Mediterr. J. Math. (2022) 19:4
https://doi.org/10.1007/s00009-021-01886-3

Mediterranean Journal of Mathematics

Threshold ECDSA with an Offline Recovery Party

Michele Battagliola, Riccardo Longo, Alessio Meneghetti and Massimiliano Sala

Abstract. A (t, n)-threshold signature scheme enables distributed signing among n players such that any subset of size at least t can sign, whereas any subset with fewer players cannot. Our goal is to produce digital signatures that are compatible with an existing centralized signature scheme: the key-generation and signature algorithms are replaced by a communication protocol between the players, but the verification algorithm remains identical to that of a signature issued using the centralized algorithm. Starting from the threshold scheme for the ECDSA signature due to Gennaro and Goldfeder, we present the first protocol that supports multiparty signatures with an offline participant during the key-generation phase and that does not rely on a trusted third party. Under standard assumptions on the underlying algebraic and geometric problems (e.g. the Discrete Logarithm Problem for an elliptic curve and the computation of eth root on semi-prime residue rings), we prove our scheme secure against adaptive malicious adversaries.

Mathematics Subject Classification. Primary 94Axx; Secondary 12Exx, 14Hxx, 68Wxx.

Keywords. 94A60 cryptography, 12E20 finite fields, 14H52 elliptic curves, 94A62 authentication and secret sharing, 68W40 analysis of algorithms.

1. Introduction

A (t, n)-threshold signature scheme enables distributed signing among n players such that any subset of size t can sign, whereas any subset with fewer players cannot. The first Threshold Multi-Party Signature Scheme was a protocol for ECDSA signatures proposed by Gennaro et al. [16] where $t + 1$ parties out of $2t + 1$ were required to sign a message. Later MacKenzie and Reiter proposed and then improved another scheme [26,27], which has later been furthermore enhanced [10,11,23]. The first scheme supporting a general

Published online: 22 November 2021

(t, n)-threshold was proposed in [15], improved in [4] and in [14]. A parallel approach has been taken by Lindell and Nof in [24]. In [21] the authors introduce a refresh mechanism, for proactive security against the corruption of different actors in time, that does not require all parties to be online, and in [7] the authors take a similar approach and propose a protocol that streamlines signature generation and include proactive security mechanisms. Currently there is a large effort of standardization for threshold signatures, as can be seen in [5].
The schemes proposed in the previous papers produce signatures that are compatible with an existing centralized signature scheme. In this context, the key-generation and signature algorithms are replaced by a communication protocol between the parties, while the verification algorithm remains identical to that of a signature issued using the centralized algorithm.

The need for joint signatures arises frequently in the world of cryptocurrencies, where 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. Three possible solutions are:

- to rely on a trusted third-party custodian that takes responsibility of key management, but this kind of centralization may form single points of failure and juicy targets for criminal takeovers (there have already been plenty of examples of said events in the past [8]).
- to use multi-sig wallets (available for some cryptocurrencies, like Bitcoin [29]) 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. Unfortunately, this approach is not supported by every cryptocurrency (e.g. Ethereum [6]) and such wallets are very easily identifiable.
- to distribute the control of the wallet through advanced multi-signature schemes, in particular with threshold-like policies. This solution may be reached by threshold multi-sig compatible with centralised digital signatures, as in the aforementioned work by Gennaro and Goldfeder and related papers.

Regarding this last option, there is a potential problem in real-life applications of these protocols: the recovery party (that allows to recover the wallet funds in case of key loss) is usually 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 the bank participate in the enrollment of every user.

In this paper, following the latter approach, we propose a protocol in which the recovery party is involved only once (in a preliminary set-up), and afterwards it is not involved until a lost account must be recovered. Our protocol may be seen as an adaption of that in [14]. We prove its security against (adaptive) adversaries by relying on standard assumptions on the underlying

algebraic and geometric problems, such as the strong RSA assumption on semi-prime residue rings and the DDH assumption on elliptic curves.

Organization We present some preliminaries in Sect. 2, we describe our protocol in Sect. 3. Another practical problem is to derive many keys from a single secret, for example to efficiently manage multiple wallets. Therefore, we provide also an extension of our protocol that can work with key-derivation in Sect. 3.6. In Sect. 4 we state the security property that we claim for our protocol, with a proof similar to that in [14], which however requires several subtle modifications to tackle the off-line situation. The adaption of our proof to the key-derivation extension is cumbersome but easy and so we do not provide it here. Finally in Sect. 5 we draw our conclusions.

2. Preliminaries

In this section we present some preliminary definitions and primitives that will be used in the protocol and its proof of security.

In the following when we say that an algorithm is *efficient* we mean that it runs in (expected) polynomial time in the size of the input, possibly using a random source.

2.1. Assumptions

Our proof is based on two assumptions: the decisional Diffie and Hellman [3] (from now on DDH) and the RSA [32] assumption.

Definition 2.1. (*DDH assumption*) Let $\mathbb{G}$ be a cyclic group with generator g and order n. Let a, b, c be random elements of $\mathbb{Z}_n$. The decisional Diffie–Hellman assumption, from now on DDH assumption, states that no efficient algorithm can distinguish between the two distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c).

Definition 2.2. (*RSA assumption*) Let $N = pq$ with both p, q safe primes. Let e be an integer such that e and $\phi(N)$ are coprime.

- The *RSA assumption* states that given a random element $s \in \mathbb{Z}_N^*$ no efficient algorithm can find x such that $x^e = s \mod N$.
- The *Strong RSA assumption* states that given a random element $s \in \mathbb{Z}_N^*$ no efficient algorithm can find $x, e_0 \neq 1$ such that $x^{e_0} = s \mod N$.

2.2. Zero-Knowledge Proofs

In the protocol various zero-knowledge proofs (ZKP) [18] are used to enforce the respect of the passages prescribed by the specifications. In fact in the proof of security we can exploit the soundness of these sub-protocols to extract valuable information from the adversary, and their zero-knowledge property to simulate correct executions even without knowing some secrets. We can do so because we see the adversary as a (black-box) algorithm that we can call on arbitrary input, and crucially we have the faculty to rewind its execution.

In particular we use ZKP *of Knowledge* (ZKPoK) to guarantee the usage of secret values that properly correspond to the public counterpart: the

Schnorr protocol for discrete logarithms (see [34] and Appendix A.1) and an integer-factorization proof (see [31] and Appendix A.2) for Paillier keys. The soundness property of a ZKPoK guarantees that the adversary must know the secret input, and opportune rewinds and manipulations of the adversary's execution during the proof allows us to extract those secrets and use them in the simulation. Conversely exploiting the zero-knowledge property we can trick the adversary in believing that we know our secrets even if we don't, thus we still obtain a correct simulation of our protocol form the adversary's point of view.

The other ZKP used is a range proof (see [14,26] and appendix A.3) that guarantees a proper execution of the share conversion protocol of Sect. 2.5, however the same discussion of [14], Sect. 5 (about how the security is not significatively affected removing this proof) applies to our protocol.

2.3. Paillier Cryptosystem

In our protocol we use the Paillier cryptosystem, a partially homomorphic asymmetric encryption scheme presented by Paillier in [30]. Suppose that Bob wants to send an encrypted message to Alice. The workflow of the algorithm is the following:

- **Key-generation:**
  1. Alice chooses two large primes p, q uniformly at random, such that $\gcd(pq, (p-1)(q-1)) = 1$.
  2. Alice sets $N = pq$ and $\lambda = \text{lcm}(p-1, q-1)$.
  3. Alice picks $\Gamma \in \mathbb{Z}^*_{N^2}$ uniformly at random. In this context $\mathbb{Z}^*_{N^2}$ indicates the ring of units of $\mathbb{Z}_{N^2}$.
  4. Alice checks that N divides the order of Γ. To do so it is sufficient (see [30, Section 3]) to compute $\mu = (L(g^\lambda \mod N^2))^{-1} \mod N$, where $L(x)$ is the quotient of the Euclidean division $\frac{x-1}{N}$, i.e. the largest integer value k such that $x - 1 \geq kN$.
  5. The public encryption key is (Γ, N). The private encryption key is (μ, λ).
- **Message encryption:**
  1. To send a message $m \in \mathbb{Z}_N$ to Alice Bob picks $r \in \mathbb{Z}^*_N$.
  2. Bob computes $c = \Gamma^m r^N \mod N^2$.
- **Decryption**
  1. Alice computes $m = L(c^\lambda \mod N^2) \cdot \mu \mod N$.

Let E and D be the encryption and decryption functions respectively. Given two plaintexts $m_1, m_2 \in Z_N$, their associated ciphertexts $c_1, c_2 \in Z^*_{N^2}$ and $a \in Z_N$ then we define $+_E$ and $\times_E$ as follows:

$$\begin{aligned} c_1 +_E c_2 &= c_1 c_2 \mod N^2, \\ a \times_E c_1 &= c_1^a \mod N^2. \end{aligned}$$

Then the homomorphic properties of the Pallier cryptosystem are:

$$\begin{aligned} D(c_1 +_E c_2) &= m_1 + m_2 \mod N, \\ D(a \times_E c_1) &= a m_1 \mod N. \end{aligned}$$

2.4. ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA), presented in [20], is a variant of the Digital Signature Algorithm (DSA) [22] which uses elliptic curve cryptography.

Suppose Alice wants to send a signed message m to Bob. Initially, they agree on a cryptographic hash function [33] H, an elliptic curve E, a base point $\mathcal{B}$ for E, with n the order of $\mathcal{B}$ a prime. For any point $\mathcal{P} \in E$ we use the notation $\mathcal{P}_x$ to denote the value of the first coordinate of the point $\mathcal{P}$. The protocol works as follows:

1. Alice creates a key-pair consisting of a private integer d, selected uniformly at random in the interval $[1, n-1]$ and the public point $\mathcal{Q} = d\mathcal{B}$.
2. Alice computes $e = H(m)$ and picks k uniformly at random in the interval $[1, n-1]$.
3. Alice computes the point $\mathcal{R} = k^{-1}\mathcal{B}$.
4. Alice computes $s = k(e + rd)$ with $r = \mathcal{R}_x$.

The signature is the pair (r, s).
To verify the signature Bob performs the following steps:

1. Bob checks that $r, s \in [1, n-1]$,
2. Bob computes $e = H(m)$,
3. Bob computes $u_1 = es^{-1} \mod n$ and $u_2 = rs^{-1} \mod n$,
4. Bob computes the point $\mathcal{U} = u_1\mathcal{B} + u_2\mathcal{Q}$,
5. checks that $r \equiv \mathcal{U}_x \mod n$.

2.5. A Share Conversion Protocol

Assume that we have two parties, Alice and Bob, holding two secrets, respectively $a, b \in \mathbb{Z}_q$ with q prime, such that $ab = x \mod q$. We can imagine a, b as private shards of a shared secret x. Alice and Bob would like to perform a multiplicative-to-additive conversion to compute $\alpha, \beta \in \mathbb{Z}_q$ such that $\alpha + \beta = x \mod q$. In this section we will present a protocol for this based on the Paillier Encryption Scheme (see Sect. 2.3). We assume that Alice has a public key $A = (N, \Gamma)$, and E_A will indicate the Pallier encryption with A. Moreover we need a value $K > q$.

1. Alice initiates the protocol:
   - Alice computes $\mathsf{c}_A = E_A(a)$ and sends it to Bob.
   - Alice proves in ZK that $a < K$ via the first range proof explained in Appendix A.3.
2. Bob generates his shard β:
   - Bob computes $\mathsf{c}_B = b \times_E \mathsf{c}_A +_E E_A(\beta') = E_A(ab + \beta')$ where β' is chosen uniformly at random in $\mathbb{Z}_N$.
   - Bob sends c_B to Alice.
   - Bob proves in ZK that $b < K$ via the second range proof presented in Appendix A.3.
   - If $B = g^b$ is public Bob proves in ZK that he knows b, β' such that $g^b = B$ and $\mathsf{c}_B = b \times_E \mathsf{c}_A +_E E_A(\beta')$.
3. Alice decrypts c_B to obtain α'.
4. Alice obtains her shard $\alpha = \alpha' \mod q$, Bob obtains his shard $\beta = -\beta'$.

The protocol takes two different names depending on whether $B = g^b$ is public or not. In the first case we refer to this protocol as MtAwc (Multiplicative to Additive with check), because Bob performs the extra check at the end, in the second we refer to it simply as MtA.

For more details about the protocol and the security proof see [14].

2.6. Feldman-VSS

Feldman's VSS scheme [13] is a verifiable secret sharing scheme built on top of Shamir's scheme [36]. A secret sharing scheme is verifiable if auxiliary information is included, that allows players to verify the consistency of their shares. We use a simplified version of Feldman's protocol: if the verification fails the protocol does not attempt to recover excluding malicious participants, instead it aborts altogether. In a sense we consider *somewhat honest* participants, for this reason we do not need stronger schemes such as [17,35]. The scheme works as follows:

1. A cyclic group $\mathbb{G}$ of prime order p is chosen, as well as a generator $g \in \mathbb{G}$. The group $\mathbb{G}$ must be chosen such that the discrete logarithm is hard to compute.
2. The dealer computes a random polynomial P of degree t with coefficients in $\mathbb{Z}_p$, such that $P(0) = s$ where s is the secret to be shared.
3. Each of the n share holders receive a value $P(1), ..., P(n) \mod p$. So far, this is exactly Shamir's scheme.
4. To make these shares verifiable, the dealer distributes commitments to the coefficients of p. Let $P(X) = s + \sum_{i=1}^{n} a_i X^i$, then the commitments are $c_0 = g^s$ and $c_i = g^{a_i}$ for $i > 0$.
5. Any party can verify its share in the following way: let α be the share received by the i-th party, then it can check if $\alpha = P(i)$ by checking if the following equality holds:

$$g^{\alpha} = \prod_{j=0}^{t} c_j^{(i^j)} = g^s \prod_{j=1}^{t} g^{a_j(i^j)} = g^{s+\sum_{j=1}^{t} a_j(i^j)} = g^{P(i)}.$$

In the proof we will need to simulate a $(2, 2)$-threshold instance of this protocol without knowing the secret value s.

Let us use an additive group with generator $\mathcal{B}$, and let $\mathcal{Y} = s\mathcal{B}$, the simulation proceeds as follows:

- the dealer selects two random values a, b and forces $P(1) = a$, $P(2) = b$;
- then it computes:

$$c_1 = (a\mathcal{B} - \mathcal{Y}), \tag{1}$$

$$c_2 = \frac{1}{2}(b\mathcal{B} - \mathcal{Y}); \tag{2}$$

- the other players can successfully verify their shards, checking that

$$a\mathcal{B} = \mathcal{Y} + c_1 = \mathcal{Y} + a\mathcal{B} - \mathcal{Y}, \tag{3}$$

$$b\mathcal{B} = \mathcal{Y} + 2c_2 = \mathcal{Y} + 2 \cdot \frac{1}{2}(b\mathcal{B} - \mathcal{Y}). \tag{4}$$

3. Protocol Description

In this section we describe the details of our protocol. After some common parameters are established, one player chooses a long-term asymmetric key and then can go offline, leaving the proper generation of the signing key to the remaining two participants. For this reason the signature algorithm is presented in two variants, one used jointly by the two players who performed the key-generation, and one used by the offline player and one of the others. More specifically the protocol is comprised by four phases:

1. **Setup phase** (Sect. 3.1): played by all the parties, it is used to decide common parameters. Note that in many contexts these parameters are mandated by the application, so the parties merely acknowledge them, possibly checking they respect the required security level.
2. **Key-generation** (Sect. 3.2): played by only two parties, from now on P_1 and P_2. It is used to create a public key and the private shards for each player.
3. **Ordinary signature** (Sect. 3.4): played by P_1 and P_2. As the name suggests this is the normal use-case of the protocol.
4. **Recovery signature** (Sect. 3.5): played by P_3 and one between P_1 and P_2. This models the unavailability of one player, with P_3 stepping up as a replacement.

From here on with the notation "P_i does something", we mean that both P_1 and P_2 perform the prescribed task independently. Similarly, the notation "P_i sends something to P_j" means that P_1 sends to P_2 and P_2 sends to P_1.

3.1. Setup Phase

This phase involves all the participants and is used to decide the parameters of the algorithm.
The parameters involved are the following:

Player 1 and 2		**Player 3**	
Input:	–	**Input:**	–
Private output:	–	**Private output:**	sk_3
Public output:	$E, \mathcal{B}, q, H$	**Public output:**	pk_3

P_3 chooses an asymmetric encryption algorithm and a key pair $(\mathsf{pk}_3, \mathsf{sk}_3)$, then it publishes pk_3, keeping sk_3 secret. pk_3 is the key that P_1 and P_2 will use to communicate with P_3. The algorithm which generates the key pair $(\mathsf{sk}_3, \mathsf{pk}_3)$ and the encryption algorithm itself are unrelated to the signature algorithm, but it is important that both of them are secure.
More formally we require that the encryption protocol has the property of IND-CPA [1,28]:

Definition 3.1. (*IND-CPA*) Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme. Let us define the following experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}^b$ parameterized by a bit b:

1. The challenger runs $\mathsf{Gen}(1^k)$ to get sk and pk, the secret and public keys. Then it gives pk to $\mathcal{A}$.
2. $\mathcal{A}$ outputs two messages (m_0, m_1) of the same length.
3. The challenger computes $\mathsf{Enc}(\mathsf{pk}, m_b)$ and gives it to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a bit b'(if it aborts without giving any output, we just set $b' = 0$). The challenger returns b' as the output of the game.

We say that Π has the property of being *indistinguishable under chosen plaintext attacks* (IND-CPA) or simply *secure against a chosen plaintext attack*, if for any k and any probabilistic polynomial time adversary $\mathcal{A}$ the function

$$\mathsf{Adv}(\mathcal{A}) = \mathbb{P}[\mathcal{C}^1(\mathcal{A}, k) = 1] - \mathbb{P}[\mathcal{C}^0(\mathcal{A}, k) = 1], \tag{5}$$

i.e. $\mathsf{Adv}(\mathcal{A}) = \mathbb{P}[b' = b] - \mathbb{P}[b' \neq b]$, is negligible.

Then P_1 and P_3 need to agree on a secure hash function H, an elliptic curve E with group of points of prime order q, and a generator $\mathcal{B} \in E$ of said group. The order identifies the ring $\mathbb{Z}_q$ used for scalar values.

3.2. Key-Generation

Player 1		**Player 2**	
Input:	pk_3	**Input:**	pk_3
Private output:	p_1, q_1, ω_1	**Private output:**	p_2, q_2, ω_2
Shared secrets:	$\mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}, d$	**Shared secrets:**	$\mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}, d$
Public output:	$\Gamma_1, N_2, \Gamma_2, \mathcal{Y}$	**Public output:**	$\Gamma_2, N_1, \Gamma_1, \mathcal{Y}$

The protocol proceeds as follows:

1. Secret key-generation and communication:
   (a) P_i generates a Paillier public key (N_i, Γ_i) and its corresponding secret key (p_i, q_i).
   (b) P_i selects randomly $u_i, \sigma_{3,i} \in \mathbb{Z}_q$.
   (c) P_i computes $[\mathsf{KGC}_i, \mathsf{KGD}_i] := \mathrm{Com}(u_i\mathcal{B})$.
   (d) P_i computes $[\mathsf{KGCS}_i, \mathsf{KGDS}_i] := \mathrm{Com}(\sigma_{3,i}\mathcal{B})$.
   (e) P_i sends KGC_i, KGCS_i, and (N_i, Γ_i) to P_j.
   (f) P_i sends $\mathsf{KGD}_i, \mathsf{KGDS}_i$ to P_j.
   (g) P_i computes $u_j\mathcal{B} = \mathrm{Ver}(\mathsf{KGC}_j, \mathsf{KGD}_j)$ and $\sigma_{3,j}\mathcal{B} = \mathrm{Ver}(\mathsf{KGCS}_j, \mathsf{KGDS}_j)$.
2. Feldman VSS protocol and generation of P_3's data:
   (a) P_i selects randomly $m_i \in \mathbb{Z}_q$.
   (b) P_i sets $f_i(X) = u_i + m_i X$, and $\sigma_{i,1} = f_i(2), \sigma_{i,2} = f_i(3), \sigma_{i,3} = f_i(1)$. Then P_i computes and distributes the shards $c_{i,j}$ for the Feldman-VSS, as described in Sect. 2.6.
   (c) Everyone checks the integrity and consistency of the shards according to the VSS protocol.
   (d) P_i encrypts $\sigma_{i,3}$ and $\sigma_{3,i}$ with pk_3 to obtain $\mathsf{rec}_{i,3}$.
   (e) P_i sends $\sigma_{i,j}, m_i\mathcal{B}, \mathsf{rec}_{i,3}$ to P_j.
3. P_i computes the private key $x_i = \sigma_{1,i} + \sigma_{2,i} + \sigma_{3,i}$.

4. P_i proves in ZK that it knows x_i using Schnorr's protocol of Appendix A.1.
5. P_i proves in ZK that it knows p_i, q_i such that $N_i = p_i q_i$ using the integer factorization ZKP of Appendix A.2.
6. Public key-generation and shares conversion:
   (a) the public key is $\mathcal{Y} = \sum_{i=1}^{3} u_i \mathcal{B}$, where $u_3\mathcal{B} = 3(\sigma_{3,1}\mathcal{B}) - 2(\sigma_{3,2}\mathcal{B})$. So $u_3 = 3\sigma_{3,1} - 2\sigma_{3,2}$. From now on we will set $u = \sum_{i=1}^{3} u_i$. Obviously $u\mathcal{B} = \mathcal{Y}$.
   (b) the private key of P_1 is $\omega_1 = 3x_1$, while the private key of P_2 is $w_2 = -2x_2$. We can observe that $w_1 + w_2 = u$.
   (c) P_1 and P_2 can compute the common secret $d = (\sigma_{2,1}\sigma_{2,3}\mathcal{B})_x$ that will be used for key derivation.

Observation 1. We define $u_3 = 3\sigma_{3,1} - 2\sigma_{3,2}$ because we need to be consistent with the Feldman-VSS protocol. Indeed, suppose that $\sigma_{3,2}$ and $\sigma_{3,1}$ are valid shards of a Feldman-VSS protocol where the secret is u_3. Since there is an m_3 such that $\sigma_{3,2} = u_3 + m_3 \cdot 3$ and $\sigma_{3,1} = u_3 + m_3 \cdot 2$, we have that:

$$3\sigma_{3,1} - 2\sigma_{3,2} = 3u_3 + 6m_3 - 2u_3 - 6m_3 = u_3.$$

Note that $u_3\mathcal{B}$ can be computed by both P_1 and P_2, but they cannot compute u_3.

3.3. Signature Algorithm

This protocol is used by two players, called P_A and P_B, to sign messages. P_1, P_2, and P_3 take the role of either P_A or P_B depending on the situation, see Sects. 3.4 and 3.5.

The participants agree on a message M to sign. The goal of this protocol is to produce a valid ECDSA signature (r, s) for the public key y.

The parameters involved are:

Player A		**Player B**	
Input:	$M, \omega_A, \Gamma_A, p_A, q_A, N_B, \Gamma_B, \mathcal{Y}$	**Input:**	$M, \omega_B, \Gamma_B, p_B, q_B, N_A, \Gamma_A, \mathcal{Y}$
Public output:	(r, s)	**Public output:**	(r, s)

The protocol works as follows:

1. Commitment phase:
   (a) P_i picks randomly $k_i, \gamma_i \in \mathbb{Z}_q$.
   (b) P_i computes $\mathcal{G}_i = \gamma_i\mathcal{B}$ and $[\Delta_i, D_i] = \text{Com}(\mathcal{G}_i)$. We define $\gamma = \gamma_A + \gamma_B$ and $k = k_A + k_B$.
   (c) P_i sends Δ_i to P_j.
2. Multiplicative to additive conversion:
   (a) P_i re-computes the parameters for Paillier encryption from its keys: $\lambda_i = \text{lcm}(p_i - 1, q_i - 1), \mu_i = (L_i(\Gamma_i^{\lambda_i} \mod N_i^2))^{-1} \mod N_i$ where $L_i(x) = \frac{x-1}{N_i}$.

(b) P_A and P_B run MtA(k_A, γ_B) to get respectively $\alpha_{A,B}, \beta_{A,B}$ such that $k_A\gamma_B = \alpha_{A,B} + \beta_{A,B}$. They also run it on k_B, γ_A to get respectively $\beta_{B,A}$ and $\alpha_{B,A}$.
(c) P_i sets $\delta_i = k_i\gamma_i + \alpha_{i,j} + \beta_{j,i}$.
(d) P_A and P_B run MtAwc(k_A, ω_B) to get respectively $\mu_{A,B}$ $\nu_{A,B}$ such that $k_A\omega_B = \mu_{A,B} + \nu_{A,B}$. They also run it on k_B, ω_A to get respectively $\nu_{B,A}$ and $\mu_{B,A}$.
(e) P_i sets $\sigma_i = k_i\omega_i + \mu_{ij} + \nu_{ji}$. We can observe that $\sum \sigma_i = ku$.
(f) P_i sends δ_i to P_j.
(g) P_A and P_B compute $\delta = \delta_A + \delta_B$ and $\delta^{-1} \mod q$.

3. Decommitment phase and ZKP:
(a) P_i sends D_i.
(b) P_i computes $\mathcal{G}_j = \text{Ver}(\Delta_j, D_j)$.
(c) P_i proves in ZK that it knows γ_i such that $\gamma_i\mathcal{B} = \mathcal{G}_i$, using Schnorr's protocol.
(d) P_A and P_A set $\mathcal{R} = \delta^{-1}(\mathcal{G}_A + \mathcal{G}_B)$ and $r = \mathcal{R}_x$. We can observe that $\delta = \gamma k$ and $\mathcal{G}_A + \mathcal{G}_B = \gamma\mathcal{B}$, so $\mathcal{R} = k^{-1}\mathcal{B}$.

4. Signature generation:
(a) Both players set $m = H(M)$.
(b) P_i computes $s_i = mk_i + r\sigma_i$.
(c) P_i picks uniformly at random $l_i, \rho_i \in \mathbb{Z}_q$, and computes $\mathcal{W}_i := s_i\mathcal{R} + l_i\mathcal{B}$, $\mathcal{Z}_i = \rho_i\mathcal{B}$, and then $[\hat{\Delta}_i, \hat{D}_i] = \text{Com}(\mathcal{W}_i, \mathcal{Z}_i)$.
(d) P_i sends $\hat{\Delta}_i$ to P_j.
(e) P_i sends $\hat{D}_i$ to P_j.
(f) P_i computes $[\mathcal{W}_j, \mathcal{Z}_j] := \text{Ver}(\hat{\Delta}_j, \hat{D}_j)$.
(g) Each P_i proves in ZK that it knows s_i, l_i, ρ_i such that $\mathcal{W}_i = s_iR + l_i\mathcal{B}$ and $\mathcal{Z}_i = \rho_i\mathcal{B}$ (if a ZKP fails, the protocol aborts).
(h) P_A and P_A compute $\mathcal{W} := -m\mathcal{B} - ry + \mathcal{W}_A + \mathcal{W}_B$ and $\mathcal{Z} := \mathcal{Z}_A + \mathcal{Z}_B$.
(i) Each P_i computes $\mathcal{U}_i := \rho_i\mathcal{W}$, $\mathcal{T}_i := l_i\mathcal{Z}$ and $[\tilde{\Delta}_i, \tilde{D}_i] := \text{Com}(\mathcal{U}_i, \mathcal{T}_i)$.
(j) P_i sends $\tilde{\Delta}_i$ to P_j.
(k) P_i sends $\tilde{D}_i$ to P_j.
(l) P_i computes $[\mathcal{U}_j, \mathcal{T}_j] := \text{Ver}(\tilde{\Delta}_j, \tilde{D}_j)$.
(m) If $\mathcal{T}_1 + \mathcal{T}_2 \neq \mathcal{U}_1 + \mathcal{U}_2$ the protocol aborts.
(n) P_i sends s_i.
(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.

3.4. Ordinary Signature

This is the case where P_1 and P_2 wants to sign a message m. They run the signature algorithm of Sect. 3.3 with the following parameters (supposing P_1 play the roles of P_A and P_2 of P_B):

Player A		**Player B**	
Input:	$M, \omega_1, \Gamma_1, p_1, q_1, N_2, \Gamma_2, \mathcal{Y}$	**Input:**	$M, \omega_2, \Gamma_2, p_2, q_2, N_1, \Gamma_1, \mathcal{Y}$
Public output:	(r, s)	**Public output:**	(r, s)

3.5. Recovery Signature

If one between P_1 and P_2 is unable to sign, then P_3 has to come back online and a recovery signature is performed.

We have to consider two different cases, depending on who is offline. First we consider the case in which P_2 is offline, therefore P_1 and P_3 sign. The parameters involved are:

Player 1		**Player 3**	
Input:	$M, \omega_1, \Gamma_1, p_1, q_1, \mathcal{Y}$ $\mathsf{rec}_{2,3}, \mathsf{rec}_{1,3}$	**Input:**	M, sk_3
Public output:	(r, s)	**Public output:**	(r, s)

The workflow in this case is:

1. Communication:
   (a) P_1 contacts P_3, which comes back online.
   (b) P_1 sends y and $\mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$ to P_3.
2. Paillier keys generation and exchange:
   (a) P_3 generates a Paillier public key (N_3, Γ_3) and its relative secret key (p_3, q_3).
   (b) P_i sends N_i, Γ_i to the other party.
   (c) P_i proves to P_j that it knows p_i, q_i such that $N_i = p_i q_i$ using integer factorization ZKP.
3. P_3's secrets generation:
   (a) P_3 decrypts $\mathsf{rec}_{1,3}$ and $\mathsf{rec}_{2,3}$ with its private key sk_3, getting $\sigma_{1,3}$, $\sigma_{3,1}$, $\sigma_{2,3}, \sigma_{3,2}$.
   (b) P_3 computes $x_3 = \sigma_{1,3} + 2\sigma_{3,1} - \sigma_{3,2} + \sigma_{2,3}$.
   (c) P_i proves in ZK that it knows x_i using Schnorr's protocol.
4. Signature generation:
   (a) P_1 computes $\tilde{\omega}_1 := -\frac{1}{3}\omega_1$.
   (b) P_3 computes $\omega_3 := 2x_3$.
   (c) P_1 and P_3 perform the Signature Algorithm of Sect. 3.3 as P_A and P_B respectively, where the P_1 uses $\tilde{\omega}_1$ in place of ω_A and P_3 uses ω_3 in place of ω_B (the other parameters are straightforward).

We consider now the second case in which P_1 is offline, therefore P_2 and P_3 sign.

The parameters involved are:

Player 2		**Player 3**	
Input:	$M, \omega_2, \Gamma_2, p_2, q_2, \mathcal{Y}$ $\mathsf{rec}_{2,3}, \mathsf{rec}_{1,3}$	**Input:**	M, sk_3
Public output:	(r, s)	**Public output:**	(r, s)

The workflow of this case is:

1. Communication:
   (a) P_2 contacts P_3, which comes back online.
   (b) P_2 sends y and $\mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$ to P_3.
2. Paillier keys generation and exchange:
   (a) P_3 generates a Paillier public key (N_3, Γ_3) and its relative secret key (p_3, q_3).
   (b) P_i sends N_i, Γ_i to the other party.
   (c) P_i proves to P_j that it knows p_i, q_i such that $N_i = p_i q_i$ using integer factorization ZKP.
3. P_3's secrets generation:
   (a) P_3 decrypts $\mathsf{rec}_{1,3}$ and $\mathsf{rec}_{2,3}$ with its private key sk_3, getting $\sigma_{1,3}$, $\sigma_{3,1}, \sigma_{2,3}, \sigma_{3,2}$.
   (b) P_3 computes $x_3 = \sigma_{1,3} + 2\sigma_{3,1} - \sigma_{3,2} + \sigma_{2,3}$.
   (c) P_i proves in ZK that it knows x_i using Schnorr's protocol.
4. Signature generation:
   (a) P_2 computes $\tilde{\omega}_2 := \frac{1}{4}\omega_2$.
   (b) P_3 computes $\omega_3 := \frac{3}{2}x_3$.
   (c) P_2 and P_3 perform the Signature Algorithm of Sect. 3.3 as P_A and P_B respectively, where the P_2 uses $\tilde{\omega}_2$ in place of ω_A and P_3 uses ω_3 in place of ω_B (the other parameters are straightforward).

Observation 2. We define $\tilde{\omega}_i$ in this way since we need $\tilde{\omega}_i + \omega_3 = u$, with $i \in \{1, 2\}$. Moreover we define x_3 in this way for the same reasons explained in Observation 1.

3.6. Key Derivation

In many applications it is useful to deterministically derive multiple signing keys from a single *master* key (see e.g. BIP32 for Bitcoin wallets [37]), and this practice becomes even more useful since in our protocol the key-generation is a multi-party computation.

To perform the key derivation we need a derivation index i and the common secret d created during the key-generation protocol.
The derivation is performed as follows:

- P_1 and P_2 perform the key derivation:
  - $\omega_1 \to \omega_1^i = \omega_1 + 3H(d||i)$,
  - $\omega_2 \to \omega_2^i = \omega_2 - 2H(d||i)$,
  - $\mathcal{Y} \to \mathcal{Y}^i = y + H(d||i)\mathcal{B}$.
- P_1 and P_3 perform the key derivation:

  - $\omega_1 \to \omega_1^i = \omega_1 - H(d||i)$,
  - $\omega_3 \to \omega_3^i = \omega_3 + 2H(d||i)$,
  - $\mathcal{Y} \to \mathcal{Y}^i = y + H(d||i)\mathcal{B}$.
- P_2 and P_3 perform the key derivation:
  - $\omega_2 \to \omega_2^i = \omega_2 - \frac{1}{2}H(d||i)$,
  - $\omega_3 \to \omega_3^i = \omega_3 + \frac{3}{2}H(d||i)$,
  - $\mathcal{Y} \to \mathcal{Y}^i = y + H(d||i)\mathcal{B}$.

Observation 3. We observe that the algorithm outputs valid keys, such that, for example:

$$(\omega_1^i + \omega_2^i)\mathcal{B} = y^i.$$

Since $(\omega_1^i + \omega_2^i) = \omega_1 + \omega_2 + H(d||i)$ we have that:

$$(\omega_1^i + \omega_2^i)\mathcal{B} = (\omega_1 + \omega_2 + H(d||i))\mathcal{B} = \mathcal{Y} + H(d||i)\mathcal{B} = \mathcal{Y}^i.$$

With the same procedure we can prove that also the other pairs of derived keys are consistent.

4. Security Proof

As customary for digital signature protocols, we state the security of our scheme as an *unforgeability* property, defined as follows (adapted from the classical definition introduced in [19]):

Definition 4.1. We say that a (t, n)-threshold signature scheme is unforgeable if no malicious adversary who corrupts at most $t - 1$ players can produce with non-negligible probability the signature on a new message m, given the view of **Threshold-Sign** on input messages $m_1, ..., m_k$ (which the adversary adaptively chooses), as well as the signatures on those messages.

Referring to this definition the security of our protocol derives from the following theorem, whose proof is the topic of this section:

Theorem 4.1. *Assuming that*

- *the ECDSA signature scheme is unforgeable,*
- *the strong RSA assumption holds,*
- $(\mathrm{Com}, \mathrm{Ver})$ *is a non malleable commitment scheme,*
- *the DDH assumption holds,*
- *and that encryption algorithm used by* P_3 *is IND-CPA,*[1]

the threshold ECDSA protocol is unforgeable.

The proof will use a classical game-based argument, our goal is to show that if there is an adversary $\mathcal{A}$ that forges the threshold scheme with a non-negligible probability $\varepsilon > \lambda^{-c}$, for a polynomial $\lambda(x)$ and $c > 0$, we can build a forger $\mathcal{F}$ that forges the centralised ECDSA scheme with non-negligible probability as well.

[1] In this proof we focus on the unforgeability property. We discuss other security aspects, such as recovery resiliency, in Sect. 5.

Since the algorithm presented is a $(2,3)$-threshold signature scheme the adversary will control one player and $\mathcal{F}$ will simulate the remaining two. Since the role of P_3 is different we have to consider two distinct cases: one for $\mathcal{A}$ controlling P_3 and one for $\mathcal{A}$ controlling one between P_1 and P_2 (whose roles are symmetrical). The second case is way more interesting and difficult, so it will be discussed first, and for now we suppose without loss of generality that $\mathcal{A}$ controls P_2.

Definition 4.2. (*Security Game*:) The security game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ is defined as follows:

- $\mathcal{C}$ runs the preliminary phase and sets up the parameters, $\mathcal{C}$ controls both P_1 and P_3.
- $\mathcal{C}$ and $\mathcal{A}$ participate in the key-generation algorithm.
- $\mathcal{A}$ chooses adaptively some messages $m_1, ..., m_l$ for some $l > 0$ and asks for a signature on them. $\mathcal{A}$ could either participate in the signature or it can query to $\mathcal{C}$ a signature generated by P_1 and P_3.
- Eventually $\mathcal{A}$ outputs a new message $m \neq m_i \forall i$ and a valid signature for it with probability at least ε.

If we denote with $\tau_{\mathcal{A}}$ the adversary's tape and with τ_i the tape of the honest player P_i we can write:

$$\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}[\mathcal{A}(\tau_{\mathcal{A}})_{P_i(\tau_i)} = \texttt{forgery}] \geq \varepsilon, \tag{6}$$

where $\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}$ means that the probability is taken over the random tape $\tau_{\mathcal{A}}$ of the adversary and the random tape τ_i of the honest player, while $\mathcal{A}(\tau_{\mathcal{A}})_{P_i(\tau_i)}$ is the output of the iteration between the adversary $\mathcal{A}$, running on tape $\tau_{\mathcal{A}}$, and the player P_i, running on tape τ_i . We say that an adversary's random tape $\tau_{\mathcal{A}}$ is good if:

$$\mathbb{P}_{\tau_i}[\mathcal{A}(\tau_{\mathcal{A}})_{P_i(\tau_i)} = \texttt{forgery}] \geq \frac{\varepsilon}{2}. \tag{7}$$

Now we have the following Lemma, introduced in [14]:

Lemma 4.1. *If $\tau_{\mathcal{A}}$ is chosen uniformly at random, then the probability of choosing a good one is at least $\frac{\varepsilon}{2}$.*

Proof. In the proof we will simplify the notation writing $\mathcal{A}(\tau_{\mathcal{A}}, \tau_i) = \texttt{forgery}$ instead of $\mathcal{A}(\tau_{\mathcal{A}})_{P_i(\tau_i)} = \texttt{forgery}$. Moreover we write b to identify a good tape, while c will be a bad one. We can rewrite Eq. (6) in this way:

$$\begin{aligned} A &= \mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}(\tau_{\mathcal{A}} = b, A(\tau_{\mathcal{A}}, \tau_i) = \texttt{forgery}) + \mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}(\tau_{\mathcal{A}} = c, A(\tau_{\mathcal{A}}, \tau_i) = \texttt{forgery}) \\ &= \mathbb{P}_{\tau_{\mathcal{A}},\tau_i}(\tau_{\mathcal{A}} = b)\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}(A(\tau_{\mathcal{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathcal{A}} = b) \\ &\quad + \mathbb{P}_{\tau_{\mathcal{A}},\tau_i}(\tau_{\mathcal{A}} = c)\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}(A(\tau_{\mathcal{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathcal{A}} = c). \end{aligned} \tag{8}$$

Trivially we have that $\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}(A(\tau_{\mathcal{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathcal{A}} = b) < 1$, and from the definition of good tape in Eq. (7) we get:

$$\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}(A(\tau_{\mathcal{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathcal{A}} = c) < \frac{\varepsilon}{2}. \tag{9}$$

Now we want to solve for $x = \mathbb{P}_{\tau_{\mathcal{A}},\tau_i}(\tau_{\mathcal{A}} = b)$, so we get:

$$\varepsilon \leq A < x \cdot 1 + (1 - x) \cdot \frac{\varepsilon}{2} = x\left(1 - \frac{\varepsilon}{2}\right) + \frac{\varepsilon}{2}, \tag{10}$$

that leads us to the conclusion:

$$x \geq \frac{\varepsilon - \frac{\varepsilon}{2}}{1 - \frac{\varepsilon}{2}} \geq \frac{\varepsilon}{2 - \varepsilon} \geq \frac{\varepsilon}{2}. \tag{11}$$

□

From now on we will suppose that the adversary is running on a good random tape.

Now we describe the simulation for the key-generation protocol. The forger $\mathcal{F}$ receives from its challenger the public key $\mathcal{Y}_c = x\mathcal{B}$ for the centralised ECDSA, a public key E_1 for Paillier and a public key pk_3 for the asymmetric encryption scheme.

The simulation proceeds rewinding $\mathcal{A}$ and repeating the following steps until the public key has been correctly generated (i.e. $\mathcal{Y} \neq \perp$), which happens if $\mathcal{A}$ sends a correct decommitment on behalf of P_2 also after the rewind of step 5:

1. $\mathcal{A}$ computes and broadcasts KGC_2 and KGCS_2.
2. $\mathcal{F}$ selects random values $u_1, \sigma_{3,1}$, computes $[\mathsf{KGC}_1, \mathsf{KGD}_1] = \mathrm{Com}(u_1\mathcal{B})$, $[\mathsf{KGCS}_1, \mathsf{KGCS}_1] = \mathrm{Com}(\sigma_{3,1}\mathcal{B})$and sends $\mathsf{KGC}_1, \mathsf{KGCS}_1$.
3. Each player P_i broadcasts its decommitments and the Feldman-VSS values. Let $u_i\mathcal{B}, \sigma_{3,i}\mathcal{B}$ be the values decommitted.
4. Each player sends their Paillier public key E_i.
5. at this point $\mathcal{F}$ computes $\hat{u}\mathcal{B} = \mathcal{Y}_c - u_2\mathcal{B} - 3\sigma_{3,1}\mathcal{B} + 2\sigma_{3,2}\mathcal{B}$ and rewinds $\mathcal{A}$ to its commitment of KGC_1 (step 2). We remark that $\mathcal{F}$ does not know $\hat{u}$ but only $\hat{u}\mathcal{B}$.
6. $\mathcal{F}$ sends $\hat{\mathsf{KGC}}_1$, the commitment corresponding to $\hat{u}\mathcal{B}$. The commitment for $\sigma_{3,1}\mathcal{B}$ remains the same.
7. if $\mathcal{A}$ refuses to decommit then $u_i\mathcal{B}$ is set to $\perp$.
8. $\mathcal{F}$ simulates the VSS (since $\mathcal{F}$ is not able to compute the random polynomial $f(x)$) as explained in Sect. 2.6.
9. $\mathcal{F}$ participates in the ZK, extracting the secret values from $\mathcal{A}$.
10. the remaining steps remain the same. If $u_i = \perp$ for some i the public key y is set to $\perp$. It is important to note that P_1 does not know the value of $\sigma_{1,3}$ and therefore uses a random value to compute $\mathsf{rec}_{1,3}$.

Now we prove that the simulation terminates in expected polynomial time, that it is indistinguishable from the real protocol and that it terminates with output $\mathcal{Y}_c$ except with negligible probability.

Lemma 4.2. *The simulation terminates in expected polynomial time and it is indistinguishable from the real protocol.*

Proof. Since $\mathcal{A}$ is running on a good random tape we know that it will correctly decommit with probability at least $\frac{\varepsilon}{2}$, then we need to rewind only a polynomial number of times, since the expected number of iterations we need to perform is $\frac{2}{\varepsilon} < 2\lambda^c$. The first difference between the real protocol and the simulated one is that $\mathcal{F}$ does not know the discrete logarithm of $\hat{u}\mathcal{B}$ and so it needs to perform a "fake" Feldman-VSS. This is indistinguishable from a real Feldman-VSS since they have both the same distribution, as shown in

Sect. 2.6. The other difference is that $\mathcal{F}$ does not know the value of $\sigma_{1,3}$ and therefore uses a random value to compute $\mathsf{rec}_{1,3}$, but since the encryption algorithm is IND-CPA for the adversary this ciphertext is indistinguishable from the real one. □

Lemma 4.3. *For a polynomially large fraction of possible values for the input $\mathcal{Y}_c$ the simulation terminates with output $\mathcal{Y}_c$ except with negligible probability.*

Proof. First we prove that if the simulation terminates correctly (i.e. with output which is not $\perp$) then it terminates with $\mathcal{Y}_c$ except with negligible probability.

This is a consequence of the non-malleability property of the commitment scheme. Indeed, if $\mathcal{A}$ correctly decommits KGC_2 twice it must do so to the same string, no matter what P_1 decommits to (except with negligible probability). Therefore, due to our choice for $\hat{u}\mathcal{B}$ we have that the output is exactly $\mathcal{Y}_c$.

Now we prove that the simulation ends correctly for a polynomially large fractions of the inputs.

Since $\mathcal{A}$ is running on a good random tape it decommits correctly for at least a fraction $\frac{\varepsilon}{2} > \frac{1}{2\lambda^c}$ of the possible values of $\hat{u}\mathcal{B}$. Moreover, since $\mathcal{Y}_c$ and $\sigma_{3,1}$ are chosen uniformly at random, and y_2 and $\sigma_{3,2}$ are chosen by $\mathcal{A}$ without the knowledge of the other values, we can conclude that $\hat{u}\mathcal{B} = \mathcal{Y}_c - u_2\mathcal{B} - 3\sigma_{3,1} + 2\sigma_{3,2}$ (that is fully determined before the rewind) has also uniform distribution. Then given the 1-to-1 correspondence between $\mathcal{Y}_c$ and $\hat{u}\mathcal{B}$ we can conclude that for a fraction $\frac{\varepsilon}{2} > \frac{1}{2\lambda^c}$ of the inputs the protocol will correctly terminate. □

Observation 4. In the simulation it is crucial that the adversary broadcasts KGC_2 and KGCS_2 before $\mathcal{F}$. Inverting the order will cause this simulation to fail, since after the rewind $\mathcal{A}$ could change its commitment. Due to the non-malleability property we are assured that $\mathcal{A}$ can not deduce anything about the content of these commitments, but nevertheless it could use it as a seed for the random generation of its values. In this case $\mathcal{F}$ guesses the right $\hat{u}\mathcal{B}$ only with probability $\frac{1}{q}$ where q is the size of the group, so the expected time is exponential.

It is possible to swap the order in the first step using an equivocable commitment scheme with a secret trapdoor. In this case we only need to rewind at the decommitment step, we change KCD_1 to match $\hat{u}\mathcal{B}$. In this way we could prove the security of the protocol also in the presence of a *rushing adversary* but we need an additional hypothesis regarding the commitment scheme.

Now we describe the simulation of the protocol for the signature generation. In the same way Gennaro and Goldfeder did in [14], we need to distinguish two different types of executions depending on what happens during the step 3 in the signature generation algorithm:

1. Semi-correct executions: the adversary has followed the protocol and committed-decommitted the correct k_2, then the equality $\mathcal{R} = k^{-1}\mathcal{B}$

holds, where $k = k_1 + k_2$. In this case $\mathcal{F}$ is able to correctly determine the value $s_1\mathcal{R}$ and therefore to correctly terminate the simulation.
2. Non-semi-correct executions: the value decommitted by $\mathcal{A}$ is not the correct k_2, then the value k^{-1} is not the proper one. We will show that a simulation that is not semi-correct will fail with high probability since the value $\mathcal{U}_1$ contributed by P_1 is indistinguishable from a random one. This allows us to simulate the protocol by simply using a random $\tilde{s}_1$ for P_1 instead of the correct one.

We note that it is impossible to distinguish the two cases a priori, so the idea is to guess if an execution will be semi-correct or not. In the semi-correct executions the simulator will be able to extract the "signature shard" of P_1 and to terminate the simulation successfully, in the non-semi-correct ones our simulator will guess a random signature causing the protocol to abort (we will show that in the non-semi-correct execution the real protocol aborts with high probability).

We now present the simulation for a semi-correct execution. We recall that $\mathcal{F}$ does not know the secret values of P_1 (ω_1 and the secret key corresponding to its Paillier public key) but it knows the secret values of P_2. In the following simulation $\mathcal{F}$ aborts whenever the original protocol is supposed to abort.

1. Both players pick randomly k_i, γ_i and broadcast Δ_i.
2. Both players execute the MtA protocol for $k\gamma$. Since P_1 can not decrypt $\alpha_{1,2}$, $\mathcal{F}$ sets it at random.
3. Both players execute the MtAwc protocol for $k\omega$. From the ZKP $\mathcal{F}$ extracts $\nu_{1,2}$. Since $\mathcal{F}$ does not know ω_1 it sends a random $\mu_{2,1}$ to P_2. At this point $\mathcal{F}$ knows σ_2.
4. Both players execute the protocol, revealing δ_i and setting $\delta = \delta_1 + \delta_2$.
5. Both players send D_i.
6. $\mathcal{F}$ queries its signature oracle and receives a signature (r, s) for m. It computes $\mathcal{R} = ms^{-1}\mathcal{B} + rs^{-1}\mathcal{Y}$. We note that, in centralised ECDSA, we have $s\mathcal{R} = k(m + ru)\mathcal{R} = m\mathcal{B} + ru\mathcal{B} = m\mathcal{B} + r\mathcal{Y}$. Then the $\mathcal{R}$ we compute in our simulation is the correct value in the centralised algorithm.
7. $\mathcal{F}$ rewinds $\mathcal{A}$ and changes its commitment to $\hat{\mathcal{G}}_1 = \delta^{-1}\mathcal{R} - \mathcal{G}_2$. In this way we have that $\delta(\hat{\mathcal{G}}_1 + \mathcal{G}_2) = \mathcal{R}$. In the steps after the new commitment P_1 uses the old values of γ_1 and k_1 since it does not know the correct ones.
8. At this point $\mathcal{F}$ knows the value s_2 and then it can compute the right s_1 as $s_1 = s - s_2$.
9. P_1 and P_2 follow the remaining part of the protocol normally.

Observation 5. As in the case of the enrollment phase we need that $\mathcal{A}$ speaks first, to rewind to our commitment phase without changing $\mathcal{A}$'s random tape. The same argument about equivocable commitment schemes and rushing adversaries applies here as well.

Lemma 4.4. *Assuming that*

- *The strong RSA assumption holds,*
- *we use a non malleable commitment scheme,*

then the simulation has the following properties:

- *on input m it outputs a valid signature (r, s) or aborts,*
- *it is computationally indistinguishable from a semi-correct real execution.*

Proof. The differences between the real and the simulated views are the following:

- P_1 does not know the discrete logarithm of $\mathcal{G}_1$. Moreover $\mathcal{G}_1 \neq \gamma_1 \mathcal{B}$.
- in the real protocol $\mathcal{R} = (k_1 + k_2)^{-1}\mathcal{B}$, in this simulation $\mathcal{R}$ is chosen by the signing oracle.

We have that $c_1 = E(\gamma_1)$ is sent during the MtA protocol. In order to distinguish between a real execution and a simulated one an adversary should detect if c_1 is the encryption of a random plaintext or if it is the encryption of $\log_{\mathcal{B}}(\mathcal{G}_1)$. The strong RSA assumption assures the semantic security of Paillier's encryption, so this is infeasible.

In the same way let $e_1 = E(k_1)$. We can write $\mathcal{R} = (\hat{k}_1 + k_2)^{-1}\mathcal{B}$, where we could imagine $\hat{k}_1$ sent by a random oracle (in reality we never calculate $\hat{k}_1$ in the simulation because we compute directly $\mathcal{R}$). Then $(\hat{k}_1 + k_2)\mathcal{R} = \mathcal{B}$, so $\hat{k}_1\mathcal{R} = \mathcal{B} - k_2\mathcal{R}$ and $\hat{k}_1 = \log_{\mathcal{R}}(\mathcal{B} - k_2\mathcal{R})$. With the same argument as before we can conclude that the simulation is indistinguishable from the real execution. It is worth noting that we are simulating a semi-correct execution with a non-semi-correct one, but since they are indistinguishable this is fine.

Now let (r, s) be the signature that $\mathcal{F}$ receives by its oracle in the sixth point of the protocol. Note that the change of the commitment after the rewind does not change the view for $\mathcal{A}$ given the hiding and non-malleabilty properties of the commitment scheme and the considerations above, so as a consequence of the non-malleability property of the commitment scheme the decommitment is consistent and we have that if the protocol terminates it does so with output (r, s). □

Now we show how to simulate the protocol for a non semi-correct execution, i.e. when the value decommitted by $\mathcal{A}$ is not the real k_2.

1. the simulator runs the semi-correct simulation from the first to the sixth point.
2. $\mathcal{F}$ does not rewind $\mathcal{A}$ to fix the value of $\mathcal{R}$. Instead it runs the protocol normally.
3. $\mathcal{F}$ chooses $\tilde{s_1} \in \mathbb{Z}_q$ and $\mathcal{U}_1$ at random and uses these values in the last part of the protocol.

The only difference between the semi-correct simulation and this one is the choice of s_1 and $\mathcal{U}_1$. The reason is that in the semi-correct simulation $\mathcal{F}$ can arbitrarily fix $\mathcal{R}$ because the values decommitted by $\mathcal{A}$ are the real ones, instead in the non-semi-correct simulation this is impossible since the value k_1

does not match anymore. Therefore $\mathcal{F}$ tries to make the protocol fail choosing $\mathcal{U}_1$ and s_1 at random.

We divide the proof in two different steps: first we prove that a real non-semi-correct execution is indistinguishable from a simulation in which P_1 uses the right s_1 but outputs a random $\mathcal{U}_1$ and then we prove that this second intermediate simulation is indistinguishable from the one described above, with both s_1 and $\mathcal{U}_1$ chosen randomly.

Lemma 4.5. *Assuming that*

- *the DDH assumption holds,*
- $\mathrm{Com}, \mathrm{Ver}$ *is a non malleable commitment scheme,*

then the simulation is computationally indistinguishable from a non-semi-correct real execution.

Proof. As anticipated we construct three games between $\mathcal{F}$ and $\mathcal{A}$. In the first one, G_0, the simulator will simply run the real protocol. In the game G_1, $\mathcal{F}$ follows the real protocol but chooses $\mathcal{U}_1$ randomly. Finally, in G_2, $\mathcal{F}$ runs the simulation previously described, with both s_1 and $\mathcal{U}_1$ chosen at random. Now we proceed to prove the indistinguishability of G_0 and G_1 and then of G_1 and G_2.

Let us assume that there is an adversary $\mathcal{A}_0$ that can distinguish between G_0 and G_1. We show that this contradicts the DDH assumption.
Let $\tilde{\mathcal{A}} = a\mathcal{B}, \tilde{\mathcal{B}} = b\mathcal{B}, \tilde{\mathcal{C}} = c\mathcal{B}$ be the DDH challenge where $c = ab$ or c is random in $\mathbb{Z}_q$. The distinguisher $\mathcal{F}_0$ runs $\mathcal{A}_0$, simulating the key-generation phase so that $\mathcal{Y} = b\mathcal{B}$. It can do that by rewinding the adversary and changing its decommitment to $u_1\mathcal{B} = \mathcal{Y} - u_2\mathcal{B} - \sigma_{3,1}\mathcal{B} + 2\sigma_{3,2}\mathcal{B}$, making $\mathcal{Y} = \tilde{\mathcal{B}}$. Thanks to the ZKP $\mathcal{F}_0$ extracts the values of x_2 (and then of ω_2) from the adversary, but does not know b (and therefore not x_1 nor $\omega_1 = b - \omega_2$). In this simulation we can also suppose that $\mathcal{F}_0$ knows the secret key associated to E_1, its public key of the Paillier cryptosystem (we can do this since we are not making any reduction to the security of the encryption scheme).

At this point $\mathcal{F}_0$ runs the signature generation protocol for a non-semi-correct execution. It runs the protocol normally till the MtA and MtAwc part of the signature protocol. It knows γ_1, k_1, since it runs P_1 normally, and γ_2, since it extracts its value from the adversary. Therefore $\mathcal{F}_0$ knows k such that $\mathcal{R} = k^{-1}\mathcal{B}$. Since we suppose that $\mathcal{F}_0$ knows the secret key associated to E_1 it can also know $\mu_{1,2}$ obtained from the MtAwc protocol on input ω_2 and k_1. Since it does not know ω_1, during the MtAwc protocol on input ω_1 and k_2 it sends a random $\mu_{2,1}$ and sets:

$$\nu_{2,1} = k_2\omega_1 - \mu_{2,1}. \tag{12}$$

So at the end of the MtAwc we have that:

$$\sigma_1 = k_1\omega_1 + \mu_{1,2} + \nu_{2,1}; \tag{13}$$

using Eq. (12) in Eq. (13) we get

$$\sigma_1 = k_1\omega_1 + k_2\omega_1 - \mu_{2,1} + \mu_{1,2} = \tilde{k}\omega_1 + \mu_{1,2} - \mu_{2,1},$$

where $\tilde{k} = k_1 + k_2$ and we have that $\tilde{k} \neq k$ since we are in a non-semi-correct execution. Remembering that $\omega_1 = b - \omega_2$ we can substitute again, obtaining

$$\sigma_1 = \tilde{k}b - \tilde{k}\omega_2 + \mu_{1,2} - \mu_{2,1}, \tag{14}$$

and $\mathcal{F}_0$ knows every value in the equation except b, so let us group these known values and set $\mu_1 = \tilde{k}\omega_2 + \mu_{1,2} - \mu_{2,1}$. Thus $\mathcal{F}_0$ can successfully compute:

$$\sigma_1\mathcal{B} = \tilde{k}b\mathcal{B} + \mu_1\mathcal{B} = \tilde{k}\tilde{\mathcal{B}} + \mu_1\mathcal{B}, \tag{15}$$

and therefore also:

$$s_1\mathcal{R} = (k_1 m + r\sigma_1)k^{-1}\mathcal{B} = (k_1 m + r\mu_1)k^{-1}\mathcal{B} + \tilde{k}rk^{-1}\tilde{\mathcal{B}}. \tag{16}$$

We now proceed to the last part of the simulation. $\mathcal{F}_0$ selects a random l_1 and sets $\mathcal{W}_1 = s_1\mathcal{R}_1 + l_1\mathcal{B}$. Instead of following the algorithm $\mathcal{F}_0$ does not choose a random ρ_1 but sets implicitly $\rho_1 = a$ and sends $\mathcal{A}_1 = a\mathcal{B} = \tilde{\mathcal{A}}$. During the ZKP that it simulates (since it does not know a nor s_1) it extracts s_2, l_2, ρ_2 from the adversary. Let us define $s = k^{-1}s_2$. We note that:

$$\begin{aligned}
\mathcal{W} &= -m\mathcal{B} - r\mathcal{Y} + \mathcal{W}_1 + \mathcal{W}_2 && (17)\\
&= (l_1 + l_2)\mathcal{B} + s_1\mathcal{R} + (s - m)\mathcal{B} - r\mathcal{Y} && (18)\\
&= l\mathcal{B} + t_1\mathcal{B} + t_2\tilde{\mathcal{B}}, && (19)
\end{aligned}$$

where $t_1 = k^{-1}(k_1 m + r\mu_1) + s - m$ and $t_2 = k^{-1}\tilde{k}r - r$. We note that in a not-semi-correct execution $\tilde{k} \neq k$ and then $t_2 \neq 0$. Finally $\mathcal{F}_0$ computes $\mathcal{T}_1 = l_1\mathcal{A}$ correctly but for $\mathcal{U}_1$ it outputs $\mathcal{U}_1 = (l + t_1)\tilde{\mathcal{A}} + t_2\tilde{\mathcal{C}}$ and aborts. More explicitly we have that:

$$\begin{aligned}
\mathcal{U}_1 &= (l + t_1)\tilde{\mathcal{A}} + t_2\tilde{\mathcal{C}} && (20)\\
&= (l + k^{-1}(k_1 m + r\mu_1) + s - m)\tilde{\mathcal{A}} + (k^{-1}\tilde{k}r - r)\tilde{\mathcal{C}} && (21)\\
&= \left(l_1 + l_2 + k^{-1}(k_1 m + r\mu_1) + s - m\right) a\mathcal{B} + (k^{-1}\tilde{k}r - r)c\mathcal{B} && (22)
\end{aligned}$$

If we have $c = ab$ this equation can be further simplified to:

$$\begin{aligned}
\mathcal{U}_1 &= a(l_1 + l_2)\mathcal{B} + ak^{-1}(k_1 m + r\mu_1 + \tilde{k}rb) + a(s - m)\mathcal{B} - abr\mathcal{B} && (23)\\
&= a(l_1 + l_2)\mathcal{B} + ak^{-1}s_1\mathcal{B} + ak^{-1}s_2\mathcal{B} - am\mathcal{B} - abr\mathcal{B} && (24)\\
&= a(l_1 + l_2)\mathcal{B} + ak^{-1}(s_1 + s_2)\mathcal{B} - a(br + m)\mathcal{B}, && (25)
\end{aligned}$$

and we note also that:

$$\begin{aligned}
a\mathcal{W} &= a\mathcal{W}_1 + a\mathcal{W}_2 - am\mathcal{B} - ar\mathcal{Y} && (26)\\
&= as_1\mathcal{R} + al_1\mathcal{B} + as_2\mathcal{R} + al_2\mathcal{B} - am\mathcal{B} - arb\mathcal{B} && (27)\\
&= a(l_1 + l_2)\mathcal{B} + ak^{-1}(s_1 + s_2)\mathcal{B} - a(br + m)\mathcal{B}. && (28)
\end{aligned}$$

Then we can conclude that if $c = ab$ we have $\mathcal{U}_1 = aV = \rho_1\mathcal{W}$, as in G_0, otherwise $\mathcal{U}_1$ is a random group element as in G_1. Then if a distinguisher for G_0 and G_1 exists it can be also used to win a DDH challenge as we described above, so G_0 and G_1 are indistinguishable.

Now we deal with the indistinguishability of G_1 and G_2. We recall that the difference between the protocols G_1 and G_2 is that in G_2 we use a random $\tilde{s}_1$ during the last part and then we have a random $\tilde{\mathcal{W}}_1 = \tilde{s}_1\mathcal{R} + l_1\mathcal{B}$. Once again we will prove that G_1 is indistiguishable from G_2 performing a reduction

to the DDH assumption. The idea is to show that $\mathcal{W}_1 = s_1\mathcal{R} + l_1\mathcal{B}$ and $\tilde{\mathcal{W}}_1 = \tilde{s}_1\mathcal{R} + l_1\mathcal{B}$ are indistinguishable due to the random value $l_1\mathcal{B}$ added to both of them.

Let $\tilde{\mathcal{A}} = (a-d)\mathcal{B}$, $\tilde{\mathcal{B}} = b\mathcal{B}$, $\tilde{\mathcal{C}} = ab\mathcal{B}$ be the DDH challenge where $d = 0$ or d is a random value of $\mathbb{Z}_q$. The simulator proceeds with the regular protocol until step 4c Then $\mathcal{F}_0$ broadcasts $\mathcal{W}_1 = s_1\mathcal{R} + \tilde{\mathcal{A}}$ and $\mathcal{A}_1 = \tilde{\mathcal{B}}$. It simulates the ZKP (since it does not know l_1 and ρ_1) and extracts s_2, l_2, ρ_2. Then it computes $\mathcal{U}_1$ as a random element and $\mathcal{T}_1 = \tilde{\mathcal{C}} + \rho_2\tilde{\mathcal{A}} = ab\mathcal{B} + \rho_2(a-d)\mathcal{B}$.

When $d = 0$ we have $\tilde{\mathcal{A}} = a\mathcal{B}$, so $a = l_1$, $b = \rho_1$ and we have that $\mathcal{W}_1 = s_1R + l_1\mathcal{B}$ and:

$$\mathcal{T}_1 = ab\mathcal{B} + \rho_2 a\mathcal{B} = l_1\rho_1\mathcal{B} + l_1\rho_2\mathcal{B} = l_1(\rho_1 + \rho_2)\mathcal{B}, \tag{29}$$

so $\mathcal{T}_1 = l_1\mathcal{A}$ as in G_1. Otherwise, when $d \neq 0$ we have that $\tilde{\mathcal{A}} = a\mathcal{B} - d\mathcal{B}$ with a randomly distributed d, then this is equivalent to have:

$$\mathcal{W}_1 = s_1\mathcal{R} + (a-d)\mathcal{B} = \tilde{s}_1\mathcal{R} + a\mathcal{B}, \tag{30}$$

with $\tilde{s}_1 = s_1 - dk^{-1}$, that is uniformly distributed thanks to d, and $\mathcal{T}_1 = l_1\mathcal{A}$ as in G_2. The key idea is that we use the random value d to change the fixed value s_1 to a random and unknown $\tilde{s}_1$ during the computation of $\mathcal{W}_1$. Therefore, under the DDH assumption, G_1 and G_2 are indistinguishable. Then G_0 is indistinguishable from G_2 as we wanted to prove. □

Now we have to deal with the recovery signature algorithm. Since the core algorithm remains the same we can use the two proofs already explained, we only need to change the setup phase in which the third player recovers its secret material. In this section we still examine the case in which $\mathcal{A}$ controls one between P_1 or P_2 and $\mathcal{F}$ controls P_3. Things are a little bit different if $\mathcal{A}$ controls P_3 since it does not perform the enrollment phase (this case is much easier).

Trivially if $\mathcal{A}$ asks for a recovery signature between the two honest parties $\mathcal{F}$ can simply query its oracle and output whatever the oracle outputs. So we can limit ourselves to deal with the case where the adversary participates in the signing process.

Without loss of generality we suppose that $\mathcal{A}$ controls P_2. The simulation works as follows:

1. P_2 sends $\mathcal{Y}, \mathsf{rec}_{1,2}, \mathsf{rec}_{1,3}$ to P_3.
2. P_i generates a Paillier public key (N_i, Γ_i) and sends it to the other party.
3. P_i proves in ZK that it knows the matching secret for its public Paillier key.
4. P_3 can not decrypt the values received in step 1, so it simulates the ZKP about x_3 and at the same time it can extract the secret value x_2 from P_2. Note that the inability to decrypt is not a problem since most of the data is useless (the random values sent by $\mathcal{F}$ during the enrolment phase), so it would not have been able to compute x_3 nor ω_3 anyway. However the simulator remembers the correct values of $\mathsf{rec}_{1,2}, \mathsf{rec}_{1,3}$, so if $\mathcal{A}$ does not send the proper ones it can abort the simulation, as it would happen in a real execution since P_3 can not recover a secret key shard that matches $\mathcal{Y}$.

5. P_2 computes its $\tilde{x}_2$ and $\tilde{\omega}_2$ from its original shards.
6. P_2 and P_3 perform the signing algorithm with the above simulation. Also in this case $\mathcal{F}$ does not know its secret key, but we remark that this is fine since it can use the signing oracle.

In the case of P_3 being dishonest the simulation is much more easier. During the enrolment phase $\mathcal{F}$ can produce random shards to send to P_3 during the recovery signature phase and output directly its original ECDSA challenge. Then with the same algorithm as before it can perform the signing protocol.

Now we are ready to prove Theorem 4.1.

Proof. Let $Q < \lambda^c$ be the maximum number of signature queries that the adversary makes. In the real protocol the adversary will output a forgery after $l < Q$ queries, either because it stops submitting queries or because the protocol aborts. In our simulation we try to guess if a simulation will be semi-correct or not choosing a random $i \in [0, Q]$. We have two cases:

- if $i = 0$ we assume that all the executions are semi-correct and then we always use the semi-correct algorithm described previously.
- if $i \neq 0$ we assume that the first $i - 1$ are semi-correct, but the ith is not. In this case we use the semi-correct algorithm for every execution except for the ith one, for which we use the non-semi-correct one, then we abort.

As we previously proved we produce an indistinguishable view for the adversary, then $\mathcal{A}$ will produce a forgery with the same probability as in a real execution. Then the probability of success of our forger $\mathcal{F}$ is $\frac{\varepsilon^3}{8Q}$ and it is the product of:

- the probability of choosing a good random tape for $\mathcal{A}$, that is at least $\frac{\varepsilon}{2}$, a shown in Lemma 4.1,
- the probability of hitting a good public key, that also is at least $\frac{\varepsilon}{2}$, as shown in Lemmas 4.2 and 4.3,
- the probability of guessing the right index i, that is $\frac{1}{Q}$,
- the probability of $\mathcal{A}$ to successfully produce a forgery on a good random tape, that is $\frac{\varepsilon}{2}$ as shown in Eq. 7.

Under the security of the ECDSA signature scheme the probability of producing a forgery must be negligible, which implies that ε must negligible too, contradicting the hypothesis that $\mathcal{A}$ has non-negligible probability of forging the scheme. □

5. Conclusions

Although decentralized signature algorithms have been known for a while, we are aware of only few proposals for algorithms that are able to produce signatures indistinguishable from a standard one. Moreover, the protocol described in this work is, as far as we know, the first example of a threshold multi-signature allowing the presence of an off-line participant. Regarding the protocol's specification, the main difference w.r.t. [14] lies in the key-generation phase. Specifically, the idea is to have two active participants to simulate the

action of the third one. This step is possible due to the uniqueness property of polynomial interpolation that gives a bijection between points and coefficients, that combined to the preserved uniform distribution in $\mathbb{Z}_p$ allows us to "invert" the generation of the shares, that are later recovered by the offline party thanks to an asymmetric encryption scheme. A second divergence is that we have managed to avoid the use of equivocable commitments under the assumption that in some specific steps (see Observation 4) we can consider the adversary to not be rushing.

The main efficiency bottleneck is in the massive usage of ZKPs, which are necessary to guarantee the security of the signature itself against black-box adversaries, as hinted by the security proof of Sect. 4. Nevertheless, there are implementations that use our protocol to resiliently manage bitcoin wallets [9].

In our security analysis we focused on the unforgeability of the signature, however with an offline party (and more so in the application context of crypto-assets management) there is another security aspect worthy of consideration: the resiliency of recovery in the presence of a malicious adversary. Of course if the offline party is malicious and unwilling to cooperate in fund recovery there is nothing we can do about it, however the security can be strengthened if we consider that one of the online parties may corrupt the recovery material. In this case a generic CPA asymmetric encryption scheme is not sufficient to prevent malicious behaviour, because we need a verifiable scheme that allows the parties to prove that the recovery material is consistent, just like they prove that they computed the shards correctly. In this way the adversary is unable to corrupt the recovery material and then there is always a pair of players that is able to sign. Indeed without these protection measures a malicious user could convince an unsuspecting victim to set up a $(2, 3)$-threshold wallet together, and sending bogus recovery data the attacker can later on blackmail the victim to sign transactions of their choice otherwise it will refuse to collaborate in future signatures, effectively freezing the funds since the recovery party has been neutralised.

An interesting topic of further analysis surely regards provable public-key encryption schemes, for example we see potential solutions that could exploit the homomorphic properties of Paillier or ElGamal [12] cryptosystems.

Other future research steps involve the generalisation to (t, n)-threshold schemes with more than one offline party, as well as to different standard signatures. Regarding the latter, there is a variant of our protocol whose signatures are indistinguishable from EdDSA [2,25], and we are working on its security proof.

Acknowledgements

The core of this work is contained in the first author's MSC thesis that would like to thank his two supervisors, the second and fourth author, and Telsy S.p.A. for their support during the work. Part of the results presented here have been carried on within the EU-ESF activities, call PON Ricerca

e Innovazione 2014-2020, project Distributed Ledgers for Secure Open Communities. The second and third authors are members of the INdAM Research group GNSAGA. We would like to thank Conio s.r.l. and its co-CEO Vincenzo di Nicola for their support. We also thank Gaetano Russo, Federico Mazzone, and Zsolt Levente Kucsván that worked on the implementation and provided valuable feedback. The authors would like to thank the anonymous referees.

Appendix A. Zero Knowledge Proofs

A.1 Schnorr Protocol

The Schnorr Protocol is a zero-knowledge proof for the discrete logarithm. Let $\mathbb{G}$ be a group of prime order p with generator g. Let $h \in \mathbb{G}$ be a random element in $\mathbb{G}$. The prover $\mathcal{P}$ wants to prove to a verifier $\mathcal{V}$ that it knows the discrete logarithm of h, i.e. it knows $x \in \mathbb{Z}_p$ such that $g^x = h$.
So the common inputs are $\mathbb{G}, g$ and h, while the secret input of $\mathcal{P}$ is x.

The protocol works as follows:

1. $\mathcal{P}$ picks r uniformly at random in $\mathbb{Z}_p$ and computes $u = g^r$. It sends u to $\mathcal{V}$.
2. $\mathcal{V}$ picks c uniformly at random $\in \mathbb{Z}_p$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $z = r + cx$ and sends z to $\mathcal{V}$.
4. $\mathcal{V}$ computes g^z. If $\mathcal{P}$ really knows x it holds that $g^z = uh^c$. If the equality does not hold, the verifier rejects.

A detailed proof about the security of the algorithm can be found in [34].

A.1.1 Schnorr Protocol Simulation. We need to simulate the Schnorr protocol in two different ways: first we need to use it to extract the adversary's secret value, then we need to simulate it without knowing our secret value, tricking the opponent. We can use the Schnorr protocol to extract the value x from the adversary in this way:

1. Follow the standard protocol until the third point, obtaining z.
2. Rewind the adversary to the second point and pick $c' \neq c$.
3. Follow the remaining part of the protocol, obtaining z'.
4. We can compute $\frac{z-z'}{c-c'} = \frac{(c-c')x}{c-c'} = x$.

Proof. (Sketch) Since the only extra hypothesis for c' is that $c' \neq c$ we can suppose that c' has uniform distribution as well. Moreover z, once the verifier sent c the value of z is fixed, so the rewinding technique does not cause any problem. □

At the same time we need to be able to simulate the protocol without knowing x. The simulation works as follows:

1. Follow the protocol until the second point, obtaining c.
2. Rewind the adversary to the first point. The simulator picks r randomly and computes $u' = g^{-xc+r} = (g^x)^{-c}g^r$. Under the discrete logarithm

assumption and since r, c are random element, this is indistinguishable from g^r.
3. The simulator sends u' and the adversary sends c again.
4. The simulator sends $z = r - cx + cx = r$.
5. The adversary checks that $g^z = g^r = u'(g^x)^c = g^{-xc}g^r g^{xc}$.

Proof. (Sketch) The tricky point of the simulation is the third point, when we need that the adversary sends the same c it has previously sent, since sending a different r could change the random choice of c . This could be achieved introducing an equivocable commitment scheme, in this way we need only to change the decommitment value after receiving the adversary commitment. □

A.2 Integer Factorization Proof

We now present a well-known ZK proof for the integer factorization problem. Let k be the security parameter and $N = pq$ with neither p nor q small. Let A, B, l be such that[2] $l \log B = \theta(k)$, $(N - \phi(N))lB < A < N$. Let $z_1, \ldots, z_k \in \mathbb{Z}_N^*$ be chosen uniformly at random. The protocol consists in the repetition l times of the following sub-protocol:

1. The prover $\mathcal{P}$ picks $r \in \{0, ..., A-1\}$ at random and computes $x_i = z_i^r \mod N$ for all $i \in \{1, \ldots, k\}$.
2. $\mathcal{P}$ sends every x_i to the verifier $\mathcal{V}$.
3. $\mathcal{V}$ picks a random integer $e \in \{0, ..., B-1\}$ and sends it to $\mathcal{P}$.
4. $\mathcal{P}$ computes $y = r + e(N - \phi(N))$ and sends it to $\mathcal{V}$.
5. $\mathcal{V}$ checks that $0 \leq y < A$ and that $z_i^{y-Ne} = x_i \mod N$ for all i. If it does not hold the protocol fails.

A detailed explanation of the protocol and the security proof, as well as proofs about the soundness and the completeness of it, can be found in [31].

A.2.1 Integer Factorization Proof. We need to simulate the protocol without knowing the factorization of n. A detailed description and proof of a simulation can be found in [31]. Alternatively, at the price of an equivocable commitment, we could simplify the simulation as follows:

1. Follow the point of the protocol choosing a random e_r and y_r and compute $x_i = z_i^{y-ne}$.
2. Receive e from the opponent. If $e = e_r$ then the simulation could end, otherwise rewind the opponent and change the pair e_r, y_r. Repeat this step until $e = e_r$
3. The opponent sends e again, the simulator sends y'.

Proof. (Sketch) Clearly $z^{y'-ne} = x_i \mod n$ and $y' < A$ holds by construction.

We observe that a good pair e_r, y_r is obtained with probability $\frac{1}{B}$ and so the complexity of all the simulation is lB. □

[2] ϕ is the Euler's totient function that counts the positive integers up to a given integer n that are relatively prime to n, while θ is the Bachmann–Landau symbol that means that the argument is bounded both above and below.

A.3 Range Proof

We need two protocols to ensure that the share conversion protocol explained in Sect. 2.5 will run correctly: one started by the initiator and the other started by the receiver.

For the first one the common inputs are a Paillier public key (Γ, N), the ciphertext $c \in \mathbb{Z}_{N^2}$, an RSA modulus M product of two safe primes, and $h_1, h_2 \in \mathbb{Z}_M^*$. The prover knows $m \in \mathbb{Z}_q$ and $r \in \mathbb{Z}_N^*$ such that $c = \Gamma^m r^N \mod N^2$ where q is the order of the group used during the share conversion protocol previously described. At the end of the protocol the verifier is convinced that $m \in [-q^3, q^3]$.

The protocol works as follows:

1. $\mathcal{P}$ picks randomly $\alpha \in \mathbb{Z}_{q^3}$, $\beta \in \mathbb{Z}_N^*$, $\gamma \in \mathbb{Z}_{q^3 M}$, $\rho \in \mathbb{Z}_{qM}$.
2. $\mathcal{P}$ computes $z = h_1^m h_2^\rho \mod M$, $u = \Gamma^\alpha \beta^N \mod N^2$, and $\omega = h_1^\alpha h_2^\gamma \mod M$.
3. $\mathcal{P}$ sends z, u and ω to $\mathcal{V}$.
4. $\mathcal{V}$ picks e at random and sends it to $\mathcal{P}$.
5. $\mathcal{P}$ computes $s = r^e \beta \mod N$, $s_1 = em + \alpha$ and $s_2 = e\rho + \gamma$.
6. $\mathcal{P}$ sends s, s_1 and s_2 to $\mathcal{V}$.
7. $\mathcal{V}$ checks if $s_1 \leq q^3, u = \Gamma_1^s s^N c^{-e} \mod N^2$ and $h_1^{s_1} h_2^{s_2} = z^e \omega \mod M$.

In the second protocol the prover wants to show that $|b| \leq q^3$ and that it knows b, β' such that $g^b = B$ and $\mathsf{c}_B = (b \times_E \mathsf{c}_A) +_E E_A(y)$ (this second part only during the Share Conversion Protocol with check). The common inputs are B, the Pallier public key (Γ, N), and the ciphertexts c_1, c_2 (that are the Paillier ciphertexts $\mathsf{c}_A, \mathsf{c}_B$). $\mathcal{P}$ also knows $b \in \mathbb{Z}_q, y \in \mathbb{Z}_N$ and $c_2 = c_1^b \Gamma^y r^N \mod N^2$. The protocol works as follows:

1. $\mathcal{P}$ picks $\alpha \in \mathbb{Z}_{q^3}$, $\rho, \sigma, \tau \in \mathbb{Z}_{qM}$, $\rho' \in \mathbb{Z}_{q^3 M}$, $\beta, \gamma \in \mathbb{Z}_N^*$ uniformly at random.
2. $\mathcal{P}$ computes:
   - $z = h_1^b h_2^\rho \mod M$,
   - $z' = h_1^\alpha h_2^{\rho'} \mod M$,
   - $t = h_1^y h_2^\sigma \mod M$,
   - $u = g^\alpha$,
   - $v = c_1^\alpha \Gamma^\gamma \beta^N \mod N^2$,
   - $\omega = h_1^\gamma h_2^\tau \mod M$.
3. $\mathcal{P}$ sends z, z', t, v, ω, u to $\mathcal{V}$.
4. $\mathcal{V}$ picks $e \in \mathbb{Z}_q$ uniformly at random and sends it to $\mathcal{P}$.
5. $\mathcal{P}$ computes:
   - $s = r^e \beta \mod N$,
   - $s_1 = eb + \alpha$,
   - $s_2 = e\rho + \rho'$,
   - $t_1 = ey + \gamma$,
   - $t_2 = e\sigma + \tau$.
6. $\mathcal{P}$ sends s, s_1, s_2, t_1 and t_2 to $\mathcal{V}$.
7. $\mathcal{V}$ checks if:
   - $s_1 \leq q^3$,

- $g^{s_1} = B^e u$,
- $h_1^{s_1} h_2^{s_2} = z^e z' \mod M$,
- $h_1^{t_1} h_2^{t_2} = \omega t^e \mod M$,
- $c_1^{s_1} s^N \Gamma^{t_1} = c_2^e v \mod N^2$.

For a security proof of this protocol see [14].

References

[1] Bellare, M., Rogaway, P.: Introduction to modern cryptography. (2005). https://web.cs.ucdavis.edu/rogaway/~classes/227/spring05/boc

[2] Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High speed high-security signatures. J. Cryptogr. Eng. **2**(2), 77–89 (2012)

[3] Boneh, D.: The decision Diffie–Hellman problem. In: International Algorithmic Number Theory Symposium. Springer, pp. 48–63 (1998)

[4] Boneh, D., Gennaro, R., Goldfeder, S.: Using level-1 homomorphic encryption to improve threshold DSA signatures for Bitcoin wallet security. In: International Conference on Cryptology and Information Security in Latin America. Springer, pp. 352–357 (2017)

[5] Brandao, L.T.A.N., Davidson, M., Vassilev, A.: NIST roadmap toward criteria for threshold schemes for cryptographic primitives. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8214A.pdf. Accessed: 27 Aug 2020

[6] Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)

[7] Canetti, R., Makriyannis, N., Peled, U.: UC non-interactive, proactive, threshold ECDSA. IACR Cryptol. ePrint Arch. **2020**, 492 (2020)

[8] Chohan, U.W.: The problems of cryptocurrency thefts and exchange shutdowns. In: Available at SSRN 3131702 (2018)

[9] Di Nicola, V.: Custody at Conio-part 3. https://medium.com/conio/custody-at-conio-part-3-623292bc9222 (2020)

[10] Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: IEEE Symposium on Security and Privacy (SP), vol. 2018. IEEE, pp. 980–997 (2018)

[11] Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ECDSA from ECDSA assumptions: the multiparty case. In: IEEE Symposium on Security and Privacy (SP), vol. 2019. IEEE, pp. 1051–1066 (2019)

[12] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory **31**(4), 469–472 (1985)

[13] Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (SFCS 1987), pp. 427–438 (1987)

[14] Gennaro, R., Goldfeder, S.: Fast multiparty threshold ECDSA with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, p. 11791194 (2018)

[15] Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In: International Conference on Applied Cryptography and Network Security. Springer, pp. 156–174 (2016)

[16] Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold DSS signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp 354–371 (1996)

[17] Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp 295–310 (1999)

[18] Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In: Proceedings of the 27th Annual Symposium on Foundations of Com-puter Science, pp 174–187 (1986)

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

[20] Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Sec. **1**, 36–63 (2001). https://doi.org/10.1007/s102070100002

[21] Kondi, Y., Magri, B., Orlandi, C., Shlomovits, O.: Refresh when you wake up: proactive threshold wallets with offline devices. IACR Cryptol. ePrint Arch. **2019**, 1328 (2019)

[22] Kravitz, D.W.: Digital signature algorithm. US Patent 5,231,668 (1993)

[23] Lindell, Y.: Fast secure two-party ECDSA signing. In: Annual International Cryptology Conference. Springer, pp 613–644 (2017)

[24] Lindell, Y., Nof, A.: 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. ACM, pp. 1837–1854 (2018)

[25] Longo, R., Meneghetti, A., Sala, M.: Threshold multi-signature with an offline recovery party. https://eprint.iacr.org/2020/023 (2020)

[26] MacKenzie, P., Reiter, M.K.: Two-party generation of DSA signatures. In: Annual International Cryptology Conference. Springer, pp. 137–154 (2001)

[27] MacKenzie, P., Reiter, M.K.: Two-party generation of DSA signatures. Int. J. Inf. Secur. **2**(3–4), 218–239 (2004)

[28] Marcedone, A., Orlandi, C.: Obfuscation $\Rightarrow$ (IND-CPA security $\not\Rightarrow$ circular security). In: International Conference on Security and Cryptography for Networks. Springer, pp. 77–90 (2014)

[29] Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. Tech. rep, Manubot (2019)

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

[31] Poupard, G., Stern, J.: Short proofs of knowledge for factoring. In: International Workshop on Public Key Cryptography. Springer, pp. 147–166 (2000)

[32] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. In: Communications of the ACM 21.2, pp. 120–126 (1978)

[33] Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: International Workshop on Fast Software Encryption. Springer, pp. 371–388 (2004)

[34] Schnorr, C.-P.: Efficient identification and signatures for smart cards. In: Conference on the Theory and Application of Cryptology. Springer, pp. 239–252 (1989)

[35] Schoenmakers, B.: A simple publicly verifiable secret sharing scheme and its application to electronic voting. In: Annual International Cryptology Conference. Springer, pp. 148–164 (1999)

[36] Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176. (**ISSN: 0001-0782**)

[37] Wuille, P.: Hierarchical deterministic wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (2012)

Michele Battagliola, Riccardo Longo, Alessio Meneghetti and Massimiliano Sala
Department of Mathematics
University of Trento
Via Sommarive, 14, Povo
38123 Trento
Italy
e-mail: `alessio.meneghetti@unitn.it`

Michele Battagliola
e-mail: `michele.battagliola@studenti.unitn.it`

Riccardo Longo
e-mail: `riccardolongomath@gmail.com`

Massimiliano Sala
e-mail: `maxsalacodes@gmail.com`

Received: August 25, 2020.
Revised: January 22, 2021.
Accepted: October 4, 2021.