible that the procedure fails before completion and the actors must abort (and possibly start over).

4.1 EdDSA - Assumptions

The curve has order $2^c q$, where q is prime and the scalars are in $\mathbb{Z}_q$. The group G is the group generated by the base point $\mathcal{B}$.

4.2 EdDSA - Preliminary Phase

The Preliminary Phase occurs just once, at the beginning of the protocol. The actors involved in this phase are the custodian A and the recovery server B. This is the workflow of this phase:

1. B generates a non-ephemeral private/public key pair $(\mathsf{sk}_B, \mathsf{pk}_B)$ associated to some encryption scheme with plaintext space $\mathbb{Z}_q$.
2. B sends the public key pk_B to A.

Note 10. The encryption algorithm which generates the key pair $(\mathsf{sk}_B, \mathsf{pk}_B)$ is unrelated to the signature algorithm.

Note 11. B keeps the private key sk_B secret and never reveals it to other actors.

4.3 EdDSA - Enrollment Phase

This phase occurs whenever a new client C contacts A. The actors involved in this phase are A and C.

Player A

Input: pk_B

Output: BIP39: w_A, r'_A; Private: $\mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB}, \mathcal{R}, d$; Public: $\mathcal{A}$.

Player C

Input: –

Output: BIP39: w_C, r'_C; Private: $\mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB}, \mathcal{R}, d$; Public: $\mathcal{A}$.

The workflow of this phase is:

1. Recovery public key communication:
   a) A sends pk_B to C.

2. Secrets generation:

   a) A picks randomly $u_A \in \mathbb{Z}_q$.
      C picks randomly $u_C \in \mathbb{Z}_q$.
   b) A computes $[\mathsf{KGC}_A, \mathsf{KGD}_A] := \mathrm{Com}(u_A \mathcal{B})$.
      C computes $[\mathsf{KGC}_C, \mathsf{KGD}_C] := \mathrm{Com}(u_C \mathcal{B})$.
   c) A sends KGC_A to C.
      C sends KGC_C to A.
   d) A sends KGD_A to C.
      C sends KGD_C to A.
   e) A gets $y_C := \mathrm{Ver}(\mathsf{KGC}_C, \mathsf{KGD}_C)$.
      C gets $y_A := \mathrm{Ver}(\mathsf{KGC}_A, \mathsf{KGD}_A)$.
   f) A picks randomly $m_A \in \mathbb{Z}_q$.
      C picks randomly $m_C \in \mathbb{Z}_q$.
   g) A sets $f_A(x) := u_A + m_A x \mod q$, and $\sigma_{AB} := f_A(1)$, $\sigma_{AA} := f_A(2)$, $\sigma_{AC} := f_A(3)$.
      C sets $f_C(x) := u_C + m_C x \mod q$, and $\sigma_{CB} := f_C(1)$, $\sigma_{CA} := f_C(2)$, $\sigma_{CC} := f_C(3)$.
   h) A picks randomly $\sigma_{BA} \in \mathbb{Z}_q$.
      C picks randomly $\sigma_{BC} \in \mathbb{Z}_q$.
   i) A encrypts σ_{AB} and σ_{BA} with the public key pk_B, getting rec_{AB}.
      C encrypts σ_{CB} and σ_{BC} with the public key pk_B, getting rec_{CB}.

3. Shards communication:

   a) A sends σ_{AC}, $m_A\mathcal{B}$, $\sigma_{BA}\mathcal{B}$, $\sigma_{AB}\mathcal{B}$, rec_{AB} to C.
      C sends σ_{CA}, $m_C\mathcal{B}$, $\sigma_{BC}\mathcal{B}$, $\sigma_{CB}\mathcal{B}$, rec_{CB} to A.

4. Private key generation:

   a) A computes $x_A := \sigma_{AA} + \sigma_{BA} + \sigma_{CA}$.
      C computes $x_C := \sigma_{AC} + \sigma_{BC} + \sigma_{CC}$.

5. Second secret generation:

   a) Steps 2, 3 and 4 are repeated (except passages 2.a and 2.e) so that:
      A has also elements $x'_A, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB}$;
      C has also elements $x'_C, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB}$

6. ZK proofs:

   a) A proves in ZK that A knows x_A, x'_A using Schnorr's protocol.
      C proves in ZK that C knows x_C, x'_C using Schnorr's protocol.

7. Public key generation and shares conversion:

   a) A and C compute the public key $\mathcal{A} := y_A + y_B + y_C$, where $y_B := 3(\sigma_{BA}\mathcal{B}) - 2(\sigma_{BC}\mathcal{B})$, and $y_{rec} := (\sigma_{AB}\mathcal{B}) + 2(\sigma_{BA}\mathcal{B}) - (\sigma_{BC}\mathcal{B}) + (\sigma_{CB}\mathcal{B})$.
   b) A computes $w_A := 3x_A$ and $r'_A := 3x'_A$.
      C computes $w_C := -2x_C$ and $r'_C := -2x'_C$.
   c) A and C compute $\mathcal{R} := y'_A + y'_B + y'_C = (r'_A + r'_C)\mathcal{B}$, where $y'_B = 3\sigma'_{BA}\mathcal{B} - 2\sigma'_{BC}\mathcal{B}$, and $y'_{rec} := (\sigma'_{AB}\mathcal{B}) + 2(\sigma'_{BA}\mathcal{B}) - (\sigma'_{BC}\mathcal{B}) + (\sigma'_{CB}\mathcal{B})$.
   d) A and C compute the common secret $d := \sigma_{BA}\sigma_{BC}\mathcal{B}$.

Note 12. Defining $x := u_A + u_B + u_C$ and $x' := u'_A + u'_B + u'_C$, we have that $w_A + w_C = x$, $y = x\mathcal{B}$ and $r'_A + r'_C = x'$.

4.4 EdDSA - Signature Phase

Two actors P_1 and P_2 want to sign a message M.

Each actor P_i is supposed to know two secrets $w_{P_i}, r'_{P_i} \in \mathbb{Z}_q$ and the public key $\mathcal{A}$.

In the case of the usual signature between the custodian P_1 = A and the client P_2 = C, everything needed has been computed during the "Enrollment Phase".

Player P_1

Input: $M, w_{P_1}, r'_{P_1}, \mathcal{A}, \mathcal{R}$.

Output: BIP39: $-$; Private: $-$; Public: (R, S).

Player P_2

Input: $M, w_{P_2}, r'_{P_1}, \mathcal{A}, \mathcal{R}$.

Output: BIP39: $-$; Private: $-$; Public: (R, S).

The workflow of this phase is:

1. Signature generation - component 1 (R):
   a) Each P_i computes $R_i := r'_i H(\mathcal{R} \| M)\mathcal{B}$
   b) P_1 computes $[\mathsf{KGC}_{P_1}, \mathsf{KGD}_{P_1}] := \mathrm{Com}(R_1)$.
      P_2 computes $[\mathsf{KGC}_{P_2}, \mathsf{KGD}_{P_2}] := \mathrm{Com}(R_2)$.
   c) P_1 sends KGC_{P_1} to P_2.
      P_2 sends KGC_{P_2} to P_1.
   d) P_1 sends KGD_{P_1} to P_2.
      P_2 sends KGD_{P_2} to P_1.
   e) P_1 gets $R_2 := \mathrm{Ver}(\mathsf{KGC}_{P_2}, \mathsf{KGD}_{P_2})$.
      P_2 gets $R_1 := \mathrm{Ver}(\mathsf{KGC}_{P_1}, \mathsf{KGD}_{P_1})$.
   f) P_1 and P_2 compute $R := R_1 + R_2$.

   Note 13. $R = (r'_1 + r'_2)H(\mathcal{R} \| M)\mathcal{B}$ is the first component of the signature.

2. Signature generation - component 2 (S):
   a) Each P_i computes $S_i := r'_i H(\mathcal{R} \| M) + w_i H(R \| \mathcal{A} \| M)$
   b) P_1 computes $[\mathsf{KGC}'_{P_1}, \mathsf{KGD}'_{P_1}] := \mathrm{Com}(S_1)$.
      P_2 computes $[\mathsf{KGC}'_{P_2}, \mathsf{KGD}'_{P_2}] := \mathrm{Com}(S_2)$.
   c) P_1 sends KGC'_{P_1} to P_2.
      P_2 sends KGC'_{P_2} to P_1.

d) P_1 sends KGD'_{P_1} to P_2.
P_2 sends KGD'_{P_2} to P_1.

e) P_1 gets $S_2 := \mathrm{Ver}(\mathsf{KGC}'_{P_2}, \mathsf{KGD}'_{P_2})$.
P_2 gets $S_1 := \mathrm{Ver}(\mathsf{KGC}'_{P_1}, \mathsf{KGD}'_{P_1})$.

f) P_1 and P_2 compute $S := S_1 + S_2$.

> **Note 14.** $S = (r'_1+r'_2)H(\mathcal{R}\|M)+(w_1+w_2)H(R\|\mathcal{A}\|M)$ is the second component of the signature.

3. Signature check:

a) P_1 and P_2 check that $2^cS\mathcal{B} = 2^cR + 2^cH(R\|\mathcal{A}\|M)\mathcal{A}$. If a check is not true, the protocol aborts, otherwise the signature is (R, S).

4.5 EdDSA - Key derivation

In order to perform the key derivation, we need a derivation index i and a common secret d.

We use the following substitutions:

- A and C perform key derivation:

$$\begin{aligned}
w_A &\mapsto w_A^i := w_A + 3H(d\|i)\\
r'_A &\mapsto {r'_A}^i := r_A + 3H(d\|i)\\
w_C &\mapsto w_C^i := w_C - 2H(d\|i)\\
r'_C &\mapsto {r'_C}^i := r_C - 2H(d\|i)\\
\mathcal{A} &\mapsto \mathcal{A}^i := \mathcal{A} + H(d\|i)\mathcal{B}.\\
\mathcal{R} &\mapsto \mathcal{R}^i := \mathcal{R} + H(d\|i)\mathcal{B}.
\end{aligned}$$

- A and B perform key derivation:

$$\begin{aligned}
w_A &\mapsto w_A^i := w_A - H(d\|i)\\
r_A &\mapsto {r'_A}^i := r'_A - H(d\|i)\\
w_B &\mapsto w_B^i := w_B + 2H(d\|i)\\
r'_B &\mapsto {r'_B}^i := r'_B + 2H(d\|i)\\
\mathcal{A} &\mapsto \mathcal{A}^i := y + H(d\|i)\mathcal{B}.\\
\mathcal{R} &\mapsto \mathcal{R}^i := y + H(d\|i)\mathcal{B}.
\end{aligned}$$

- C and B perform key derivation:

$$\begin{aligned}
w_C &\mapsto w_C^i := w_C - \frac{1}{2}H(d\|i)\\
r_C &\mapsto {r'_C}^i := r'_C - \frac{1}{2}H(d\|i)\\
w_B &\mapsto w_B^i := w_B + \frac{3}{2}H(d\|i)\\
r_B &\mapsto {r'_B}^i := r'_B + \frac{3}{2}H(d\|i)\\
\mathcal{A} &\mapsto \mathcal{A}^i := \mathcal{A} + H(d\|i)\mathcal{B}.\\
\mathcal{R} &\mapsto \mathcal{R}^i := \mathcal{R} + H(d\|i)\mathcal{B}.
\end{aligned}$$

Note 15. $x^i := 3s^i_A - 2s^i_C = x + H(d\|i)$, $x'^i := 3r'^i_A - 2r'^i_C = x + H(d\|i)$, see Note 12.

4.6 EdDSA - Mnemonic

We use the BIP39 mnemonic encoding (see Appendix A) to save the secret values. In particular the mnemonic for the client C (which is the party most susceptible to key loss) is computed at the end of the Enrollment Phase as:

$$\mathsf{mnemo} = \text{BIP39\_encode}((w_C, r'_C, (d\|\mathcal{A}\|\mathcal{R}\|\mathsf{rec}_{AB}\|\mathsf{rec}_{CB}\|\mathsf{rec}'_{AB}\|\mathsf{rec}'_{CB}))$$

Later on, C can re-initialize itself from its mnemonic mnemo with the help of A:

- First A sends C the values $d, \mathcal{A}, \mathcal{R}, \mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB}$
- then C retrieves its secret:

$$w_C = \text{BIP39\_decode}(\mathsf{mnemo}, 512, d, \mathcal{A}, \mathcal{R}, \mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB})$$

Once this steps are successfully completed C can resume its normal operation.

Note 16. In BIP39_decode function the length 512 is the maximum bit representation of the secrets.

4.7 EdDSA - Recovery Phase

Player B

Input: M, sk_B

Output: BIP39: $-$; Private: $-$; Public: (R, S).

We consider the case in which one of the following actors is not able to sign.

Player A

Input: $M, \mathcal{A}, \mathcal{R}, \mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB}$

Output: BIP39: $-$; Private: $-$; Public: (R, S).

Player C

Input: $M, \mathcal{A}, \mathcal{R}, \mathsf{rec}_{AB}, \mathsf{rec}_{CB}, \mathsf{rec}'_{AB}, \mathsf{rec}'_{CB}$

Output: BIP39: $-$; Private: $-$; Public: (R, S).

We consider now the case in which the client C is not able to sign, while A and B want to sign.

The workflow of this case is:

1. Communication:
   a) A contacts B, which comes back online.
   b) A sends $\mathcal{A}$, $\mathcal{R}$, rec_{AB}, rec_{CB}, rec'_{AB}, rec'_{CB} to B.
2. B's key creation:
   a) B decrypts rec_{AB}, rec_{CB}, rec'_{AB} and rec'_{CB} with its private key sk_B, getting: σ_{AB}, σ_{BA}, σ_{CB}, σ_{BC}, σ'_{AB}, σ'_{BA}, σ'_{CB}, σ'_{BC}.
   b) B computes:
      - $x_B := \sigma_{AB} + 2\sigma_{BA} - \sigma_{BC} + \sigma_{CB}$;
      - $x'_B := \sigma'_{AB} + 2\sigma'_{BA} - \sigma'_{BC} + \sigma'_{CB}$.
   c) B proves in ZK that B knows x_B, x'_B using Schnorr's protocol.
3. Signature:
   a) A computes $\widetilde{w}_A := -\frac{1}{3}w_A$ and $\widetilde{r}'_A := -\frac{1}{3}{r_A}'$.
      B computes $w_B := 2x_B$ and $r'_B := 2x'_B$.
   b) A and B perform the "Signature Phase" (see 4.4) as P_1 and P_2 respectively, where the A uses $\widetilde{w}_A$ and $\widetilde{r}'_A$ in place of w_A and r'_A

We consider now the case in which the custodian A is not able to sign, while C and B want to sign.

The workflow of this case is:

1. Communication:
   a) C contacts B, which comes back online.
   b) C sends $\mathcal{A}$, $\mathcal{R}$, rec_{AB}, rec_{CB}, rec'_{AB}, rec'_{CB} to B.
2. B's key creation:
   a) B decrypts rec_{AB}, rec_{CB}, rec'_{AB} and rec'_{CB} with its private key sk_B, getting: σ_{AB}, σ_{BA}, σ_{CB}, σ_{BC}, σ'_{AB}, σ'_{BA}, σ'_{CB}, σ'_{BC}.
   b) B computes:
      - $x_B := \sigma_{AB} + 2\sigma_{BA} - \sigma_{BC} + \sigma_{CB}$;
      - $x'_B := \sigma'_{AB} + 2\sigma'_{BA} - \sigma'_{BC} + \sigma'_{CB}$.
   c) B proves in ZK that B knows x_B, x'_B using Schnorr's protocol.
3. Signature:
   a) C computes $\widetilde{w}_C := \frac{1}{4}w_C$ and $\widetilde{r}'_C := \frac{1}{4}{r_C}'$.
   b) B computes $w_B := \frac{3}{2}x_B$. and $r'_B := \frac{3}{2}x'_B$.
   c) C and B perform the "Signature Phase" (see 4.4) as P_1 and P_2 respectively, where the C uses $\widetilde{w}_C$ and $\widetilde{r}'_C$ in place of w_C and r'_C.

Note 17. The recovery server B can calculate the common secret $d = \sigma_{BA}\sigma_{BC}\mathcal{B}$ decrypting, using its private key, the values rec_{AB} and rec_{CB} from which it obtains σ_{BA} and σ_{BC}. Since the generator $\mathcal{B}$ of the group is publicly known, B can compute the value d and perform the signature also for any derived key.

5 Security Claims and Conclusion

Both protocols presented in the previous sections are provable secure in the standard model, although the actual proofs are omitted here, they will be included in a more complete future version. More precisely the signatures are proven unforgeable following the standard notion of existential unforgeability against chosen message attacks (EU-CMA) as introduced in [15], adapted for a threshold multi-party scheme.

Definition 5.1 (Existential Unforgeability of a 2-out-of-3 Threshold Signature Protocol)**.** *Consider a malicious Probabilistic Polynomial Time (*PPT*) adversary $\mathcal{A}$ who participates in a 2-out-of-3 threshold signature protocol $S = (\mathrm{KeyGen}, \mathrm{Sig}, \mathrm{Ver})$ for the creation of a public key pk and in creation of signatures on adaptively chosen messages of its choosing. Let M be the set of messages queried by $\mathcal{A}$. The protocol is said to be existentially unforgeable if there is no such PPT adversary $\mathcal{A}$ that can produce a signature on a message $m \notin M$, except with negligible probability.*

The main result is the following:

Theorem 5.1. *Assuming that:*

- *the original signature scheme is unforgeable;*
- *the Strong RSA Assumption holds;*
- *there is a non-malleable commitment scheme;*
- *the DDH Assumption holds;*

then our threshold scheme is unforgeable.

The design of these schemes aims to solve the real-world problem of the custody of digital assets, in particular cryptocurrencies, adding resiliency to key loss without sacrificing control and decentralization.

References

[1] A. B. Association et al. Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa). *ANSI X9*, pages 62–1998.

[2] L. Association et al. Libra white paper. *Internet access:,[accessed June 19, 2019]*, 2019.

[3] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted edwards curves. In *International Conference on Cryptology in Africa*, pages 389–405. Springer, 2008.

[4] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[5] D. Boneh, R. Gennaro, and S. Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security, 2017.

[6] V. Buterin et al. Ethereum white paper. *GitHub repository*, pages 22–23, 2013.

[7] U. W. Chohan. The problems of cryptocurrency thefts and exchange shutdowns. *Available at SSRN 3131702*, 2018.

[8] J. Doerner, Y. Kondi, E. Lee, and A. Shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 980–997. IEEE, 2018.

[9] J. Doerner, Y. Kondi, E. Lee, and A. Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1051–1066. IEEE, 2019.

[10] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[11] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Annual International Cryptology Conference*, pages 16–30. Springer, 1997.

[12] R. Gennaro and S. Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194. ACM, 2018.

[13] R. Gennaro, S. Goldfeder, and A. Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156–174. Springer, 2016.

[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold dss signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 354–371. Springer, 1996.

[15] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[16] L. Goodman. Tezos—a self-amending crypto-ledger white paper. *URL: https://www. tezos. com/static/papers/white_paper. pdf*, 2014.

[17] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted edwards curves revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 326–343. Springer, 2008.

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

[19] D. W. Kravitz. Digital signature algorithm, July 27 1993. US Patent 5,231,668.

[20] Y. Lindell. Fast secure two-party ecdsa signing. In *Annual International Cryptology Conference*, pages 613–644. Springer, 2017.

[21] Y. Lindell and A. Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1837–1854. ACM, 2018.

[22] P. MacKenzie and M. K. Reiter. Two-party generation of dsa signatures. In *Annual International Cryptology Conference*, pages 137–154. Springer, 2001.

[23] P. MacKenzie and M. K. Reiter. Two-party generation of dsa signatures. *International Journal of Information Security*, 2(3-4):218–239, 2004.

[24] D. Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, page 32, 2015.

[25] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[26] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[27] M. Palatinus, P. Rusnak, A. Voisine, and S. Bowe. Mnemonic code for generating deterministic keys. *Online at https://github. com/bitcoin/bips/blob/master/bip-0039. mediawiki*, 2013.

[28] G. Poupard and J. Stern. Short proofs of knowledge for factoring. In *International Workshop on Public Key Cryptography*, pages 147–166. Springer, 2000.

[29] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[30] P. Wuille. Bip32: Hierarchical deterministic wallets. *h ttps://github. com/genjix/bips/blob/master/bip-0032. md*, 2012.

A BIP39-like Mnemonic

The BIP39 mnemonic encoding allows to save binary data as a list of common words that can be memorized and later used to reconstruct the original bytes.

The protocol uses this method to save critical private data in order to avoid the Recovery Signature Phase as long as the mnemonic is available. Normally it is the client C that takes advantage of this enhancement (since it is the party most susceptible to key loss), but the same steps may be performed by the custodian A.

We now describe the functions BIP39_encode and BIP39_decode that can be used by an Actor P to retrieve its secret values and resume normal operativity.

A.1 BIP39_encode

It is the function that computes the mnemonic mnemo, with input a primary secret sec and a complementary secret comp.

- Let $I :=$ H_Indexes(sec) be a list of unique indexes I_j pseudo-randomly generated from the secret sec, and denote with z_i the i-th bit of the binary representation of a generic element z, and denote with $\#z$ the bit-length of z.
- C constructs a sequence of l bits $b := (b_i)_{i=0\ldots l-1}$ where

$$b_i := \begin{cases} \mathsf{sec}_i & 0 \leq i < \#z \\ H(\mathsf{comp})_{I_{i-\#z}} & \#z \leq i < l \end{cases}$$

- P computes the checksum bits $\mathsf{check} := \mathrm{Checksum}(b)$ and the list of words

$$\mathsf{mnemo} := \mathrm{Mnemonic}(b, \mathsf{check})$$

A.2 BIP39_decode

It is the function that retrieves a primary secret sec of length ℓ and checks the validity of the complementary secret comp, with input a mnemonic mnemo a complementary secret comp, and the bit-length of the secret ℓ.

- The bits encoded by mnemo are retrieved and separated from the checksum:

$$(b, \mathsf{check}) = \mathrm{Retrieve}(\mathsf{mnemo})$$

- The validity of the checksum is verified: $\mathsf{check} = \mathrm{Checksum}(b)$
- The first ℓ bits of b are extracted as the value sec and the set of indexes $I :=$ H_Indexes(sec) is computed
- The integrity of comp is checked verifying that

$$b_i = H(\mathsf{comp})_{I_{i-\ell}} \qquad \ell \leq i < l$$

- if all the checks are successful, the algorithm outputs sec, otherwise it fails.

B Paillier encryption scheme

In this appendix we describe the additively homomorphic encryption scheme of Paillier [26].

B.1 Key Generation

This function generates the public and private keys for the encryption scheme.

1. Choose two random safe primes P and Q, different and having the same length.
2. Compute $N := PQ$ and $\lambda := \mathrm{lcm}(P-1, Q-1)$.
3. Pick $\Gamma \in \mathbb{Z}_{N^2}^*$ such that its order is a multiple of N: as explained in [26] it is enough to pick randomly Γ and check whether the following inverse exists $\mu := (L(\Gamma^\lambda \mod N^2))^{-1} \pmod N$, where the function L is the integer division quotient $L(u) := (u-1)/N$.
4. Return the public key (N, Γ) and the private key (P, Q, λ, μ).

Note 18. Note that just by the knowledge of Γ, P, Q, we can compute all the other values of the key:

$$N = PQ, \qquad \lambda = \mathrm{lcm}(P-1, Q-1), \qquad \mu = (L(\Gamma^\lambda \mod N^2))^{-1} \pmod N$$

B.2 Encryption

Given as input the public key (N, Γ) and a message $m \in \mathbb{Z}_N$, this function encrypts the message m with the public key (N, Γ) in the following way:

1. Pick randomly $r \in \mathbb{Z}_N^*$.
2. Return the ciphertext $c := \Gamma^m r^N \pmod{N^2}$.

B.3 Decryption

Given as input the public key (N, Γ), the private key (λ, μ) and a ciphertext $c \in \mathbb{Z}_{N^2}^*$, this function decrypts the ciphertext c with the private key (λ, μ) in the following way:

1. Return the plaintext $m := L(c^\lambda \mod N^2) \cdot \mu \pmod N$.

B.4 Homomorphic properties

Let E and D denote the encryption and decryption functions respectively.

Given two ciphertexts $c_1, c_2 \in \mathbb{Z}_{N^2}^*$, associated to the plaintexts $m_1, m_2 \in \mathbb{Z}_N$ respectively, i.e. $c_1 = E(m_1)$ and $c_2 = E(m_2)$, then we have that

$$c_1 +_E c_2 := c_1 c_2 \pmod{N^2}$$

$$D(c_1 +_E c_2) = m_1 + m_2 \mod N.$$

Similarly, given a ciphertext $c = E(m)$ and a number $a \in \mathbb{Z}_N$, we have that

$$a \times_E c := c^a \pmod{N^2}$$

$$D(a \times_E c) = am \pmod N.$$

C Commitments

In this appendix we describe the commitment system that we use in the protocol.

We assume to work on a group G.

C.1 Commitment function

Com
Input: $g_1, \dots, g_n \in G$
Output: C string, D string

1. Generates a random value R.
2. Transform (via bijection) each element g_i into a sequence of bits z_i.
3. Set C to be the hash of $R \| z_1 \| \dots \| z_n$.
4. Set $D := (R, z_1, \dots, z_n)$.

Note 19. If g_i is a point of an elliptic curve, then a possible bijection is to take z_i as the concatenation of its two components.

C.2 Decommitment function

Ver
Input: C string, D string
Output: $g_1, \dots, g_n \in G$ / false

1. Get the values $R, z_1, \dots, z_n$ from the tuple D.
2. Set C' to be the hash of $R \| z_1 \| \dots \| z_n$.
3. If $C' \neq C$ return false.
4. Transform (via the bijection) each integer z_i into the corresponding element g_i.
5. Return $g_1, \dots, g_n$.

D Feldman's verifiable secret sharing protocol

In cryptography, a secret sharing scheme is verifiable if auxiliary information is included that allows players to verify that their shares are consistent and define a unique secret. More formally, verifiable secret sharing ensures that even if the dealer is malicious there is a well-defined secret that the players can later reconstruct.

A VSS scheme is the protocol by Paul Feldman [10], which is based on Shamir's secret sharing scheme combined with any homomorphic encryption scheme.

D.1 Shamir's SS

In order to share a secret $s \in \mathbb{Z}_q$ so that at least $t+1$ shares are necessary to reconstruct it, the dealer generates a random polynomial of degree t:

$$p(x) = s + a_1 x + a_2 x^2 + \cdots + a_t x^t \in \mathbb{Z}_q[x]$$

(note that $p(0) = s$). The secret shares are the evaluations $s_i := p(i) \mod q$.

D.2 Feldman's VSS

It extends the Shamir's SS protocol in the following way. We consider a group G with generator g where the DLOG is (supposed to be) hard.

The dealer also publishes commitments to the coefficients of p:

- $c_0 := g^s$
- $c_i := g^{a_i}$ for every $i \in \{1, \ldots, t\}$

Using this auxiliary information, each player P_i can check its share s_i for consistency (i.e. that s_i is actually $p(i) \mod q$), by checking:

$$g^{s_i} = \prod_{j=0}^{t} c_j^{i^j} = c_0 c_1^i \cdots c_t^{i^t}.$$

D.3 Our case

In our case $t = 1$ and $n = 3$, so the whole VSS to share the secret $s \in \mathbb{Z}_q$ becomes:

- The dealer picks randomly $a_1 \in \mathbb{Z}_q$ and considers the line $p(x) = s + a_1 x \in \mathbb{Z}_q[x]$.
- The dealer sends to the players $s_1 := p(1) \mod q$, $s_2 := p(2) \mod q$, and $s_3 := p(3) \mod q$.
- The dealer publishes $c_0 := g^s$ and $c_1 := g^{a_1}$.

E Multiplicative to additive share conversion protocol

Assume that two actors Alice and Bob have multiplicative shares $a, b \in \mathbb{Z}_q$ respectively of a secret $x = ab \mod q$. In this appendix we show a protocol that lets Alice an Bob to convert their multiplicative shares into additive shares $\alpha, \beta \in \mathbb{Z}_q$ of x, i.e. such that $x = \alpha + \beta \mod q$. This protocol is based on an additively homomorphic encryption scheme, like the one presented in appendix B.

E.1 MtA (Multiplicative to Additive) share conversion protocol

Assumptions:

- q is a known prime.
- Alice has a secret $a \in \mathbb{Z}_q$.
- Bob has a secret $b \in \mathbb{Z}_q$.
- Alice is associated with a Paillier public key (N, Γ).
  The encryption function associated will be denoted by $E_A(\cdot)$.
- K is a public integer such that $K > q$ and $N > K^2 q$.

MtA protocol:

1. Alice
   - Sends $c_A := E_A(a)$ to Bob.
   - Proves in ZK that $a < K$ via a range proof.
2. Bob
   - Picks randomly $\beta' \in \mathbb{Z}_N$.
   - Computes $c_B := (b \times_E c_A) +_E E_A(\beta')$ via homomorphic properties.
   - Sets $\beta := -\beta' \mod q$.
   - Sends c_B to Alice.
   - Proves in ZK that $b < K$.
3. Alice
   - Decrypts c_B getting α'.
   - Sets $\alpha := \alpha' \mod q$.

E.2 MtAwc (Multiplicative to Additive with check) share conversion protocol

Assumptions:

- q is a known prime.
- Alice has a secret $a \in \mathbb{Z}_q$.
- Bob has a secret $b \in \mathbb{Z}_q$.
- Alice is associated with a Paillier public key (N, Γ).
  The encryption function associated will be denoted by $E_A(\cdot)$.
- K is a public integer such that $K > q$ and $N > K^2 q$.
- There is a public group G of order q and generator g.
- $B := g^b$ is public.

MtAwc protocol:

1. Alice
   - Sends $c_A := E_A(a)$ to Bob.
   - Proves in ZK that $a < K$ via a range proof.
2. Bob
   - Picks randomly $\beta' \in \mathbb{Z}_N$.
   - Computes $c_B := (b \times_E c_A) +_E E_A(\beta')$ via homomorphic properties.
   - Sets $\beta := -\beta' \mod q$.
   - Sends c_B to Alice.
   - Proves in ZK that $b < K$.
   - Proves in ZK that he knows b, β' such that $B = g^b$ and $c_B = (b \times_E c_A) +_E E_A(\beta')$.
3. Alice
   - Decrypts c_B getting α'.
   - Sets $\alpha := \alpha' \mod q$.

Note 20. Both for the MtA and MtAwc protocols we require that $K \sim q^3$ and that $N \sim q^8$. Indeed, a typical choice of parameters is q 256-bit long, K 768-bit long and N 2048-bit long.

F Zero-Knowledge proofs

A Zero-Knowledge (ZK) proof is an interactive protocol between a prover who wants to convince a verifier that an object belongs to a language (proof of membership) or that he knows a secret information (proof of knowledge), without revealing anything about his secret knowledge.

Note 21. The following Zero-Knowledge proofs are described within the context of a group G, with generator g, in multiplicative notation.

F.1 Proof of knowledge of integer factorization

This proof has been taken from [28].

Given a public number n, the prover wants to prove to a verifier that he knows some prime numbers whose product is n, without giving any information about this decomposition.

Let us consider:

- n the public integer, whose number of digits in its binary expansion is denoted by $|n|$.
- k the security parameter.
- A, B, l, K integers which depend a priori on k and $|n|$.
- $z_1, \ldots, z_K$ random elements of $\mathbb{Z}_n^*$.

A round of the proof consists in the following steps:

a) The prover picks randomly an integer $r \in \{0, \ldots, A-1\}$.

b) The prover computes $x_i := z_i^r \pmod{n}$ for $i = 1, \ldots, K$ and sends them to the verifier.

c) The verifier picks randomly an integer $e \in \{0, \ldots, B-1\}$ and sends it to the prover.

d) The prover computes $y := r + (n - \phi(n)) \cdot e$ and sends it to the verifier.

e) The verifier checks that $0 \leq y < A$ and that $z_i^{y-ne} = x_i \pmod{n}$ for $i = 1, \ldots, K$.

A complete proof consists in repeating l times the elementary round.

In order to guarantee the soundness, the completeness and the security of the protocol, the choice of the parameters must respect the following constraints:

- $l \cdot \log B = \theta(k)$.
- $(n - \phi(n))lB < A < n$.
- $K \approx k$.

Note 22. The protocol works only if n does not have small prime factors.

Note 23. The z_i can be pseudo-randomly generated from a seed of the form $h(n, i)$ where h is a hash function.

Note 24. The x_i can be precomputed.

Note 25. It is possible to optimize the protocol by replacing $x_1, \dots, x_K$ by their hash value $H(x_1, \dots, x_K)$, where H is an appropriate hash function.

F.2 ZK proofs for the MtA protocol

These proofs have been taken from [12].

In these proofs the Verifier uses the following auxiliary data:

- An RSA modulus $\tilde{N}$, which is the product of two safe primes $\tilde{P} = 2\tilde{p} + 1$ and $\tilde{Q} = 2\tilde{q} + 1$ with $\tilde{p}, \tilde{q}$ primes.
- Two values $h_1, h_2 \in \mathbb{Z}^*_{\tilde{N}}$.

Note 26. The initialization protocol must be augmented with each actor generating those values, together with a proof that they are of the correct form (see [11]).

F.2.1 Range Proof

This proof is run in both MtA and MtAwc protocols, when Alice wants to prove to Bob that $a < K$.

In that context we have that:

- The Prover (Alice) is provided with:
  - A Paillier public key $(N, \Gamma) \in \mathbb{Z} \times \mathbb{Z}^*_{N^2}$.
  - A value $m \in \mathbb{Z}_q$ (which is the a in MtA(wc) protocol).
  - The ciphertext $c = \Gamma^m r^N \pmod{N^2} \in \mathbb{Z}_{N^2}$ with respect to Paillier's scheme, where $r \in \mathbb{Z}^*_N$ is the random quantity generated during the encryption.
- The Verifier (Bob), at the end of the protocol, is convinced that $m \in [-q^3, q^3]$.
  This is consistent with the choice of picking $K \sim q^3$.

Here we describe the protocol:

1. The Prover picks randomly $\alpha \in \mathbb{Z}_{q^3}$, $\beta \in \mathbb{Z}^*_N$, $\gamma \in \mathbb{Z}_{q^3\tilde{N}}$, $\rho \in \mathbb{Z}_{q\tilde{N}}$.
   The Prover computes $z := h_1^m h_2^\rho \pmod{\tilde{N}}$, $u := \Gamma^\alpha \beta^N \pmod{N^2}$, $w := h_1^\alpha h_2^\gamma \pmod{\tilde{N}}$.
   The Prover sends z, u, w to the Verifier.
2. The Verifier picks randomly $e \in \mathbb{Z}_q$.
   The Verifier sends e to the Prover.
3. The Prover computes $s := r^e \beta \pmod{N}$, $s_1 := em + \alpha$, $s_2 := e\rho + \gamma$.
   The Prover sends s, s_1, s_2 to the Verifier.
4. The Verifier checks that $s_1 \le q^3$, $u = \Gamma^{s_1} s^N c^{-e} \pmod{N^2}$, $h_1^{s_1} h_2^{s_2} z^{-e} = w \pmod{\tilde{N}}$.

F.2.2 Respondent ZK Proof for MtA

This proof is run in MtA protocol, when Bob wants to prove to Alice that $b < K$.

In that context we have that:

- Everyone knows:
  - The Paillier public key $(N, \Gamma) \in \mathbb{Z} \times \mathbb{Z}^*_{N^2}$ associated to the Verifier (Alice).
  - The ciphertext $c_1 \in \mathbb{Z}_{N^2}$ (corresponding to c_A).
  - The ciphertext $c_2 \in \mathbb{Z}_{N^2}$ (corresponding to c_B).
- The Prover (Bob) is provided with:
  - A value $x \in \mathbb{Z}_q$ (corresponding to b).
  - A value $y \in \mathbb{Z}_N$ (corresponding to β').
  - The ciphertext $c_2 = c_1^x \Gamma^y r^N \pmod{N^2} \in \mathbb{Z}_{N^2}$ with respect to Paillier's scheme, where $r \in \mathbb{Z}^*_N$ is the random quantity generated during the encryption of y.
- The Verifier (Alice), at the end of the protocol, is convinced that $x \in [-q^3, q^3]$ and that the Prover (Bob) knows $x \in \mathbb{Z}_q$, $y \in \mathbb{Z}_N$, $r \in \mathbb{Z}^*_N$ such that $c_2 = c_1^x \Gamma^y r^N \pmod{N^2}$.

Here we describe the protocol:

1. The Prover picks randomly $\alpha \in \mathbb{Z}_{q^3}$, $\rho \in \mathbb{Z}_{q\tilde{N}}$, $\rho' \in \mathbb{Z}_{q^3\tilde{N}}$, $\sigma \in \mathbb{Z}_{q\tilde{N}}$, $\beta \in \mathbb{Z}^*_N$, $\gamma \in \mathbb{Z}^*_N$, $\tau \in \mathbb{Z}_{q\tilde{N}}$.
   The Prover computes $z := h_1^x h_2^\rho \pmod{\tilde{N}}$, $z' := h_1^\alpha h_2^{\rho'} \pmod{\tilde{N}}$, $t := h_1^y h_2^\sigma \pmod{\tilde{N}}$, $v := c_1^\alpha \Gamma^\gamma \beta^N \pmod{N^2}$, $w := h_1^\gamma h_2^\tau \pmod{\tilde{N}}$.
   The Prover sends z, z', t, v, w to the Verifier.
2. The Verifier picks randomly $e \in \mathbb{Z}_q$ and sends it to the Prover.
3. The Prover computes $s := r^e \beta \pmod{N}$, $s_1 := ex + \alpha$, $s_2 := e\rho + \rho'$, $t_1 := ey + \gamma$, $t_2 := e\sigma + \tau$.
   The Prover sends s, s_1, s_2, t_1, t_2 to the Verifier.
4. The Verifier checks that $s_1 \leq q^3$, $h_1^{s_1} h_2^{s_2} = z^e z' \pmod{\tilde{N}}$, $h_1^{t_1} h_2^{t_2} = t^e w \pmod{\tilde{N}}$, $c_1^{s_1} s^N \Gamma^{t_1} = c_2^e v \pmod{N^2}$.

F.2.3 Respondent ZK Proof for MtAwc

This proof is run in MtAwc protocol, when Bob wants to prove to Alice that $b < K$ and that he knows b, β' such that $B = g^b$ and $c_B = (b \times_E c_A) +_E E_A(\beta')$.

In that context we have that:

- Everyone knows:
  - The Paillier public key $(N, \Gamma) \in \mathbb{Z} \times \mathbb{Z}^*_{N^2}$ associated to the Verifier (Alice).
  - The ciphertext $c_1 \in \mathbb{Z}_{N^2}$ (corresponding to c_A).
  - The ciphertext $c_2 \in \mathbb{Z}_{N^2}$ (corresponding to c_B).
  - A value $X \in G$ (corresponding to g^b in multiplicative notation or to $b\mathcal{B}$ in additive notation).

- The Prover (Bob) is provided with:
  - A value $x \in \mathbb{Z}_q$ (corresponding to b).
  - A value $y \in \mathbb{Z}_N$ (corresponding to β').
  - The ciphertext $c_2 = c_1^x \Gamma^y r^N \pmod{N^2} \in \mathbb{Z}_{N^2}$ with respect to Paillier's scheme, where $r \in \mathbb{Z}_N^*$ is the random quantity generated during the encryption.
- The Verifier (Alice), at the end of the protocol, is convinced that $x \in [-q^3, q^3]$ and that the Prover (Bob) knows $x \in \mathbb{Z}_q$, $y \in \mathbb{Z}_N$, $r \in \mathbb{Z}_N^*$ such that $c_2 = c_1^x \Gamma^y r^N \pmod{N^2}$ and $X = g^x$.

Here we describe the protocol:

1. The Prover picks randomly $\alpha \in \mathbb{Z}_{q^3}$, $\rho \in \mathbb{Z}_{q\tilde{N}}$, $\rho' \in \mathbb{Z}_{q^3\tilde{N}}$, $\sigma \in \mathbb{Z}_{q\tilde{N}}$, $\beta \in \mathbb{Z}_N^*$, $\gamma \in \mathbb{Z}_N^*$, $\tau \in \mathbb{Z}_{q\tilde{N}}$.
   The Prover computes $u := g^\alpha$, $z := h_1^x h_2^\rho \pmod{\tilde{N}}$, $z' := h_1^\alpha h_2^{\rho'} \pmod{\tilde{N}}$, $t := h_1^y h_2^\sigma \pmod{\tilde{N}}$, $v := c_1^\alpha \Gamma^\gamma \beta^N \pmod{N^2}$, $w := h_1^\gamma h_2^\tau \pmod{\tilde{N}}$.
   The Prover sends u, z, z', t, v, w to the Verifier.

2. The Verifier picks randomly $e \in \mathbb{Z}_q$ and sends it to the Prover.

3. The Prover computes $s := r^e \beta \pmod{N}$, $s_1 := ex + \alpha$, $s_2 := e\rho + \rho'$, $t_1 := ey + \gamma$, $t_2 := e\sigma + \tau$.
   The Prover sends s, s_1, s_2, t_1, t_2 to the Verifier.

4. The Verifier checks that $s_1 \leq q^3$, $g^{s_1} = X^e u \in G$, $h_1^{s_1} h_2^{s_2} = z^e z' \pmod{\tilde{N}}$, $h_1^{t_1} h_2^{t_2} = t^e w \pmod{\tilde{N}}$, $c_1^{s_1} s^N \Gamma^{t_1} = c_2^e v \pmod{N^2}$.

F.3 Schnorr's proofs

This proof has been taken from [29].

`schnorr_proof`	
Input:	Prover, Verifier, $q \in \mathbb{Z}$, G group, $g \in G$, $x \in \mathbb{Z}_q$, $y \in G$
Output:	true/false

We are in the context of a cyclic group G of cardinality q, with generator g. The Prover knows x such that $y = g^x$. The Verifier knows y. The Prover wants to prove to the Verifier that he knows such an x.

1. The Prover picks randomly $r \in \mathbb{Z}_q$.
   The Prover computes $t := g^r$.
   The Prover sends t to the Verifier.

2. The Verifier picks randomly $c \in \mathbb{Z}_q$.
   The Verifier sends c to the Prover.

3. The Prover computes $s := r + cx$.
   The Prover sends s to the Verifier.

4. The Verifier checks that $g^s = ty^c$.

This proof has been taken from [12].

`schnorr_proof2`
Input: Prover, Verifier, $q \in \mathbb{Z}$, G group, $g \in G$, $x_1 \in \mathbb{Z}_q$, $x_2 \in \mathbb{Z}_q$, $y_1 \in G$, $y_2 \in G$
Output: true/false

We are in the context of a cyclic group G of cardinality q, with generator g. The Prover knows x_1, x_2 such that $y_2 = y_1^{x_1} g^{x_2}$. The Verifier knows y_1, y_2. The Prover wants to prove to the Verifier that he knows such x_1, x_2.

1. The Prover picks randomly $r_1, r_2 \in \mathbb{Z}_q$.
   The Prover computes $t := {y_1}^{r_1} g^{r_2}$.
   The Prover sends t to the Verifier.

2. The Verifier picks randomly $c \in \mathbb{Z}_q$.
   The Verifier sends c to the Prover.

3. The Prover computes $s_1 := r_1 + cx_1 \pmod{q}$, $s_2 := r_2 + cx_2 \pmod{q}$.
   The Prover sends s_1, s_2 to the Verifier.

4. The Verifier checks that ${y_1}^{s_1} g^{s_2} = t{y_2}^{c}$.