EXTENSIBLE DECENTRALIZED SECRET SHARING AND APPLICATION TO SCHNORR SIGNATURES

MICHELE BATTAGLIOLA, RICCARDO LONGO, AND ALESSIO MENEGHETTI

Abstract. Starting from links between coding theory and secret sharing we develop an extensible and decentralized version of Shamir Secret Sharing, that allows the addition of new users after the initial shares distribution.

On top of it we design a totally decentralized (t, n)-threshld Schnorr signature scheme that needs only t users online during the key generation phase, while the others join later.

Under standard assumptions we prove our scheme secure against adaptive malicious adversaries. Furthermore, we show how our security notion can be strengthen when considering a rushing adversary. 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 centralized Schnorr scheme with non-negligible probability.

1. Introduction

Decentralized systems are slowly becoming a desirable alternative to centralized ones, due to the advantages of distributing the management of data, such as avoiding single-points-of-failures or the secure storage of crypto-assets. For them to become a viable alternative, it is necessary to use secure decentralized cryptoghraphic schemes. In particular, digital signature schemes assume a central role in this setting, as hinted by the amount of recent works on multi-user schemes and threshold variant of signature protocols (see e.g. [1, 2, 5, 4]).

Roughly summarizing, a threshold variant of a scheme is composed by three algorithms: a multi-party key-generation algorithm, a multi-party signature algorithm, and a verification algorithm which is the same as the one of the centralized scheme. In this context, the multi-party key-generation algorithms usually employ a secret-sharing scheme to obtain a set of shares of the keys used to sign and verify signatures. The most established scheme is Shamir's Secret Sharing Scheme, a secure protocol based on polynomial evaluation. Even though efficient and secure, this scheme relies on a central authority that generates the shares and provides them to the participants to the threshold scheme. In order to achieve a completely decentralized scheme it is therefore necessary to utilize a variant of Shamir's scheme in which the dealer is not a single authority, such as for example the scheme described in [7, Section 5.2]. In this work we present a (t, n) generalisation of Schnorr's $(2, 3)$-Theshold Signature [1], using a decentralized secret-sharing scheme, where the dealer is replaced by a set of users.

Moreover, we also generalize the concept of offline participant defined in [2] with the introduction of an "extensible" key generation, developing a protocol that needs only t participants online during the key generation phase but later can be extended to involve new parties. Indeed, any set of at least t users is able to generate new shards of the private key, allowing the addition of new participants (which have the same characteristics of the initial users) without altering neither the threshold nor the public key.

1.1. Organization.

We start with some preliminaries in Section 2, showing how results from coding theory could be useful in building a Secret Sharing Scheme, defining some notation, and introducing some cryptographic primitives used later on. Then, in Section 3,

we formally define our extensible decentralized secret sharing we use for the key generation algorithm.

In Section 4 we describe the (t, n)-threshold Schnorr Signature. Finally, in Section 5 we prove the security of our protocol.

2. Preliminaries

2.1. **Notation.** We use the symbol || to indicate the concatenation of bit-strings.

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 structures (i.e. sets, groups, rings, fields and elliptic curves). When speaking about a generic group $\mathbb{G}$, we use multiplicative notation unless stated otherwise.

When describing communication steps we will use two indexes, the first denotes the sender, while the second the receiver (i.e. the symbol $x_{i,j}$ denotes that the value x was generated by party i and sent to party j).

With an abuse of notation we sometimes say that a list is a subset of a set. In this context we simply mean that every element of that list is an element of the set.

2.2. **From MDS Codes to Secret Sharing.** Let $\mathbb{F}_q$ be the finite field with q elements and let α be an agreed-upon primitive element of $\mathbb{F}_q$. Let $\{p^{(i)}\}_{i=1,\dots,\tau} \subseteq \mathbb{F}_q[x]$ be a set of τ polynomials of degree $t-1$, so $p^{(i)} = \sum_{k=0}^{t-1} p_k^{(i)} x^k$, where $p_k^{(i)} \in \mathbb{F}_q$ is the k-th coefficient of the polynomial $p^{(i)}$.

Let $p = \sum_{i=1}^{\tau} p^{(i)}$, with coefficients $p_k = \sum_{i=1}^{\tau} p_k^{(i)}$ for $k = 0, \dots, t-1$, and define $\beta_j = p(\alpha^j)$. Note that, if we define $\beta_{i,j} = p^{(i)}(\alpha^j)$ for $i \in \{1, \dots, \tau\}$ and $j \in \{1, \dots, q-1\}$, then we have that $\beta_j = \sum_{i=1}^{\tau} \beta_{i,j}$.

Definition 1. Let $J = [j_1, \dots, j_n]$ be a list of $1 \leq n \leq q-1$ distinct integers in $\{0, \dots, q-1\}$. We define G_J as the $(t \times n)$ matrix

$$G_J = \left[\alpha^{j \cdot k}\right]_{k \in \{0,\dots,t-1\},\, j \in J}$$

If $n = 1$ then $J = [j]$ and we sometimes simply use G_j instead of $G_{[j]}$.

Lemma 1. *For any $t \leq n \leq q-1$ and for any $J = [j_1, \dots, j_n]$, the matrix G_J is the generator matrix of a punctured $[n, t]_q$ Reed-Solomon code. In particular:*

- *G_J has maximum rank for any $J = [j_1, \dots, j_n]$;*
- *if $n = t$ then G_J is invertible;*
- *if $n = q-1$ then G_J is a standard generator matrix (given as a Vandermonde matrix) of a $[q-1, t]_q$ Reed-Solomon code.*

Lemma 1 summarizes the properties of the matrix defined in Definition 1 and the link with Reed-Solomon codes [8]. An interested reader can refer to [9] for a comprehensive introduction to Coding Theory with a focus on Reed-Solomon codes and algebraic codes. We remark that the link with Reed-Solomon codes derives from the matrix in Definition 1.

An alternative and more general approach would be to use any $t \times n$ matrix with coefficients in $\mathbb{F}_q$. In this case Lemma 1 would become a summary of the required properties that the matrix should satisfy in order to achieve similar results. In particular, we remark that it is possible to substitute our definition with the one of Extended Generalized Reed-Solomon codes, a choice that would allow a broader set of acceptable parameters (e.g. in Definition 1 n can be at most $q+1$ instead of $q-1$). We focus however on Vandermonde matrices to explicit the parallelism with the classical version of Shamir's Secret Sharing Scheme.

Now we show that, since p has degree at most $t-1$, given any list $J \subseteq \{1, \dots, q-1\}$ of cardinality at least t, with the list of evaluations $[\beta_j]_{j \in J}$ it is possible to interpolate the

polynomial p. That is, the coefficients p_k can be reconstructed and therefore the evaluation $p(\gamma)$ in any element $\gamma \in \mathbb{F}_q$ can be computed.

Proposition 1. *Let $J = [j_1, \ldots, j_t]$ be a list of t distinct integers in $\{1, \ldots, n\}$, and let G_J be the square matrix constructed as in Definition 1. Then:*

$$(p_0, \ldots, p_{t-1}) = (\beta_{j_1}, \ldots, \beta_{j_t}) \cdot G_J^{-1}.$$

Proof. For any $j \in \{1, \ldots, q\}$ we have that $\beta_j = p(\alpha^j) = \sum_{k=0}^{t-1} p_k \cdot (\alpha^j)^k = (p_0, \ldots, p_{t-1}) G_j$, thus:

$$(p_0, \ldots, p_{t-1}) \cdot G_J = (\beta_{j_1}, \ldots, \beta_{j_t}). \tag{2.1}$$

By Lemma 1, since J has cardinality t, then G_J is invertible, so we can multiply both sides of Equation (2.1) by G_J^{-1} and conclude our proof. □

Proposition 2. *Let h be any integer in $\{1, \ldots, n\}$, let $J = [j_1, \ldots, j_t]$ be a list of t distinct integers in $\{1, \ldots, n\}$, and let e_ℓ be the ℓ-th element of the standard basis of $(\mathbb{F}_q)^t$. Then:*

$$\beta_h = \sum_{\ell=1}^{t} f(\beta_{j_\ell}, h, J, \ell),$$

where for any $\ell \in \{1, \ldots, t\}$ we define the function f as:

$$f(x, h, J, \ell) = x \cdot e_\ell G_J^{-1} G_h. \tag{2.2}$$

Proof. Observe that $e_\ell \cdot G_J^{-1}$ is the ℓ-th row of G_J^{-1}. By linearity, from Proposition 1 we have:

$$(p_0, \ldots, p_{t-1}) = \sum_{\ell=1}^{t} \beta_{j_\ell} e_\ell \cdot G_J^{-1}.$$

So:

$$\sum_{\ell=1}^{t} f(\beta_{j_\ell}, h, J, \ell) = \sum_{\ell=1}^{t} \beta_{j_\ell} e_\ell G_J^{-1} G_h = (p_0, \ldots, p_{t-1}) G_h = \beta_h,$$

as shown in the proof of Proposition 1. □

An interesting consequence of Proposition 2 is that t distinct shares are sufficient to compute any other share. However, observe that it is possible to obtain β_{j_ℓ} from $f(\beta_{j_\ell}, h, J, \ell)$, since both G_J and G_h can be easily computed even without knowing anything about the polynomials. This means that Proposition 2 should not be used directly to distribute new shares of a secret, in order to preserve the privacy of the old shares.

A simple workaround is to split these secret values. Let $b_{h,J,\ell,k}$ be chosen at random in $\mathbb{F}_q$ for $k \in \{1, \ldots, t\} \setminus \{\ell\}$, and set $b_{h,J,\ell,\ell} = f(\beta_{j_\ell}, h, J, \ell) - \sum_{k=1, k\neq\ell}^{t} b_{h,J,\ell,k}$. If we define $b_{h,J,k} = \sum_{\ell=1}^{t} b_{h,J,\ell,k}$, then we have that:

$$\sum_{k=1}^{t} b_{h,J,k} = \sum_{k=1}^{t} \left(\sum_{\ell=1}^{t} b_{h,J,\ell,k} \right) = \sum_{\ell=1}^{t} \left(\sum_{k=1}^{t} b_{h,J,\ell,k} \right) = \sum_{\ell=1}^{t} f(\beta_{j_\ell}, h, J, \ell) = \beta_h \tag{2.3}$$

Note that the random values are completely canceled out only when summing all the $b_{h,J,k}$, this means that the values β_{j_ℓ} remain hidden, so this is a safe way to generate new shares.

2.3. **Commitments.** A commitment scheme [3] is composed by two algorithms:

- $\mathrm{Com}(m, r)$**:** which given the message m to commit and some random value r (sometimes we will omit this randomness in our notation) outputs the commitment `KGC` and the decommitment `KGD`.
- $\mathrm{Ver}(\texttt{KGC}, \texttt{KGD})$**:** which given a commitment and its decommitment outputs the committed message m if the verification succeeds, $\perp$ otherwise.

A commitment scheme must have the following two properties:

- **Binding:** given $\texttt{KGC}$, it is infeasible to find values $m' \neq m$ and $\texttt{KGD}$, $\texttt{KGD}'$ such that $\mathrm{Ver}(\texttt{KGC}, \texttt{KGD}) = m$ and $\mathrm{Ver}(\texttt{KGC}, \texttt{KGD}') = m'$. We say that the commitment is *perfectly binding* if the hiding property holds even if the adversary has unbounded computational power.
- **Hiding:** Let $[\texttt{KGC}_1, \texttt{KGD}_1] = \mathrm{Com}(m_1, r_1)$ and $[\texttt{KGC}_2, \texttt{KGD}_2] = \mathrm{Com}(m_2, r_2)$ with $m_1 \neq m_2$, then it is infeasible for an attacker having only $\texttt{KGC}_1$, $\texttt{KGC}_2$, m_1 and m_2 to distinguish which $\texttt{KGC}_i$ corresponds to which m_i. We say that the commitment is *perfectly hiding* if the hiding property holds even if the adversary has unbounded computational power.

Notice that perfect hiding and perfect binding are mutually exclusive properties, in fact in a perfectly binding commitment $\texttt{KGC}$ can be decommitted in at most one way, so a computationally unbounded adversary can violate the hiding property via a brute-force search.

For our Extensible Decentralized Verifiable Secret Sharing Scheme, described in Section 3, we need a homomorphic commitment, that is a commitment HCom for which the following properties hold for all $m_0, m_1, z_0, z_1, \gamma \in \mathbb{F}_q$:

$$\mathrm{HCom}(m_0; z_0) \cdot \mathrm{HCom}(m_1; z_1) = \mathrm{HCom}(m_0 + m_1; z_0 + z_1),$$
$$\mathrm{HCom}(m_0; z_0)^\gamma = \mathrm{HCom}(\gamma \cdot m_0; \gamma \cdot z_0).$$

The Pedersen commitment [6], based on the difficulty of the discrete logarithm, is a perfectly hiding homomorphic commitment scheme which works as follows:

Setup: let $\mathbb{G}$ be a group of prime order q where the DLOG problem is hard (for the binding property to hold), and g, h be random generators of $\mathbb{G}$, then the message space of the commitment scheme is $\mathbb{Z}_q$, the randomizer space is $\mathbb{Z}_q$ and the commitment space is $\mathbb{G}$;

Commitment: to commit to $m \in \mathbb{Z}_q$ using the randomizer $z \in \mathbb{Z}_q$, the committer computes $C = \mathrm{HCom}(m, z) = g^m \cdot h^z$;

Verification: the decommitment is the pair (m, z), and $\mathrm{Ver}(C, m, z)$ simply outputs m if $C = g^m \cdot h^z$, $\perp$ otherwise.

3. Extensible Decentralized Verifiable Secret Sharing Protocol

In this section we will give a brief description of our decentralized variant of the Verifiable Secret Sharing Scheme by Pedersen [6], which includes the feature of adding new users.

Let $P_1, \ldots, P_n$ be n parties participating in the secret sharing scheme, and let $t \leq n$ be the chosen threshold.

We assume that q is big enough that, given n polynomials of degree d sampled uniformly at random, the probability of their sum to be of degree $d' < d$ is negligible. Finally, we use a homomorphic commitment HCom as defined in Section 2.3.

3.1. Secret Generation.

The distributed secret generation algorithm is carried out by the first $\tau \leq n$ parties $\{P_1, \ldots, P_\tau\}$, and proceeds as follows:

(1) Each P_i for $i \in \{1, \ldots, \tau\}$ generates a secret polynomial $p^{(i)} \in \mathbb{F}_q[x]$ of degree $t-1$, by sampling the coefficients $p_k^{(i)}$ uniformly at random in $\mathbb{F}_q$.
(2) The constant term p_0 of the summation polynomial p (see Section 2.2) is implicitly defined as the secret to be shared. Note that no single party P_i for any i knows this secret.
(3) Each P_i samples another random polynomial $z^{(i)} \in \mathbb{F}_q[x]$ of degree $t-1$, and uses its coefficients to compute and publish the commitments to the coefficients of their secret polynomial $p^{(i)}$: $C_{0,i,k} = \mathrm{HCom}\left(p_k^{(i)}; z_k^{(i)}\right)$.

(4) After having received every single commitment $C_{0,j,k}$, for $j \in \{1,\dots,\tau\}$ and $k \in \{0,\dots,t-1\}$, each P_i sends to each P_j the polynomial evaluations $\beta_{i,j} = p^{(i)}(\alpha^j)$ and $\gamma_{i,j} = z^{(i)}(\alpha^j)$.
(5) Each P_i for $i \in \{1,\dots,\tau\}$ sends $(\beta_{i,j}, \gamma_{i,j})$ also to every party P_j for $j \in \{\tau+1,\dots,n\}$.
(6) Exploiting the homomorphic properties of the commitment, each P_i for $i \in \{1,\dots,n\}$ checks the values received against the published commitments:

$$\mathrm{HCom}(\beta_{j,i};\gamma_{j,i}) \stackrel{?}{=} \prod_{k=0}^{t-1} (C_{0,j,k})^{(\alpha^i)^k}, \tag{3.1}$$

for $j \in \{1,\dots,\tau\}$.
(7) If all of these checks pass, each P_i sets its share of the newly generated secret as $\beta_i = \sum_{j=1}^{\tau} \beta_{j,i}$, and saves the checking value $\gamma_i = \sum_{j=1}^{\tau} \gamma_{j,i}$.

3.2. **Secret Reconstruction.** If $J \subseteq \{1,\dots,q\}$ is a list of t distinct indexes, then with the vector of shares $(\beta_j)_{j\in J}$ it is possible to reconstruct the secret p_0 as follows:

$$p_0 = (\beta_j)_{j\in J} \cdot G_J^{-1} \cdot e_1^T,$$

which is a direct consequence ofProposition 1. Let $\ell \in \{1,\dots,t\}$ be the position of j inside the list J, note that the Shamir β_j can be converted into an additive share ω_j:

$$\begin{aligned} \omega_j &= \beta_j e_\ell \cdot G_J^{-1} \cdot e_1^T \\ p_0 &= \sum_{j\in J} \omega_j \end{aligned} \tag{3.2}$$

3.3. **Addition of New Parties.** Let $J = \{j_1,\dots,j_t\} \subseteq \{1,\dots,n\}$ be a set of cardinality t. The parties $\{P_i\}_{i\in J}$ can collaborate to add the new party P_{n+1} (i.e. generate its share β_{n+1}) with the following algorithm:

(1) Each P_{j_ℓ} for $\ell \in \{1,\dots,t\}$ chooses uniformly at random $b_{n+1,J,\ell,k}, z_{n+1,J,\ell,k} \in \mathbb{F}_q$ for $k \in \{1,\dots,t\} \setminus \{\ell\}$, and sets $b_{n+1,J,\ell,\ell} = f(\beta_{j_\ell}, n+1, J, \ell) - \sum_{k=1,k\neq\ell}^{t} b_{n+1,J,\ell,k}$, $z_{n+1,J,\ell,\ell} = f(\gamma_{j_\ell}, n+1, J, \ell) - \sum_{k=1,k\neq\ell}^{t} z_{n+1,J,\ell,k}$, where $f(x, n+1, J, \ell)$ is defined as in Equation (2.2).
(2) Each P_{j_ℓ} publishes the commitments $C_{n+1,J,\ell,k} = \mathrm{HCom}\left(b_{n+1,J,\ell,k}; z_{n+1,J,\ell,k}\right)$ for $k \in \{1,\dots,t\}$.
(3) After having received every single commitment $C_{n+1,J,\ell,k}$, for $\ell, k \in \{1,\dots,t\}$, each P_{j_ℓ} checks the coherence of these commitments with the ones published during the generation phase:

$$\prod_{k=1}^{t} C_{n+1,J,\ell,k} \stackrel{?}{=} \left(\prod_{k=0}^{t-1} \left(\prod_{j=1}^{\tau} C_{0,j,k} \right)^{(\alpha^\ell)^k} \right)^{e_\ell G_J^{-1} G_{n+1}}, \tag{3.3}$$

for $\ell \in \{1,\dots,t\}$ (G_J and G_{n+1} are defined as in Definition 1), and:

$$\prod_{k=1}^{t} \prod_{\ell=1}^{t} C_{n+1,J,\ell,k} \stackrel{?}{=} \prod_{k=0}^{t-1} \left(\prod_{j=1}^{\tau} C_{0,j,k} \right)^{(\alpha^{n+1})^k}. \tag{3.4}$$

If everything checks out, P_{j_ℓ} sends to each P_{j_k} the values $b_{n+1,J,\ell,k}$ and $z_{n+1,J,\ell,k}$, for $\ell, k \in \{1,\dots,t\}$.

(4) Each P_{j_ℓ} checks the consistency of the data received and the committed values:

$$\mathrm{HCom}\left(b_{n+1,J,k,\ell}; z_{n+1,J,k,\ell}\right) \stackrel{?}{=} C_{n+1,J,k,\ell},$$

for $k \in \{1, \ldots, t\}$, then sets $b_{n+1,J,\ell} = \sum_{k=1}^{t} b_{n+1,J,k,\ell}$, $z_{n+1,J,\ell} = \sum_{k=1}^{t} z_{n+1,J,k,\ell}$, and sends them to P_{n+1}.

(5) P_{n+1} retrieves its share as: $\beta_{n+1} = \sum_{\ell=1}^{t} b_{n+1,J,\ell}$, and the checking value as: $\gamma_{n+1} = \sum_{\ell=1}^{t} z_{n+1,J,\ell}$. Then it checks their consistency with the commitments by verifying:

$$\mathrm{HCom}\left(b_{n+1,J,\ell}; z_{n+1,J,\ell}\right) \stackrel{?}{=} \prod_{k=1}^{t} C_{n+1,J,k,\ell}, \tag{3.5}$$

for $\ell \in \{1, \ldots, t\}$, and Equations (3.3) and (3.4).

Note that at the end of the procedure, P_{n+1} has its own secret values just like the other parties, so it can participate in the secret reconstruction or the addition of further parties.

3.4. **Security of the Secret Sharing.** For the detailed security of the initial Secret Generation protocol see [6].

To prove the security of the Addition of New Parties we need to show that an adversary controlling at most $t-1$ participants is not able to learn anything about the secret of the other parties. Initially we suppose that the adversary does not control P_{n+1}, but only $t-1$ out of the t parties which perform the protocol to add P_{n+1}. WLOG we can suppose that these parties are $P_1, \ldots, P_t$ and that the adversary controls $P_2, \ldots, P_t$.

We can notice that in step 1 there is a (t, t) additive secret sharing of $f(\beta_1, n+1, J, 1)$, with dealer P_1, verified with a homomorphic commitment. This is secure and does not leak any information about β_1 or β_{n+1}.

The following steps do not require any additional computation or communication involving the secret $b_{n+1,J,1,1}$, so the security is trivial.

Now we need to deal with the case of the adversary controlling P_{n+1} and $t-2$ among $P_1, \ldots, P_t$. WLOG we can suppose that the adversary controls $P_3, \ldots, P_t$.

The same considerations as before hold for step 1. However, now the adversary is also able to learn $b_{n+1,J,1}$ and $b_{n+1,J,2}$ after step 5. Since in the computation of $b_{n+1,J,1}$ and $b_{n+1,J,2}$ there are two unknown and uniformly distributed addends, the adversary is not able to learn anything more.

4. Threshold Schnorr Signature

In this section we describe a possible use case of our Secret Sharing Scheme: a (t, n)-threshold variant of Schnorr's digital signature algorithm with offline participants. For our construction we need a group $\mathbb{G}$ of prime order q with generator g where the DLOG problem is assumed to be hard. Note that this means that the field $\mathbb{F}_q$ is isomorphic to the ring $\mathbb{Z}_q$, so we will write $\mathbb{Z}_q$ from now on. Moreover the hardness of DLOG implies that the size of q is exponential in the security parameter, thus any practical application necessary has a number of users $n \ll q$. Finally, we require that at least t users are online for the setup, in the following we suppose there are exactly t online parties in the key generation phase, namely $P_1, \ldots, P_t$.

The protocol is dividend into four algorithms:

(1) **Setup Phase** (4.1): in this phase all the players interact to set some common parameters. Note that in a practical implementation this phase can be performed ahead of time without any real communication, because these parameters are usually fixed (e.g. for Bitcoin applications which have to use secp256k1 and SHA-256).

(2) **Key-Generation** (4.2): this phase is performed by parties $P_1, \ldots, P_t$ to create the public key for the signature scheme and the private shares for themselves.

(3) **Signature Algorithm** (4.3): this phase is performed whenever any group of t parties wants to produce a signature.
(4) **Participant Addition**(4.4) performed by any group of at least t parties to create new shares for a new player.

From now on "P_i does something" means that all the parties involved in that phase perform the specified action.

4.1. **Setup Phase.** The aim of this phase is to make the starting parties $P_1, \dots, P_K$ to agree on all the parameters required in the protocol.

Player $1, \dots, t$	
Input:	
Private Output:	
Public Output:	$\mathbb{G}, g, q, H, \alpha$

$P_1, \dots, P_t$ have to establish a group $\mathbb{G}$ of prime order q with generator g in which the discrete logarithm problem is considered to be hard, a secure hash function H whose outputs can be interpreted as elements of $\mathbb{Z}_q$, and a primitive element α of $\mathbb{Z}_q$. Lastly the agree on a common instance of a commitment scheme Com.

4.2. **Key generation.** In this phase, the starting parties $P_1, \dots, P_\tau$ produce a common public key $\mathcal{A}$ and each obtains a share of the corresponding private key.

Player i	
Input:	
Private Output:	β_i
Public Output:	$\mathcal{A}$

(1) Secret key generation and communication:
  (a) P_i randomly chooses $a_i \in \mathbb{Z}_q$ and sets $\mathcal{A}_i = g^{a_i}$;
  (b) P_i randomly chooses a polynomial $p^{(i)}$ of degree $t-1$ such that $p^{(i)}(0) = a_i$.
  (c) P_i computes $[\texttt{KGC}_i, \texttt{KGD}_i] = \mathrm{Com}(\mathcal{A}_i)$;
  (d) P_i publishes $\texttt{KGC}_i$
  (e) P_i publishes $\texttt{KGD}_i$
  (f) P_i gets $\mathcal{A}_j$ for $1 \le j \le t, i \ne j$.
(2) Shards verification and private key computation:
  (a) P_i computes $\beta_{i,j} = p^{(i)}(\alpha^j)$ and sends it to player P_j;
  (b) P_i checks the integrity and consistency of the shards $\beta_{j,i}$, as per Section 3;
  (c) P_i proves in ZK the knowledge of a_i using Schnorr's protocol.
(3) P_i compute its private key $\beta_i = \sum_{j=1}^{t} \beta_{i,j}$.
(4) The public key is $\mathcal{A} = \prod_{i=1}^{\tau} \mathcal{A}_i$. Implicitly we set $\sum_{i=1}^{\tau} = a$.

4.3. **Signature Algorithm.** This algorithm is used when a set J of at least t players agrees to sign a message M. WLOG we suppose that $J = \{j_1, \dots, j_t\}$.

The parameters involved are:

Player j_i	
Input:	$M, \omega_{j_i}, \mathcal{A}$
Public Output:	(s, e)

The protocol proceeds as follows.

(1) Generation of r:
  (a) P_{j_i} randomly chooses $k_i \in \mathbb{Z}_q$;
  (b) P_{j_i} computes $r_i = g^{k_i}$;

(c) P_{j_i} computes $[\texttt{KGC}_i, \texttt{KGD}_i] = \text{Com}(r_i)$ and sends $\texttt{KGC}_i$;
(d) once every $\texttt{KGC}_j$ for $j \in J$ has been received, P_{j_i} sends $\texttt{KGD}_i$;
(e) P_{j_i} computes $r_\ell = \text{Ver}([\texttt{KGC}_\ell, \texttt{KGD}_\ell])$ for each $\ell = 1, \dots, t$;
(f) P_{j_i} computes $r = \prod_{\ell=1}^{t} r_\ell$.

(2) Generation of s:
(a) P_{j_i} converts its Shamir share β_{j_i} to an additive share ω_{j_i} such that $\sum_{j \in J} \omega_j = a$, as in Equation (3.2);
(b) P_{j_i} computes $e = H(r||M)$ and $s_i = k_i - \omega_{j_i} e$;
(c) P_{j_i} computes $[\texttt{KGC}'_i, \texttt{KGD}'_i] = \text{Com}(s_i)$ and sends $\texttt{KGC}'_i$;
(d) once every $\texttt{KGC}'_j$ for $j \in J$ has been received, P_{j_i} sends $\texttt{KGD}'_i$;
(e) P_{j_i} computes $s_\ell = \text{Ver}([\texttt{KGC}'_\ell, \texttt{KGD}'_\ell])$ for each $\ell = 1, \dots, t$;
(f) P_{j_i} computes $s = \sum_{\ell=1}^{t} s_\ell$.

(3) P_{j_i} computes $r_v = g^s \mathcal{A}^e$ and checks that $H(r_v||M) = e$.

The output signature is (s, e). If a check fails, the protocol aborts.

4.4. **Participant Addition.** This protocol allows any set J of at least t users to add a new user P_m to the protocol. After the protocol P_m will have the same powers (i.e. can sign and add new users) of the other users.

The parameters involved are (for sake of simplicity $J = \{j_1, \dots, j_t\}$:

Player j_i	
Input:	$\beta_{j_i}, \mathcal{A}$
Private Output:	

Player m	
Input:	$\mathsf{sk}_m, \mathsf{pk}_m$
Private Output:	β_m

The protocol works as follows:

(1) P_m publishes its public key pk_m that $P_{j_1}, \dots, P_{j_t}$ will use to communicate with it;
(2) Additive Secret Sharing
(a) P_{j_i} transforms its Shamir share β_{j_i} in an additive share ω_i, as in eq. (3.2);
(b) P_{j_i} performs an additive secret sharing of $\omega_i = \sum_l \omega_{i,l}$;
(c) P_{j_i} sends $\omega_{i,l}$ to P_{j_l};
(d) P_{j_i} publishes $g^{\omega_{i,l}}$ for each l. All the values are stored in a public matrix Ω.
(e) P_{j_i} verifies to have received correct shares as explained in Section 3.3
(3) Share distribution
(a) P_{j_i} computes $\bar{\omega_i} = \sum_l \omega_{l,i}$;
(b) P_{j_i} encrypts $\omega_{l,i}$ with pk and sends it to P_m;
(4) Key reconstruction and verification
(a) P_m computes its private key $\beta_m = \sum_i \bar{\omega_i}$
(b) P_m verifies to have received correct shares as explained in Section 3.3

Observation 1. The symmetric encryption algorithm used by P_m should be IND-CPA secure, in order to maintain the security of the protocol. In the security proof we will suppose that the algorithm is indeed secure and the key generation protocol generating $\mathsf{pk}_m, \mathsf{sk}_m$ was ran correctly.

5. Security Proof

In this section we discuss the security of the scheme in terms of the unforgeability properties defined below.

Definition 2 (Unforgeability)**.** A (k, n)-threshold signature scheme is unforgeable if no malicious adversary who corrupts at most $k - 1$ players can produce the signature on a new message m with non-negligible probability, given the view of the threshold sign on input messages $m_1, \dots, m_t$ (adaptively chosen by the adversary), as well as the signatures on those messages.

The unforgeability of our protocol is formally stated in the following theorem:

Theorem 1. *Assuming that:*

- *the Schnorr signature scheme instantiated on the group $\mathbb{G}$ of prime order q with the hash function H is unforgeable;*
- $\mathrm{Com}, \mathrm{Ver}$ *is a non-malleable commitment scheme;*
- *the Decisional Diffie-Hellman Assumption holds;*

our threshold protocol is unforgeable.

In Section 5.4 we will prove the theorem by showing that if there is an adversary $\mathcal{A}$ able to forge a signature for the threshold scheme with non negligible probability $\epsilon > \lambda^{-c}$ with λ a polynomial and $c > 0$, then it is possible to build a forger $\mathcal{F}$ that forges a signature for the centralized Schnorr scheme also with non negligible probability. The simulation works by having an oracle that feeds inputs for the centralized scheme to $\mathcal{F}$, our goal is to respond by generating a signature exploiting $\mathcal{A}$. First, it has to simulate the key generation protocol in order to match the key received from the oracle, then it can proceed with the signature part. The core of this setup is that if $\mathcal{A}$ is able to crack our protocol, $\mathcal{F}$ will take advantage of that and will also create a forgery for the centralized version of the oracle.

Following the definition of unforgeability, $\mathcal{A}$ will control $t-1$ players while $\mathcal{F}$ controls the remaining ones. It is important to notice that this $t-1$ players that the adversary control can either be present from the start, during the key generation, or be added later, with the participant. In the following we will suppose that every malicious party is involved in the key generation and that $t = \tau$, i.e. only t parties are involved in it. The proof where the malicious parties are distributed differently is analogous and could be easily deduce.

The adversary interacts by first participating in the key generation part to generate a public key $\mathcal{A}$, then starts requesting signatures on some messages m_i. Here it can either take part in the process or not. Eventually $\mathcal{A}$ outputs a message $m \neq m_i \ \forall i$ and its valid signature with probability at least ϵ, where this is taken over the random tapes of the adversary and the honest player, respectively $\tau_{\mathcal{A}}$ and τ_i. So we can write that

$$\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}(\mathcal{A}(\tau_{\mathcal{A}})_{P_i(\tau_i)} = \text{forgery}) \geq \epsilon, \tag{5.1}$$

where $\mathcal{A}(\tau_{\mathcal{A}})_{P_i(\tau_i)}$ is the output of $\mathcal{A}$ at the end of this process and $\mathbb{P}_{\tau_i,\tau_{\mathcal{A}}}$ denotes that the probability is taken over the random tape τ_i and the adversary tape $\tau_{\mathcal{A}}$.

We say that a random tape is *good* if

$$\mathbb{P}_{\tau_i}(\mathcal{A}(\tau_{\mathcal{A}})_{P_i(\tau_i)} = \text{forgery}) \geq \frac{\epsilon}{2}. \tag{5.2}$$

We recall the following useful lemma, stated and proved in [2].

Lemma 2. *If $\tau_{\mathcal{A}}$ is chosen uniformly at random, the probability that $\tau_{\mathcal{A}}$ is good is at least $\frac{\epsilon}{2}$.*

5.1. **Key generation simulation.** Now we see into details how the key generation phase is simulated. $\mathcal{F}$ receives from the challenger the public key $\mathcal{A}_c$ for the centralized Schnorr protocol. The simulation proceeds as follows:

(1) $\mathcal{F}$ selects a random values $\beta_{1,j} = p^{(i)}(j)$ for $j = 2, ..., t$ and a_1.
(2) $\mathcal{F}$ computes the commitment $[\mathtt{KGC}_1, \mathtt{KGD}_1] = \mathrm{Com}(\mathcal{A}_1)$;
(3) $\mathcal{F}$ sends $\mathtt{KGC}_1$, then, after receiving $\mathtt{KGC}_j$, it sends $\mathtt{KGD}_1$;
(4) $\mathcal{F}$ gets $(\mathcal{A}_j) = \mathrm{Ver}(\mathtt{KGC}_j, \mathtt{KGD}_j)$ for all j;
(5) Now $\mathcal{F}$ knows all the parameters needed in the computation of $\mathcal{A}$, so it rewinds $\mathcal{A}$ to step 3, aiming to get $\mathcal{A} = \mathcal{A}_c$;
(6) $\mathcal{F}$ computes $\hat{\mathcal{A}} = \frac{\mathcal{A}_c}{\mathcal{A}_2\mathcal{A}_3}$, computes the commitment $[\hat{\mathtt{KGC}}_1, \hat{\mathtt{KGD}}_1] = \mathrm{Com}(\hat{\mathcal{A}})$, and sends it to $\mathcal{A}$, so that it will receive $\hat{\mathcal{A}}$ as $\mathcal{A}_1$ which leads to $\mathcal{A} = \mathcal{A}_c$. Notice that $\mathcal{F}$ does not know the discrete logarithm $\hat{a}$ of $\hat{\mathcal{A}}$.

(7) $\mathcal{F}$ sends $\beta_{1,j}$ to P_j.
(8) $\mathcal{F}$ simulates a fake verification protocol with (see e.g. [6]) since it cannot compute a polynomial $\hat{p}^{(1)}$ such that $\hat{p}^{(1)}(j) = \beta_{1,j}$ and $\hat{p}^{(1)}(j) = \hat{a}$.
(9) $\mathcal{F}$ participates in the ZK proofs rewinding $\mathcal{A}$ and selecting appropriate challenges in order to extract the secret key of each party controlled by $\mathcal{A}$.

Note that at the end of the protocol, $\mathcal{F}$ does not know its private key β_1, but $\mathcal{F}$ will still be able to complete correctly the signing part by querying the oracle.

The proof of the correctness of the simulation is stated in the following lemmas. The proofs are trivial and use the same argument of the one presented in [2].

Lemma 3. *If the Decisional Diffie-Hellman assumption holds, then the simulation terminates in expected polynomial time and is indistinguishable from the real protocosl.*

Proof. Since $\mathcal{A}$ is running on a good random tape we know that it will correctly decommit with probability at least $\frac{\epsilon}{2}$. Therefore the rewinding is performed at most a polynomial number of times, since the expected number of iterations is $\frac{2}{\epsilon} = 2\lambda^c$. The only difference from the real protocol is that $\mathcal{F}$ does not know the discrete logarithm of $\hat{\mathcal{A}}$ and so it performs a fake verification protocol. However, this is indistinguishable from a real one since they both have the same distribution. □

Lemma 4. *For a polynomial number of inputs the simulation terminates with output $\mathcal{A}_c$ except with negligible probability.*

Proof. First we prove that if the simulation terminates correctly, then it terminates with $\mathcal{A}_c$ except with negligible probability. This is because of the non-malleability property of the commitment scheme: if $\mathcal{A}$ correctly decommits twice it must do so to the same string, no matter what P_1 decommits (except with negligible probability). Because of the construction of $\hat{\mathcal{A}}$, the output is $\mathcal{A}_c$

Now we prove that the simulation ends correctly for a polynomially large fractions of inputs. Since $\mathcal{A}$ is running on a good random tape, it decommits correctly for at least $\frac{\epsilon}{2} > \frac{1}{2\lambda^c}$ inputs. Since $\mathcal{A}_c$ is chosen at random we have that $\hat{\mathcal{A}}$ is uniformly distributed . We can conclude that for a fraction $\frac{\epsilon}{2} > \frac{1}{2\lambda^c}$ of the inputs, the protocol will terminate correctly. □

Observation 2. It is important that in step 3 the adversary sends $\texttt{KGC}_j$ and $\texttt{KGD}_j$ before $\mathcal{F}$, so that after the rewinding $\mathcal{A}$ cannot change its commitment.

If the order were inverted, $\mathcal{A}$ could also use the commitment of $\mathcal{F}$ to generate its value.

Assuming the non-malleability property, $\mathcal{A}$ does not deduce anything about the content of the commitment, but it could still use it as a seed for a random generator.

If this were to happen, $\mathcal{F}$ can guess $\hat{\mathcal{A}}$ with probability $\frac{1}{q}$ with q the size of the group, making the expected time exponential.

It is possible to swap the order in the commitment step using an equivocable commitment scheme with a secret trapdoor. In this case we only need to rewind at the decommitment step and change KCD_1 in order to match $\hat{\mathcal{A}}$.

5.2. **Security of the addition of new users.** Intuitively the security follows by the fact that $\omega_{i,l}$ are uniformly distributed and are shards of a full threshold additive secret sharing of ω_i. This ensures that $\bar{\omega}_i$ is also a full threshold additive secret sharing of $\sum_i \omega_i$.

The correctness of the algorithm is an immediate consequence of Proposition 2, as noticed in Section 3.

Moreover we can notice that the adversary is forced to act semi-honestly thanks to the verification steps explained in Section 3.4.

5.3. **Signature generation simulation.** After the the key generation and the addition of new user, $\mathcal{F}$ has to deal with the signature requests issued by $\mathcal{A}$. When $\mathcal{A}$ asks for a

signature, $\mathcal{F}$ performs a simulation while having access to the signing oracle that uses the previously created public key. In this section we will suppose that the adversary has the maximum power possible, i.e. he controls $t-1$ participant. The cases where he controls less participants can be dealt in the same way. Without loss of generality we will suppose that $\mathcal{A}$ controls $P_2, ..., P_t$.

First we note that after the key generation/participant addition, the simulator knows every secret of the adversary, since he is able to extract them during the ZKPs in the protocol.

For this reason $\mathcal{F}$ can fully predict what $\mathcal{A}$ will output and, while it does not know any secret key of P_1, it knows everything of $P_2, ..., P_t$.

The simulation proceeds as follows:

(1) $\mathcal{A}$ chooses a message m to sign;
(2) $\mathcal{F}$ queries its signing oracle for a signature for m corresponding to the public key $\mathcal{A}$ and gets (s_f, e_f);
(3) P_i randomly chooses $k_i \in \mathbb{Z}_q^*$, then computes $r_i = g^{k_i}$ and $[\mathtt{KGC}_i, \mathtt{KGD}_i] = \mathrm{Com}(r_i)$;
(4) P_i sends $\mathtt{KGC}_i$, then, after receiving $\mathtt{KGC}_j$, sends $\mathtt{KGD}_i$ and gets $r_i = \mathrm{Ver}([\mathtt{KGC}_i, \mathtt{KGD}_i])$;
(5) $\mathcal{F}$ rewinds $\mathcal{A}$ to step 4;
(6) $\mathcal{F}$ computes $\hat{r}_1 = \frac{r_f}{\prod_{j=2}^{t} r_j}$, then compute the commitment $[\hat{\mathtt{KGC}}_1, \hat{\mathtt{KGD}}_1] = \mathrm{Com}(\hat{r}_1)$ and sends $\hat{\mathtt{KGC}}_1$ to $\mathcal{A}$ so it receives $\hat{r}_1$ as r_1 which leads to $r = r_f$;
(7) P_i computes $r = \prod_{i=1}^{t} r_i$, $e = H(r||m)$, and $s_i = k_i - \omega_i e$ ($\mathcal{F}$ picks s_1 at random);
(8) P_i computes $[\mathtt{KGC}'_i, \mathtt{KGD}'_i] = \mathrm{Com}(s_i)$, then sends $\mathtt{KGC}'_i$;
(9) P_i sends $\mathtt{KGD}'_i$ and gets $s_i = \mathrm{Ver}([\mathtt{KGC}'_i, \mathtt{KGD}'_i])$;
(10) $\mathcal{F}$ computes $r'_j = g^{s_j} \cdot g^{-e\omega_j}$ for each $j = 2, ..., t$, then if $r_2 = r'_2$ it rewinds $\mathcal{A}$ to step 8, otherwise it sends s_1 and aborts;
(11) $\mathcal{F}$ computes $\hat{s}_1 = s_f - \sum_{j=2}^{t} s_j$, then computes the commitment $[\hat{\mathtt{KGC}}'_1, \hat{\mathtt{KGD}}'_1] = \mathrm{Com}(\hat{s}_1)$ and sends $\hat{\mathtt{KGC}}'_1$ to $\mathcal{A}$ so it receives $\hat{s}_1$ as s_1 which leads to $s = s_f$;
(12) P_i computes $s = \sum_{j=1}^{t} s_j$ and $r_v = g^s \mathcal{A}^e$, then checks that $H(r_v||m) = e$. If a check fails the protocol aborts, otherwise the signature is (s, e).

Lemma 5. *If* Com *is a secure non-malleable commitment scheme, the protocol above is a perfect simulation of the centralized one and terminates correctly with output* (s_f, e_f).

Proof. The simulation is identical to the real protocol except that here $\mathcal{F}$ does not know its secret shards. Nevertheless it is still able to retrieve the correct value from $\mathcal{A}$ by rewinding it. As above, if the protocol terminates, by construction it will terminate with output (s_f, e_f). If $\mathcal{A}$ is dishonest or refuses to decommit some values, the protocol aborts. Note that the check of step 10 is introduced to preserve any abort that the adversary may cause by sending an invalid s_1. □

5.4. **Proof of the unforgeability property.** Now we are able to prove Theorem 1:

Proof. Let $Q < \lambda^c$ be the maximum number of signature queries that the adversary makes. As we previously proved, our simulator produces a view of the protocol that the adversary cannot distinguish from the real one, therefore $\mathcal{A}$ will produce a forgery with the same probability as in a real execution. Then the probability of success of our forger $\mathcal{F}$ is $\frac{\epsilon^3}{8}$ which is the product of the probability of the following independent events:

(1) choosing a good random tape for $\mathcal{A}$, whose probability is at least $\frac{\epsilon}{2}$ as per Lemma 2;
(2) getting a good public key, whose probability is at least $\frac{\epsilon}{2}$ as shown in Lemma 3 and 4;
(3) $\mathcal{A}$ successfully produces a forgery, whose probability is again $\frac{\epsilon}{2}$ (5.2).

Under the assumption on the security of the Schnorr signature scheme, the probability of success of $\mathcal{F}$ must be negligible, which implies that ϵ must be negligible too, contradicting the hypothesis that $\mathcal{A}$ has a non-negligible probability of forging a signature for the scheme. □

References

[1] M. Battagliola, A. Galli, R. Longo, and A. Meneghetti. "A Provably-Unforgeable Threshold Schnorr Signature With an Offline Recovery Party". In: *Proceedings http://ceur-ws. org ISSN* 1613 (2022), p. 0073.

[2] M. Battagliola, R. Longo, A. Meneghetti, and M. Sala. "Threshold ECDSA with an Offline Recovery Party". In: *Mediterranean Journal of Mathematics* 19.1 (2022), pp. 1–29.

[3] G. Brassard, D. Chaum, and C. Crépeau. "Minimum disclosure proofs of knowledge". In: *Journal of computer and system sciences* 37.2 (1988), pp. 156–189.

[4] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1769–1787. ISBN: 9781450370899. DOI: 10.1145/3372297.3423367. URL: https://doi.org/10.1145/3372297.3423367.

[5] R. Gennaro and S. Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1179–1194. ISBN: 9781450356930. DOI: 10.1145/3243734.3243859. URL: https://doi.org/10.1145/3243734.3243859.

[6] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology — CRYPTO '91*. Ed. by J. Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140.

[7] T. P. Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing". In: *Annual international cryptology conference*. Springer. 1991, pp. 129–140.

[8] I. S. Reed and G. Solomon. "Polynomial codes over certain finite fields". In: *Journal of the society for industrial and applied mathematics* 8.2 (1960), pp. 300–304.

[9] R. M. Roth. *Introduction to coding theory*. Vol. 47. 18-19. IET, 2006, p. 4.

(M. Battagliola) University of Trento, Department of Mathematics, michele.battagliola@unitn.it

(R. Longo) University of Trento, Department of Mathematics, riccardolongomath@gmail.com

(A. Meneghetti) University of Trento, Department of Mathematics, alessio.meneghetti@unitn.it