



## Research Article

Annalisa Cimatti\*, Francesco De Sclavis, Giuseppe Galano, Sara Giammusso, Michela Iezzi, Antonio Muci, Matteo Nardelli, and Marco Pedicini

# Dynamic-FROST: Schnorr threshold signatures with a flexible committee

<https://doi.org/10.1515/jmc-2024-0045>

received November 29, 2024; accepted January 21, 2025

**Abstract:** Threshold signatures enable any subgroup of predefined cardinality  $t$  out of a committee of  $n$  participants to generate a valid, aggregated signature. Although several  $(t, n)$ -threshold signature schemes exist, most of them assume that the threshold  $t$  and the set of participants do not change over time. Practical applications of threshold signatures might benefit from the possibility of updating the threshold or the committee of participants. Examples of such applications are consensus algorithms and blockchain wallets. In this article, we present Dynamic-FROST (D-FROST) that combines FROST, a Schnorr threshold signature scheme, with CHURP, a dynamic proactive secret sharing scheme. The resulting protocol is the first Schnorr threshold signature scheme that accommodates changes in both the committee and the threshold value without relying on a trusted third party. Besides detailing the protocol, we present a proof of its security: as the original signing scheme, D-FROST preserves the property of existential unforgeability under chosen-message attack.

**Keywords:** proactive secret sharing, threshold signatures, decentralization, FROST, CHURP

**MSC 2020:** 94A60, 11T71, 68P25

## 1 Introduction

A threshold signature allows any subgroup of  $t$  signers out of  $n$  participants to generate a signature which cannot be forged by any subgroup with fewer than  $t$  members. The signature is generated collaboratively using a *single* group public key, which is the same size of a single-party public key. Threshold signature schemes offer scalability and confidentiality: the length of the aggregated signature remains constant and does not increase with  $t$  or  $n$ , and the identity of actual signers remains confidential, as it is not disclosed by the aggregated signature.

---

\* **Corresponding author:** Annalisa Cimatti, Mathematics and Physics Department, Roma Tre University, Rome, Italy,  
e-mail: annalisa.cimatti@uniroma3.it

**Francesco De Sclavis:** Directorate General for Information Technology, Bank of Italy, Rome, Italy,  
e-mail: francesco.desclavis@bancaditalia.it

**Giuseppe Galano:** Directorate General for Information Technology, Bank of Italy, Rome, Italy; Department of Computer Science,  
University of Pisa, Pisa, Italy, e-mail: giuseppe.galano2@bancaditalia.it

**Sara Giammusso:** Directorate General for Information Technology, Bank of Italy, Rome, Italy, e-mail: sara.giammusso@bancaditalia.it

**Michela Iezzi:** Directorate General for Information Technology, Bank of Italy, Rome, Italy, e-mail: michela.iezzi@bancaditalia.it

**Antonio Muci:** Directorate General for Information Technology, Bank of Italy, Rome, Italy, e-mail: antonio.muci@bancaditalia.it

**Matteo Nardelli:** Directorate General for Information Technology, Bank of Italy, Rome, Italy, e-mail: matteo.nardelli@bancaditalia.it

**Marco Pedicini:** Mathematics and Physics Department, Roma Tre University, Rome, Italy, e-mail: marco.pedicini@uniroma3.it

**Schnorr threshold signatures and FROST.** Among threshold signature schemes, FROST [1] leverages the additive property of Schnorr signatures to produce a joint one that looks like a simple, single Schnorr signature. Although other schemes have been proposed [2], e.g., based on Rivest-Shamir-Adleman (RSA) or elliptic curve digital signature algorithm (CEDSA), the characteristics of Schnorr signatures facilitate more straightforward implementations; for this reason, Schnorr signatures have been recently included in the Bitcoin codebase<sup>1</sup>. Furthermore, FROST has many desirable properties for decentralized applications: it uses Perdersen’s Distributed Key Generation (DKG) algorithm and constructs signatures in such a way that no central dealer is required to generate and distribute keys or to sign; it achieves Existential Unforgeability under Chosen-Message Attack (EUF-CMA) [1]; it achieves efficient communication by reducing the protocol to just two rounds. Some variants, like ROAST (RObust Asynchronous Schnorr Threshold signatures) [3], also guarantee that the signing session eventually terminates successfully if at least  $t$  participants cooperate.

**Motivation for dynamic-FROST.** FROST signatures have a fixed committee and a fixed threshold  $t$ . For some applications, it might be interesting to allow the committee or the threshold to change. A naïve solution to this problem consists in simply generating a new group secret and distribute new shares among the updated participants. However, changing the secret is not always practical, and we offer two examples of applications where this is particularly relevant. First, advanced self-custodial cryptocurrency wallets might require a FROST-powered dynamic threshold signature that enables users to alter the set of signers, but without moving funds to a new address, i.e., without modifying the group public key through a blockchain transaction. In 2023, the Human Rights Foundation announced it would award 1 bitcoin to any mobile wallet that successfully implements such a feature<sup>2</sup>. Second, threshold signatures can be employed by a committee of validators in a permissioned blockchain to authenticate new blocks, as outlined in the study of Benedetti *et al.* [4]. In this scenario, the composition of the validators’ committee might evolve over time – due to governance adjustments, security incidents, or simply to rotate members – and thus, the set of signers or the threshold would need to be updated accordingly. In such cases, changing the group public key would require upgrading all participants’ nodes to recognize the new one; failing to do so would mean that blocks signed with the new group secret would not be considered valid by those participants who did not upgrade.

**Proactive secret sharing (PSS).** In principle, a dynamic committee or a dynamic threshold can be achieved by addressing four simple sub-problems: (i) to remove a participant, (ii) to add a participant, (iii) to decrease the threshold, or (iv) to increase the threshold. Some of the sub-cases can be tackled with different techniques while allowing the group secret to remain unchanged. For example, decreasing the threshold is essentially equivalent to having an additional share in the  $t$ -of- $n$  scheme, which is exposed to all participants; adding a participant is equivalent to jointly producing a new share, which can be obtained from previous shares, as in *repairable threshold schemes* [5]. However, composing different techniques while still being able to assess the security properties of the protocol is not an easy task. A more desirable approach would be to find a unified solution that works for all the previously mentioned sub-cases. A possible starting point is PSS, introduced by Herzberg *et al.* [6], which periodically updates the secret shares while leaving the group secret unchanged, thus reinforcing Shamir’s secret sharing [7]. The idea behind this method is very straightforward: adding a polynomial with zero constant term to the one used to generate the secrets will not change the group secret, but only the secret shares. Indeed, in Shamir’s secret sharing, the group secret is the constant term of the polynomial, while the secret shares are the values of the polynomial at various indices. PSS schemes build upon this idea, but differentiate between each other along three main dimensions. First, they can be *dynamic* when they support dynamic committees, namely, when they allow to change both the members and the cardinality of the committee. Second, they can be either *centralized* or *decentralized*, depending on whether the new shares are distributed with the aid of a central trusted dealer or not. Third, depending on the assumption about the communication channels of participants, they can be *synchronous* (if message delays are bounded), *asynchronous* (if message delays are unbounded), or *partially synchronous* (if communication channels are asynchronous until a Global Stabilization Time event, and synchronous after). As described in Section 2, we analyzed several PSS schemes and selected CHURP as our favorite candidate.

---

<sup>1</sup> Schnorr Signatures for secp256k1: <https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki>.

<sup>2</sup> Human Rights Foundation Bounties, accessed on April 07, 2024: <https://hrfbounties.org/>.

**CHURP PSS.** CHURP is a dynamic PSS (DPSS) scheme, which does not rely on a trusted dealer, works in a synchronous setting, and can be used to accommodate changes in both the committee and threshold as long as  $t - 1 < \frac{n}{2}$ . The basic idea is to generate a two-variable polynomial (instead of a one-variable one, like in Shamir's method) that has two different degrees in the two variables: the lower-degree variable is used to distribute polynomial shares (called *full shares*), which will be used to perform signatures; the higher-degree one is used to pass a set of polynomials (called *reduced shares*) to a new committee; specific points on these polynomials can be used by the new committee to generate new full shares. In such a way, both the committee and the threshold can be changed. In practice, this change is done by constructing and adding a two-variable polynomial with a zero constant term, similarly to the study of Herzberg et al. [6]. More details can be found in the study of Maram et al. [8] and in Section 3.3.

**Our contribution.** In this article, we introduce a novel protocol called *Dynamic FROST* (D-FROST), which combines FROST with CHURP [8] to accommodate dynamic committees and threshold changes in a FROST threshold signature. The idea behind CHURP is based upon a technique outlined in the study of Herzberg et al. [6], which is based on two-variable polynomials and places it far away from FROST, which uses one-variable polynomials. To combine these two approaches, we define a new scheme that provides a bridge between the two protocols and we prove its security properties. To blend FROST and CHURP together, after FROST Key Generation, we transition to a *steady state*, i.e., a state in which CHURP can be executed. This means that we generate a bivariate polynomial that returns the previously generated secret shares and the group secret at various indices. In practice, we generate a set of polynomials, whose constant terms are the secret shares, and then, we interpolate them to create a bivariate polynomial. Once we are in a steady state, CHURP is executed, and then, FROST signatures can be made with the newly generated shares. Then, periodically, at fixed intervals called *epochs*, CHURP is executed again and new FROST signatures can be performed; there is no need to repeat the key generation or the transition to a steady state. To the best of our knowledge, this is the first protocol that allows Schnorr-based threshold signatures with a dynamic committee and a dynamic threshold, without changing the group public key. We formally prove that the resulting protocol inherits both FROST's and CHURP's properties: the signature is still EUF-CMA secure, and proactivizing the shares does not reveal additional information to malicious participants.

**Article organization.** The rest of this article is organized as follows: after reviewing related works in Section 2, we outline FROST and CHURP in Section 3; the description of our D-FROST protocol can be found in Section 4, and a complete proof of its security can be found in Section 5.

## 2 Related work

### 2.1 Threshold signature schemes

As introduced in Section 1, a threshold signature scheme allows any subgroup of  $t$  signers out of  $n$  participants to generate a signature for a message  $m$ . Formally, a threshold signature scheme can be defined as follows.

**Definition 1. A threshold signature scheme**  $(G, S, C, V)$  is a tuple of four efficient algorithms:

- $G$  is a probabilistic **key generation algorithm** that is invoked as

$$(pk, pk_c, s_1, \dots, s_n) \xleftarrow{\$} G(n, t)$$

to generate a  $(t, n)$  shared key. It outputs a public key  $pk$ , a **combiner public key**  $pk_c$ , and  $n$  **signing key shares**,  $s_1, \dots, s_n$ .

- $S$  is a (possibly) probabilistic **signing algorithm** that is invoked as

$$\sigma'_i \leftarrow S(s_i, m),$$

where  $s_i$  is one of the key shares generated by  $G$ ,  $m$  is a message, and  $\sigma'_i$  is a signature share for  $m$  using  $s_i$ .

- $C$  is a deterministic **combiner algorithm** that is invoked as

$$\sigma \leftarrow C(pk_c, m, J, \{\sigma'_j\}_{j \in J}),$$

where  $pk_c$  is the combiner public key,  $m$  is a message,  $J$  is a subset of  $[n]$  of size  $t$ , and each  $\sigma'_j$  is the signature share for  $m$  of  $j \in J$ . The algorithm either outputs a signature  $\sigma$ , or outputs a special message  $blame(J^*)$ , where  $J^*$  is a nonempty subset of  $J$ .

- Intuitively, the message  $blame(J^*)$  indicates that the provided signature shares  $\sigma'_j$  for  $j \in J^*$  are invalid.
- $V$  is a deterministic **verification algorithm** as in a signature scheme invoked as

$$V(pk, m, \sigma),$$

and outputs either accept or reject.

- **Correctness:** the verification algorithm should accept a properly constructed signature; specifically, for all possible outputs  $(pk, pk_c, s_1, \dots, s_n)$  of  $G(n, t)$ , all messages  $m$ , and all  $t$ -size subsets  $J$  of  $[n]$ , we have

$$\Pr[V(pk, m, C(pk_c, m, J, \{S(s_j, m)\}_{j \in J})) = \text{accept}] = 1.$$

Different threshold signature schemes have been defined so far. Shoup [9] defined one of the most used threshold signature schemes, which is based on RSA [10–13]. It requires a trusted, centralized dealer for key generation, and then uses non-interactive signature share generation and signature verification protocols.

Gennaro et al. [14] proposed a threshold DSA signature scheme, with  $n \geq 2t - 1$ , where a trusted centralized dealer is adopted. The more general, threshold-optimal case is then presented in the study of Gennaro et al. [15], where they propose a dealer-less approach supporting the case  $n \geq t$ . However, DKG is costly and impractical. Then, Gennaro and Goldfeder [16,17] presented an ECDSA-based protocol supporting efficient DKG, which obtains faster signing than Gennaro et al. [15] and requires less data to be transmitted. In a closely related work, Lindell et al. [18] proposed an efficient threshold ECDSA scheme, which employs different methods to neutralize any adversarial behavior. Differently from the study of Gennaro and Goldfeder [16], this protocol revolves around a modification of the ElGamal encryption scheme. Using an ElGamal signature scheme, Noack and Spitz [19] propose a dynamic threshold signature scheme, which does not rely on a trusted third party. It has the nice property of not changing the public key while adding or removing a certain number of nodes.

A detailed (and more extensive) review of threshold ECDSA schemes can be found in the study of Aumasson et al. [20]. Although ECDSA is fast and secure, aggregated signatures cannot be easily obtained with it.

Conversely, BLS [21] and Schnorr [22] schemes can be easily transformed into threshold schemes by supporting the sum of partial signatures with no overhead [23]. In particular, Boldyreva [24] proposed the most widely adopted approach for threshold BLS signatures. Here, the DKG does not require a trusted dealer, and the signature generation does not require participant interaction (or any zero-knowledge proof). It can only tolerate up to  $t - 1 < n/2$  malicious parties, but it allows us to periodically renew the secret shares.

Recently, Tomescu et al. [25] proposed a more efficient BLS signature scheme, that improves signing and verification time. Threshold BLS signature schemes rely on pairing-based cryptography [21] and can perform signing operations in a single round among participants.

Schnorr signatures received increased interest recently, and they have been included in the Bitcoin protocol<sup>3</sup>. Komlo and Goldberg [1] proposed FROST, an efficient Schnorr-based threshold scheme, whereby signing can be performed in two rounds, or optimized to a single round with preprocessing. FROST is currently considered the most efficient scheme for generating Schnorr threshold signatures [26]. Ruffing et al. [3] proposed ROAST, a wrapper protocol around FROST that provides liveness guarantees in the presence of malicious nodes and asynchronous networks.

We prioritize efficiency over robustness, so we assume FROST as the starting point of our work. FROST's efficiency comes from another valuable feature, which is the ability to perform signing operations asynchronously.

---

<sup>3</sup> [https://en.bitcoin.it/wiki/BIP\\_0340](https://en.bitcoin.it/wiki/BIP_0340).

## 2.2 Possible solutions

To find the solution that best fits our problem, we consider many DPSS schemes, namely, PSS schemes that involve dynamic committees. For the sake of clarity, we report here the formal definition of a DPSS, as presented in the study of Hu et al. [27].

**Definition 2.** A DPSS protocol consists of the following three algorithms:

- $\langle s_i, \pi_i \rangle_{P_i^1 \in C^1} \leftarrow \text{Share}(t, n, s, 1^\kappa)$ : This algorithm shares a secret to the initial committee  $C^1$ . It takes as inputs a threshold  $t$ , a committee size  $n$ , a secret value  $s$ , and a security parameter  $\kappa$  in unary form. Each node  $P_i^1 \in C^1$  outputs a share-proof tuple  $\langle s_i, \pi_i \rangle$ .
- $\langle s'_j, \pi'_j \rangle_{P_j^{e+1} \in C^{e+1}} \leftarrow \text{Handoff}(\langle s_i, \pi_i \rangle_{P_i^e \in C^e})$ : This algorithm allows the new committee  $C^{e+1}$  to obtain refreshed shares from the old committee  $C^e$ . Each old node  $P_i^e \in C^e$  inputs a share-proof tuple  $\langle s_i, \pi_i \rangle$ , and each new node  $P_j^{e+1} \in C^{e+1}$  outputs a refreshed tuple  $\langle s'_j, \pi'_j \rangle$ .
- $v \leftarrow \text{Reconstruct}(t, \langle s_i, \pi_i \rangle_{i \in I})$ : This algorithm reconstructs the secret. It takes as inputs a threshold  $t$  and a set of share-proof tuples  $\langle s_i, \pi_i \rangle_{i \in I}$ , where  $I \subseteq [n]$  with  $|I| \geq t$ , and outputs a reconstructed secret  $v$ .

DPSS protocols can be classified into three categories, based on whether they use a synchronous, partially synchronous, or asynchronous network.

In D-FROST, we suppose to be in a synchronous setting, since FROST works synchronously during KeyGen and Preprocess. Thus, there is no need for a DPSS that operates in an asynchronous network, especially if it weakens the protocol. In particular, all asynchronous and partially synchronous DPSS schemes in the literature (e.g., Schultz's MPSS [28], COBRA [29], Robust Asynchronous DPSS [30]) require the presence of a dealer, giving up decentralization. Moreover, they are less efficient than many synchronous techniques and have lower threshold bounds. We therefore opt for a synchronous protocol.

To the best of our knowledge, the most recent and efficient synchronous PSS schemes with dynamic committees are CHURP [8], Benhamouda et al.'s [31], and Goyal et al.'s [32]. However, the protocol by Benhamouda et al. involves a dealer and the one by Goyal et al. demands that the secret  $s$  is held by a client. Since we want the secret to be hidden from everybody, none of these schemes suits our purpose. Thus, we select CHURP as the best solution, which is a highly efficient and decentralized protocol with a large upper bound on the threshold.

As stated in Section 1, D-FROST is the first Schnorr-based threshold signature scheme that allows modifications to the committee and to the threshold, without changing the group public key.

The first scheme to achieve something similar is that in the study of Battagliola et al. [33], which enables a group of  $t$  participants to add a new node to the committee. Thus, this system only achieves one of the properties we desire.

An improvement is accomplished by the SPRINT protocol [34], which allows both to remove and add a participant. Even though this scheme tolerates dynamic committees, it does not allow threshold changes, and therefore, it is less flexible than D-FROST.

## 3 Background

### 3.1 FROST

FROST [1] is a Schnorr threshold signature scheme that allows a group of  $t$  out of  $n$  nodes to sign a message  $m$  with a signature that is indistinguishable from a single-party Schnorr signature. It is a decentralized protocol, where each participant has the same power, except for the **signature aggregator** (SA). SA is a semi-trusted node that has the ability to report misbehaving participants and to publish the group signature at the end of the protocol. The SA role might also be assigned to an external party that has access to all public keys. In the following, we provide an overview of the protocol. For the sake of completeness, we detail FROST's KeyGen, Preprocess( $\pi$ ), and Sign( $m$ ), respectively, in Algorithms 1–3.

---

**Algorithm 1** KeyGen

---

**Input:** committee  $C = \{P_i\}_{i \in [n]}$ , threshold  $t$ .**Output:** each  $P_i$  holds a  $(t, n)$ -share  $s_i$  of the secret  $s$ .**Round 1**

1. Every participant  $P_i$  samples  $t$  random values  $(a_{i0}, \dots, a_{i(t-1)}) \xleftarrow{\$} \mathbb{Z}_q$  and uses these values as coefficients to define a degree  $t - 1$  polynomial  $f_i(x) = \sum_{j=0}^{t-1} a_{ij}x^j$ .
2. Every  $P_i$  computes a proof of knowledge to the corresponding secret  $a_{i0}$  by calculating  $\sigma_i = (R_i, \mu_i)$ , such that  $k \xleftarrow{\$} \mathbb{Z}_q, R_i = g^k, c_i = H(i, \Phi, g^{a_{i0}}, R_i), \mu_i = k + a_{i0}c_i$ , with  $\Phi$  being a context string to prevent replay attacks.
3. Every participant  $P_i$  computes a public commitment  $\vec{C}_i = \langle \Phi_{i0}, \dots, \Phi_{i(t-1)} \rangle$ , where  $\Phi_{ij} = g^{a_{ij}}, 0 \leq j \leq t - 1$ .
4. Every  $P_i$  broadcasts  $\vec{C}_i, \sigma_i$  to all other participants.
5. Upon receiving  $\vec{C}_l, \sigma_l$  from participants  $1 \leq l \leq n, l \neq i$ , participant  $P_i$  verifies  $\sigma_l = (R_l, \mu_l)$ , aborting on failure, by checking  $R_l \stackrel{?}{=} g^{\mu_l} \Phi_{l0}^{-c_l}$ , where  $c_l = H(l, \Phi, \Phi_{l0}, R_l)$ .
6. Upon success, participants delete  $\{\sigma_l : 1 \leq l \leq n\}$ .

**Round 2**

1. Each  $P_i$  securely sends to each other participant  $P_l$  a secret share  $(l, f_i(l))$ , deleting  $f_i$  and each share afterward except for  $(i, f_i(i))$ , which they keep for themselves.
  2. Each  $P_i$  verifies their shares by calculating  $g^{f_i(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} \Phi_{ik}^{i^k \bmod q}$ , aborting if the check fails.
  3. Each  $P_i$  calculates their long-lived private signing share by computing  $s_i = \sum_{l=1}^n f_l(i)$ , stores  $s_i$  securely, and deletes each  $f_l(i)$ .
  4. Each  $P_i$  calculates their public verification share  $Y_i = g^{s_i}$ , and the group's public key  $Y = \prod_{j=1}^n \Phi_{j0}$ . Any participant can compute the public verification share of any other participant by calculating  $Y_i = \prod_{j=1}^n \prod_{k=0}^{t-1} \Phi_{jk}^{i^k \bmod q}$ .
- 

---

**Algorithm 2** Preprocess( $\pi$ )

---

Let  $j$  be a counter for a specific nonce/commitment share pair, and  $\pi$  be the number of pairs generated at a time**Input:**  $\pi = \text{number of nonce/commitment share pairs}$ **Output:** each  $P_i$  publishes  $(i, \langle (D_{ij}, E_{ij}) \rangle_{j \in [\pi]})$ 

1. Create an empty list  $L_i$ . Then, for  $1 \leq j \leq \pi$ , perform the following:

- 1.a Sample single-use nonces  $(d_{ij}, e_{ij}) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ .
  - 1.b Derive commitment shares  $(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$
  - 1.c Append  $(D_{ij}, E_{ij})$  to  $L_i$ . Store  $((d_{ij}, D_{ij}), (e_{ij}, E_{ij}))$  for later use in signing operations.
2. Publish  $(i, L_i)$  to a predetermined location, as specified by the implementation.
- 

**Protocol details.** Let  $\mathbb{G}$  be a group of prime order  $q$ , and let  $g$  be a generator of  $\mathbb{G}$ . Let  $\{P_i\}_{i \in [n]}$  denote the set of participants, where  $[n] = \{1, \dots, n\}$ . The protocol starts with a secret sharing scheme that distributes the secret  $s \in \mathbb{Z}_q$  in  $n$  secret shares  $s_i$ , one for each  $P_i$ , such that  $t$  shares are enough to reconstruct  $s$  and  $t - 1$  participants cannot learn any information about  $s$ . The key generation scheme used by FROST is a modified version of Pedersen's DKG. The idea behind this scheme is to generate a random polynomial  $f(x) \in \mathbb{Z}_q[x]$  such that  $\deg_f = t - 1$  and  $f(0) = s$ . Each  $P_i$  is given the value  $f(i) = s_i$ , which is its secret share of  $s$ , and thus can compute its public key  $Y_i := g^{s_i}$ . Every time a misbehaving node is detected, FROST aborts in order to avoid rogue-key attacks [35]. To collectively reconstruct  $s$ ,  $t$  nodes might perform Lagrange interpolation with their shares and obtain

$s = \sum_{i=1}^t \lambda_i s_i$ , where  $\lambda_i = \prod_{j \neq i} \frac{j}{j-i}$ . However, the secret is never directly recovered by any node, as otherwise such a node could sign messages independently from the others. Instead, Lagrange interpolation is indirectly used during the signing operations. The group public key is  $Y = g^s$ . The polynomial  $f(x)$  is generated in a decentralized way, by adding polynomials  $f_i(x)$  randomly generated by each participant, and each share  $s_i$  is recovered by the correspondent participant without help from a particular node. This is an important feature of Pedersen's DKG, as FROST values decentralization and there is no trusted dealer who knows the secret.

FROST's KeyGen (Algorithm 1) is Pedersen's DKG with a slight modification, which consists in a zero-knowledge proof of knowledge, computed by each participant, of their corresponding secret  $a_{i0} = f_i(0)$ . Thanks to this change, the upper bound on the threshold  $t$  is raised from  $\frac{n}{2}$  to  $n$  without losing security against rogue-key attacks. Once KeyGen is completed, the protocol proceeds with Preprocess( $\pi$ ), which is a preprocessing stage reported in Algorithm 2. Here, each  $P_i$  creates and publishes  $\pi$  pairs of commitments  $(D_{ij}, E_{ij}) = (g^{d_{ij}}, g^{e_{ij}})$ , where  $d_{ij}$  and  $e_{ij}$  are the random elements of  $\mathbb{Z}_q$ . Each pair of commitments is used for a single signature and discarded afterwards. If the committee needs to sign a new message and there are no more available commitments, the Preprocess( $\pi$ ) protocol is executed again.

The last part of the protocol is the signing phase. During  $\text{Sign}(m)$ , which is described in Algorithm 3, SA selects the set  $S$  of nodes that will sign the message  $m$ . This set is made of  $\alpha$  signing nodes, where  $t \leq \alpha \leq n$ . Then, SA obtains the next available commitment for each  $P_i$  and creates  $B = \langle (i, D_i, E_i) \rangle_{i \in S}$ . Once all nodes have received  $B$ , they validate  $m$  and compute  $\rho_l = H_1(l||m||B)$ ,  $l \in S$ , where  $H_1$  is a hash function mapping to  $\mathbb{Z}_q^*$ . Next, they derive the group commitment  $R = \prod_{l \in S} D_l \cdot (E_l)^{\rho_l}$  and the challenge  $c = H_2(R||Y||m)$ , where  $H_2$  is also a hash function. Then, each  $P_i$  computes  $z_i = d_i + e_i \cdot \rho_i + \lambda_i \cdot s_i \cdot c$  and returns it to SA. The SA verifies the validity of  $z_i$ , for  $i \in S$ . If every response is correct, SA computes  $z = \sum_{i \in S} z_i$ . Finally, the signature  $\sigma = (R, z)$  is published.

Note that the way  $R$  is calculated binds the message, the set of signing participants, and the pairs  $(D_i, E_i)_{i \in S}$  to each signature share. This binding method prevents the adversary from changing anything or combining signature shares across disjoint signing operations, which makes the protocol resistant to the Drijvers attack.

**Synchronicity assumptions.** During the first two phases of the protocol (KeyGen and Preprocess( $\pi$ )), FROST requires a synchronous network, while the signing phase can be performed asynchronously.

---

### Algorithm 3 $\text{Sign}(m)$

---

Let SA be the signature aggregator,  $S$  be the set of signers, and  $Y$  be the group public key. Let  $B = \langle (i, D_i, E_i) \rangle_{i \in S}$  be the ordered list of indices and commitment shares, corresponding to each participant  $P_i$ , and let  $L_i$  be the set of commitment shares for  $P_i$  that were published during the preprocess stage. Let  $H_1, H_2$  be hash functions whose outputs are in  $\mathbb{Z}_q^*$ .

**Input:** a message  $m$  and the list  $B = \langle (i, D_i, E_i) \rangle_{i \in S}$

**Output:** a signature  $\sigma = (R, z)$  and  $m$

1. SA begins by fetching the next available commitment for each participant  $P_i$  from  $L_i$  and constructs  $B$ .
  2. For each  $i \in S$ , SA sends  $P_i$  the tuple  $(m, B)$ .
  3. After receiving  $(m, B)$ , each  $P_i$  first validates the message  $m$  and then checks  $D_l, E_l \in \mathbb{G}^*$  for each commitment in  $B$ , aborting if either check fails.
  4. Each  $P_i$  then computes the set of binding values  $\rho_l = H_1(l||m||B)$ ,  $l \in S$ . Each  $P_i$  then derives the group commitment  $R = \prod_{l \in S} D_l \cdot (E_l)^{\rho_l}$  and the challenge  $c = H_2(R||Y||m)$ .
  5. Each  $P_i$  computes their response using their long-lived secret share  $s_i$  by computing  $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$ , using  $S$  to determine the  $i$ th Lagrange coefficient  $\lambda_i$ .
  6. Each  $P_i$  securely deletes  $((d_i, D_i), (e_i, E_i))$  from their local storage and then returns  $z_i$  to SA.
  7. The signature aggregator SA performs the following steps:
    - 7.a Derive  $\rho_i = H_1(i||m||B)$  and  $R_i = D_{ij} \cdot (E_{ij})^{\rho_i}$  for  $i \in S$ , and subsequently  $R = \prod_{i \in S} R_i$  and  $c = H_2(R||Y||m)$ .
    - 7.b Verify the validity of each response by checking  $g^{z_i} = R_i \cdot Y_i^{c \cdot \lambda_i}$  for each signing share  $z_i$ ,  $i \in S$ .  
If the equality does not hold, identify and report the misbehaving participant and then abort.  
Otherwise, continue.
    - 7.c Compute the group's response  $z = \sum_{i \in S} z_i$ .
    - 7.d Publish  $\sigma = (R, z)$  along with  $m$ .
-

### 3.2 Security of FROST

The protocol is proved to be EUF-CMA secure under the Discrete Logarithm (DL) assumption in the random oracle model.

**Definition 3.** A signature scheme is **existentially unforgeable under chosen message attack**, or **EUF-CMA secure**, if the adversary cannot forge a signature on a chosen message  $m$  that was not previously signed by the oracle.

The scheme is also secure against the Drijvers attack [36] and the ROS solver [37]. In particular, this means that the protocol is secure against a concurrent adversary, i.e., an adversary that can open simultaneous signing sessions at once. As stated in Section 3.1, the protocol is resistant to rogue-key attacks too.

FROST's proof of security uses the general forking algorithm (see Algorithm 4), which we denote by  $\text{GF}_A$ , and the general forking lemma by Bellare and Neven [38]. The symbol  $\$$  indicates random sampling.

**Theorem 1.** (General Forking lemma) *Fix an integer  $q \geq 1$  and a set  $O$  of size  $h \geq 2$ . Let  $A$  be a randomized algorithm that on input  $x, h_1, \dots, h_q$  returns a pair, the first element of which is an integer in the range  $0, \dots, q$  and the second element of which we refer to as a side output. Let  $\text{IG}$  be a randomized algorithm that we call the input generator. The accepting probability of  $A$ , denoted  $\text{acc}$ , is defined as the probability that  $J \geq 1$  in the experiment*

$$x \xleftarrow{\$} \text{IG}; \quad h_1, \dots, h_q \xleftarrow{\$} O; \quad (J, \sigma) \leftarrow A(x, h_1, \dots, h_q).$$

Let  $\text{frk} = \Pr[b = 1 : x \xleftarrow{\$} \text{IG}; (b, \sigma, \sigma') \xleftarrow{\$} \text{GF}_A(x)]$ . Then,

$$\text{frk} \geq \text{acc} \cdot \left( \frac{\text{acc}}{q} - \frac{1}{h} \right).$$

---

#### Algorithm 4 $\text{GF}_A(x)$

---

1. Pick random coins  $\rho$
  2.  $h_1, \dots, h_q \xleftarrow{\$} O$
  3.  $(J, \sigma)$  or  $\perp \leftarrow A(x, \{h_1, \dots, h_q\}; \rho)$
  4. **If**  $J = 0$ , **then return**  $(0, \epsilon, \epsilon)$
  5.  $h'_1, \dots, h'_q \xleftarrow{\$} O$
  6.  $(J', \sigma')$  or  $\perp \leftarrow A(x, \{h_1, \dots, h_{J-1}, h'_1, \dots, h'_q\}; \rho)$
  7. **If**  $J \neq J' \vee h_j \neq h'_j$  **then return**  $(0, \epsilon, \epsilon)$
  8. **Return**  $(1, \sigma, \sigma')$ .
- 

The adversary in FROST's proof of security is supposed to be *active* and *static* with the power to corrupt up to  $t - 1$  nodes, including SA. In particular, a static adversary decides which nodes are corrupted at the beginning of the protocol; thus, FROST does not achieve adaptive security, in which the adversary adaptively selects corrupted nodes during the execution of the protocol.

### 3.3 CHURP

CHURP [8] is a DPSS scheme, started by a group  $C = \{P_i\}_{i \in [n]}$  of nodes that  $(t, n)$ -share a secret  $s$ . CHURP allows  $C$  to go through a proactivation phase (*handoff*) in which the committee passes the secret to a new possibly disjoint group  $C' = \{P'_i\}_{i \in [n']}$ . Table 1 summarizes the main notation used throughout this article.

Initially, the secret is shared among nodes in  $C$  via a bivariate polynomial  $B(x, y)$  such that  $B(0, 0) = s$  and  $\deg_B = \langle t - 1, 2t - 2 \rangle$ . Each  $P_i$  holds the  $(2t - 2)$ -degree polynomial  $B(i, y)$ , which we refer to as *full share*. Then, during the handoff,  $B(x, y)$  is proactivized into a new polynomial  $B'(x, y)$  such that  $B'(0, 0) = B(0, 0) = s$ . Here, we suppose that the threshold is fixed for ease of exposition. Nevertheless, in this phase, both the threshold and the number of participants can be changed, as long as  $t - 1 < \frac{n}{2}$ . The reason behind this bound is that the adversary is given the power to corrupt up to  $t - 1$  nodes from each committee, so the total number of corrupted nodes is at most  $2t - 2$ ; then, the previous inequality follows from the fact that  $2t - 2 < n$  must hold.

To protect the secret during the handoff against  $2t - 2$  possibly corrupted nodes, the threshold is raised to  $2t - 1$ . This is the main reason for using a bivariate polynomial, as it allows to switch dimensions easily. Indeed,  $s$  can be distributed both with the  $(t, n)$ -shares  $s_i = B(i, 0)$  and the  $(2t - 1, n)$ -shares  $s_j = B(0, j)$ . In particular, during the handoff, the participants hold polynomial shares  $B(x, j)$ , which we refer to as *reduced shares* (since a higher threshold gives less power to a single share). These shares are used to distribute, to all the members of the new committee, the new proactivized full shares  $B'(i, y)$ , that are independent of the old ones.

This protocol is executed periodically, at the beginning of a fixed interval of time called *epoch*.

**Invariants.** To preserve integrity of the secret while transmitting it to a new committee, CHURP makes use of the Kate, Zaverucha, Goldberg (KZG) scheme [39], a polynomial commitment protocol: it allows a user to commit to a polynomial  $P(x)$  and to prove that  $P(i)$  is the result of the evaluation  $P(x)$  at some index  $i$ .

**Definition 4.** A **polynomial commitment scheme** consists of six algorithms:

- $\text{Setup}(1^\lambda, t)$  generates an appropriate algebraic structure  $G$  and a commitment public–private key pair  $(\text{pk}, \text{sk})$  to commit to a polynomial of degree  $\leq t$ . For simplicity, we add  $G$  to the public key  $\text{pk}$ .  $\text{Setup}$  is run by a trusted or distributed authority. Note that  $\text{sk}$  is not required in the rest of the scheme.
- $\text{Commit}(\text{pk}, \Phi(x))$  outputs a commitment  $C$  to a polynomial  $\Phi(x)$  for the public key  $\text{pk}$ , and some associated decommitment information  $d$ . (In some constructions,  $d$  is null.)
- $\text{Open}(\text{pk}, C, \Phi(x), d)$  outputs the polynomial  $\Phi(x)$  used while creating the commitment, with decommitment information  $d$ .
- $\text{VerifyPoly}(\text{pk}, C, \Phi(x), d)$  verifies that  $C$  is a commitment to  $\Phi(x)$ , created with decommitment information  $d$ . If so, the algorithm outputs 1; otherwise, it outputs 0.
- $\text{CreateWitness}(\text{pk}, \Phi(x), i, d)$  outputs  $(i, \Phi(i), w_i)$ , where  $w_i$  is a witness for the evaluation  $\Phi(i)$  of  $\Phi(x)$  at the index  $i$  and  $d$  is the decommitment information.
- $\text{VerifyEval}(\text{pk}, C, i, \Phi(i), w_i)$  verifies that  $\Phi(i)$  is indeed the evaluation at the index  $i$  of the polynomial committed in  $C$ . If so, the algorithm outputs 1; otherwise, it outputs 0.

After a successful handoff, the system is in a *steady state*, which means that the following three invariants must hold for the new committee  $\{P_i\}_{i \in [n]}$  and the new polynomial  $B(x, y)$ :

- INV-SECRET: the secret  $s$  is the same across handoffs.
- INV-STATE: each node  $P_i$  holds a full share  $B(i, y)$  and a proof to the correctness thereof. Specifically, the full share  $B(i, y)$  is a  $(2t - 2)$ -degree polynomial, and hence can be uniquely represented by  $2t - 1$  points  $\{B(i, j)\}_{j \in [2t-1]}$ . The proof is a set of witnesses  $\{W_{B(i, j)}\}_{j \in [2t-1]}$ .
- INV-COMM: KZG commitments to reduced shares  $\{B(x, j)\}_{j \in [2t-1]}$  are available to all nodes.

These invariants ensure that some important properties are satisfied. In particular, INV-SECRET guarantees that the secret remains the same throughout the whole protocol, while INV-STATE and INV-COMM guarantee the correctness of the scheme. Indeed, during the handoff, nodes in the new committee can verify the correctness of reduced shares (and, thus, the correctness of dimension-switching), using the commitments and the witnesses.

**Setup.** First, the protocol selects an initial committee  $C^{(0)}$  and each participant is given a private/public key pair, where public keys are known to all nodes in the system. Then, the generation of the secret and the setup of KZG are executed by a trusted party or a committee with at least one honest participant.

**Communication model.** Nodes in CHURP have two ways to communicate with each other: a blockchain available to everyone, on which nodes can publish and read messages, or peer-to-peer channels to send

**Table 1:** Notation used in CHURP and D-FROST

Notation	Description
$C^{(e-1)}, C^{(e)}$	Old, new committee
$B(x, y)$	Bivariate polynomial used to share the secret
$\langle t, k \rangle$	Degree of $x, y$ terms in $B$
$RS_i(x) = B(x, i)$	Reduced share held by $P_i$
$FS_i(y) = B(i, y)$	Full share held by $P_i$
$C_{B(x, j)}$	KZG commitment to $B(x, j)$
$W_{B(i, j)}$	Witness to evaluation of $B(x, j)$ at $i$
$W'_{B(i, j)}$	Witness to evaluation of $B(i, y)$ at $j$
$Q(x, y)$	Bivariate proactivation polynomial
$U'$	Subset of nodes chosen to participate in handoff
$\lambda_i$	Lagrange coefficients

and receive messages. Both communication methods are supposed to be *synchronous*, i.e., once a message is sent, it is received within a finite period of time  $T$ . Synchronicity in peer-to-peer channels is required only for performance, not for liveness, secrecy, or integrity. This kind of communication is used only in the optimistic path, and if a message takes too long to deliver, the protocol switches to the pessimistic path, where all communications happen on-chain, as explained below.

The use of a blockchain for communication is not strictly necessary, and it could be replaced with any other kind of reliable bulletin board. However, this change would require additional work that goes beyond the scope of our work, so we employ a blockchain for broadcast communication.

**Protocol details.** CHURP is made of three subprotocols: Opt-CHURP, Exp-CHURP-A, and Exp-CHURP-B. The first one is the optimistic path, while the others are the pessimistic ones. When CHURP is started, Opt-CHURP is executed by default. To speed up the protocol, most communications take place off-chain. If a fault is detected, the protocol switches to Exp-CHURP-A: from this point forward, nodes communicate on-chain only. This allows participants to perform verifiable accusations and expel corrupted nodes from the committee. If a breach in the underlying assumptions of the KZG scheme is detected, the protocol switches to Exp-CHURP-B. This pessimistic path is proved to be secure under the DL assumption, but it lowers the bound on the threshold to  $t - 1 < \frac{n}{3}$ . In D-FROST, we suppose all necessary assumptions hold, so we only consider the first two paths and exclude Exp-CHURP-B. Moreover, since the difference between the two paths is only in the communication model, the two protocols are largely the same, and we only explain Opt-CHURP in the next paragraph. More details on Opt-CHURP and Exp-CHURP-A are included in Sections 3.3.1 and 3.3.2, respectively.

---

**Algorithm 5** Opt-ShareReduce
 

---

**Public Input:**  $\{C_{B(x, j)}\}_{j \in [2t-1]}$

**Input:** Set of nodes  $\{P_i\}_{i \in [n]}$  where each node  $P_i$  is given  $\{B(i, j), W_{B(i, j)}\}_{j \in [2t-1]}$ . Set of nodes  $\{P'_j\}_{j \in [n']}$  such that  $n' \geq 2t - 1$

**Output:**  $\forall j \in [2t - 1]$ , node  $P'_j$  outputs  $B(x, j)$ .

1. Order  $\{P'_j\}$  based on the lexicographic order of their public keys.
  2. Choose the first  $2t - 1$  nodes, denoted as  $U'$ , without loss of generality,  $U' = \{P'_j\}_{j \in [2t-1]}$ .
  3. Node  $P_i$ :
    - 3.a  $\forall j \in [2t - 1]$ , send a point and witness  $(B(i, j), W_{B(i, j)})$  to  $U'_j$  off-chain.
  4. Node  $U'_j$ :
    - 4.a Wait and receive  $n$  points and witnesses,  $\{(B(i, j), W_{B(i, j)})\}_{i \in [n]}$ .
    - 4.b  $\forall i \in [n]$ , invoke  $\text{VerifyEval}(C_{B(x, j)}, i, B(i, j), W_{B(i, j)})$ .
    - 4.c Interpolate any  $t$  verified points to construct  $B(x, j)$ .
-

**Opt-CHURP.** Even though the protocol supports changes to both the threshold and the number of nodes in the committee, in this section, we assume that the threshold is fixed and  $n$  can change. In particular, if  $t_e$  is the threshold in epoch  $e$ , we set  $t_{e-1} = t_e = t$ . In Section 3.3.3, we explain how threshold changes are managed.

Opt-CHURP is divided into three stages. The first one is Opt-ShareReduce (Algorithm 5), which allows a set  $U'$  of  $2t - 1$  members of the new committee  $C' = \{P'_i\}_{i \in [n']}$  to recover the reduced shares  $B(x, j)$ . This is done by interpolating  $t$  verified points  $B(i, j)$ . To check the validity of the points, nodes use the KZG commitments and witnesses produced by members of the old committee.

---

**Algorithm 6** Opt-Proactivize
 

---

**Public Input:**  $\{C_{B(x,j)}\}_{j \in [2t-1]}$ .

**Input:** Set of nodes  $\{P'_i\}_{i \in [n']}$ . Let  $U' = \{P'_j\}_{j \in [2t-1]}$ , each node  $U'_j$  is given  $B(x, j)$ .

**Output:**  $U'_j$  outputs success and  $B'(x, j)$  for a degree- $\langle t - 1, 2t - 2 \rangle$  bivariate polynomial  $B'(x, y)$  with  $B'(0, 0) = B(0, 0)$  or fail.

**Public Output:**  $\{C_{B'(x,j)}\}_{j \in [2t-1]}$ .

1. Invoke  $(2t - 2, 2t - 1)$ -UnivariateZeroShare among the nodes  $\{U'_j\}_{j \in [2t-1]}$  to generate shares  $\{s_j\}_{j \in [2t-1]}$ .
  2. Node  $U'_j$ :
    - 2.a Generate a random  $(t - 1)$ -degree polynomial  $R_j(x)$  such that  $R_j(0) = s_j$ .
    3. Denote the bivariate polynomial  $Q(x, y)$  where  $Q(x, j) = R_j(x) \forall j \in [2t - 1]$ .
    4. Denote the bivariate polynomial  $B'(x, y) = B(x, y) + Q(x, y)$ .
  5. Node  $U'_j$ :
    - 5.a Compute  $B'(x, j) = B(x, j) + Q(x, j)$  and  $Z_j(x) = R_j(x) - s_j$ .
    - 5.b Send  $\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$  off-chain to all nodes in  $C'$ , where  $C_{Z_j} = \text{Commit}(Z_j)$ ,  $W_{Z_j(0)} = \text{CreateWitness}(Z_j, 0)$ , and  $C_{B'(x,j)} = \text{Commit}(B'(x, j))$ .
    - 5.c Publish hash of the commitments on-chain  $H_j = H(g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)})$ .
  6. Node  $P'_i$ :
    - 6.a  $\forall j \in [2t - 1]$ , retrieve on-chain hash  $H_j$ . Also, receive  $\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$  off-chain.
    - 6.b  $\forall j \in [2t - 1]$ , if  $H_j \neq H(g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)})$  or  $\text{VerifyEval}(C_{Z_j}, 0, 0, W_{Z_j(0)})$  True or  $C_{B'(x,j)} \neq C_{B(x,j)} \times C_{Z_j} \times g^{s_j}$ , output fail.
    - 6.c If  $\prod_{j=1}^{2t-1} (g^{s_j})^{\lambda_j^{2t}} \neq 1$ , output fail.
  7. Node  $U'_j$ :
    - 7.a Output success and  $B'(x, j)$ .
- 

**Algorithm 7** Opt-ShareDist
 

---

**Public Input:**  $\{C_{B'(x,j)}\}_{j \in [2t-1]}$ .

**Input:** Set of nodes  $\{P'_i\}_{i \in [n']}$ . Let  $U' = \{P'_j\}_{j \in [2t-1]}$ , each node  $U'_j$  is given  $B'(x, j)$ .

**Output:**  $\forall i \in [n']$ ,  $P'_i$  outputs success and  $B'(i, y)$  or fail.

1. Node  $U'_j$ :
    - 1.a  $\forall i \in [n']$ , send a point and witness off-chain  $\{B'(i, j), W_{B'(i,j)}\}$  to  $P'_i$ , where  $W_{B'(i,j)} = \text{CreateWitness}(B'(x, j), i)$ .
  2. Node  $P'_i$ :
    - 2.a Wait and receive points and witnesses  $\{B'(i, j), W_{B'(i,j)}\}_{j \in [2t-1]}$ .
    - 2.b  $\forall j \in [2t - 1]$ , invoke  $\text{VerifyEval}(C_{B'(x,j)}, i, B'(i, j), W_{B'(i,j)})$ .
    - 2.c If all  $2t - 1$  points are correct, interpolate to construct  $B'(i, y)$ .
    - 2.d Output success and the full share  $B'(i, y)$ .
    - 2.e In all other cases, output fail.
-

The second phase of Opt-CHURP is Opt-Proactivize (Algorithm 6), during which the polynomial  $B(x, y)$  obtains proactivized. The idea is to add a random zero-hole polynomial  $Q(x, y)$  to  $B(x, y)$ , obtaining a new polynomial  $B'(x, y)$  such that  $B'(0, 0) = B(0, 0) = s$ ,  $\deg_{B'} = \langle t - 1, 2t - 2 \rangle$  and  $B'$  is independent from  $B$ . To create  $Q(x, y)$ , the algorithm generates a zero-hole polynomial  $P(x)$  of degree  $2t - 2$  and  $2t - 1$  zero-shares  $s_j$  such that  $s_j = P(j)$ . Then, each node in  $U'$  generates a random polynomial  $R_j(x)$  such that  $R_j(0) = s_j$ .  $Q(x, y)$  is defined as the interpolation of  $\{R_j(x)\}_{j \in [2t-1]}$ , and therefore,  $Q(x, j) = R_j(x) \forall j \in [2t-1]$ . Since  $Q(0, j) = P(j)$ , it also follows that  $Q(0, 0)$  as required. Then,  $U'_j$  can proactivize its reduced share, by defining  $B'(x, j) = B(x, j) + Q(x, j)$ . Participants prepare all necessary information to allow the others to verify validity of the updated shares. Specifically, each  $U'_j$  computes  $Z_j(x) = R_j(x) - s_j$  and sends  $\{g^{s_j}, C_{Z_j}, W_{Z_j(0)}, C_{B'(x,j)}\}$  off-chain to all members of  $C'$ . This way, nodes can check correctness of new shares without knowing either  $B'(x, j)$  or  $s_j$ . Note that, after this step, the commitments  $\{C_{B'(x,j)}\}_{j \in [2t-1]}$  are available to all nodes in  $C'$ , as required by INV-COMM.

The last part of the protocol is Opt-ShareDist (Algorithm 7). In this phase, every  $U'_j$  sends  $B'(i, j)$  to each participant  $P'_i$ , along with the witness  $W_{B'(i,j)}$  to make it verifiable.  $P'_i$  receives  $2t - 1$  points  $\{B'(i, j)\}_{j \in [2t-1]}$  and interpolates them to obtain the full share  $B'(i, y)$ . If any of these points is not valid, the algorithm returns fail. Otherwise, the process ends successfully and the committee is in steady state. Nodes in the old committee are required to delete their full shares and nodes in  $U'$  delete their reduced shares.

### 3.3.1 Details of Opt-CHURP

Full details of Opt-CHURP are provided here for ease of reading, but can also be found in the study of Maram et al. [8]:

- In Algorithm 5, the set  $U'$  of  $2t - 1$  members from the old committee reconstruct the polynomial shares  $B(x, j)$ ;
- Algorithm 6 shows how nodes in  $U'$  proactivize their reduced shares;
- Algorithm 7 describes the last phase of the handoff, in which members of the new committee recover their full shares of the proactivitized polynomial  $B'(x, y)$ .

These algorithms refer to the following auxiliary functions:

- Commit and CreateWitness are part of the KZG scheme. They generate the commitment to a polynomial and the witness to the evaluation of a polynomial at some point, respectively.
- VerifyEval( $C_{R(x)}, i, R(i), W_{R(i)}$ ) is also part of the KZG scheme, and it verifies that the evaluation of the polynomial  $R(x)$  at  $i$  gives the value  $R(i)$ .
- Given a set of nodes  $\{U_j\}_{j \in [2t-1]}$ ,  $(2t - 2, 2t - 1)$ -UnivariateZeroShare generates a random polynomial  $P(y)$  such that  $\deg_{P(y)} = 2t - 2$ ,  $P(0) = 0$  and each node  $U_j$  holds  $s_j = P(j)$ . Its functioning is shown in Algorithm 8.

---

#### Algorithm 8 $(2t - 2, 2t - 1)$ -UnivariateZeroShare

---

**Input:**  $t$ , set of  $2t - 1$  nodes  $\{U_j\}_{j \in [2t-1]}$ .

**Output:** Each node  $U_j$  outputs a share  $s_j = P(j)$  for a randomly generated degree- $(2t - 2)$  polynomial  $P(y)$  with  $P(0) = 0$ .

1 Node  $U_j$ :

- 1.a Generate a random  $(2t - 2)$ -degree polynomial  $P_j$  such that  $P_j(0) = 0$ .
  - 1.b Send a point  $P_j(i)$  to node  $U_i$  for each  $i \in [2t - 1]$ .
  - 1.c Wait to receive points  $\{P_i(j)\}_{i \in [2t-1]}$  from all other nodes.
  - 1.d Let  $P = \sum_{i=1}^{2t-1} P_i$ , compute share  $P(j) = \sum_{i=1}^{2t-1} P_i(j)$ .
-

### 3.3.2 Pessimistic path

Here, we detail the functioning of Exp-CHURP-A. Similar to Opt-CHURP, Exp-CHURP-A is composed by three subprotocols: Exp-ShareReduce (Algorithm 9), Exp-Proactivize (Algorithm 10), and Exp-ShareDist (Algorithm 11). They have the same roles as Opt-ShareReduce, Opt-Proactivize, and Opt-ShareDist, respectively. The main difference is that in Exp-CHURP-A nodes do not have access to peer-to-peer channels. Thus, when  $P_i$  wants to send a private message to  $P_j$ , it encrypts the message with  $P_j$ 's public key  $pk_j$ . Then,  $P_j$  is the only node able to read the message by decrypting it with its secret key  $sk_j$ .

All CHURP's algorithms reported in this article are identical to the ones written in Maram et al. [8], except for Exp-ShareReduce (Algorithm 9). The original version of Exp-CHURP-A uses Opt-ShareReduce instead, but this contradicts the communication model, as  $P_i$  and  $U'$  communicate off-chain. We therefore build a revised algorithm starting from Opt-ShareReduce and moving all communications on-chain.

---

#### Algorithm 9 Exp-ShareReduce

---

**Public Input:**  $\{C_{B(x,j)}\}_{j \in [2t-1]}$ .

**Input:** Set of nodes  $\{P_i\}_{i \in [n]}$  where each node  $P_i$  is given  $\{B(i,j), W_{B(i,j)}\}_{j \in [2t-1]}$ . Set of nodes  $\{P'_j\}_{j \in [n']}$  such that  $n' \geq 2t - 1$ .

**Output:**  $\forall j \in [2t - 1]$ , node  $P'_j$  outputs  $B(x, j)$ .

1. Order  $\{P'_j\}$  based on the lexicographic order of their public keys.
  2. Choose the first  $2t - 1$  nodes, denoted as  $U'$ , without loss of generality,  $U' = \{P'_j\}_{j \in [2t-1]}$ .
  3. Node  $P_i$ :
    - 3.a  $\forall j \in [2t - 1]$ , publish  $(Enc_{pk_j}(B(i,j)), g^{B'(i,j)}, W_{B(i,j)})$  on-chain.
  4. Node  $U'_j$ :
    - 4.a Decrypt the message on-chain to obtain  $\{B(i,j), W_{B(i,j)}\}_{j \in [2t-1]}$ .
    - 4.b  $\forall i \in U' \setminus U'_{\text{corrupted}}$ , invoke  $\text{VerifyEval}(C_{B(x,j)}, i, B(i,j), W_{B(i,j)})$ . If any of the checks fail, add  $i$  to  $U'_{\text{corrupted}}$ .
    - 4.c Interpolate any  $t$  verified points to construct  $B(x, j)$ .
- 

---

#### Algorithm 10 Exp-Proactivize

---

**Public Input:**  $\{C_{B(x,j)}\}_{j \in [2t-1]}$ .

**Input:** Set of  $2t - 1$  nodes  $\{U'_j\}_{j \in [2t-1]}$ . Each node  $U'_j$  is given  $B(x, j)$ .

**Output:**  $U'_j$  outputs  $B'(x, j)$  for a degree- $\langle t - 1, 2t - 2 \rangle$  bivariate polynomial  $B'(x, y)$  with  $B'(0, 0) = B(0, 0)$ .

**Public Output:**  $\{C_{B'(x,j)}\}_{j \in [2t-1]}$ .

1. Node  $U'_i$ :
  - 1.1 Generate  $\{s_{ij}\}_{j \in [2t-1]}$  that form a 0-sharing, i.e.  $\sum_{j=1}^{2t-1} \lambda_j^{2t-2} s_{ij} = 0$ .
  - 1.2 Publish  $\{g^{s_{ij}}\}_{j \in [2t-1]}$ ,  $\{Enc_{pk_j}[s_{ij}]\}_{j \in [2t-1]}$ , and zero-knowledge proofs of correctness of the encryptions on-chain.
2. Node  $U'_j$ :
  - 2.1 Decrypt  $Enc_{pk_j}[s_{ij}]$  from node  $i$  and verify  $s_{ij}$  using  $g^{s_{ij}}$  on-chain.
3. Node  $U'_j$ :
  - 3.a If any adversarial node  $i$  is detected in step 2.1, add it to  $U'_{\text{corrupted}}$  and publish  $s_{ji}$ .
  - 3.b Set  $s_j = \sum_{i \in U' \setminus U'_{\text{corrupted}}} s_{ij}$ .
  - 3.c Execute steps 2.a, 3, 4, 5.a and 5.b of Opt-Proactivize with messages posted on the chain in step 5.b.

4. Node  $P'_i$ :
    - 4.a Execute step 6.b of Opt-Proactivize. If it outputs `fail`, add  $j$  to  $U'_{\text{corrupted}}$ . Nodes in  $U'$  discard shares by executing step 5.b again.
  5. Node  $P_i$ :
    - 5.a For all malicious nodes  $j$  detected in step 2.1 and 4.a, publish point and witness  $\{B(i, j), W_{B(i, j)}\}$  on-chain.
- 

**Algorithm 11** Exp-ShareDist
 

---

**Public Input:**  $\{C_{B'(x, j)}\}_{j \in [2t+1]}$ .

**Input:** Set of nodes  $\{P'_i\}_{i \in [n']}$ . Let  $U' = \{P'_j\}_{j \in [2t-1]}$ , each node  $U'_j$  is given  $B'(x, j)$ .

**Output:**  $\forall i \in [n]$ ,  $P'_i$  outputs  $B'(i, y)$ .

1. Node  $U'_j$ :
    - 1.a  $\forall i \in [n]$ , publish  $Enc_{pk_i}(B'(i, j))$ ,  $g^{B'(i, j)}$ ,  $w'_{ij}$  on-chain, where  $w'_{ij} = \text{CreateWitness}(B'(x, j), i)$ .  
Also, publish zero-knowledge proofs of correctness of the encryption.
  2. Node  $P'_i$ :
    - 2.a Decrypt the message on-chain to obtain  $\{B'(i, j), w'_{ij}\}_{j \in [2t-1]}$ .
    - 2.b  $\forall j \in U' \setminus U'_{\text{corrupted}}$ , invoke  $\text{VerifyEval}(C_{B'(x, j)}, i, B'(i, j), w'_{ij})$ . If any of the checks fail, add  $j$  to  $U'_{\text{corrupted}}$ .
  3. Node  $P_i$ :
    - 3.a Publish  $B(i, j)$ ,  $w_{ij}$  for any new adversarial node  $j$  detected above.
  4. Node  $U'_i$ :
    - 4.a Publish  $s_{ij}$  for any new adversarial node  $j$  detected above and discard shares by executing step 3.b in Exp-Proactivize.
  5. Node  $P'_i$ :
    - 5.a  $\forall j \in U'_{\text{corrupted}}$ , validate their reduced shares posted by the old committee  
by  $\forall i \in [n]$ ,  $\text{VerifyEval}(C_{B(x, j)}, i, B(i, j), w_{ij})$ .
    - 5.b  $\forall j \in U'_{\text{corrupted}}$ , interpolate any  $t$  verified points to construct  $B(x, j)$ . Set  $B'(i, j) = B(i, j) + \sum_{i \in \text{honest}} s_{ij}$ .
    - 5.c Interpolate all  $B'(i, j)$  for  $j \in [2t-1]$  to construct  $B'(i, y)$ .
    - 5.d Output the full share  $B'(i, y)$ .
- 

### 3.3.3 Changing the threshold

**Increasing the threshold.** To increase the threshold from  $t_{e-1}$  to  $t_e$ , CHURP runs the proactivization phase with parameter  $t = t_e$ , i.e., during the proactivization protocol, a bivariate zero-hole polynomial  $Q(x, y)$  of degree  $\langle t_e - 1, 2t_e - 2 \rangle$  is generated. Each node  $i$  holds a  $(t_e - 1)$ -degree polynomial  $Q(x, i)$  and commitments to  $\{Q(x, i)\}_{i \in [2t-1]}$  are publicly available. The rest of the proactivization follows without modification, besides the fact that now each node  $i$  holds two polynomials with different degrees:  $B'(x, i)$ , i.e.,  $(t_{e-1} - 1)$ -degree, and  $Q(x, i)$ , i.e.,  $(t_e - 1)$ -degree. Thus, the proactivized global polynomial  $B'(x, y)$  is of degree  $\langle t_e - 1, 2t_e - 2 \rangle$ , concluding the threshold upgrade.

**Decreasing the threshold.** The idea is to create  $2(t_{e-1} - t_e)$  virtual nodes, denoted as  $V$ , and execute the handoff protocol between  $C = C^{(e-1)}$  and  $C' = C^{(e)} \cup V$ , assuming the threshold remains  $t_{e-1}$ . Details are shown in Algorithm 12.

**Algorithm 12** Decreasing the threshold.

1. Choose a subset  $U \subseteq C'$  of  $2t_e - 1$  nodes. For notational simplicity, suppose  $U = \{1, \dots, 2t_e - 1\}$  and  $V = \{2t_e, \dots, 2t_{e-1} - 1\}$ . Each node  $i \in U$  recovers a reduced share  $RS_i^{(e-1)}(x) = B(x, i)$ . In addition,  $C$  publishes reduced shares for virtual nodes:  $RS_j^{(e-1)}(x) = B(x, j)$ , for  $j \in V$ .
2.  $U$  executes the proactivation phase and collectively generates a  $(t_e - 1, 2t_e - 2)$ -degree bivariate zero-hole polynomial  $Q(x, y)$ . At the end of this phase, each node  $i \in U$  has  $Q(x, i)$ .
3. Let  $v = \sum_{j \in V} \lambda_j^{(2t_{e-1}-2)} RS_j^{(e-1)}(0)$ . Each node  $i \in U$  incorporates virtual nodes' state and updates its state as
$$RS_i^{(e)}(x) = \frac{\lambda_i^{(2t_e-1-2)}}{\lambda_i^{(2t_e-2)}} \left( RS_i^{(e-1)}(x) + \frac{v}{\lambda_j^{(2t_{e-1}-2)}(2t_e-1)} \right) + Q(x, i)$$
where  $\lambda^{(2t_e-1-2)}$  and  $\lambda^{(2t_e-2)}$  are Lagrange coefficients for corresponding thresholds. One can verify that  $RS_i^{(e)}(x)$  are  $(2t_e - 2)$ -sharing of the secret, i.e.,  $B(0, 0)$  can be calculated from any  $2t_e - 1$  of  $RS_i^{(e)}(x)$ .
4. Each node  $i \in U$  sends to every node  $j \in C'$  a point  $RS_i^{(e)}(j)$ . The full share of node  $j \in C'$  consists of  $2t_e - 1$  points  $\{RS_i^{(e)}(j) = B'(i, j)\}_{i \in U}$ , from which  $j$  can compute  $FS_j(y) = B'(j, y)$ .

### 3.4 Security of CHURP

CHURP's proof of security is done under some non-standard assumptions. In particular, the protocol presumes validity of the  $(t - 1)$ -SDH (Strong Diffie Hellman) assumption, as it is required by the KZG scheme.

**Definition 5.**  $(t - 1)$ -SDH: Let  $\alpha \in \mathbb{Z}_p^*$ . Given as input a tuple  $\langle g, g^\alpha, \dots, g^{\alpha^{t-1}} \rangle \in \mathbb{G}^t$ , for every probabilistic polynomial time (PPT) adversary  $A_{t-1}$ , the probability  $\Pr[A_{t-1}(g, g^\alpha, \dots, g^{\alpha^{t-1}}) = \langle c, g^{\frac{1}{\alpha+c}} \rangle] = \epsilon(k)$  for any value of  $c \in \mathbb{Z}_p \setminus \{-\alpha\}$ , where  $\epsilon$  is a negligible function and  $k$  is a security parameter.

Additionally, in order to guarantee verifiability of threshold changes, CHURP uses a modified version of the KZG scheme based on the  $q$ -PKE assumption.

**Definition 6.**  $q$ -PKE ( $q$ -Power Knowledge of Exponent): Given values  $g, g^x, \dots, g^{x^q}, g^\alpha, \dots, g^{\alpha x^q} \in \mathbb{G}$ , it is infeasible to find  $(c, \hat{c}) \in \mathbb{G}^2$  s.t.  $\hat{c} = c^\alpha$  without knowing  $a_0, \dots, a_q$  s.t.  $c = \prod_{i=0}^q (g^{x^i})^{a_i}$  and  $\hat{c} = \prod_{i=0}^q (g^{\alpha x^i})^{a_i}$ .

The adversary  $A$  is therefore computationally bounded.  $A$  is supposed to be *active* and *adaptive*, which means that nodes can be corrupted at any time. This kind of adversary is stronger than the one in FROST, which is static and thus can corrupt only a fixed set of nodes.

Once a node is corrupted, it remains corrupted until the end of the current epoch.  $A$  can corrupt up to  $t - 1$  nodes of  $C^{(e-1)}$  and  $t - 1$  nodes of  $C^{(e)}$ .

Under the previous assumptions, CHURP satisfies the following properties:

- **Secrecy:** if  $A$  corrupts no more than  $t - 1$  nodes in a committee of any epoch,  $A$  learns no information about the secret  $s$ .
- **Integrity:** if  $A$  corrupts no more than  $t - 1$  nodes in each of the committees  $C^{(e-1)}$  and  $C^{(e)}$ , after the handoff, the shares for honest nodes can be correctly computed and the secret  $s$  remains intact.

## 4 D-FROST signature scheme

D-FROST is the result of merging FROST and CHURP, obtaining a flexible and dynamic version of FROST. To the best of our knowledge, this is the first protocol that allows us to change both the group of signers and the threshold, without changing the secret, in a signature scheme with FROST (and, more generally, in a Schnorr-based threshold scheme). The protocol is started by a group of  $n$  nodes that wish to sign messages with some threshold  $t$ . Then, the committee performs CHURP's handoff to enter the first epoch and begins to sign messages. After a predetermined amount of time, which is the duration of an epoch, the group proactivizes its shares and it potentially changes the threshold  $t$  and/or the set and number  $n$  of participants. Note that epochs should not last too long, in order to allow changes often enough. On the other hand, they should last long enough to avoid unavailability of the system.

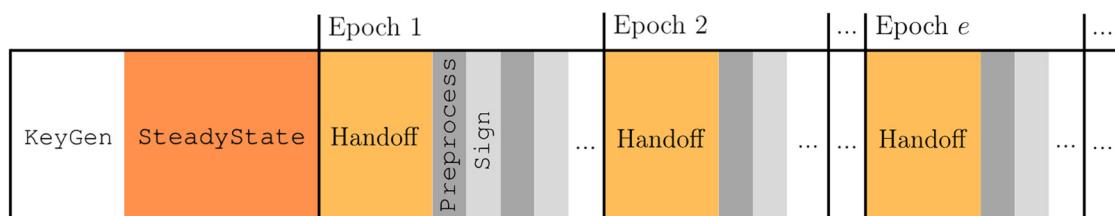
**Setup.** The setup phase selects an initial committee  $C = \{P_i\}_{i \in [n]}$  and a threshold  $t$ . Each  $P_i$  is given a private/public key pair, and public keys are known to all nodes. These keys are used to encrypt and decrypt messages in the pessimistic path. All nodes have access to the blockchain on which messages are posted. To conform with CHURP, we also suppose that  $t - 1 < \frac{n}{2}$ . The setup of the KZG scheme is performed by the committee in order to build a totally decentralized scheme. D-FROST works in a synchronous setting, since FROST requires a synchronous setting during KeyGen and Preprocess( $\pi$ ). The role of the SA is assigned to a random member of the committee.

**Protocol.** The protocol (Figure 1) is composed as follows:

- KeyGen: FROST's key generation scheme (Algorithm 1);
- SteadyState (Algorithm 13): sets the system to a steady state, so that the first committee is ready to enter CHURP;
- In each epoch, perform:
  - CHURP's handoff: Opt-CHURP or Exp-CHURP-A;
  - Until the current epoch ends or a malicious node is detected, repeat:
    - \* Preprocess( $\pi$ ) (Algorithm 2);
    - \* Sign( $m$ ) (Algorithm 3).

The initial phase of the protocol happens before entering the first epoch, and it consists of the execution of KeyGen and SteadyState. KeyGen generates  $(t, n)$ -shares of the secret  $s$  for the first committee. The protocol SteadyState creates the polynomials and commitments necessary to run CHURP. Details of the protocol SteadyState are described in the next subsection. If a misbehaving participant is detected at any point during the execution of these two algorithms, the protocol is immediately aborted to ensure system integrity. Subsequently, the nodes can restart the procedure.

At the beginning of each epoch, the shares are proactivized by CHURP, and the threshold  $t$  and/or the set and number  $n$  of participants can be changed. The protocol executes the optimistic path by default, where we suppose that there are no adversarial nodes. Then, if a participant misbehaves during Opt-CHURP, the protocol switches to the pessimistic path Exp-CHURP-A.



**Figure 1:** Dynamic-FROST. At the beginning of the execution, D-FROST runs the KeyGen algorithm and transits to the steady state. Then, it proceeds in epochs which consist of a single CHURP's handoff and multiple rounds of Preprocess( $\pi$ ) and Sign( $m$ ).

In each epoch, FROST's  $\text{Preprocess}(\pi)$  and  $\text{Sign}(m)$  can be performed multiple times. If a participant misbehaves during this phase, no more signing sessions are allowed until the end of the current epoch, following FROST's requirement to abort on misbehavior. Since we use the same signing process as FROST, D-FROST signatures are Schnorr signatures. This is another valuable property of the scheme.

## 4.1 Transition to a steady state

The main difference between KeyGen and CHURP is that the former creates a one-variable polynomial, while the latter uses a bivariate polynomial to share the secret. Moreover, in order to enter CHURP's handoff, a committee must be in a steady state, which means that the three invariants INV-SECRET, INV-STATE, and INV-COMM must hold. In particular, the KZG commitments  $\{C_{B(x,j)}\}_{j \in [2t-1]}$  should be published on-chain and each  $P_i$  should hold  $B(i, y)$  and  $\{W_{B(i,j)}\}_{j \in [2t-1]}$ . For this reason, we designed an additional protocol, called `SteadyState`, that creates  $B(x, y)$  and gives every node the necessary information. Remember that  $B(x, y)$  is a  $\langle t-1, 2t-2 \rangle$ -degree polynomial such that  $B(0, 0) = s$  and  $B(i, 0) = s_i$  for each  $i \in [n]$ .

This phase is performed after a successful execution of KeyGen, so each  $P_i$  in the first committee already holds its  $(t, n)$ -share  $s_i$  of the secret  $s$ .

`SteadyState` (Algorithm 13) works as follows. First,  $2t-1$  members of the committee are chosen and stored in the set  $\mathcal{U} = \{U_j\}_{j \in [2t-1]}$ . Then, each  $U_i \in \mathcal{U}'$ , where  $\mathcal{U}' = \{U_i\}_{i \in [t]} \subseteq \mathcal{U}$ , randomly creates a polynomial  $B(i, y)$  such that  $\deg_{B(i,y)} = 2t-2$  and  $B(i, 0) = s_i$ . The same node publishes the KZG commitment  $C_{B(i,y)}$  and sends  $(B(i, j), W'_{B(i,j)})$  to every  $U_j \in \mathcal{U}$ , where  $W'_{B(i,j)} = \text{CreateWitness}(B(i, y), j)$ . Note that, in general,  $W'_{B(i,j)} \neq W_{B(i,j)}$ . Note also that the protocol works even if this step is performed by a set  $\mathcal{U}'$  of nodes not included in  $\mathcal{U}$ . However, here, we use members of  $\mathcal{U}$  to place ourselves in the worst-case scenario for the security proof.

In the next step of the scheme,  $U_j$  verifies correctness of the received points and, if one them fails, it returns `fail`. Otherwise,  $U_j$  interpolates the points in order to compute  $B(x, j)$  and publishes  $C_{B(x,j)}$ , along with  $\{W_{B(i,j)}\}_{i \in [n]}$ . Then,  $U_j$  sends  $B(i, j)$  to each  $P_i \in \mathcal{C}$ .

At this stage, each node  $U_i$  in  $\mathcal{U}'$  verifies the correctness of the points  $\{B(i, j)\}_{j \in [2t-1]}$  it received and checks that they match the values it sent at the beginning of the protocol. At the same time, all other nodes  $P_\ell \in \mathcal{C} \setminus \mathcal{U}'$  verify the validity of the received points and interpolate them to construct  $B(\ell, y)$ . Additionally, they verify that  $B(\ell, 0) = s_\ell$  to guarantee the integrity of the group secret.

---

### Algorithm 13 `SteadyState`( $\mathcal{C}, \{s_i\}_{i \in [n]}$ )

---

**Input:** committee  $\mathcal{C} = \{P_i\}_{i \in [n]}$ ,  $(t, n)$ -shares  $\{s_i\}_{i \in [n]}$  of the secret  $s$  for each  $P_i$ .

**Output:**  $P_i$  outputs `success`,  $\{W_{B(i,j)}\}_{j \in [2t-1]}$  and  $B(i, y)$ , or `fail`.

**Public output:**  $\{C_{B(x,j)}\}_{j \in [2t-1]}$ .

1. Choose  $2t-1$  nodes in  $\mathcal{C}$  at random, denoted as  $\mathcal{U} = \{U_j\}_{j \in [2t-1]}$ .
2. For each  $i \in [t]$ :
  - 2.a  $U_i$  creates a random polynomial  $B(i, y)$  such that  $\deg_{B(i,y)} = 2t-2$  and  $B(i, 0) = s_i$ .
  - 2.b  $U_i$  computes the KZG commitment  $C_{B(i,y)}$  and publishes it on-chain.
  - 2.c  $U_i$  sends off-chain  $(B(i, j), W'_{B(i,j)})$  to  $U_j$ , where  $W'_{B(i,j)} = \text{CreateWitness}(B(i, y), j)$ , for each  $j \in [2t-1]$ .
3. For each  $j \in [2t-1]$ :
  - 3.a  $U_j$  verifies that the points it received are correct using  $\text{VerifyEval}(C_{B(i,y)}, i, B(i, j), W'_{B(i,j)})$ .
  - 3.b If any of the checks fail, return `fail`.
  - 3.c  $U_j$  interpolates  $\{B(i, j)\}_{i \in [t]}$  to construct  $B(x, j)$ , then computes  $C_{B(x,j)}$  and publishes it on-chain, along with  $W_{B(i,j)} = \text{CreateWitness}(B(x, j), i)$  for each  $i \in [n]$ .

- 3.d  $U_j$  sends off-chain  $B(i, j)$  to  $P_i$ , for each  $i \in [n]$ .
4. For each  $U_i \in \mathcal{U}' = \{U_i\}_{i \in [t]}$ :
- 4.a  $U_i$  verifies that the evaluation of  $B(x, j)$  at  $i$  returns  $B(i, j)$  via  $\text{VerifyEval}(C_{B(x, j)}, i, B(i, j), W_{B(i, j)})$ .  
 $U_i$  also verifies that  $B(i, j)$  is the same point it originally sent to  $U_j$ . If any of the checks fail return `fail`.
- 4.b  $U_i$  returns `success`.
5. For each  $\ell$  such that  $P_\ell \in C \setminus \mathcal{U}'$ :
- 5.a  $P_\ell$  verifies that the evaluation of  $B(x, j)$  at  $\ell$  returns  $B(\ell, j)$  via  $\text{VerifyEval}(C_{B(x, j)}, \ell, B(\ell, j), W_{B(\ell, j)})$ .  
If any of the checks fail return `fail`.
- 5.b  $P_\ell$  interpolates  $\{B(\ell, j)\}_{j \in [2t-1]}$  to build  $B(\ell, y)$  and verifies that  $B(\ell, 0) = s_\ell$ . If the check holds return `success`, otherwise return `fail`.

## 5 Proof of security

We know that FROST is EUF-CMA secure under the random oracle model, so our goal is to prove that D-FROST achieves the same kind of security. In Section 5.1, we prove that the transition to a steady state does not reveal additional information on the group secret, and that the constant term of the generated polynomial equals the group secret. Then, in Section 5.2, we prove that, in each epoch, the combination of CHURP with FROST signatures is still secure. Finally, in Section 5.3, we bring it all together.

SteadyState is a new scheme, so we have to prove that it is secure first. In particular, we prove that the properties of *secrecy* and *integrity* hold. We move on by proving that the combination of CHURP, Preprocess ( $\pi$ ), and  $\text{Sign}(m)$  in an arbitrary epoch is EUF-CMA secure. Finally, we claim that the whole protocol is secure thanks to the independency of the shares in different epochs.

**Assumptions.** Since the key generation scheme and the signing phase are identical to the ones in FROST, our protocol inherits its protection against some kinds of attacks: rogue-key attacks, the ROS solver, and the Drijvers attack. In particular, we can assume a *concurrent* adversary because security against the last two kinds of attack implies security against such an adversary. The attacker is also assumed to be *active* and *static*. This type of adversary is the same as in FROST. While CHURP is secure against a stronger adversary, more precisely an adaptive one, we assume a static one to preserve the security of FROST. Moreover, CHURP requires some nonstandard assumptions, i.e.,  $(t-1)$ -SDH and  $q$ -PKE, so we suppose that these hold. Therefore, D-FROST achieves the same level of security as FROST does, minus making some additional assumptions caused by the integration of CHURP.

### 5.1 Steady state

To prove the security of SteadyState, we need to show that the following properties are satisfied:

- **Secrecy:** an adversary corrupting a set of at most  $t-1$  parties cannot learn anything about the secret  $s$ ;
- **Integrity:** it must hold that  $B(0, 0) = s$ .

By proving that these hold, we assure that nodes in the first committee enter the handoff phase with the correct full shares and that no information leaks during this phase.

Note that, by corrupting  $t-1$  participants, an adversary  $A$  obtains the following information (other than the public information that is available to everyone):

- for all corrupt  $U_i \in \mathcal{U}'$ :  $\{B(i, j), W_{B(i, j)}\}_{j \in [2t-1]}$  and the full share  $B(i, y)$ ;
- for all corrupt  $U_j \in \mathcal{U}$ :  $\{B(i, j), W_{B(i, j)}\}_{i \in [t]}$  and the reduced share  $B(x, j)$ ;
- for all corrupt  $P_\ell \in C \setminus \mathcal{U}'$ :  $\{B(\ell, j)\}_{j \in [2t-1]}$  and the full share  $B(\ell, y)$ .

The following two theorems prove secrecy and integrity, respectively.

**Theorem 2.** *If a PPT adversary  $A$  corrupts no more than  $t - 1$  nodes in the committee, the information received by  $A$  in `SteadyState` is random and independent of the secret  $s$ .*

**Proof.** In the worst case, when all  $t - 1$  corrupted nodes are in  $\mathcal{U}' = \{U_i\}_{i \in [t]}$ ,  $A$  learns  $t - 1$  shares  $B(i, y)$  and  $t - 1$  shares  $B(x, j)$ . For a  $\langle t - 1, 2t - 2 \rangle$ -degree bivariate polynomial, any  $t - 1$  shares of  $B(i, y)$  and  $t - 1$  shares of  $B(x, j)$  are random and independent of  $s = B(0, 0)$ .

Moreover, based on the DL assumption, the witnesses  $W'_{B(i,j)}$  are computationally zero-knowledge by the KZG scheme, so the PPT adversary cannot learn additional information from them.  $\square$

**Theorem 3.** *After `SteadyState`, either honest nodes in  $\mathcal{U} = \{U_j\}_{j \in [2t-1]}$  hold the correct shares  $B(x, j)$  and honest nodes in  $C$  hold the correct shares  $B(i, y)$  of  $B(x, y)$  such that  $B(0, 0) = s$ , or at least  $t$  honest nodes in  $C$  output fail.*

**Proof.** Nodes  $U_j \in \mathcal{U}$  verify the validity of the points  $\{B(i, j)\}_{i \in [t]}$  received by nodes  $U_i \in \mathcal{U}'$ . If any of the checks fail and  $U_j$  is honest, then  $U_j$  returns `fail`. In this case, at least  $t$  honest members of  $\mathcal{U}$  will return `fail`, since there are at most  $t - 1$  corrupted nodes and  $|\mathcal{U}| = 2t - 1$ .

If no member of  $\mathcal{U}$  returned `fail`, then nodes  $U_i$  verify that all the shares  $B(x, j)$  computed by nodes  $U_j$  are correct, in the sense that, evaluated at  $i$ , they give the original points  $B(i, j)$ . If this is not the case, all honest nodes in  $\mathcal{U}'$  return `fail`.

At the same time, all other members  $P_\ell$  of  $C$  verify that the received points are correct. If all the checks are satisfied, they interpolate the points to obtain a polynomial  $B(\ell, y)$ . Then, each  $P_\ell$  verifies that  $B(\ell, 0) = s_\ell$ , which holds if all nodes in  $\mathcal{U}$  are honest. The reason behind this is that the points  $\{B(i, j)\}_{i \in [t], j \in [2t-1]}$  define a unique  $\langle t - 1, 2t - 2 \rangle$ -degree polynomial  $B(x, y)$  such that  $B(i, 0) = s_i$  for all  $i \in [n]$ . Otherwise, honest nodes in  $C \setminus \mathcal{U}'$  output `fail`. Note that, if some corrupted nodes are in  $\mathcal{U}'$ , then the integrity of the secret is still preserved. Indeed, even if just one node  $U_i$  uses a polynomial  $B(i, y)$  such that  $B(i, 0) \neq s_i$  to share the points, then for  $P_\ell$  the equality  $B(\ell, 0) = s_\ell$  does not hold. There are at most  $t - 1$  corrupted nodes in the whole committee and we know that in case of misbehavior all honest nodes return `fail`. Since the number of participants  $n$  satisfies  $n \geq 2t - 1$ , the number of honest nodes in  $C$  is at least  $t$ .

$B(x, y)$  is generated using Lagrange interpolation:

$$B(x, y) = \sum_{j=1}^{2t-1} B(x, j) \prod_{r \neq j} \frac{r - y}{r - j},$$

where

$$B(x, j) = \sum_{i=1}^t B(i, j) \prod_{k \neq i} \frac{k - x}{k - i}.$$

If all nodes in  $\mathcal{U}$  are honest, then for all  $j \in [2t - 1]$ ,  $B(0, j) = s_j$ , where  $\{s_j\}_{j \in [2t-1]}$  are  $(t, n)$ -shares of  $s$ . Thus, the equality  $B(0, 0) = s$  holds.  $\square$

## 5.2 Security in each epoch

We now prove EUF-CMA security for the combination of CHURP, `Preprocess`( $\pi$ ), and `Sign`( $m$ ) in an arbitrary epoch. We want to show that any PPT adversary  $F$  that corrupts no more than  $t - 1$  participants cannot forge D-FROST signatures. We prove this by contradiction: if  $F$  is able to forge D-FROST signatures, then it is possible to compute the DL of the public key  $Y$ , revealing  $s$  and thus breaking CHURP's property of secrecy.

The idea is to use  $F$  as a subroutine of a simulation that is forked by the general forking algorithm to forge two signatures  $\sigma = (R, z)$ ,  $\sigma' = (R, z')$  such that  $c \neq c'$ . Then, the DL of  $Y$  can be computed as  $s = \frac{z - z'}{c - c'}$ .

**Theorem 4.** *If the property of secrecy in CHURP holds, then D-FROST is EUF-CMA secure against an active adversary that corrupts no more than  $t - 1$  nodes during an arbitrary epoch.*

**Assumptions.** Let  $n = 2t - 1$  and suppose w.l.o.g. that the corrupted nodes are  $\{P_i\}_{i \in [t-1]}$ . Assume also that the set  $U'$  of nodes that participate in the handoff contains  $\{P_i\}_{i \in [t]}$  and that the set of signers  $S$  during Preprocess  $(\pi)$  and  $\text{Sign}(m)$  is  $\{P_i\}_{i \in [t]}$ . Other nodes honestly follow CHURP's protocol, and they do not take part in the signing phase, so there is no need to further specify their behavior. Since this is the worst-case scenario (the adversary has the most power possible, as there is only one honest node and the adversary controls the others), this assumption is reasonable. We analyze security using the optimistic path of CHURP, but the proof in the pessimistic case is essentially the same.

**Proof.** The algorithms used in our proof are the following:

- $F$  is a forger that, with non-negligible probability  $\epsilon$  and in polynomial time  $\tau$ , can forge a signature for a public key  $Y$  in one epoch of D-FROST by corrupting  $t - 1$  nodes, with the limitation of making at most  $n_r$  random oracle queries. One of the corrupted nodes has the role of SA;
- $A$  simulates the honest participant  $P_t$  and answers to random oracle queries made by  $F$  during the handoff phase, Preprocess( $\pi$ ), and  $\text{Sign}(m)$ ;
- $D$  is the coordination algorithm that, given the public key  $Y$ , invokes the others in order to compute  $s = \text{dlog}(Y)$ .

Let us take a closer look at how these algorithms work.

**A.** During the handoff phase,  $A$  simulates the honest participant  $P_t$ . Note that  $P_t$  is part of  $U'$ , so during Opt-ShareReduce the old committee sends  $P_t$  all the necessary information to take part in CHURP's protocol correctly. In particular,  $A$  obtains  $P_t$ 's secret share  $s_t$ . Then,  $A$  just needs to follow the scheme as an honest node would do.

To answer  $F$ 's queries, the algorithm initializes an array  $T$  where it will store its responses and a counter  $ctr = 0$ . Then, every time the forger asks for the value of  $H(g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)})$ ,  $A$  proceeds as follows: if  $T(g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)}) = \perp$ , set  $T(g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)}) = h_{ctr}$ ,  $ctr = ctr + 1$ , then return  $T(g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)})$ . Note that, as in the general forking lemma,  $T(g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)}) = \perp$  indicates that the query with input  $g^{s_j} || C_{Z_j} || W_{Z_j(0)} || C_{B'(x,j)}$  has not been answered previously. If that is the case,  $A$  responds with the first available value  $h_{ctr}$ .

$A$  also simulates  $P_t$  during Preprocess( $\pi$ ) and  $\text{Sign}(m)$ , knowing  $s_t$ . The algorithm initializes a counter  $ctr = 0$  and two arrays  $T_1, T_2$  to keep track of the answers it already gave to  $F$ 's queries. If there is no value stored in  $T_i$  under key  $x$ ,  $T_i(x) = \perp$ .  $A$  also initializes an array  $J_2$  to memorize the index  $j$  of  $h_j$  such that  $T_2(R || Y || m) = h_j$ .

$A$  answers  $F$ 's queries using random values  $h_1, \dots, h_{n_r}$  given by the general forking algorithm  $\text{GF}_A$  as follows:

- $H_1(i || m || B)$ : if  $T_1(i || m || B) = \perp$ , set  $T_1(i || m || B) = h_{ctr}$ ,  $ctr = ctr + 1$ . Return  $T_1(i || m || B)$ ;
- $H_2(R || Y || m)$ : if  $T_2(R || Y || m) = \perp$ , set  $T_2(R || Y || m) = h_{ctr}$ ,  $J_2(R || Y || m) = ctr$ ,  $ctr = ctr + 1$ . Return  $T_2(R || Y || m)$ .

After running  $F$ ,  $A$  verifies that  $F$  succeeded in forging a signature  $\sigma = (R, z)$  on a message  $m$ . This happens when  $F$  returns  $(R, z)$  such that  $R = Y^c g^z$ ,  $c = T_2(R || Y || m)$ . If this is the case,  $A$  returns  $(J, \sigma)$ , where  $J$  is such that  $h_j = T_2(R || Y || m)$ . Otherwise,  $A$  outputs  $(\perp, \perp)$ .

**D.** First,  $D$  (Algorithm 14) executes  $\text{GF}_A(Y)$  to obtain two signatures with the required properties. Remember that  $F$  forges a valid signature with non-negligible probability  $\epsilon$ , so  $A$  succeeds with the same probability. Therefore, thanks to the general forking lemma,  $\text{GF}_A$  outputs  $(1, h_j, h'_j, \sigma, \sigma')$  with probability  $frk \geq \epsilon \cdot (\frac{\epsilon}{n_r} - \frac{1}{h})$ , where  $n_r$  is the number of random elements  $(h_1, \dots, h_{n_r}) \in O^{n_r}$  given as input to  $A$  and  $h$  is the size of  $O$ . Note that  $frk$  is non-negligible, since  $\frac{\epsilon^2}{n_r}$  and  $\frac{\epsilon}{h}$  are both non-negligible, but the first term grows faster and therefore dominates.

With negligible probability  $1 - frk$ ,  $\text{GF}_A$  outputs  $\perp$  and  $D$  returns  $\perp$ . Otherwise,  $D$  computes and returns  $s = \frac{z' - z}{h'_j - h_j}$ . This value is the DL of  $Y$ . In fact, the signatures are  $\sigma = (R, z)$  and  $\sigma' = (R, z')$ , so they both use the same commitment  $R$ . This is true because  $\text{GF}_A$  returned  $J = J'$ , which means that  $F$  forged two signatures on

the same message  $m_j$ , so in both executions  $R$  is calculated as  $R = \prod_{i \in S} D_{ij} E_{ij}^{\rho_i}$ , where  $\rho_i = H_1(i \| m_j \| B)$ . The next thing the protocol does after calculating  $R$  is to compute  $c = H_2(R \| Y \| m_j)$ . The trick is that, starting exactly from this query, the simulation gives different answers ( $h_j \neq h'_j$ ) to  $F$ 's queries in the two executions. This way, we obtain  $g^z = RY^{h_j}$  from the first run of  $A$  and  $g^{z'} = RY^{h'_j}$  from the second one. So,  $R = g^z g^{-s \cdot h_j} = g^{z'} g^{-s \cdot h'_j}$  holds, and it implies  $z - s \cdot h_j = z' - s \cdot h'_j$ . Thus,  $D$  can compute  $s$  as  $\frac{z' - z}{h'_j - h_j}$ .  $\square$

---

**Algorithm 14**  $D(Y)$ 


---

**Input:** Group's public key  $Y$ .

**Output:**  $s = dlog(Y)$ .

1.  $(1, h_j, h'_j, \sigma, \sigma')$  or  $\perp \leftarrow \text{GF}_A(Y)$
  2. **If**  $\perp$ , **then return**  $\perp$ .
  3. Parse  $\sigma, \sigma'$  as  $(R, z), (R, z')$ .
  4.  $s = \frac{z' - z}{h'_j - h_j}$
  5. **Return**  $s$ .
- 

### 5.3 Security of D-FROST

The last step is to show that the whole protocol is secure, in the sense that it is EUF-CMA secure and it preserves the secrecy and integrity of  $s$ .

Secrecy and integrity hold in KeyGen, thanks to Pedersen's DKG, and in each epoch thanks to CHURP. In Section 5.1, we proved that these properties are valid also in SteadyState, so they hold throughout the duration of the protocol.

CHURP assures that the shares in one epoch are independent of the old ones, so the adversary does not obtain any additional data by putting together information learned during different epochs. In particular, information learned in previous epochs cannot be used for the purpose of forging signatures in the current epoch. Therefore, proving the unforgeability of D-FROST signatures reduces to what is proved in Section 5.2, concluding our proof of security.

## 6 Conclusion

Threshold signatures are applicable to a variety of use cases, and FROST works well for this purpose with its Schnorr-based algebraic simplicity and its communication efficiency. In order to extend the possible applications to cases where both the committee and the threshold of signers are variable, in this work, we devised a new protocol, which periodically updates the committee and, possibly, the threshold, using CHURP, a DPSS. We also proved that combining the two protocols preserves their security.

**Future work.** Possible improvements to our work can be done in two directions: achieving robustness, i.e., guaranteeing that SteadyState and signing sessions end in a valid state, and adapting the protocol to epochs of variable length. The lack of robustness in D-FROST stems from the presence of SteadyState and FROST, both of which are non-robust schemes. When transitioning to a steady state, the protocol might abort due to a misbehaving node, as outlined in Section 4: to prevent this, SteadyState might switch to a pessimistic path on misbehavior, similar to CHURP (see Section 3.3), to expel corrupted nodes. On the other hand, the execution of the protocol in each epoch might abort during the signing phase, ending with no valid signature. However, as mentioned in Section 2.1, robust FROST variants, like ROAST, have been proposed. A possible extension to our work is to join ROAST with CHURP, and include a pessimistic case Exp-SteadyState for the SteadyState algorithm. While the merge of ROAST with CHURP is straightforward and does not provide a drastic change to

the protocol, the design of Exp-SteadyState and the security of the resulting scheme remain to be assessed precisely. Furthermore, since CHURP is periodically executed at fixed time intervals, it remains to see what happens if committee and threshold changes are done only when requested (and not periodically), i.e., if the execution of CHURP is delayed and more FROST signatures are produced in the meantime. A possible way to introduce this kind of flexibility is to use a consensus algorithm.

**Acknowledgement:** This work was presented at CIFRIS24, <https://www.decifris.it/cifris24>, the second congress of De Cifris. The work was conducted when the corresponding author (Annalisa Cimatti) was at Roma Tre University. She is now affiliated with the University of Bern, Switzerland. All views are those of the authors and do not necessarily reflect the position of Bank of Italy.

**Funding information:** This work was partially supported by Project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

**Author contributions:** F.D.S., G.G., and M.P. designed and supervised the project. A.C., F.D.S., G.G., and M.N. worked on the main conceptual ideas and developed the proof outline. S.G., M.I., and A.M. provided useful insights for the discussion and actively participated in the investigation phase. A.C. contributed to the original writing of the manuscript. F.D.S., G.G., M.N., and M.P. contributed to the review and editing of the writing. All authors have accepted responsibility for the entire content of this manuscript and consented to its submission to the journal, reviewed all the results, and approved the final version of the manuscript.

**Conflict of interest:** Prof. Marco Pedicini is a member of the Editorial Board of the Journal of Mathematical Cryptology but was not involved in the review process of this article.

## References

- [1] Komlo C, Goldberg I. FROST: flexible round-optimized Schnorr threshold signatures. In: Dunkelman O, Jacobson Jr MJ, O'Flynn C, editors. *Selected areas in cryptography*. Cham: Springer; 2021. p. 34–65.
- [2] Sedghighadikolaei K, Yavuz AA. A comprehensive survey of threshold digital signatures: NIST standards, post-quantum cryptography, exotic techniques, and real-world applications. *arXiv:2311.05514 [cs.CR]*, 2024. <https://arxiv.org/abs/2311.05514>.
- [3] Ruffing T, Ronge V, Jin E, Schneider-Bensch J, Schröder D. ROAST: Robust Asynchronous Schnorr Threshold Signatures. In: Proc. of ACM SIGSAC CCS '22. New York, NY, USA: ACM; 2022. p. 2551–64.
- [4] Benedetti M, De Sclavis F, Favorito M, Galano G, Giammusso S, Muci A, et al. Certified Byzantine consensus with confidential quorum for a bitcoin-derived permissioned DLT. In: Proc. of the 5th Distributed Ledger Technology Workshop; 2023.
- [5] Laing TM, Stinson DR. A survey and refinement of repairable threshold schemes; 2017. *Cryptology ePrint Archive*, 2017/1155.
- [6] Herzberg A, Jarecki S, Krawczyk H, Yung M. Proactive secret sharing or: how to cope with perpetual leakage. In: Proc. of CRYPTO '95. Berlin, Heidelberg: Springer-Verlag; 1995. p. 339–52.
- [7] Shamir A. How to share a secret. *Commun ACM*. 1979 nov;22(11):612–3.
- [8] Maram SKD, Zhang F, Wang L, Low A, Zhang Y, Juels A, et al. CHURP: dynamic-committee proactive secret sharing. In: Proc. of ACM SIGSAC CCS '19. ACM; 2019. p. 2369–86.
- [9] Shoup V. Practical threshold signatures. In: Preneel B, editor. *Advances in Cryptology - EUROCRYPT 2000*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2000. p. 207–20.
- [10] Cachin C, Kursawe K, Shoup V. Random oracles in constantinople: practical asynchronous byzantine agreement using cryptography. *J Cryptol*. 2005;18(3):219–46.
- [11] Golan Gueta G, Abraham I, Grossman S, Malkhi D, Pinkas B, Reiter M, et al. SBFT: A scalable and decentralized trust infrastructure. In: Proc. of IEEE/IFIP DSN '19; 2019. p. 568–80.
- [12] Yin M, Malkhi D, Reiter MK, Gueta GG, Abraham I. HotStuff: BFTConsensus with linearity and responsiveness. In: Proc. of ACM PODC '19. New York, NY, USA: ACM; 2019. p. 347–56.
- [13] Thai QT, Yim JC, Yoo TW, Yoo HK, Kwak JY, Kim SM. Hierarchical byzantine fault-tolerance protocol for permissioned blockchain systems. *J Supercomput*. 2019;75(11):7337–65.
- [14] Gennaro R, Jarecki S, Krawczyk H, Rabin T. Robust threshold DSS signatures. *Inform Comput*. 2001;164(1):54–84.
- [15] Gennaro R, Goldfeder S, Narayanan A. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: Manulis M, Sadeghi AR, Schneider S, editors. *Applied cryptography and network security*. Cham: Springer; 2016. p. 156–74.

- [16] Gennaro R, Goldfeder S. Fast multiparty threshold ECDSA with fast trustless setup. In: Proc. of ACM SIGSAC CCS '18. ACM; 2018. p. 1179–94.
- [17] Gennaro R, Goldfeder S. One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, 2020/540. 2020.
- [18] Lindell Y, Nof A. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: Proc. of ACM SIGSAC CCS '18. ACM; 2018. p. 1837–54.
- [19] Noack A, Spitz S. Dynamic threshold cryptosystem without group manager. IACR Cryptol ePrint Arch. 2008:380.
- [20] Aumasson JP, Hamelink A, Shlomovits O. A survey of ECDSA threshold signing. IACR Cryptol ePrint Arch. 2020;2020:1390.
- [21] Boneh D, Lynn B, Shacham H. Short signatures from the weil pairing. J Cryptol. 2004;17(4):297–319.
- [22] Schnorr CP. Efficient identification and signatures for smart cards. In: Conference on the Theory and Application of Cryptology. Springer; 1989. p. 239–52.
- [23] Ergezer S, Kinkelin H, Rezabek F. A survey on threshold signature schemes. Seminar IITM SS 20, Network Architectures and Services, Technical University of Munich; 2020.
- [24] Boldyreva A. Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-Group signature scheme. In: Desmedt YG, editor. Public Key Cryptography–PKC 2003. Berlin, Heidelberg: Springer; 2002. p. 31–46.
- [25] Tomescu A, Chen R, Zheng Y, Abraham I, Pinkas B, Gueta GG, et al. Towards scalable threshold cryptosystems. In: Proc. of the 2020 IEEE Symposium on Security and Privacy; 2020. p. 877–93.
- [26] Crites E, Komlo C, Maller M. How to prove Schnorr assuming Schnorr: security of multi- and threshold signatures. Cryptology ePrint Archive. 2021.
- [27] Hu B, Zhang Z, Chen H, Zhou Y, Jiang H, Liu J. DyCAPS: asynchronous dynamic-committee proactive secret sharing; 2022. Cryptology ePrint Archive, Paper 2022/1169. <https://eprint.iacr.org/2022/1169>.
- [28] Schultz D, Liskov B, Liskov M. MPSS: Mobile proactive secret sharing. ACM Trans Inf Syst Secur. 2010;13(4):32.
- [29] Vassantlal R, Alchieri EAP, Ferreira B, Bessani AN. COBRA: Dynamic proactive secret sharing for confidential BFT services. Proceedings of 2022 IEEE Symposium on Security and Privacy (SP). 2022:1335–53.
- [30] Yurek T, Xiang Z, Xia Y, Miller A. Long live the honey badger: Robust asynchronous DPSS and its applications. In: Proc. of 32nd USENIX Security. vol. 8; 2023. p. 5413–30.
- [31] Benhamouda F, Gentry C, Gorbunov S, Halevi S, Krawczyk H, Lin C, et al. Can a public blockchain keep a secret?; 2020. Cryptology ePrint Archive, 2020/464.
- [32] Goyal V, Kothapalli A, Masserova E, Parno B, Song Y. Storing and retrieving secrets on a blockchain; 2020. Cryptology ePrint Archive, 2020/504.
- [33] Battagliola M, Longo R, Meneghetti A. Extensible decentralized secret sharing and application to Schnorr signatures; 2022. Cryptology ePrint Archive, Paper 2022/1551. <https://eprint.iacr.org/2022/1551>.
- [34] Benhamouda F, Halevi S, Krawczyk H, Ma Y, Rabin T. SPRINT: High-Throughput Robust Distributed Schnorr Signatures; 2023. Cryptology ePrint Archive, 2023/427.
- [35] Bellare M, Boldyreva A, Staddon J. Multi-recipient encryption schemes: Security notions and randomness re-use. In: PKC. vol. 2003; 2003. p. 85–99.
- [36] Drijvers M, Edalatnejad K, Ford B, Kiltz E, Loss J, Neven G, et al. On the Security of Two-Round Multi-Signatures; 2018. Cryptology ePrint Archive, 2018/417.
- [37] Benhamouda F, Lepoint T, Loss J, Orrù M, Raykova M. On the (in)security of ROS; 2020. Cryptology ePrint Archive, 2020/945.
- [38] Bellare M, Neven G. Multi-signatures in the plain public-Key model and a general forking lemma. In: Proc. of ACM CCS '06. ACM; 2006. p. 390–9.
- [39] Kate A, Zaverucha GM, Goldberg I. Constant-size commitments to polynomials and their applications. In: Abe M, editor. Advances in Cryptology - ASIACRYPT 2010. Berlin, Heidelberg: Springer; 2010. p. 177–94.