Mediterr. J. Math. (2023) 20:253
https://doi.org/10.1007/s00009-023-02452-9
1660-5446/23/050001-30
published online June 22, 2023

Mediterranean Journal of Mathematics

Provably Unforgeable Threshold EdDSA with an Offline Participant and Trustless Setup

Michele Battagliola, Riccardo Longo, Alessio Meneghetti and Massimiliano Sala

Abstract. We present an EdDSA-compatible multi-party digital signature scheme that supports an offline participant during the key-generation phase, without relying on a trusted third party. Under standard assumptions, we prove our scheme secure against adaptive malicious adversaries. Using a classical game-based argument, we prove that if there is an adversary capable of forging the scheme with non-negligible probability, then we can build a forger for the original EdDSA scheme with non-negligible probability. The scheme requires only two communication rounds in the signature generation phase and avoids expensive multi-party evaluation of cryptographic hash functions. We present our solution in a setting where two parties generate the keys and compute signatures, with a third party which can be brought online after the key generation when one of the other parties becomes unavailable. This setting is a perfect fit for custodial solutions where partially trusted services are employed by a user to increase resiliency. We provide also a possible solution to retain the resiliency of the recovery in the presence of a malicious party.

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 is a multi-party computation protocol that enables a subset of at least t among n authorized players to jointly perform digital signatures. The flexibility and security advantages of threshold

protocols have become of central importance in the research for new cryptographic primitives [1] with particular attention to threshold versions of EdDSA and Schnorr Signature Schemes [2].

Starting from the highly influential work of Gennaro et al. [3], several authors proposed both novel schemes [4–6] and improvements to existing protocols [7–14].

Usually, threshold schemes translate in the multi-party setting a well-established signature scheme, while producing signatures that are compatible with the original version. Then their security is proved with a reduction to the original scheme, like the proof presented in Refs. [15,16]. The key-generation and signature algorithms are replaced by a communication protocol between the parties, while the compatibility is achieved by keeping the verification algorithm of the original algorithm. This approach streamlines the insertion of the new protocol in the cryptographic landscape, because verification is compatible with established solutions and the resulting security derives from standard assumptions.

Recently, in Ref. [15], the authors propose an ECDSA-compatible $(2, 3)$-threshold multi-signature protocol in which one of the users plays the role of recovery party: a user involved only once in a preliminary setup prior even to the key-generation step of the protocol. More precisely, only two parties are active in the key-generation step of the protocol, and by a secure multi-party protocol, they create their own private keys together with some additional data to be eventually sent to the third non-active player. In case of need, the third player receives and uses this additional data to generate its private key and can, therefore, eventually participate in the signature phase with one of the other two. In this paper, we propose an EdDSA-compatible variant of Ref. [15] where again the key-generation algorithm of the protocol does not require the active involvement of all three players.

The $(2, 3)$-threshold setting is limited, but is the first step towards a generalized (t, n)-threshold and a good fit for custodial services. These kind of services offer to a user the possibility to improve the resiliency of signature generation. This is particularly important in the world of cryptocurrencies, where digital signatures determine ownership rights and control over assets. This means that resiliency against key loss is essential to retain ownership of the assets, since there is no central authority that can restore ownership of a digital token once the private key of the wallet is lost.

To avoid relying on a trusted third-party custodian that takes full responsibility and control of keys (which exposes them to criminal takeovers, with plenty of incidents already happened [17]), or multi-sig wallets (whose availability is very blockchain-dependent, moreover, such wallets are very easily identifiable), a threshold signature compatible with the original digital signature scheme is a perfect solution. In this approach, the key is generated in collaboration with a partially trusted custodial service and a partially trusted recovery service. The custodial service collaborates with the user to compute the signatures, and if the user loses their private-key material, the custodial and recovery services can collaborate to restore ownership.

However, in real-life applications, the recovery service 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. For this very reason, our setting, where a party can stay offline during the key generation and only participate in the recovery, is a perfect fit. More details on how $(2, 3)$-threshold multi-signatures with offline parties are applied in custody services for crypto-assets can be found in Ref. [18].

EdDSA offers better performance than ECDSA, but the latter is at first glance better suited for a multi-party environment: the presence of hash computations in EdDSA is indeed not readily-compatible with an MPC setting. In order to work around the problem, we work with a variant of EdDSA whose outputs are indistinguishable from those of the standard version, and we adopt some techniques similar to those in Ref. [19] to deal with the deterministic nature of the protocol.

We prove the protocol secure against adaptive adversaries by reducing it to the classical EdDSA scheme, assuming the security of a non-malleable commitment scheme, the strength of the underlying hash function and an IND-CPA encryption scheme. Moreover, we make some considerations about the resiliency of the recovery, an interesting aspect due to the presence of an offline party, analyzing possible changes that allow us to achieve this higher level of security.

Our Contribution and Related Works As noted in the recent NIST roadmap [2], one of the most challenging task of designing a threshold version of EdDSA is the distributed deterministic nonce generation. In particular, in Sections 4.3.2 and 4.3.3, the authors suggested the usage of MPC techniques or ZKP to overcome the problem of checking the correct computation of the nonce.

In Ref. [20], the authors propose an elegant MPC-based solution suitable for HashEdDSA, a variant of EdDSA where the message to be signed is hashed in advance using a collision-resistant hash function. To obtain an efficient threshold EdDSA, the authors also propose to replace SHA512 and SHAKE256 with an MPC-friendly hash function such as Rescue. Notice that the authors rely on Q2 access structures, thus t is bounded to be small enough to guarantee that no union of two unqualified sets covers the whole set of parties, in particular $t < \frac{n}{2} + 1$.

Following the footsteps of previous works about Schnorr signatures with deterministic nonce generation, such as Ref. [21] and particularly Ref. [19], in this paper, we move away from MPC techniques and instead we use ZKP to prove that the random nonce is indeed generated correctly. This allows us to design a threshold version of EdDSA without the need to distribute the computation of any hash function.

It is worth mentioning that dropping the deterministic nonce generation requirement leads to the possibility of having more straightforward schemes, such as Refs. [22,23].

Organization We present some preliminaries in Sect. 2. We describe our protocol in Sect. 3, in particular in Sect. 3.6, we provide a protocol extension that

includes key derivation. In Sect. 4, we state and prove the security properties of our protocol. 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.

Notation We use the symbol $||$ to indicate the concatenation of bit-strings. Sometimes we slightly abuse the notation and concatenate a bit-string M with an elliptic curve point $\mathcal{P}$, in those cases, we assume that there has been fixed an encoding φ that maps elliptic curve points into bit-strings, so $M||\mathcal{P} := M||\varphi(\mathcal{P})$.

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.

We use a blackboard-bold font to indicate algebraic structure (i.e., sets, groups, rings, fields and elliptic curves), and a calligraphic font to denote points over elliptic curves. About elliptic curves, we distinguish the notation for the curve used for the signature and for the auxiliary curve used in the deterministic nonce generation: the latter are characterized by a prime symbol.

2.1. Decisional Diffie–Hellman Assumption

Our proof is based on the Decisional Diffie–Hellman [24] (from now on DDH).

Definition 2.1. (*Group Families*) A group family is a set of finite cyclic groups $\{\mathbb{G}_\mathfrak{p}\}$ where $\mathfrak{p}$ ranges over an infinite index set. We assume there is a polynomial time (in $|\mathfrak{p}|$) algorithm that given $\mathfrak{p}$ and two elements in $\mathbb{G}_\mathfrak{p}$ outputs their sum.

Definition 2.2. (*Instance Generator*) An instance generator $\texttt{InstanceGen}$ for a group family $\{\mathbb{G}_\mathfrak{p}\}$ is a randomized algorithm that, given an integer n, runs in polynomial time in n and outputs some random index $\mathfrak{p}$ and a generator $\mathcal{B}$ of $\mathbb{G}_\mathfrak{p}$. Note that for each n, $\texttt{InstanceGen}$ induces a distribution on the set of indexes $\mathfrak{p}$.

The index $\mathfrak{p}$ encodes the group parameters, for example, in the case of the group of points of elliptic curves $\mathfrak{p} = (q, a, b)$ denotes the elliptic curve $E_{a,b}/\mathbb{F}_q$.

Definition 2.3. (*DDH Assumption*) Let $\{\mathbb{G}_\mathfrak{p}\}$ be a family of (additive) cyclic group with parameters $\mathfrak{p}$. A Decisional Diffie–Hellman (DDH) algorithm for $\{\mathbb{G}_\mathfrak{p}\}$ is a probabilistic polynomial algorithm $\mathfrak{A}_{\texttt{DDH}}$ satisfying, for some fixed $\alpha > 0$ and sufficiently large n:

$$|\mathbb{P}(\mathfrak{A}_{\texttt{DDH}}(\mathfrak{p}, \mathcal{B}, x\mathcal{B}, y\mathcal{B}, xy\mathcal{B}) = 1) - |\mathbb{P}(\mathfrak{A}_{\texttt{DDH}}(\mathfrak{p}, \mathcal{B}, x\mathcal{B}, y\mathcal{B}, z\mathcal{B}) = 1)| > \frac{1}{n^\alpha}$$

where $\mathcal{B}$ is a generator of $\mathbb{G}_\mathfrak{p}$ and the probability is over the random choice of $\mathfrak{p}$ and $\mathcal{B}$ according to the distribution induced be $\texttt{InstanceGen}(n)$, the

random choice of x, y, z in the range $[1, |\mathbb{G}_{\mathfrak{p}}|]$, and the random bits used by $\mathfrak{A}_{\mathtt{DDH}}$.

The group family $\{\mathbb{G}_{\mathfrak{p}}\}$ satisfies the DDH assumption is there is no DDH algorithm for it.

2.2. Cryptographic Hash Functions

In the EdDSA scheme (and therefore, in our threshold protocol), a cryptographic hash function H is used as a *Pseudo-Random Number Generator* (PRNG), employed to derive secret scalars and nonces.

For this reason, we map the output of the hash function onto the field $\mathbb{Z}_q$ where q is a prime and the order of the base point $\mathcal{B}$ used in EdDSA (i.e., $\mathcal{B}$ generates a subgroup of elliptic curve points with prime order q), and we require H to behave like a *Random Oracle*. We formalize our requirements with the following definition.

Definition 2.4. (*Good PRNG*) Let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a function that maps bit-strings of arbitrary length into elements of $\mathbb{Z}_q$. H is a *Good PRNG* if no efficient algorithm can distinguish between the distributions of $H(S)$ and x, where both $x \in \mathbb{Z}_q$ is chosen uniformly at random, and $S \in \{0,1\}^*$ is a bit-string that embeds at least n bits of entropy, with $2^n < q < 2^{n+1}$.

This definition is not standard, but precisely captures exactly what we need from a hash function to generate a good nonce. Also note that the stronger classical definition of a Random Oracle, that is usually used to study the security of EdDSA, perfectly satisfies our definition.

For secret scalars, EdDSA uses the hash function in a slightly more complicated way, in order to prevent timing leaks in poor implementations, put a lower bound on standard attacks, and embed the curve cofactor into the scalar, so that even a multiplication by an adversary-controlled point would not leak information about the secret (although note that this does not happen in the EdDSA scheme). More precisely, EdDSA uses the following PRNG.

Let $H : \{0,1\}^b \to \{0,1\}^n$ be a function that maps bit-strings of length b into bit-strings of length $c \leq n \leq b$, with $q < 2^{n+2-c}$, and $c \in \{2,3\}$, and $\psi : \{0,1\}^n \to \mathbb{Z}_q$ is defined as

$$\psi(h) = 2^{n+1} + \sum_{i=c}^{n} 2^i h_i \mod q. \tag{1}$$

EdDSA generates some scalars as $\psi(H(\mathtt{k})) \in \mathbb{Z}_q$, where $\mathtt{k} \in \{0,1\}^b$ is chosen uniformly at random.

Note that if $\mathcal{B}$ is the generator of an additive group of order q in which the discrete logarithm problem is hard, it is infeasible to distinguish $\mathfrak{A} = \psi(H(\mathtt{k}))\mathcal{B}$ and $x\mathcal{B}$ if both $\mathtt{k} \in \{0,1\}^b$, $x \in \mathbb{Z}_q$ are chosen uniformly at random and H is a PRF [25].

2.3. EdDSA

Edwards-curve Digital Signature Algorithm (EdDSA) [25] is a digital signature scheme based on twisted Edwards curves. It is designed to be faster than the previously developed schemes without sacrificing security.

EdDSA has several parameters: a prime field $\mathbb{F}_p$; an integer b with $2^{b-1} > p$; a $(b-1)$-bit encoding of elements of the finite field $\mathbb{F}_p$ (if omitted it is assumed to be the classical *little-endian* encoding); a cryptographic hash function H producing $2b$-bit outputs; an integer $c \in \{2, 3\}$ associated to the cofactor of the curve, an integer n with $c \leq n \leq b$ (secret scalars are $n+1$ bits long); a non-zero square element $a \in \mathbb{F}_p$; a non-square element d of $\mathbb{F}_p$; a point $\mathcal{B} \neq (0, 1)$ of the curve described by the equation:

$$ax^2 + y^2 = 1 + dx^2y^2, \tag{2}$$

and a prime q such that $q\mathcal{B} = 0$ and $2^c q$ is the number of points of the curve. Elliptic curve points are encoded as b-bit strings that are the $(b-1)$-bit encoding of their second coordinate y, followed by a *sign bit* that is set if the $(b-1)$-bit encoding of the first coordinate x is lexicographically larger than the $(b-1)$-bit encoding of $-x$. When we concatenate a point and a bit-string (e.g., $\mathcal{P}||S$), we implicitly encode the point into a bit-string as explained above.

Given the parameters $(p, b, H, c, n, a, d, \mathcal{B}, q)$ described above, the protocol works as follows:

1. Choose a random $b-bit$ string $\mathtt{k}$, that will be the secret key.
2. Compute $H(\mathtt{k}) = (h_0, ..., h_{2b-1})$.
3. Compute $a = \psi(h_0||\dots||h_{n-1})$ (where ψ is the same as Eq. (1)), the public key is set to be $\mathcal{A} = a\mathcal{B}$.
4. To sign a message M compute $r = H(h_b||...||h_{2b-1}||M)$ (interpreting the digest as an integer), and $\mathcal{R} = r\mathcal{B}$.
5. The signature is $(\mathcal{R}, S)$, where $S = (r + aH(\mathcal{R}||\mathcal{A}||M)) \mod q$.
6. to verify the signature check if $2^c S\mathcal{B} = 2^c\mathcal{R} + 2^c H(\mathcal{R}||\mathcal{A}||M)\mathcal{A}$.

2.4. Encryption Scheme

In our protocol, we need an asymmetric encryption scheme to communicate with the offline party. The minimum requirement we ask for our protocol to be secure is that the encryption scheme chosen by the offline party has the property of IND-CPA [26,27], i.e.,

Definition 2.5. Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme. Let us define the following experiment between an adversary $\mathfrak{A}$ and a challenger $\mathcal{C}^b$ parametrized 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 $\mathfrak{A}$.
2. $\mathfrak{A}$ 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 $\mathfrak{A}$.
4. $\mathfrak{A}$ 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 is secure against a chosen plaintext attack if for any k and any probabilistic polynomial time adversary $\mathfrak{A}$ the function

$$\mathsf{Adv}(\mathfrak{A}) = \mathbb{P}[\mathcal{C}^1(\mathfrak{A}, k) = 1] - \mathbb{P}[\mathcal{C}^0(\mathfrak{A}, k) = 1], \tag{3}$$

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

This hypothesis will be enough to prove the unforgeability of the protocol, but it is possible to achieve an higher notion of security using more sophisticated encryption scheme that supports ZKP for the Discrete Logarithm. This will be more clearly explained in Sect. 4.1.

2.5. Commitment Schemes

A commitment scheme [28] is composed by two algorithms:

- $\text{Com}(M) : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$: takes in input the value M to commit[1] and, using a random source, outputs the commitment string C and the decommitment string D.
- $\text{Ver}(C, D) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$: takes the commitment and decommitment strings C, D and outputs a message M' if the input pair is valid, $\perp$ otherwise.[2]

We require a commitment scheme to have the following properties:

- **Correctness:** for every value M, it holds $\text{Ver}(\text{Com}(M)) = M$.
- **Binding:** for every commitment string C it is infeasible to find $M \neq M'$ and $D \neq D'$ such that $\text{Ver}(C, D) = M$ and $\text{Ver}(C, D') = M'$ with both $M, M' \neq \perp$.
- **Hiding:** Let $(C, D) = \text{Com}(M_b)$ with $b \in \{0,1\}$, $M_0 \neq M_1$, then it is infeasible for an attacker that may choose $M_0 \neq M_1$ and sees only C, to correctly guess b with more than negligible advantage.
- **Non-malleability:** Given $(C, D) = \text{Com}(M)$ such that $\text{Ver}(C, D) = M$, it is infeasible for an adversary $\mathfrak{A}$ to produce C', D' such that $\text{Ver}(C', D') = M'$ with M' *related* to M, that is $\mathfrak{A}$ can only create commitments to values that are independent from M.

2.6. Zero-Knowledge Proofs

In the protocol, various Zero-Knowledge Proofs (ZKP) [29] 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 of rewinding its execution.

In particular, we use ZKP *of Knowledge* (ZKPoK) to guarantee the usage of secret values that properly correspond to the public counterpart, specifically the Schnorr protocol for discrete logarithms, and its variant that proves that two public values are linked to the same secret (see Refs. [30, 31] and Appendix A.1). 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

[1] In the protocol and the simulations we implicitly encode every value we need to commit into a bit-string, assuming there is a standard encoding understood by all parties.

[2] Again, in the protocol we implicitly decode valid decommitment outputs (i.e., $\neq\perp$) into the original value, assuming that the decoding is also standard and understood by all parties.

extract those secrets and use them in the simulation. Conversely, by exploiting the zero-knowledge property we can trick the adversary in believing that we know our secrets even if we do not, thus we still obtain a correct simulation of our protocol from the adversary's point of view.

However, Schnorr protocol requires a prime order group, so we implicitly use the Ristretto technique [32] for constructing prime order elliptic curve groups, and we transform elliptic curve points in Ristretto points for these computations. This method extends Mike Hamburg's Decaf [33] approach to cofactor elimination to support cofactor-8 curves such as Curve25519 [34,35] (the standard EdDSA curve). We refer to the original sources for more details about this approach.

2.7. Feldman VSS

Feldman-VSS scheme [36] is a verifiable secret sharing scheme built on top of Shamir's scheme [37]. 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 Refs. [38,39].

The scheme works as follows:

1. A cyclic group $\mathbb{G}$ of prime order q is chosen, as well as a generator $\mathcal{B} \in \mathbb{G}$. 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 $t-1$ with coefficients in $\mathbb{Z}_q$, such that $P(0) = s$ where $s \in \mathbb{Z}_q$ is the secret to be shared.
3. Each of the n share holders receive a value $P(i) \in \mathbb{Z}_q$. So far, this is exactly Shamir's scheme.
4. To make these shares verifiable, the dealer distributes commitments to the coefficients of P. Let $P(X) = s + \sum_{i=1}^{t-1} a_i X^i$, then the commitments are $\mathcal{C}_0 = s\mathcal{B}$ and $\mathcal{C}_i = a_i\mathcal{B}$ for $i \in \{1, \ldots, t-1\}$.
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 verifying if the following equality holds:

$$\alpha\mathcal{B} = \sum_{j=0}^{t-1} (i^j)\mathcal{C}_j = s\mathcal{B} + \sum_{j=1}^{t-1} a_j(i^j)\mathcal{B} = \left(s + \sum_{j=1}^{t-1} a_j(i^j)\right)\mathcal{B} = P(i)\mathcal{B}.$$

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 sets $\mathcal{C}_0 = \mathcal{Y}$ and, depending on whether the adversary is P_1 or P_2, it computes:

$$\mathcal{C}_1 = a\mathcal{B} - \mathcal{Y}; \tag{4}$$

in the case the adversary is P_1, or

$$\mathcal{C}_1 = \tfrac{1}{2}(b\mathcal{B} - \mathcal{Y}); \tag{5}$$

in the case the adversary is P_2.

- In either case, the other player can successfully verify their shards, performing the corresponding check:

$$a\mathcal{B} = \mathcal{Y} + \mathcal{C}_1 = \mathcal{Y} + a\mathcal{B} - \mathcal{Y} \tag{6}$$

or

$$b\mathcal{B} = \mathcal{Y} + 2\mathcal{C}_1 = \mathcal{Y} + 2 \cdot \tfrac{1}{2}(b\mathcal{B} - \mathcal{Y}). \tag{7}$$

2.8. Deterministic Nonce Generation

One of the peculiar features of EdDSA is that it is a deterministic signature algorithm, in the sense that it does not require the generation of a random nonce.

To achieve the same feature, we rely on a verifiable random nonce generator: roughly each player chooses a random seed during the Key Generation algorithm, and each time a signature is produced, it is proven that the nonce used in the signature algorithm is coherent with the seed.

In particular, we use Purify [19], a Pseudo-Random Function (PRF) purely based on elliptic curves.

Let $\mathbb{E}$ be an Edwards curve for the EdDSA algorithm, with a point $\mathcal{B}$ of order q, Purify requires the choice of a second elliptic curve $\mathbb{E}'$ over $\mathbb{F}_{q^2}$ whose group of points is cyclic of order q' is generated by a point $\mathcal{B}'$ and is such that the DDH assumption holds. In particular the participants fix a quadratic non-residue $\delta \in \mathbb{F}_q^*$ and find $a, b \in \mathbb{F}_q$ such that

- The equation $y^2 = x^3 + ax + b$ defines an elliptic curve $\mathbb{E}_1$ over $\mathbb{F}_q$ of a prime order q_1 in which the DDH assumption holds;
- The equation $y^2 = x^3 + a\delta^2 x + b\delta^3$ defines an elliptic curve $\mathbb{E}_2$ over $\mathbb{F}_q$ of a prime order $q_2 \neq q_1$ in which the DDH assumption holds;

Then define $\mathbb{E}'$ as the elliptic curve defined by the equation $y^2 = x^3 + ax + b$ over $\mathbb{F}_{q^2}$. It is possible to prove that there is an efficiently computable and invertible isomorphism $\phi : \mathbb{E}' \to \mathbb{E}_1 \times \mathbb{E}_2$.

Let $z \in \{0,1\}^*$ be a string, we define the hash function

$$H_{\mathtt{Pur}}(z) = \phi^{-1}(H_1(z), H_2(z)) \tag{8}$$

where H_1 and H_2 are hash functions onto $\mathbb{E}_1$ and $\mathbb{E}_2$, respectively.

Now let $f : \mathbb{E}' \to \mathbb{Z}_q$ be the function defined as follows:

$$f(\mathcal{Q}) = \begin{cases} 0 & \text{if } \mathcal{Q} = 0_{\mathbb{E}'} \\ x_0 & \text{if } \mathcal{Q} = (x_0 + x_1\sqrt{\delta}, y_0 + y_1\sqrt{\delta}) \end{cases} \tag{9}$$

It is possible to prove that the uniform distribution over $\mathbb{Z}_q$ is statistically close to $f(U_{\mathbb{E}'})$, where $U_{\mathbb{E}'}$ is the uniform distribution over $\mathbb{E}'$. So, if z is a random uniformly distributed string, u is distributed uniformly in $\mathbb{Z}_{q'}$, and H_1, H_2 behave like random oracles, we have that $f(uH_{\mathtt{Pur}}(z))$ is uniformly distributed in $\mathbb{Z}_q$.

The crucial aspect of this construction is that it allows the possibility of building a non-interactive ZKP for the relation $\mathcal{U}' = u'\mathcal{B}'$ and $\mathcal{R} = f(u\mathcal{V}')\mathcal{B}$ where $\mathcal{B}, \mathcal{B}'$ are defined as before, $\mathcal{V}'$ is a public random point of $\mathbb{E}'$ and u' is the private input of the prover. This allows the construction of a verifiable pseudo-random nonce generator. In particular the ZKP is described in Ref. [19] and makes use of the Bulletproof framework [40].

Formally, the security of Purify is stated in the following Lemma:

Lemma 2.1. *Let $\mathbb{E}'$ be an elliptic curve over $\mathbb{F}_{q^2}$ whose group of points generated by a point $\mathcal{B}'$ is cyclic of order q' and is such that the DDH assumption holds. Let u be a random element of $\mathbb{Z}_{q'}$ and $H_{\mathtt{Pur}}$ and f be defined as, respectively, in Eqs. (8) and (9). Then, $H_{\mathtt{Pur}}$ is indistinguishable from a random oracle onto $\mathbb{E}'$ and thus also $f(uH_{\mathtt{Pur}}(\cdot))$ is indistinguishable from a random oracle onto $\mathbb{Z}_q$.*

Moreover, it is possible to build a secure non-interactive ZKP for the relation $\mathcal{U}' = u'\mathcal{B}'$ and $\mathcal{R} = f(u\mathcal{V}')\mathcal{B}$ where $\mathcal{B}, \mathcal{B}'$ are public data defined as before, $\mathcal{V}'$ is a public random point of $\mathbb{E}'$ and u' is the private input of the prover.

Observation 1. As it will be clear later, in our construction, the signature is deterministic as long as the set of signers is fixed. To achieve a deterministic signature that depends only on the message, an alternative solution is the usage of a multi-party symmetric cipher with authenticated MAC key such as MiMC [41] and the Marvellous [42] family combined with a threshold secret sharing of the key. However, while being suited for multi-party nonce generation, these protocols have the drawback of requiring expensive precomputation steps, that are cumbersome in our settings.

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 (called P_1 and P_2) who performed the key generation, and one used by the offline player (P_3) 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.

In order to obtain a deterministic signature scheme, we need to rely on a verifiable nonces generation algorithm. We choose to use Purify, described in Ref. [19]. Starting from some secrete parameters and their committed value, this algorithm allows every party to check whether the other party has computed the correct random value or not.

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. We also assume that every communication between parties is made on a private and authenticated channel.

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:	$\mathbb{E}, \mathcal{B}, q, H$ $\mathbb{E}', \mathcal{B}', q',$	**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 (sk_3, pk_3) and the encryption algorithm itself are unrelated to the signature algorithm, but it is important that both of them are secure. We require the encryption protocol to be IND-CPA, see Sect. 2.4 and Sect. 4.1 for more details.

Then P_1 and P_2 need to agree on a secure hash function H whose outputs we interpret as elements of $\mathbb{Z}_q$, a twisted Edwards elliptic curve $\mathbb{E}$ with cofactor 2^c, and a generator $\mathcal{B} \in \mathbb{E}$ of a subgroup of points of prime order q. The order identifies the ring $\mathbb{Z}_q$ used for scalar values. Lastly they need to agree on the Purify parameters, in particular they choose a second elliptic curve $\mathbb{E}'$ over $\mathbb{F}_{q^2}$ and a base point $\mathcal{B}' \in \mathbb{E}'$ which generates a group of points of order q'.

3.2. Key Generation

The parameters involved are:

Player 1		**Player 2**	
Input:	pk_3	**Input:**	pk_3
Private Output:	ω_1, r'_1	**Private Output:**	ω_2, r'_2
Shared Secret:	$\mathcal{D}$	**Shared Secret:**	$\mathcal{D}$
Public Output:	$\mathsf{rec}_{1,3}$, $\mathsf{rec}_{2,3}$, $\mathcal{A}$, X_1, X_2	**Public Output:**	$\mathsf{rec}_{1,3}$, $\mathsf{rec}_{2,3}$, $\mathcal{A}$, X_1, X_2

The protocol proceeds as follows:

1. Secret key generation and communication:
   a. P_i picks randomly $a_i, y_{3,i}, m_i \in \mathbb{Z}_q$, $r'_i \in \mathbb{Z}_{q'}$, and sets $\mathcal{A}_i = a_i\mathcal{B}$, $\mathcal{Y}_{3,i} = y_{3,i}\mathcal{B}$, $\mathcal{R}'_i = r'_i\mathcal{B}'$, $\mathcal{M}_i = m_i\mathcal{B}$.
   b. P_i computes $[\mathsf{KGC}_i, \mathsf{KGD}_i] = \mathrm{Com}((\mathcal{A}_i, \mathcal{Y}_{3,i}, \mathcal{R}'_i, \mathcal{M}_i))$.
   c. P_i sends KGC_i to P_j.
   d. P_i sends KGD_i to P_j.
   e. P_i gets $(\mathcal{A}_j, \mathcal{Y}_{3,j}, \mathcal{R}'_j, \mathcal{M}_j) = \mathrm{Ver}(\mathsf{KGC}_j, \mathsf{KGD}_j)$, and saves the pairs $X_1 = (\mathcal{A}_1, \mathcal{R}'_1)$ and $X_2 = (\mathcal{A}_2, \mathcal{R}'_2)$.
2. Feldman VSS and generation of P_3's data:
   a. P_i sets $f_i(x) = a_i + m_i x$ and computes $y_{i,j} = f_i(j)$ for $j \in \{1, 2, 3\}$.
   b. P_i encrypts $y_{i,3}, y_{3,i}$ with pk_3, let $\mathsf{rec}_{i,3}$ be the pair of ciphertexts obtained.
   c. P_i sends $y_{i,j}, \mathsf{rec}_{i,3}$ to P_j.
   d. If the asymmetric encryption algorithm supports DLOG verification, the encryption $\mathsf{rec}_{i,3}$ is accompanied by two NIZKPs: the first one proves that the first ciphertext in $\mathsf{rec}_{i,3}$ is the encryption of the DLOG of $\mathcal{Y}_{i,3} = \mathcal{A}_i + 3\mathcal{M}_i$, the second NIZKP proves that the second ciphertext is the encryption of the DLOG of $\mathcal{Y}_{3,i}$. P_i checks the NIZKPs attached to $\mathsf{rec}_{j,3}$.
   e. P_i checks, as in the Feldman-VSS described in Sect. 2.7, the integrity and consistency of the shards $y_{j,i}$, by verifying whether $\mathcal{Y}_{j,i} = \mathcal{A}_j + i\mathcal{M}_j$, where $\mathcal{Y}_{j,i} = y_{j,i}\mathcal{B}$.
   f. P_i computes $x_i = y_{1,i} + y_{2,i} + y_{3,i}$.
3. P_i proves in ZK the knowledge of x_i using Schnorr's protocol of Appendix A.1.
4. Public key and shards generation:
   a. the public key is $\mathcal{A} = \sum_{i=1}^{3} \mathcal{A}_i$, where $\mathcal{A}_3 = 2\mathcal{Y}_{3,1} - \mathcal{Y}_{3,2}$, so that $a_3 = 2y_{3,1} - y_{3,2}$. From now on we will set $a = \sum_{i=1}^{3} a_i$, obviously $a\mathcal{B} = \mathcal{A}$.
   b. P_1 computes $\omega_1 = 2x_1$, while P_2 computes $\omega_2 = -x_2$.
   c. P_i computes the common secret $\mathcal{D} = y_{3,i}\mathcal{Y}_{3,j}$.

Observation 2. We define $a_3 = 2y_{3,1} - y_{3,2}$ because we need to be consistent with the Feldman-VSS protocol. Indeed, suppose that $y_{3,2}$ and $y_{3,1}$ are valid

shards of a Feldman-VSS protocol where the secret is a_3. In this way, we have that $y_{3,2} = a_3 + 2m_3$ and $y_{3,1} = a_3 + m_3$, so:

$$2y_{3,1} - y_{3,2} = 2a_3 + 2m_3 - a_3 - 2m_3 = a_3.$$

Note that $\mathcal{A}_3 = a_3\mathcal{B}$ can be computed by both P_1 and P_2, but they cannot compute a_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 and the goal of this protocol is to produce a valid EdDSA signature $(\mathcal{R}, S)$ for the public key $\mathcal{A}$. The parameters involved are:

Player A		**Player** B	
Input:	$M, \omega_A, \mathcal{A}, r'_A$ X_A, X_B	**Input:**	$M, \omega_B, \mathcal{A}, r'_B$ X_A, X_B
Public Output:	$(\mathcal{R}, S)$	**Public Output:**	$(\mathcal{R}, S)$

The protocol works as follows:

1. Generation of $\mathcal{R}$:
   a. P_i computes $\mathtt{K} = H(X_A, X_B)$.
   b. P_i computes $\mathcal{V}' = H_{\mathtt{Pur}}(\mathtt{K}, M) \in \mathbb{E}'$.
   c. P_i computes $r_i = f(r'_i\mathcal{V}')$.
   d. P_i computes $\mathcal{R}_i = r_i\mathcal{B}$.
   e. P_i sends $\mathcal{R}_i$ to P_j alongside a non-interactive zero-knowledge proof that it is correct given $\mathcal{R}'_i$ (see Sect. 2.8).
   f. P_i checks the correctness of the value $\mathcal{R}_j$ received by verifying the attached NIZKP.
   g. P_i computes $\mathcal{R} = \mathcal{R}_A + \mathcal{R}_B$.
2. Generation of S:
   a. P_i computes $S_i = r_i + \omega_i H(\mathcal{R}||\mathcal{A}||M)$
   b. P_i sends S_i to P_j.
   c. P_i computes $S = S_A + S_B$.
3. P_i checks that $S\mathcal{B} = \mathcal{R} + H(\mathcal{R}||\mathcal{A}||M)\mathcal{A}$.

If any check fails the protocol aborts, otherwise the output signature is $(\mathcal{R}, S)$. Notice that the last step is the signature verification of the original EdDSA, and that if the parties are honest the signature verifies with probability 1 by design, thus the protocol is correct.

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 with the following parameters (suppose wlog that P_1 plays the roles of P_A and P_2 of P_B):

Player A		**Player** B	
Input:	$\omega_1, r'_1,$ $M, \mathcal{A}, X_1, X_2$	**Input:**	$\omega_2, r'_2,$ $M, \mathcal{A}, X_1, X_2$
Public Output:	$(\mathcal{R}, S)$	**Public Output:**	$(\mathcal{R}, S)$

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 (the method used by either P_1 or P_2 to contact P_3 is out of the scope of this paper and it is not relevant for the security discussion).

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:	$\omega_1, r'_1,$ $M, \mathcal{A}, X_1, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$	**Input:**	$\mathsf{sk}_3,$ M
Public Output:	$(\mathcal{R}, S)$	**Public Output:**	$(\mathcal{R}, S)$

The workflow in this case is:

1. Communication:
   a. P_1 contacts P_3 and sends $\mathcal{A}, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}, X_1$.
   b. P_3 decrypts everything with the private key sk_3 to recover the values $y_{1,3}$, $y_{3,1}$, $y_{2,3}$, $y_{3,2}$.
   c. P_3 computes $a_3 = 2y_{3,1} - y_{3,2}$ and $\mathcal{A}_3 = a_3\mathcal{B}$.
   d. P_3 picks randomly $r'_3 \in \mathbb{Z}_{q'}$ and computes $\mathcal{R}'_3 = r'_3\mathcal{B}'$.
   e. P_3 sends $X_3 = (\mathcal{A}_3, \mathcal{R}'_3)$ to P_1.
2. P_3's key creation:
   a. P_3 computes $x_3 = y_{1,3} + y_{2,3} + 2y_{3,2} - y_{3,1}$.
   b. P_i proves in ZK the knowledge of x_i using Schnorr's protocol (note that $x_1 = \omega_1/2$).
3. Signature generation:
   a. P_1 computes $\tilde{\omega}_1 = \frac{3}{4}\omega_1$.
   b. P_3 computes $\omega_3 = -\frac{1}{2}x_3$.
   c. P_1 and P_3 perform the Signature Algorithm as P_A and P_B, respectively, where P_1 uses $\tilde{\omega}_1$ instead of ω_A and X_1 instead of X_A, while P_3 uses ω_3 in place of ω_B and X_3 in place of X_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:

The first two steps are identical to the previous case (for the ZKP of x_2 note that $x_2 = -\omega_2$).

3. The Signature generation step proceeds as follows:

Player 2		**Player 3**	
Input:	$\omega_2, r'_2,$ $M, \mathcal{A}, X_2, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$	**Input:**	$\mathsf{sk}_3,$ M
Public Output:	$(\mathcal{R}, S)$	**Public Output:**	$(\mathcal{R}, S)$

a. P_2 computes $\tilde{\omega}_2 = -3\omega_2$.
b. P_3 computes $\omega_3 = -2x_3$.
c. P_2 and P_3 perform the Signature Algorithm as P_A and P_B, respectively, where P_2 uses $\tilde{\omega}_2$ instead of ω_A and X_2 instead of X_A and P_3 uses ω_3 in place of ω_B and X_3 in place of X_B (the other parameters are straightforward).

Observation 3. $\mathcal{R}'_3$ could be generated and published ahead of time (e.g., during the setup phase), and used for all subsequent recovery signatures. In this case, after P_3 computes X_3 for the first time then the value is fixed for all the successive executions.

The reasons why it is necessary to have a different X_i for each player will be more clear later, during the security discussion of the protocol in Sect. 4.

3.6. Key Derivation

Using the common secret $\mathcal{D}$ created during the key-generation phase, it is possible to perform key derivation, starting from ω_1, ω_2 and ω_3. This allows to update the private key of each user and the global public key, thus allowing to create more instances of the protocol, without needing further communication.

Starting from a common derivation index i, the derivation is performed as follows:

- P_1 and P_2 perform the key derivation:
  - $\omega_1 \rightarrow \omega_1^i = \omega_1 + 2H(\mathcal{D}||i)$,
  - $\omega_2 \rightarrow \omega_2^i = \omega_2 - H(\mathcal{D}||i)$;
- P_1 and P_3 perform the key derivation:
  - $\omega_1 \rightarrow \omega_1^i = \omega_1 + \frac{3}{2}H(\mathcal{D}||i)$,
  - $\omega_3 \rightarrow \omega_3^i = \omega_3 - \frac{1}{2}H(\mathcal{D}||i)$;
- P_2 and P_3 perform the key derivation:
  - $\omega_2 \rightarrow \omega_2^i = \omega_2 + 3H(\mathcal{D}||i)$,
  - $\omega_3 \rightarrow \omega_3^i = \omega_3 - 2H(\mathcal{D}||i)$;
- the public key is always updated like this: $\mathcal{A} \rightarrow \mathcal{A}^i = \mathcal{A} + H(\mathcal{D}||i)\mathcal{B}$.

Observation 4. We observe that the algorithm outputs valid keys, such that, for example:

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

Since $(\omega_1^i + \omega_2^i) = \omega_1 + \omega_2 + H(\mathcal{D}||i)$ we have that:

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

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:

Definition 4.1. We say that a (t, n)-threshold signature scheme is strongly unforgeable if no malicious adversary who corrupts at most $t-1$ players can produce with non-negligible probability a new message-signature pair (m, σ), given the view of **Threshold-Sign** on input messages $m_1, ..., m_\ell$ (which the adversary adaptively chooses), as well as the signatures $(\sigma_1, ..., \sigma_\ell)$ 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. *Let $\xi : \{0,1\}^{2b} \to \mathbb{Z}_q$ be the encoding that maps bit-strings into elements of $\mathbb{Z}_q$ via* little-endian *encoding and reduction modulo q, let $\pi : \{0,1\}^{2b} \to \{0,1\}^n$ be the function that truncates a bit-string to n bits: $\pi(h) = h_0 || \dots || h_{n-1}$. Then, assuming that:*

- *H' is a cryptographic hash function such that $H = \xi \circ H'$ is a good PRNG as per Definition 2.4, and $\pi \circ H'$ is a PRF;*
- *the EdDSA signature scheme with parameters $(p, b, H', c, n, a, d, \mathcal{B}, q)$ is strongly unforgeable;*
- *$\mathrm{Com}, \mathrm{Ver}$ is a non-malleable commitment scheme as defined in Sect. 2.5;*
- *the Decisional Diffie Hellman Assumption defined in Definition 2.3 holds for both the curves $\mathbb{E}$ and $\mathbb{E}'$;*
- *the encryption algorithm used by P_3 is IND-CPA, as per Sect. 2.4;*

our threshold protocol built with the hash function H is unforgeable.

The proof will use a classical game-based argument, our goal is to show that if there is an adversary $\mathfrak{A}$ that forges the threshold scheme with a non-negligible probability $\varepsilon > \lambda(\mathtt{k})^{-t}$, where $\mathtt{k}$ is the security parameter, for a polynomial $\lambda(x)$ and $t > 0$, then we can build a forger $\mathfrak{F}$ that forges the original EdDSA scheme with non-negligible probability as well.

Since the algorithm presented is a $(2, 3)$-threshold signature scheme, the adversary will control one player and $\mathfrak{F}$ will simulate the remaining two. Since the role of P_3 is different than those of P_2 and P_3, we have to consider two distinct cases: one where $\mathfrak{A}$ controls P_3 and one where $\mathfrak{A}$ controls 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 $\mathfrak{A}$ controls P_2.

The adversary $\mathfrak{A}$ interacts in our protocol as follows: it first participates in the key-generation protocol to generate a public key $\mathcal{A}$ for the threshold scheme, then it requests the signature on some messages $m_1, ..., m_\ell$. During this phase it can participate in the signature generation or it can query for

signatures generated by P_1, P_3. Eventually the adversary outputs a message $m \neq m_i \ \forall i$ and a valid signature on m with probability at least ε. If we denote with $\tau_{\mathfrak{A}}$ the adversary's tape and with τ_i the tape of the honest player P_i, we can write

$$\mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}[\mathfrak{A}(\tau_{\mathfrak{A}})_{P_i(\tau_i)} = \texttt{forgery}] \geq \varepsilon, \tag{10}$$

where $\mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}$ means that the probability is taken over the random tape $\tau_{\mathfrak{A}}$ of the adversary and the random tape τ_i of the honest player, while $\mathfrak{A}(\tau_{\mathfrak{A}})_{P_i(\tau_i)}$ is the output of the iteration between the adversary $\mathfrak{A}$, running on tape $\tau_{\mathfrak{A}}$, and the player P_i, running on tape τ_i.

Definition 4.2. (*Good Tape*) We say that an adversary's random tape $\tau_{\mathfrak{A}}$ is good if:

$$\mathbb{P}_{\tau_i}[\mathfrak{A}(\tau_{\mathfrak{A}})_{P_i(\tau_i)} = \texttt{forgery}] \geq \frac{\varepsilon}{2}. \tag{11}$$

Now, we have the following Lemma, introduced in Ref. [16]:

Lemma 4.1. *If $\tau_{\mathfrak{A}}$ is a tape chosen uniformly at random, the probability that it is a good one is at least $\frac{\varepsilon}{2}$.*

Proof. In the proof, we will simplify the notation writing $\mathfrak{A}(\tau_{\mathfrak{A}}, \tau_i) = \texttt{forgery}$ instead of $\mathfrak{A}(\tau_{\mathfrak{A}})_{P_i(\tau_i)} = \texttt{forgery}$. In the context of this proof, we will write b to identify a good tape, while c will be a bad one. We can rewrite Eq. 10 in this way:

$$\begin{aligned} A = \mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\tau_{\mathfrak{A}} = b, \mathfrak{A}(\tau_{\mathfrak{A}}, \tau_i) = \texttt{forgery}) + \mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\tau_{\mathfrak{A}} = c, \mathfrak{A}(\tau_{\mathfrak{A}}, \tau_i) = \texttt{forgery}) \\ = \mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\tau_{\mathfrak{A}} = b)\mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\mathfrak{A}(\tau_{\mathfrak{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathfrak{A}} = b) \\ + \mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\tau_{\mathfrak{A}} = c)\mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\mathfrak{A}(\tau_{\mathfrak{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathfrak{A}} = c). \end{aligned} \tag{12}$$

Trivially, we have that $\mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\mathfrak{A}(\tau_{\mathfrak{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathfrak{A}} = b) < 1$, and from the definition of good tape in equation 11, we get

$$\mathbb{P}_{\tau_i,\tau_{\mathfrak{A}}}(\mathfrak{A}(\tau_{\mathfrak{A}}, \tau_i) = \texttt{forgery} | \tau_{\mathfrak{A}} = c) < \frac{\varepsilon}{2}. \tag{13}$$

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

$$\varepsilon \leq A < x \cdot 1 + (1 - x) \cdot \frac{\varepsilon}{2} = x\left(1 - \frac{\varepsilon}{2}\right) + \frac{\varepsilon}{2}, \tag{14}$$

that leads us to the conclusion:

$$x \geq \frac{\varepsilon - \frac{\varepsilon}{2}}{1 - \frac{\varepsilon}{2}} \geq \frac{\varepsilon}{2 - \varepsilon} \geq \frac{\varepsilon}{2}. \tag{15}$$

□

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

Rushing Adversary In the proof, we consider also the case of a rushing adversary, i.e., an adversary that has the additional power of choosing its data after seeing the honest parties' data. In particular, in each communication round, the party corrupted by the adversary waits until it sees all the honest parties' messages and then decides what to send. In the proof, we deal with

the case of rushing adversary separately, showing in details how to change the protocol to adapt to this enhanced security model.

Key-Generation Simulation First, we need to deal with the key-generation algorithm. The simulator $\mathfrak{F}$ plays the role of P_1 and $\mathfrak{A}$ plays the role of P_2. Before starting the simulation, $\mathfrak{F}$ receives from its challenger a public key for the IND-CPA encryption algorithm and an EdDSA public key. The goal is to trick $\mathfrak{A}$ in order to force the public key output by the multi-party computation to match the EdDSA key given by the challenger.
The simulation works as follows:

1. $\mathfrak{F}$ receives from the challenger an EdDSA public key $\mathcal{A}_c$ and the public encryption key pk_3.
2. P_i picks randomly $a_i, y_{3,i}, m_i \in \mathbb{Z}_q$, $r'_i \in \mathbb{Z}_{q'}$, and sets $\mathcal{A}_i = a_i\mathcal{B}$, $\mathcal{Y}_{3,i} = y_{3,i}\mathcal{B}$, $\mathcal{R}'_i = r'_i\mathcal{B}'$, $\mathcal{M}_i = m_i\mathcal{B}$.
3. P_i computes $[\mathsf{KGC}_i, \mathsf{KGD}_i] = \mathrm{Com}((\mathcal{A}_i, \mathcal{Y}_{3,i}, \mathcal{R}'_i, \mathcal{M}_i))$.
4. P_2 sends KGC_2 to P_1.
5. P_1 sends KGC_1 to P_2. It is important that P_2 sends its commitment before P_1, see Observation 5.
6. P_i sends KGD_i to P_j.
7. P_i gets $(\mathcal{A}_j, \mathcal{Y}_{3,j}, \mathcal{R}'_j, \mathcal{M}_j) = \mathrm{Ver}(\mathsf{KGC}_j, \mathsf{KGD}_j)$.
8. At this point $\mathfrak{F}$ knows all the parameters involved in the computation of $\mathcal{A}$, the first part of the key. So it rewinds $\mathfrak{A}$ to the step 5, after the commitment of $\mathfrak{A}$, with the aim to make $\mathcal{A} = \mathcal{A}_c$.
9. $\mathfrak{F}$ computes $\hat{\mathcal{A}} = \mathcal{A}_c - \mathcal{A}_2 - 2\mathcal{Y}_{3,1} + \mathcal{Y}_{3,2}$.
10. $\mathfrak{F}$ picks randomly $y_{1,2} \in \mathbb{Z}_q$ and computes $\widehat{\mathcal{M}} = \frac{1}{2}(y_{1,2}\mathcal{B} - \hat{\mathcal{A}})$ to simulate the VSS (since $\mathfrak{F}$ is not able to compute the random polynomial $f(x)$) as explained in Sect. 2.7.
11. $\mathfrak{F}$ computes the commitment $[\hat{\mathsf{KGC}}_i, \hat{\mathsf{KGD}}_i] = \mathrm{Com}((\hat{\mathcal{A}}, \mathcal{Y}_{3,i}, \mathcal{R}'_i, \widehat{\mathcal{M}}))$ and sends it to $\mathfrak{A}$ as P_1.
12. P_1 sends $\hat{\mathsf{KGD}}$ to P_2.
13. P_i picks randomly $y_{1,3} \in \mathbb{Z}_q$ and encrypts $y_{1,3}, y_{3,1}$ with pk_3, obtaining $\mathsf{rec}_{i,3}$ ($\mathfrak{F}$ has to simulate the NIZKPs if the encryption supports DLOG verification).
14. P_i sends $y_{i,j}, \mathsf{rec}_{i,3}$ to P_j.
15. Since $\mathfrak{F}$ does not know the discrete logarithm of $\hat{\mathcal{A}}$ it can not compute x_1, so it simulates the ZKP with $\mathfrak{A}$.
16. P_2 can calculate x_2 and execute the ZKP, from which $\mathfrak{F}$ extracts the value of x_2.
17. P_i can compute the key $\mathcal{A}$, moreover P_2 can compute ω_2 (for $\mathfrak{F}$ it is impossible, since it does not know x_1).

Observation 5. In the simulation, it is crucial that the adversary broadcasts KGC_2 before $\mathfrak{F}$. Inverting the order will cause this simulation to fail, since after the rewind $\mathfrak{A}$ could change its commitment. Due to the non-malleability property, we are assured that $\mathfrak{A}$ 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, $\mathfrak{F}$ guesses the right $\hat{\mathcal{A}}$ 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 in order to match $\hat{\mathcal{A}}$ and $\hat{\mathcal{M}}$. 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.

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

Proof. The proof of the Lemma is the same as Lemma 1 in Ref. [11], with a the only difference that in our protocol $\mathfrak{F}$ also needs to simulate the computation of $\mathsf{rec}_{1,3}$.

In particular, since $\mathfrak{A}$ is running on a good random tape we know that it will correctly decommit with probability at least $\frac{\varepsilon}{2} \geq \lambda(k)^{-t}$, then we need to rewind only a polynomial number of times.

About the indistinguishability, notice that there are only two differences between the real protocol and the simulated one. The first difference is that $\mathfrak{F}$ does not know the discrete logarithm of $\hat{\mathcal{A}}$ and so it needs to simulate both the Feldman-VSS and the Schnorr protocols. Under the DDH assumption, both of them can be easily simulated in an indistinguishable way as shown in Sect. 2.7 and Appendix A.1.

The second difference is that $\mathfrak{F}$ does not correctly computes $\mathsf{rec}_{1,3}$ and instead sends a random encryption. This falls perfectly into the definition of IND-CPA security Sect. 2.4 and thus the simulated execution is indistinguishable from the real one.

With these considerations, we can conclude that the simulation of the key-generation protocol is perfect. □

Lemma 4.3. *For a polynomially large fraction of inputs $\mathcal{A}_c$ the simulation terminates with output $\mathcal{A}_c$, except with negligible probability.*

Proof. First, we prove that if the simulation terminates correctly (i.e., with output different from $\perp$) then it terminates with output $\mathcal{A}_c$ except with negligible probability.

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

Now, we prove that the simulation ends correctly for a polynomially large fractions of input. Since $\mathfrak{A}$ is running on a good random tape, it decommits correctly for at least $\frac{\varepsilon}{2} > \lambda k^{-t}$ inputs. Moreover, since $\mathcal{A}_c$ is chosen uniformly at random and $\hat{\mathcal{A}} = \mathcal{A}_c - \mathcal{A}_2 - 2\mathcal{Y}_{3,1} + \mathcal{Y}_{3,2}$ is fully determined after the rewind, we have that $\hat{\mathcal{A}}$ has also uniform distribution, then we can conclude that for at least a fraction $\frac{\varepsilon}{2} > \lambda k^{-t}$ of input the protocol will correctly terminate. □

Signature Simulation Now, we have to deal with the ordinary signature algorithm. Here, $\mathfrak{F}$ can fully predict what $\mathfrak{A}$ will output and then it can choose its shards in order to match the signature it received from its oracle. Comparing to the proofs of ECDSA threshold protocols in Refs. [10,43], we do not need to make a distinction between semi-correct and non-semi-correct executions, since we can always provide a perfect simulation that ends with the desired result (except with negligible probability).

It is important to remember that $\mathfrak{F}$ does not know the secret key of P_1 but it knows everything about P_2, since it was able to extract the secret values during the ZKPs.
The simulation works as follows:

1. $\mathfrak{A}$ chooses a message M to sign.
2. $\mathfrak{F}$ queries its signing oracle for a signature for M corresponding to the public key $\mathcal{A}$, and gets $(\mathcal{R}_f, S_f)$.
3. P_i follows the protocol normally and computes $\mathcal{R} = \mathcal{R}_1 + \mathcal{R}_2$. We can notice that $\mathfrak{F}$ is able to follow the protocol normally since it knows r_1' and, therefore, can compute $\mathcal{R}_1$.
4. $\mathfrak{F}$ computes $\hat{\mathcal{R}} = \mathcal{R}_f - \mathcal{R}_2$ and rewinds the adversary at the end of Step 2.
5. $\mathfrak{F}$ sets $\mathcal{R}_1 = \hat{\mathcal{R}}$.
6. P_i follows the protocol normally to get $\mathcal{R} = \mathcal{R}_1 + \mathcal{R}_2$.
7. $\mathfrak{F}$ simulates the ZKP using as input $\mathcal{R}_1'$ and $\hat{\mathcal{R}}$.
8. From the ZKP given by the adversary on behalf of P_2, $\mathfrak{F}$ is able to extract r_2' from the adversary, and therefore, also r_2.
9. Since $\mathfrak{F}$ knows both r_2 and ω_2, P_1 can compute S_1 as: $S_1 = S_f - r_2 - \omega_2 H(\mathcal{R}||\mathcal{A}||M)$.
10. P_i follows the protocol normally to get $S = S_1 + S_2$.
11. P_i checks that $S\mathcal{B} = \mathcal{R} + H(\mathcal{R}||\mathcal{A}||M)A$.

If any check fails the protocol aborts, otherwise the output signature is $(\mathcal{R}, S)$.

Lemma 4.4. *If Purify is secure in the sense of Lemma 2.1, then the protocol above is a perfect simulation of a real execution and terminates correctly with output* $(\mathcal{R}_f, S_f)$.

Proof. The differences between the simulation and the real protocol is that $\mathfrak{F}$ does not know the secret key ω_1 when computing S_1, and the computation of $\mathcal{R}_1$ uses a different PRF.

The lack of knowledge of the secret key is not a problem since $\mathfrak{F}$ is able to retrieve the correct values to output knowing ahead of time what $\mathfrak{A}$ should output.

About the different method used to compute $\mathcal{R}_1$, notice that the assumptions on Purify mean that in a real execution of the protocol r_i has a distribution that is indistinguishable from the uniform distribution over $\mathbb{Z}_q$, i.e., the distribution of $\mathcal{R}_i$. In the simulation $\mathcal{R}_1 = \mathcal{R}_f - \mathcal{R}_2$, where the distribution of $\mathcal{R}_2$ is indistinguishable from the uniform distribution over the group generated by $\mathcal{B}$ (as in the real protocol) and $\mathcal{R}_f$ comes from the EdDSA oracle. From our assumptions on the hash function used in EdDSA,

the distribution of $\mathcal{R}_f$ is also indistinguishable from the uniform distribution over the group generated by $\mathcal{B}$, consequently so is the distribution of $\mathcal{R}_1$.

It is straightforward that if the protocol terminates it will do so with output $(\mathcal{R}, S) = (\mathcal{R}_f, S_f)$, in fact if $\mathfrak{A}$ does not act honestly the check in the last step will fail with high probability.

Finally, the only other way that the protocol does not terminate is when the NIZKP of $\mathcal{R}_2$ does not verify. In this case, the simulation simply aborts, like in a real execution of the protocol. □

Now, we have to deal with the recovery signature. Since the core algorithm remains the same, we can use the proof just explained, we only need to change the setup phase during which the third player recovers its secret material.

First, we will examine what happens if $\mathfrak{A}$ controls one between P_1 or P_2 and $\mathfrak{F}$ controls P_3. Then, we will deal with the case in which $\mathfrak{A}$ controls P_3, that will be easier since the whole enrollment phase can be avoided.

Trivially, if $\mathfrak{A}$ asks for a recovery signature between the two honest parties $\mathfrak{F}$ can simply ask its oracle and output whatever it received from the oracle. Therefore, we can limit ourselves to deal with the case where $\mathfrak{A}$ participates in the signing process.

If $\mathfrak{A}$ controls P_2 the simulation proceeds as follows:

1. P_2 sends to P_3 $\mathcal{A}, X_2, \mathsf{rec}_{1,3}, \mathsf{rec}_{2,3}$.
2. $\mathfrak{F}$ has participated in the key-generation phase, so knows $\mathcal{Y}_{3,1}$ and $\mathcal{Y}_{3,2}$, so can compute on behalf of P_3 the value $\mathcal{A}_3 = 2\mathcal{Y}_{3,1} - \mathcal{Y}_{3,2}$.
3. P_3 picks randomly $r'_3 \in \mathbb{Z}_{q'}$ and computes $\mathcal{R}'_3 = r'_3\mathcal{B}'$.
4. P_3 sends $X_3 = (\mathcal{A}_3, \mathcal{R}'_3)$ to P_1.
5. Note that P_3 can not decrypt the values received in the first step, so it simulates the ZKP about x_3, conversely $\mathfrak{F}$ can extract x_2 from P_2.
6. P_2 computes $\tilde{\omega}_2 = -3\omega_2$. P_3 can not compute its secret key ω_3, but this is not a problem as we explained before.
7. They perform the signing algorithm with the above simulation. Also in this case $\mathfrak{F}$ does not know its own secret key, but we remark that this is fine since it knows P_2's secrets and it can use the signing oracle.

In the same way, we can deal with the case of $\mathfrak{A}$ controlling P_1.

Now, we have to deal with the last case, i.e., when P_3 is the dishonest party.

During the enrollment phase, $\mathfrak{F}$ can produce random shards, which will be sent to P_3 during the recovery signature phase, and output the public key given by the EdDSA challenger. These random shards simulate correctly the protocol for the properties of the secret sharing. In fact the only difference is that once again $\mathfrak{F}$ does not know the corresponding secret keys of one between P_1 and P_2 (one player's keys can be chosen freely, but the others are forced by the challenge public keys), but as before this is not a problem because, thanks to the oracle and the secrets it extracts from P_3, $\mathfrak{F}$ can simulate signatures with the same simulation described above.

Now, we are ready to prove Theorem 4.1.

Proof. As we previously proved, our simulator produces a view of the protocol indistinguishable from the real one for the adversary, so $\mathfrak{A}$ will produce a

forgery with the same probability as in a real execution. Then, the probability of success of our forger $\mathfrak{F}$ is at least $\frac{\varepsilon^3}{8}$, since $\mathfrak{F}$ has to succeed in

- choosing a good random tape for $\mathfrak{A}$, whose probability is at least $\frac{\varepsilon}{2}$, as shown in Lemma 4.1,
- hitting a good public key, whose probability also is at least $\frac{\varepsilon}{2}$ as shown in Lemma 4.2 and Lemma 4.3.

Under those conditions, $\mathfrak{A}$ successfully produces a forgery with probability at least $\frac{\varepsilon}{2}$ as per Eq. (11). Under the security of the EdDSA signature scheme, the probability of success of $\mathfrak{F}$ must be negligible, which implies that ε must negligible too, contradicting the hypothesis that $\mathfrak{A}$ has non-negligible probability of forging the scheme. □

Observation 6. As we said in the Signature Algorithm description, at point 1a., we need to have a different K for each pair of signers, otherwise an adversary having access to all the messages exchanged by the honest parties could steal the secret key. It suffices to ask for the signature of the same message, first the signature is performed by the honest parties, then by the adversary and a honest party, as explained in Section 4 of Ref. [19].

4.1. Resilience of the Recovery

In our security analysis, we focused on the unforgeability of the signature, however with an offline party another security aspect is 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 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 behavior, because we need a verifiable encryption scheme that allows the parties to prove that the recovery material is consistent, just like they prove that they computed the shards correctly.

In particular, we need an encryption scheme that support DLOG verification as explained in point 2f. of the key-generation algorithm. A suitable candidate is a variant of the Cramer-Shoup cryptosystem presented in Ref. [44]. This algorithm equipped with a ZKP that allow the sender to prove that the plaintext he encrypted is the discrete logarithm of a public value. In particular, since the protocol is a three-step ZKP with special soundness, completness and honest-verifier zero knowledge, it is possible to build a non-interactive ZKP using the Fiat–Shamir heuristic.

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. The protocol described in this work is, as far as we know, the first example of threshold multi-signature allowing the presence of an offline participant during key generation and whose signatures are indistinguishable from EdDSA ones.

The approach we have taken is very similar to the one presented in Refs. [10,15], although there are some key differences between the works. First of all our main 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, which allows us to "invert" the generation of the shares, thanks to the preserved uniform distribution in $\mathbb{Z}_p$. These shares are later recovered by the offline party exploiting an asymmetric encryption scheme. A second difference is that we have managed to avoid equivocable commitments, under the assumption that in some specific steps (see Observation 5) we can consider the adversary not to be rushing.

The focus of this work was to shift away from DSA-like protocols and study a more recent standard like EdDSA. We remark that ECDSA is more suited to be used in a multi-party environment: the absence of hash functions to be computed on private data allows a more straightforward adaption to a multi-party setting. Indeed, a joint computation of a standard hash function is difficult in a reasonable time. Therefore, when creating an EdDSA-compatible threshold multi-signature scheme, there is the necessity of working around this issue. Our solution is to build a variant of the EdDSA protocol, whose outputs are indistinguishable from those of the original scheme (and therefore, it preserves the security properties), thus avoiding joint hash computations.

On the other hand, multi-party EdDSA requires less message exchanges between the participants than the amount required for the threshold ECDSA protocol in Ref. [15], since less checks are needed to avoid malicious computations. In particular, the threshold ECDSA protocol requires 10 communication rounds for the signature generation, while this EdDSA protocol requires only 2. Given that efficiency is one of the reasons for preferring EdDSA over ECDSA, we have that our solution maintains this property.

A last remark worth to be mentioned is in the work-around made on the Zero-Knowledge proofs of our EdDSA scheme (as explained in Ref. [32]), which are required to work around the usage of elliptic curves whose group of points does not have prime order.

Similarly to its ECDSA counterpart, in order to guarantee the security of the signature itself against black-box adversaries, the protocol involves a large utilization of ZKPs. Despite the consequent drawbacks in terms of efficiency, our protocols has been successfully implemented and adopted in the management of Libra wallets [45].

Other future research steps involve the generalization to (t, n)-threshold schemes with more than one offline party and the extension of our notion of security. The techniques introduced in Ref. [46] are a direct evolution of the approach used in this paper to achieve the threshold key generation, so it is very likely that those techniques could be used to achieve a general (t, n)-threshold EdDSA scheme where multiple parties can be brought online after key generation.

Although our protocol is susceptible to DOS attacks on the offline party, there are many ways to overcome this apparent weakness, such as the distribution of the role of the Recovery party to multiple servers or the generalization of our scheme to more than three parties.

Acknowledgements

This work was created with the co-financing of the European Union FSE-REACT-EU, PON Research and Innovation 2014–2020 DM1062/2021. The first three authors are members of the INdAM Research Group GNSAGA. The core of this work is contained in the first author's MSC thesis supervised by the second and fourth authors. The authors would like to thank Conio s.r.l. and its co-founder 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.

Author contributions The core of this work is contained in the first author's MSC thesis supervised by the second and fourth authors. The authors would like to thank Conio s.r.l. and its co-founder Vincenzo di Nicola for their support. We also thank Gaetano Russo, Federico Mazzone, and Zsolt Levente Kucsv án who worked on the implementation and provided valuable feedback.

Funding Information Open access funding provided by Università degli Studi di Trento within the CRUI-CARE Agreement.

Data Availability Statement Not applicable.

Declarations

Conflict of Interest The authors have no relevant financial or non-financial interests to disclose. The authors have no conflicts of interest to declare that are relevant to the content of this article. The second and the third authors are members of the INdAM Research group GNSAGA. The first author acknowledges support from TIM S.p.A. through the PhD scholarship. The authors have no financial or proprietary interests in any material discussed in this article.

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 q with generator $\mathcal{B}$. Let $\mathcal{K} \in \mathbb{G}$ 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 $\mathcal{K}$, i.e., it knows $x \in \mathbb{Z}_q$ such that $x\mathcal{B} = \mathcal{K}$.
So the common inputs are $\mathbb{G}, \mathcal{B}$ and $\mathcal{K}$, while the secret input of $\mathcal{P}$ is x.
The protocol works as follows:

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

A detailed proof about the security of the algorithm can be found in Ref. [30].

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 $\tilde{c} \neq c$.
3. Follow the remaining part of the protocol, obtaining $\tilde{z}$.
4. We can compute $\frac{z-\tilde{z}}{c-\tilde{c}} = \frac{(c-\tilde{c})x}{c-\tilde{c}} = x$.

Proof. (Sketch) Since the only extra hypothesis for $\tilde{c}$ is that $\tilde{c} \neq c$ we can suppose that $\tilde{c}$ 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 $\tilde{\mathcal{U}} = (-xc + r)\mathcal{B} = -c(x\mathcal{B}) + r\mathcal{B}$. Under the discrete logarithm assumption and since r, c are random element, this is indistinguishable from $r\mathcal{B}$.

3. The simulator sends $\tilde{\mathcal{U}}$ and the adversary sends c again.
4. The simulator sends $z = r - cx + cx = r$.
5. The adversary checks that $z\mathcal{B} = r\mathcal{B} = \tilde{\mathcal{U}} + c(x\mathcal{B}) = -xc\mathcal{B} + r\mathcal{B} + xc\mathcal{B}$.

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. In general it is not possible to ensure that the same c is sent in both executions. The most common way to resolve this issue is to consider only a binary challenge space, thus the probability of getting the same c twice in a row is $\frac{1}{2}$. If the challenge space grows the probability decreases and we need an exponentially large number of repetitions. Indeed, the Schnorr Protocol is Zero Knowledge only for a binary challenge space, in the general case it is only Honest Verifier Zero Knowledge (HVZK). It is worth noticing that HVZK is a strong enough property to allow the usage of non-interactive zero-knowledge proofs, that drastically decrease the communication complexity of the protocol. For a deeper discussion about non-interactive zero knowledge proofs and their simulation, see Ref. [47]. □

A.1.2. Equality of Discrete Logarithms. This simple variant of the protocol allows to prove that two public elements are linked to the same secret value.

More formally, let $\mathbb{G}$ be a cyclic group of prime order q, let $\mathcal{B}, \bar{\mathcal{B}}$ be generators of $\mathbb{G}$, and finally let $\mathcal{K}, \bar{\mathcal{K}} \in \mathbb{G}$, $x \in \mathbb{Z}_q$. The prover knows x and wants to convince the verifier that:

$$x\mathcal{B} = \mathcal{K} \quad \text{and} \quad x\bar{\mathcal{B}} = \bar{\mathcal{K}}, \tag{16}$$

without disclosing x. The values of $\mathcal{B}$, $\mathcal{K}$, $\bar{\mathcal{B}}$ and $\bar{\mathcal{K}}$ are publicly known.
The protocol proceeds as follows:

1. The prover generates a random r and computes $\mathcal{U} = r\mathcal{B}$ and $\bar{\mathcal{U}} = r\bar{\mathcal{B}}$, then sends $(\mathcal{U}, \bar{\mathcal{U}})$ to the verifier.
2. The verifier computes a random $c \in \{0, 1\}$ and sends it to the prover.
3. The prover creates a response $s = r + c \cdot x$ and sends s to the verifier.
4. The verifier checks that $s\mathcal{B} = c\mathcal{K} + \mathcal{U}$, $s\bar{\mathcal{B}} = c\bar{\mathcal{K}} + \bar{\mathcal{U}}$. If the check fails the proof fails and the protocols aborts.
5. The previous steps are repeated $\tau = \text{poly}(\log_2(q))$ times, i.e., the number of repetitions is polynomial in the length of q (the security parameter).

A detailed analysis of the protocol and its security can be found in Ref. [48].

References

[1] Brandão, L. T. A. N., Davidson, M., Vassilev, A.: NIST Roadmap Toward Criteria for Threshold Schemes for Cryptographic Primitives. Accessed: 2020-08-27. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8214A.pdf

[2] Brandão, L. T. A. N., Davidson, M.: Notes on Threshold EdDSA/Schnorr Signatures. Accessed: 2023-05-01. https://csrc.nist.gov/publications/detail/nistir/8214b/draft

[3] 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)

[4] Canetti, R., Makriyannis, N., Peled, U.: "UC Non-Interactive, Proactive, Threshold ECDSA". In: IACR Cryptol. ePrint Arch. 2020, p. 492 (2020)

[5] Lindell, Y., Nof, A.: 11Fast 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)

[6] MacKenzie, P., Reiter, M. K.: "Two-party generation of DSA signatures". In: Annual International Cryptology Conference. Springer. pp. 137154 (2001)

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

[8] Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party thresholdECDSA from ECDSA assumptions. In: 2018 IEEE Symposium onSecurity and Privacy (SP). IEEE. pp. 980-997. (2018)

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

[10] Gennaro, R., Goldfeder, S.: "Fast multiparty threshold ecdsa with fast trustless setup". In: Proceedings of the 2018 ACM SIGSAC Confer- ence on Computer and Communications Security. ACM. pp. 1179- 1194 (2018)

[11] 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)

[12] Kondi, Y., Magri, B., Orlandi, C., Shlomovits, O.: "Refresh When You Wake Up: Proactive Threshold Wallets with Offine Devices." In: IACR Cryptol. ePrint Arch. 2019, p. 1328 (2019)

[13] Lindell, Y.: "Fast secure two-party ECDSA signing". In: Annual Inter- national Cryptology Conference. Springer. pp. 613-644 (2017)

[14] MacKenzie, P., Reiter, M. K.: "Two-party generation of DSA signatures". In: International Journal of Information Security 2.3-4, pp. 218-239 (2004)

[15] Battagliola, M., Longo, R., Meneghetti, A., Sala, M.: Threshold ECDSA with an offline recovery party. Mediterr. J. Math. **19**(4), (2022). https://doi.org/10.1007/s00009-021-01886-3

[16] Gennaro, R., Goldfeder, S.: "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: Proceedings of the 2018 ACM SIGSAC Con- ference on Computer and Communications Security. New York, NY, USA: Association for Computing Machinery, 1179-1194. (2018). isbn: 9781450356930. https://doi.org/10.1145/3243734.324385

[17] Chohan, U. W.: "The Problems of Cryptocurrency Thefts and Exchange Shutdowns". In: Available at SSRN 3131702 (2018)

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

[19] Nick, J., Ruffing, T., Seurin, Y., Wuille, P.: "MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces". In: Proceed- ings of the 2020 ACM SIGSAC Conference on Computer and Com- munications Security. CCS

'20. Virtual Event, USA: Association for Computing Machinery, 1717-1731. (2020). isbn: 9781450370899. https://doi.org/10.1145/3372297.3417236

[20] Bonte, C., Smart, N.P., Tanguy, T.: Thresholdizing HashEdDSA: MPC to the Rescue. Int. J. Inf. Secur. **20**, 879–894 (2021)

[21] Garillot, F., Kondi, Y., Mohassel, P., Nikolaenko, V.: "Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions". In: Advances in Cryptology - CRYPTO 2021. Ed. by T. Malkin and C. Peikert. Cham: Springer International Publishing, pp. 127- 156. (2021). isbn: 978-3-030-84242-0

[22] Feng, Q., Yang, K., Ma, M., He, D.: Efficient Multi-Party EdDSA Signature With Identifiable Aborts and its Applications to Blockchain. IEEE Trans. Inf. Forensics Secur. **18**, 1937–1950 (2023)

[23] Feng, Q., He, D., Luo, M., Li, Z., Choo, K.-K. R.: "Practical Secure Two-Party EdDSA Signature Generation with Key Protection and Applications in Cryptocurrency". In: 2020 IEEE 19th International Con- ference on Trust, Security and Privacy in Computing and Communica- tions (TrustCom). pp. 137-147. (2020). https://doi.org/10.1109/TrustCom50675.2020.00031

[24] Boneh, D.: "The decision diffie-hellman problem". In: International Al- gorithmic Number Theory Symposium. Springer. pp. 48-63 (1998)

[25] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: "Highspeed high-security signatures". In: Journal of Cryptographic Engineer- ing 2 (), pp. 77-89

[26] Bellare, M., Rogaway, P.: Introduction to Modern Cryptography. (2005). https://web.cs.ucdavis.edu/rogaway/classes/227/spring05/book/main.pdf

[27] 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)

[28] Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. In: Journal of computer and system sciences **37**(2), 156–189 (1988)

[29] 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)

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

[31] Shoup, V., Alwen, J.: Σ-Protocols Continued and Introduction to Zero Knowledge. (2007). https://cs.nyu.edu/courses/spring07/G22.3220-001/lec3.pdf

[32] Arcieri, T., de Valence, H., Lovecruft, I.: The Ristretto Group. (2019) https://ristretto.group/ristretto.html

[33] Hamburg, M.: "Decaf: Eliminating cofactors through point compression". In: Annual Cryptology Conference. Springer. pp. 705-723 (2015)

[34] Bernstein, D. J.: "Curve25519: new Diffie-Hellman speed records". In: International Workshop on Public Key Cryptography. Springer. pp. 207-228 (2006)

[35] Josefsson, S., Liusvaara, I.: "Edwards-curve digital signature algorithm (EdDSA)". In: Internet Research Task Force, Crypto Forum Re- search Group, RFC. Vol. 8032. (2017)

[36] Feldman, P.: "A practical scheme for non-interactive verifiable secret sharing". In: 28th Annual Symposium on Foundations of Computer Sci- ence (sfcs 1987). pp. 427-438 (1987)

[37] Shamir, A.: "How to Share a Secret". In: Commun. ACM 22.11 (Nov. 1979), 612-613. issn: 0001-0782. https://doi.org/10.1145/359168.359176.

[38] 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 Tech- niques. Springer. pp. 295-310 (1999)

[39] Schoenmakers, B.: "A simple publicly verifiable secret sharing scheme and its application to electronic voting". In: Annual International Cryp- tology Conference. Springer. pp. 148-164 (1999)

[40] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: "Bulletproofs: Short Proofs for Confidential Transactions and More". In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315- 334. (2018).https://doi.org/10.1109/SP.2018.00020.

[41] Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: Advances in Cryptology - ASIACRYPT 2016. Ed. by J. H. Cheon and T. Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 191-219. (2016). isbn: 978-3-662-53887-6

[42] Ashur, T., Dhooghe, S.: MARVELlous: a STARK-Friendly Fam- ily of Cryptographic Primitives. Cryptology ePrint Archive, Report 2018/1098. (2018). https://ia.cr/2018/1098

[43] Ashur, T., Dhooghe, S.: MARVELlous: a STARK-Friendly Fam- ily of Cryptographic Primitives. Cryptology ePrint Archive, Report 2018/1098. https://ia.cr/2018/1098.2018

[44] Camenisch, J., Shoup, V.: "Practical Verifiable Encryption and Decryption of Discrete Logarithms". In: Advances in Cryptology - CRYPTO 2003. Ed. by D. Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 126-144. (2003). isbn: 978-3-540-45146-4

[45] Libra. Libra Developer Update: Core Roadmap #3. (2020). https://mailchi.mp/263f908c91f2/libra-dev-update-first-libra-core-summit-3759697

[46] Battagliola, M., Longo, R., Meneghetti, A.: "Extensible Decentralized Secret Sharing and Application to Schnorr Signatures". In: Cryp-tology ePrint Archive (2022)

[47] Abdalla, M., An, J. H., Bellare, M., Namprempre, C.: "From Identi- fication to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security". In: Advances in Cryptol- ogy – EUROCRYPT 2002. Ed. by L. R. Knudsen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 418-433. (2002). isbn: 978-3-540-46035-0

[48] Spadafora, C., Longo, R., Sala, M.: Coercion-Resistant Blockchain- Based E-Voting Protocol. Cryptology ePrint Archive, Report 2020/674. (2020).https://eprint.iacr.org/2020/674

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

Riccardo Longo
e-mail: riccardolongomath@gmail.com

Alessio Meneghetti
e-mail: alessio.meneghetti@unitn.it

Massimiliano Sala
e-mail: maxsalacodes@gmail.com

Received: August 8, 2022.
Revised: May 3, 2023.
Accepted: June 4, 2023.