Dynamic-FROST: Schnorr Threshold Signatures with a Flexible Committee

Annalisa Cimatti$^{1[0009-0009-6942-5924]}$, Francesco De Sclavis$^{2[0009-0004-1318-3878]}$, Giuseppe Galano$^{23[0009-0008-3251-6606]}$, Sara Giammusso$^{2[0009-0009-9355-3416]}$, Michela Iezzi^{2}, Antonio Muci^{2}, Matteo Nardelli$^{2[0000-0002-9519-9387]}$, and Marco Pedicini$^{1[0000-0002-9016-074X]}$

^{1} Roma Tre University, Italy
`marco.pedicini@uniroma3.it, ann.cimatti@stud.uniroma3.it`
^{2} Bank of Italy*, Italy
`{francesco.desclavis, giuseppe.galano2, sara.giammusso, michela.iezzi, antonio.muci, matteo.nardelli}@bancaditalia.it`
^{3} University of Pisa, Italy

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 paper, we present Dynamic-FROST (D-FROST, for short) 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

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

* All views are those of the authors and do not necessarily reflect the position of Bank of Italy.

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.

Schnorr threshold signatures and FROST. Among threshold signature schemes, FROST [23] 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 [31], e.g., based on RSA or ECDSA, the characteristics of Schnorr signatures facilitate more straightforward implementations; for this reason, Schnorr signatures have been recently included in the Bitcoin codebase[4]. 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) [23]; it achieves efficient communication by reducing the protocol to just two rounds. Some variants, like ROAST [28], 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[5]. Second, threshold signatures can be employed by a committee of validators in a permissioned blockchain to authenticate new blocks, as outlined in [5]. 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. 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,

[4] Schnorr Signatures for secp256k1: https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki.

[5] Human Rights Foundation Bounties, accessed on April 07, 2024: https://hrfbounties.org/

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* [24]. 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 Herzberg et al.'s *proactive secret sharing* (PSS) [21], that periodically updates the secret shares while leaving the group secret unchanged, thus reinforcing Shamir's secret sharing [32]. 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. Proactive secret sharing 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) or *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 Proactive Secret Sharing schemes and selected CHURP as our favorite candidate.

CHURP Proactive Secret Sharing. CHURP is a dynamic PSS 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) which 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 [21]. More details can be found in [26] and in Section 3.3.

Our contribution. In this paper, we introduce a novel protocol called *Dynamic FROST* (D-FROST), which combines FROST with CHURP [26] to accomodate dynamic committees and threshold changes in a FROST threshold signature. The idea behind CHURP is based upon a technique outlined in [21], which is based on two-variables polynomials and places it far away from FROST which uses one-variable polynomials. To combine these two approaches, we define a new scheme which 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.

Paper organization. The rest of this paper is organised 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

Different threshold signature schemes have been defined so far. Shoup [33] defined one of the most used threshold signature schemes, which is based on RSA (e.g., [11,19,37,34]). It requires a trusted, centralized dealer for key generation, and then uses non-interactive signature share generation and signature verification protocols.

Gennaro et al. [18] propose 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 [17], where Gennaro et al. propose a dealer-less approach supporting the case $n \geq t$. However, DKG is costly and impractical. Then, Gennaro and Goldfeder [15,16] presented an ECDSA-based protocol supporting efficient DKG, which obtains faster signing than [17] and requires less data to be transmitted. In a closely related work, Lindell et al. [25] propose an efficient threshold ECDSA scheme, which employs different methods to neutralize any adversarial behavior. Differently from [15], this protocol revolves around a modification of the ElGamal encryption scheme. Using an ElGamal signature scheme, Noack et al. [27] 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 [1]. Although ECDSA is fast and secure, aggregated signatures cannot be easily obtained with it.

Conversely, BLS [10] and Schnorr [29] schemes can be easily transformed into threshold schemes by supporting the sum of partial signatures with no overhead [14]. In particular, Boldyreva [9] 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 to periodically renew the secret shares.

Recently, Tomescu et al. [35] proposed a more efficient BLS signature scheme, that improves signing and verification time. Threshold BLS signature schemes rely on pairing-based cryptography [10], 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[6]. Komlo and Goldberg [23] 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 [12]. Ruffing et al. [28] proposed ROAST (RObust Asynchronous Schnorr Threshold signatures), a wrapper protocol around FROST that provides liveness guarantees in 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.

2.2 Discussing Possible Solutions

To find the solution that best fits our problem, we studied many *dynamic proactive secret sharing* (DPSS) schemes, namely PSS schemes that involve dynamic committees. These can be classified in 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 the asynchronous and partially synchronous DPSS schemes we considered (e.g. Schultz's MPSS [30], COBRA [36], Robust Asynchronous DPSS [38]) 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 opted for a synchronous protocol.

To the best of our knowledge, the most recent and efficient synchronous PSS schemes with dynamic committees are CHURP [26], Benhamouda et al.'s [6] and Goyal et al.'s [20]. 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

[6] https://en.bitcoin.it/wiki/BIP_0340

we want the secret to be hidden from everybody, none of these schemes suits our purpose. Thus, we selected CHURP as the best solution, which is a highly efficient and decentralized protocol with a large upper bound on the threshold.

As stated in the introduction, 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 [2], 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 [7], 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 [23] 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 signature aggregator 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. Specific algorithms are detailed in Appendix A.

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 $[n] \coloneqq \{1, \ldots, n\}$. 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 $Y_i \coloneqq g^{s_i}$. Every time a misbehaving node is detected, FROST aborts in order to avoid rogue key attacks [3]. 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 $\lambda_i \coloneqq \prod_{j\neq i} \frac{j}{j-i}$. 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 $Y \coloneqq g^s$. 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 4) is Pedersen's DKG with a slight modification, that consists in a zero-knowledge proof of knowledge, computed by each participant, of their corresponding secret $a_{i0} \coloneqq f_i(0)$. 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(`π`)`, which is a preprocessing stage reported in Algorithm 5. Here, each P_i creates and publishes π pairs of commitments $(D_{ij}, E_{ij}) \coloneqq (g^{d_{ij}}, g^{e_{ij}})$, where d_{ij}, e_{ij} are random elements of $\mathbb{Z}_q$. Every one of these 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, this phase is executed again. The last part of the protocol is the signing phase. During `Sign(m)`, which is described in Algorithm 6, SA selects the set S of nodes that will sign the message m. This set is made of α signing nodes, where $t \leq \alpha \leq n$. Then, SA gets the next available commitment for each P_i and creates $B \coloneqq \langle (i, D_i, E_i) \rangle_{i \in S}$. Once all nodes have received B, they validate m and compute $\rho_l \coloneqq H_1(l, m, B)$, $l \in S$, where H_1 is a hash function mapping to $\mathbb{Z}_q^*$. Next, they derive the group commitment $R \coloneqq \prod_{l \in S} D_l \cdot (E_l)^{\rho_l}$ and the challenge $c \coloneqq H_2(R, Y, m)$, where H_2 is also a hash function. Then, each P_i computes $z_i \coloneqq d_i + e_i \cdot \rho_i + \lambda_i \cdot s_i \cdot c$ and returns it to SA. The signature aggregator verifies the validity of z_i, for $i \in S$. If every response is correct, SA computes $z \coloneqq \sum_{i \in S} z_i$. Finally, the signature $\sigma \coloneqq (R, z)$ is published.

Notice 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(`π`)`), FROST requires a synchronous network, while the signing phase can be performed asynchronously.

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 1. *A signature scheme is* existentially unforgeable under chosen message attack*, or* ***EUF-CMA secure****, if the adversary can not 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 [13] and the ROS solver [8]. 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 1), which we denote by GF_A, and the general forking lemma by Bellare and Neven [4]. 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 $x, h_1, \ldots, h_q$ returns a pair, the first element of which is an integer in the range $0, \ldots, q$ and the second element of which we refer to as a side output. Let IG be a randomized algorithm that we call the input generator. The accepting probability of A, denoted acc, is defined as the probability that $J \geq 1$ in the experiment*

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

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

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

Algorithm 1 $GF_A(x)$

```
1. Pick random coins $\rho$
2. $h_1, \ldots, h_q \xleftarrow{\$} O$
3. $(J, \sigma)$ or $\perp \leftarrow A(x, \{h_1, \ldots, h_q\}; \rho)$
4. If $J = 0$, then return $(0, \epsilon, \epsilon)$
5. $h'_J, \ldots, h'_q \xleftarrow{\$} O$
6. $(J', \sigma')$ or $\perp \leftarrow A(x, \{h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_q\}; \rho)$
7. If $J \neq J' \vee h_J = h'_J$ 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 adaptevely selects corrupted nodes during the execution of the protocol.

3.3 CHURP

CHURP [26] 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 proactivization phase (*handoff*) in which the committee passes the secret to a new possibly disjoint group $C' = \{P'_i\}_{i \in [n']}$.

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*.

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 proactivization polynomial
U'	subset of nodes chosen to participate in handoff
λ_i	Lagrange coefficients

Invariants. To preserve integrity of the secret while transmitting it to a new committee, CHURP makes use of the KZG scheme [22]. KZG is a polynomial commitment scheme that allows a user to commit to a polynomial $P(x)$ and to prove that $P(i)$ is the result of the evaluation of $P(x)$ at some index i.

After a successful handoff, the system is in a *steady state*, which means that the following three invariants must hold:

- 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), by 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 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 communication happens on-chain, as explained below.

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 communication takes 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 Discrete Logarithm (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 Appendix B and Appendix C respectively.

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 Appendix E we explain how threshold changes are managed.

Opt-CHURP is divided in three stages. The first one is `Opt-ShareReduce` (Algorithm 7), 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 validity of the points, nodes use the KZG commitments and witnesses produced by members of the old committee.

The second phase of Opt-CHURP is `Opt-Proactivize` (Algorithm 8), during which the polynomial $B(x,y)$ gets 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 $B'(x,j) \coloneqq B(x,j) + Q(x,j)$. Participants prepare all necessary information to allow the others to verify validity of the updated shares. Specifically, each U'_j computes $Z_j(x) \coloneqq R_j(x) - s_j$ 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. Notice 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 9). 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 get 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.

3.4 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 assumption, as it is required by the KZG scheme.

Definition 2. $(t-1)$-***SDH*** *($(t-1)$-Strong Diffie Hellman): Let $\alpha \in \mathbb{Z}_p^*$. Given as input a tuple $\langle g, g^\alpha, \ldots, g^{\alpha^{t-1}}\rangle \in \mathbb{G}^t$, for every probabilistic polynomial time (PPT) adversary A_{t-1}, the probability $Pr[A_{t-1}(g, g^\alpha, \ldots, g^{\alpha^{t-1}}) = \langle c, g^{\frac{1}{\alpha+c}}\rangle] = \epsilon(k)$ for any value of $c \in \mathbb{Z}_p \backslash \{-\alpha\}$, where ϵ 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 3. q-***PKE*** *(q-Power Knowledge of Exponent): Given values $g, g^x, \ldots, g^{x^q}, g^\alpha, \ldots, g^{\alpha x^q} \in \mathbb{G}$, it is infeasible to find $(c, \hat{c}) \in \mathbb{G}^2$ s.t. $\hat{c} = c^\alpha$ without knowing $a_0, \ldots, a_q$ 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 The 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 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. Notice 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(`π`)`. The role of the signature aggregator SA is assigned to a random member of the committee.

Protocol. The protocol is composed as follows:

- `KeyGen`: FROST's key generation scheme (Algorithm 4);
- `Opt-SteadyState` (Algorithm 2) or `Exp-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(`π`)` (Algorithm 5);
  * `Sign(m)` (Algorithm 6).

The initial phase of the protocol happens before entering the first epoch and it consists of the execution of `KeyGen` and `Opt-SteadyState` (or `Exp-SteadyState`). `KeyGen` generates (t, n)-shares of the secret s for the first committee. The protocol `Opt-SteadyState` (or `Exp-SteadyState`) creates the polynomials and commitments necessary to run CHURP. Details of the protocols `Opt-SteadyState` and `Exp-SteadyState` are described in the next subsection. 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-SteadyState` or Opt-CHURP, the protocol switches to the pessimistic path (`Exp-SteadyState` and Exp-CHURP-A).

In each epoch, FROST's `Preprocess(`π`)` and `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 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 $i \in [n]$.

The optimistic path of this protocol is called `Opt-SteadyState` and the pessimistic one is `Exp-SteadyState`. As in CHURP, in the pessimistic case nodes communicate on-chain only, while in the optimistic one they have access to secure peer-to-peer channels.

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

`Opt-SteadyState` (Algorithm 2) works as follows. First, $2t - 1$ members of the committee are chosen and stored in the set $U' = \{U'_j\}_{j\in[2t-1]}$. Then, each P_i 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 $C_{B(i,y)}$ and sends $(B(i, j), W'_{B(i,j)})$ to U'_j. Next, U'_j verifies correctness of the received points and, if one them fails, switches to `Exp-SteadyState`. U'_j interpolates the points in order to compute

$B(x,j)$ and publishes $(C_{B(x,j)}, W_{B(i,j)})$. Finally, P_i checks correctness of $B(x,j)$ for all $j \in [2t-1]$, in order to guarantee integrity of the secret.

`Exp-SteadyState` works similarly, so we omit the explanation and report its description in Appendix C.

Algorithm 2 `Opt-SteadyState` $(C, \{s_i\}_{i\in[n]}, \{(sk_i, pk_i)\}_{i\in[n]})$

Input: committee $C = \{P_i\}_{i\in[n]}$, (t,n)-shares $\{s_i\}_{i\in[n]}$ of the secret s and a pair of private and public keys (sk_i, pk_i) 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. Order $\{P_i\}_{i\in[n]}$ based on lexicographic order of their public keys.
2. Choose the first $2t-1$ nodes, denoted as $U'$.
3. $P_i$ creates a random polynomial $B(i,y)$ such that $deg_{B(i,y)} = 2t-2$ and $B(i,0) = s_i$.
4. $P_i$ computes the KZG commitment $C_{B(i,y)}$ and publishes it on-chain.
5. $P_i$ sends off-chain $(B(i,j), W'_{B(i,j)})$ to $U'_j$, where $W'_{B(i,j)} =$ CreateWitness$(B(i,y), j)$.
6. $U'_j$ verifies that the points he received are correct using VerifyEval$(C_{B(i,y)}, i, B(i,j), W'_{B(i,j)})$.
7. If any of the checks fail, switch to Exp-SteadyState.
8. $U'_j$ 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 $W_{B(i,j)} =$ CreateWitness$(B(x,j), i)$.
9. $P_i$ verifies that the evaluation of $B(x,j)$ at $i$ returns $B(i,j)$ via VerifyEval$(C_{B(x,j)}, i, B(i,j), W_{B(i,j)})$. If any of the checks fail return fail, otherwise return success.
```

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.

`Opt-SteadyState` and `Exp-SteadyState` are new schemes, so we have to prove that these are 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(`π`)` and `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 SteadyState

Opt-SteadyState. To prove the security of `Opt-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.

Notice 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 P_i: sk_i, $\{B(i,j), W'_{B(i,j)}\}_{j\in[2t-1]}$ and the full share $B(i,y)$;
- for all corrupt U'_j: $\{B(i,j), W'_{B(i,j)}\}_{i\in[n]}$ and the reduced share $B(x,j)$.

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 committe, the information received by A in* ***`Opt-SteadyState`*** *is random and independent of the secret s.*

Proof. In the worst case, when all $t-1$ corrupted nodes are in U', A learns $t-1$ shares $B(i,y)$ and $t-1$ shares $B(x,j)$. For a $\langle t-1, 2t-2\rangle$-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. □

Theorem 3. *After* ***`Opt-SteadyState`****, either all nodes $\{U'_j\}_{j\in[2t-1]}$ hold the correct shares $B(x,j)$ of $B(x,y)$ such that $B(0,0) = s$ and the commitments $C_{B(x,j)}$ are on-chain, or at least t honest nodes in C output* ***`fail`****.*

Proof. As the number of nodes $n \geq 2t-1$ and the number of corrupted nodes is at most $t-1$, each node U'_j receives at least t correct shares $B(i,j)$. As the degree on the first variable of $B(x,y)$ is $t-1$, U'_j can reconstruct $B(x,j)$ successfully.

Then, nodes in C verify that all shares $B(x,j)$ are correct. If that is the case, all honest nodes in C (at least t) output `success` and the algorithm produces $2t-1$ valid $\{B(x,j), C_{B(x,j)}\}$. Otherwise, honest nodes output `fail`.

$B(x,y)$ 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

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

We know that $\forall i \in [n]$ $B(i,0) = s_i$, where $\{s_i\}_{i\in[n]}$ are (t,n)-shares of s, so the equality $B(0,0) = s$ holds. $\square$

Exp-SteadyState. Compared to `Opt-SteadyState`, the only additional information A learns is $g^{B(i.j)}$, which is useless due to the DL assumption. Therefore, both properties are still valid in `Exp-SteadyState`.

5.2 Security in Each Epoch

We now prove EUF-CMA security for the combination of CHURP, `Preprocess(`π`)` 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 discrete logarithm 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 in order to forge two signatures $\sigma = (R,z)$, $\sigma' = (R,z')$ such that $c \neq c'$. Then, the discrete logarithm 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 take part in the handoff contains $\{P_i\}_{i\in[t]}$ and that the set of signers S during `Preprocess(`π`)` and `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:

- $\boldsymbol{F}$ is a forger that, with non-negligible probability ϵ and in polynomial time τ, 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;
- $\boldsymbol{A}_1$ simulates the honest participant P_t and answers to random oracle queries made by $\boldsymbol{F}$ during the handoff phase. Then, A_1 outputs P_t's secret share s_t;
- $\boldsymbol{A}_2$ simulates the honest participant P_t and answers to random oracle queries made by $\boldsymbol{F}$ during `Preprocess(`π`)` and `Sign(m)`;
- $\boldsymbol{D}$ is the coordination algorithm that, given the public key Y, invokes the others in order to compute $s = dlog(Y)$.

Let's take a closer look at how these algorithms work.

$\boldsymbol{A}_1$**.** During the handoff phase, A_1 simulates the honest participant P_t. Notice 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. Then, A_1 just needs to follow the scheme as an honest node would do. At the end, A_1 will return P_t's secret share s_t.

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_1 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 $T(g^{s_j}||C_{Z_j}||W_{Z_j(0)}||C_{B'(x,j)})$.

$\boldsymbol{A}_2$**.** It simulates P_t during `Preprocess(`π`)` and `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_2 also initializes an array J_2 to memorize the index j of h_j such that $T_2(R, Y, m) = h_j$.

A_2 answers F's queries using random values $h_1, ..., h_{n_r}$ given by the general forking algorithm GF_{A_2} 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 $T_2(i, m, B)$.

After running F, A_2 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_2 returns (J, σ), where J is such that $h_J = T_2(R, Y, m)$. Otherwise, A_2 outputs $\perp$.

$\boldsymbol{D}$**.** First, D (Algorithm 3) gets n_1 outputs from the random oracle H and executes $A_1(h_1, ..., h_{n_1})$ to get P_t's secret share s_t. Then, it runs $GF_{A_2}(s_t, Y)$ to get two signatures with the required properties.

Remember that F fails with negligible probability, so that A_2 returns $\perp$ with the same probability. Therefore, thanks to the general forking lemma, GF_{A_2} outputs $\perp$ with negligible probability. In this case, D returns $\perp$. Otherwise, D

computes and returns $s = \frac{z'-z}{h'_j - h_j}$. This value is the discrete logarithm of Y. In fact, the signatures are $\sigma = (R, z), \sigma' = (R, z')$, so they both use the same commitment R. This is true because GF_{A_2} 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 get $g^z = RY^{h_J}$ from the first run of A_2 and $g^{z'} = RY^{h'_J}$ from the second one. So, $R = g^z g^{-sh_J} = g^{z'} g^{-sh'_J}$ holds and it implies $z - sh_J = z' - sh'_J$. Thus, D can compute s as $\frac{z'-z}{h'_j - h_j}$.

Algorithm 3 *D(Y)*

Input: Group's public key Y.
Output: $s = dlog(Y)$.

```
1. $h_1, \ldots, h_{n_1} \xleftarrow{\$} H$
2. $s_t \leftarrow A_1(h_1, ..., h_{n_1})$
3. $(1, h_j, h'_j, \sigma, \sigma')$ or $\perp \leftarrow GF_{A_2}(s_t, Y)$
4. If $\perp$, then return $\perp$.
5. Parse $\sigma, \sigma'$ as $(R, z), (R, z')$.
6. $s = \frac{z'-z}{h'_j - h_j}$
7. 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 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 prove that these properties are valid also in `Opt-SteadyState` and `Exp-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 dynamic proactive secret sharing scheme. 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 signing sessions end with a valid signature; and adapting the protocol to epochs of variable length.

Since our protocol is based on FROST, it inherits its non-robustness. However, as mentioned in Section 2.1, robust variants, like ROAST, have been proposed. A possible extension to our work is to join ROAST with CHURP. While the extension itself is straightforward, its security remains 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.

References

1. Aumasson, J.P., Hamelink, A., Shlomovits, O.: A Survey of ECDSA Threshold Signing. IACR Cryptol. ePrint Arch. **2020**, 1390 (2020)
2. Battagliola, M., Longo, R., Meneghetti, A.: Extensible decentralized secret sharing and application to schnorr signatures. Cryptology ePrint Archive, Paper 2022/1551 (2022), https://eprint.iacr.org/2022/1551, https://eprint.iacr.org/2022/1551
3. Bellare, M., Boldyreva, A., Staddon, J.: Multi-recipient encryption schemes: Security notions and randomness re-use. In: PKC. vol. 2003, pp. 85–99 (2003)
4. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proc. of ACM CCS '06. p. 390–399. ACM (2006)
5. Benedetti, M., De Sclavis, F., Favorito, M., Galano, G., Giammusso, S., Muci, A., Nardelli, M.: Certified byzantine consensus with confidential quorum for a bitcoin-derived permissioned dlt. In: Proc. of the 5th Distributed Ledger Technology Workshop (2023)
6. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? Cryptology ePrint Archive, 2020/464 (2020)
7. Benhamouda, F., Halevi, S., Krawczyk, H., Ma, Y., Rabin, T.: Sprint: High-throughput robust distributed schnorr signatures. Cryptology ePrint Archive, 2023/427 (2023)
8. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ros. Cryptology ePrint Archive, 2020/945 (2020)

9. Boldyreva, A.: Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In: Desmedt, Y.G. (ed.) Public Key Cryptography—PKC 2003. pp. 31–46. Springer (2002)
10. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. J. Cryptol. **17**(4), 297–319 (2004)
11. Cachin, C., Kursawe, K., Shoup, V.: Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. J. Cryptol. **18**(3), 219–246 (2005)
12. Crites, E., Komlo, C., Maller, M.: How to prove schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive (2021)
13. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. Cryptology ePrint Archive, 2018/417 (2018)
14. Ergezer, S., Kinkelin, H., Rezabek, F.: A survey on threshold signature schemes. Tech. rep. (2020)
15. Gennaro, R., Goldfeder, S.: Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In: Proc. of ACM SIGSAC CCS '18. p. 1179–1194. ACM (2018)
16. Gennaro, R., Goldfeder, S.: One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, 2020/540 (2020)
17. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) Applied Cryptography and Network Security. pp. 156–174. Springer (2016)
18. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust Threshold DSS Signatures. Information and Computation **164**(1), 54–84 (2001)
19. Golan Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M., Seredinschi, D.A., Tamir, O., Tomescu, A.: SBFT: A Scalable and Decentralized Trust Infrastructure. In: Proc. of IEEE/IFIP DSN '19. pp. 568–580 (2019)
20. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. Cryptology ePrint Archive, 2020/504 (2020)
21. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: Proc. of CRYPTO '95. p. 339–352. Springer-Verlag, Berlin, Heidelberg (1995)
22. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications pp. 177–194 (2010)
23. Komlo, C., Goldberg, I.: Frost: Flexible round-optimized schnorr threshold signatures. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) Selected Areas in Cryptography. pp. 34–65. Springer (2021)
24. Laing, T.M., Stinson, D.R.: A survey and refinement of repairable threshold schemes. Cryptology ePrint Archive, 2017/1155 (2017)
25. 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. pp. 1837–1854. ACM (2018)
26. Maram, S.K.D., Zhang, F., Wang, L., Low, A., Zhang, Y., Juels, A., Song, D.: CHURP: Dynamic-committee proactive secret sharing. In: Proc. of ACM SIGSAC CCS '19. p. 2369–2386. ACM (2019)
27. Noack, A., Spitz, S.: Dynamic Threshold Cryptosystem without Group Manager. IACR Cryptol. ePrint Arch. p. 380 (2008)
28. Ruffing, T., Ronge, V., Jin, E., Schneider-Bensch, J., Schröder, D.: ROAST: Robust asynchronous schnorr threshold signatures. In: Proc. of ACM SIGSAC CCS '22. p. 2551–2564. ACM, New York, NY, USA (2022)

29. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Conference on the Theory and Application of Cryptology. pp. 239–252. Springer (1989)
30. Schultz, D., Liskov, B., Liskov, M.: Mpss: Mobile proactive secret sharing. ACM Trans. Inf. Syst. Secur. **13**(4) (2010)
31. Sedghighadikolaei, K., Yavuz, A.A.: A comprehensive survey of threshold digital signatures: Nist standards, post-quantum cryptography, exotic techniques, and real-world applications (2023)
32. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (nov 1979)
33. Shoup, V.: Practical Threshold Signatures. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. pp. 207–220. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
34. Thai, Q.T., Yim, J.C., Yoo, T.W., Yoo, H.K., Kwak, J.Y., Kim, S.M.: Hierarchical Byzantine Fault-tolerance Protocol for Permissioned Blockchain Systems. Journal of Supercomputing **75**(11), 7337–7365 (2019)
35. Tomescu, A., Chen, R., Zheng, Y., Abraham, I., Pinkas, B., Gueta, G.G., Devadas, S.: Towards Scalable Threshold Cryptosystems. In: Proc. of the 2020 IEEE Symposium on Security and Privacy. pp. 877–893 (2020)
36. Vassantlal, R., Alchieri, E.A.P., Ferreira, B., Bessani, A.N.: Cobra: Dynamic proactive secret sharing for confidential bft services. Proc. of 2022 IEEE Symposium on Security and Privacy (SP) pp. 1335–1353 (2022)
37. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT Consensus with Linearity and Responsiveness. In: Proc. of ACM PODC '19. pp. 347–356. ACM, New York, NY, USA (2019)
38. 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, pp. 5413–5430 (2023)

A FROST

Here we show the details of FROST's algorithms, which are presented in Section 3.1. In particular, Algorithm 4 shows the functioning of FROST's key generation scheme. Algorithm 5 describes the preprocess phase, in which the pairs of commitments are generated. Finally, Algorithm 6 reports details of the signing phase, during which participants create their partial signatures and the SA aggregates them to produce the group's signature.

Algorithm 4 `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 $(a_{i0}, ..., a_{i(t-1)}) \xleftarrow{\$} \mathbb{Z}_q$ 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 Φ being a context string to prevent replay attacks.
3. Every participant P_i computes a public commitment $\overrightarrow{C}_i = \langle \Phi_{i0}, ..., \Phi_{i(t-1)} \rangle$, where $\Phi_{ij} = g^{a_{ij}}$, $0 \leq j \leq t-1$.
4. Every P_i broadcasts $\overrightarrow{C}_i, \sigma_i$ to all other participants.
5. Upon receiving $\overrightarrow{C}_l, \sigma_l$ 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 $R_l \overset{?}{=} g^{\mu_l} \Phi_{l0}^{-c_l}$, 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 $g^{f_l(i)} \overset{?}{=} \prod_{k=0}^{t-1} \Phi_{lk}^{i^k \mod q}$, 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 $Y_i = \prod_{j=1}^{n} \prod_{k=0}^{t-1} \Phi_{jk}^{i^k \mod q}$.

Algorithm 5 `Preprocess(`π`)`

Let j be a counter for a specific nonce/commitment share pair, and π be the number of pairs generated at a time.
Input: π = 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 $(d_{ij}, e_{ij}) \overset{\$}{\leftarrow} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.
    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.
```

Algorithm 6 `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 $g^{z_i} \overset{?}{=} R_i \cdot Y_i^{c\cdot\lambda_i}$ 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 $m$.
```

B Opt-CHURP

In this section, we report the algorithms that compose Opt-CHURP, the description of which can be found in Section 3.3. The protocol uses some auxiliary functions that are presented in Appendix D.

In Algorithm 7, the set U' of $2t-1$ members from the old committee reconstruct the polynomial shares $B(x, j)$.

Algorithm 8 shows how nodes in U' proactivize their reduced shares.

Algorithm 9 describes the last phase of the handoff, in which members of the new committee recover their full shares of the proactivized polynomial $B'(x, y)$.

Algorithm 7 `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 VerifyEval($C_{B(x,j)}, i, B(i,j), W_{B(i,j)}$).
   4.c Interpolate any $t$ verified points to construct $B(x, j)$.
```

C Pessimistic path

This section explains the functioning of `Exp-SteadyState` and Exp-CHURP-A. They both use on-chain communication only, resulting in a slower but more robust protocol. Similarly to Opt-CHURP, Exp-CHURP-A is composed by three subprotcols: `Exp-ShareReduce` (Algorithm 10), `Exp-Proactivize` (Algorithm 11) and `Exp-ShareDist` (Algorithm 12). 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 paper are identical to the ones written in [26], except for `Exp-ShareReduce`. The original version of Exp-CHURP-A uses `Opt-ShareReduce` instead, but this contradicts the communication model,

Algorithm 8 `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 $C_{Z_j} =$ Commit$(Z_j)$, $W_{Z_j(0)} =$ CreateWitness$(Z_j, 0)$, and $C_{B'(x,j)} =$ 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 VerifyEval$(C_{Z_j}, 0, 0, W_{Z_j(0)}) \neq$ 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 9 `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 $W_{B'(i,j)} =$ 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 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.
```

as P_i and U'_j communicate off-chain. We therefore build an appropriate algorithm starting from `Opt-ShareReduce` and moving all communication on-chain.

The idea behind the construction of `Exp-SteadyState` (Algorithm 13) is essentially the same as the one for Exp-CHURP-A. In particular, `Exp-SteadyState` works similarly to `Opt-SteadyState`, with the only difference that in the former all communication happens on-chain.

Algorithm 10 `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 get $\{B(i,j), W_{B(i,j)}\}_{j\in[2t-1]}$.
   4.b $\forall i \in U' \setminus U'_{\text{corrupted}}$, invoke 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 $B(x,j)$.
```

D Auxiliary functions

CHURP, thus D-FROST, invokes 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 14.

E Changing the threshold

Increasing the threshold To increase the threshold from t_{e-1} to t_e, CHURP runs the proactivization phase with parameter $t = t_e$. That is, during the proactivization protocol, a bivariate zero-hole polynomial $Q(x,y)$ of degree $\langle t_e-1, 2t_e-$

Algorithm 11 `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 12 `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 $w'_{ij} =$ 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 get ${B'(i,j), w'_{ij}}_{j \in [2t-1]}$.
   2.b $\forall j \in U' \setminus U'_{\text{corrupted}}$, invoke 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 $\forall i \in [n]$, 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 $B'(i,y)$.
```

Algorithm 13 `Exp-SteadyState` $(C, \{s_i\}_{i \in [n]}, \{(sk_i, pk_i)\}_{i \in [n]})$

```
Input: committee $C = {P_i}_{i \in [n]}$, $(t,n)$-shares ${s_i}_{i \in [n]}$ of the secret s and a pair of private and public keys $(sk_i, pk_i)$ 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. Order ${P_i}_{i \in [n]}$ based on lexicographic order of their public keys.
2. Choose the first $2t-1$ nodes, denoted as $U'$.
3. $P_i$ creates a random polynomial $B(i,y)$ such that $deg_{B(i,y)} = 2t-2$ and $B(i,0) = s_i$.
4. $P_i$ computes the KZG commitment $C_{B(i,y)}$ and publishes it on-chain.
5. $P_i$ publishes $(Enc_{pk_j}(B(i,j)), g^{B(i,j)}, W'_{B(i,j)})$ on-chain, where $W'_{B(i,j)} =$ CreateWitness($B(i,y), j$).
6. $U'_j$ decrypts $Enc_{pk_j}(B(i,j))$ and invokes VerifyEval($C_{B(i,y)}, j, B(i,j), W'_{B(i,j)}$).
7. If any of the checks for $i$ fail, $U'_j$ adds $i$ to $U_{corrupted}$ and publishes $(B(i,j), W'_{B(i,j)})$.
8. $U'_j$ interpolates $t$ verified points $B(i,j)$ to construct $B(x,j)$.
9. $U'_j$ computes $C_{B(x,j)}$ and publishes it on-chain, along with $W_{B(i,j)} =$ CreateWitness($B(x,j), i$).
10. $P_i$ verifies that the evaluation of $B(x,j)$ at $i$ returns $B(i,j)$ via VerifyEval($C_{B(x,j)}, i, B(i,j), W_{B(i,j)}$). If any of the checks fail return fail, otherwise return success.
```

Algorithm 14 $(2t-2, 2t-1)$-`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 $P(j) = \sum_{i=1}^{2t-1} P_i(j)$
```

$2\rangle$ 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: $B'(x,i)$, that is $(t_{e-1}-1)$-degree, and $Q(x,i)$, that is (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 15.

Algorithm 15 Decreasing the threshold

```
1. Choose a subset $U \subseteq C'$ of $2t_e-1$ nodes. For notational simplicity, suppose $U = \{1,...,2t_e-1\}$ and $V = \{2t_e,...,2t_{e-1}-1\}$. 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 proactivization phase and collectively generate 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 $RS_i^{(e)}(x) = \frac{\lambda_i^{2t_{e-1}-2}}{\lambda_i^{2t_e-2}}(RS_i^{(e-1)}(x) + \frac{V}{\lambda_j^{2t_{e-1}-2}(2t_e-1)}) + Q(x,i)$, 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 $FS_j(y) = B'(j,y)$.
```