Identifiable Cheating Entity Flexible Round-Optimized Schnorr Threshold (ICE FROST) Signature Protocol

Alonso González, Hamy Ratoanina, Robin Salen,
Setareh Sharifian, Vladimir Soukharev

research@toposware.com

Toposware, Inc.

November 8, 2022

Abstract. This paper presents an Identifiable Cheating Entity (ICE) FROST signature protocol that is an improvement over the FROST signature scheme (Komlo and Goldberg, SAC 2020) since it can identify cheating participants in its Key Generation protocol.
The proposed threshold signature protocol achieves robustness in the Key Generation phase of the threshold signature protocol by introducing a cheating identification mechanism and then excluding cheating participants from the protocol. By enabling the cheating identification mechanism, we remove the need to abort the Key Generation protocol every time cheating activity is suspected. Our cheating identification mechanism allows every participant to individually check the validity of complaints issued against possibly cheating participants. Then, after all of the cheating participants are eliminated, the Key Generation protocol is guaranteed to finish successfully. On the other hand, the signing process only achieves a weak form of robustness, as in the original FROST.
We then introduce static public key variant of ICE FROST. Our work is the first to consider static private/public keys for a round-optimized Schnorr-based signature scheme. With static public keys, the group's established public and private keys remain constant for the lifetime of signers, while the signing shares of each participant are updated over time, as well as the set of group members, which ensures the long-term security of the static keys and facilitates the verification process of the generated threshold signature because a group of signers communicates their public key to the verifier only once during the group's lifetime.
Our implementation benchmarks demonstrate that the runtime of the protocol is feasible for real-world applications.

1 Introduction

Digital signatures are a primary authentication tool for many cryptographic protocols. Digital signature schemes assign a key pair consisting of a public and a private key to each user. A user can sign messages using their private key,

whereas the signature is verifiable by any entity with access to the user's public key. If the private key is known only to the signer, a secure digital signature scheme guarantees that a verifiable signature on a message is generated, and no other entity can forge their signature.

In some applications, the signer is not an individual but a group such as a network or an organization. In this case, a valid signature can only be generated if the group members approve the content of the message. This requirement has become increasingly indispensable with the advent of blockchain technologies and cryptocurrencies in the past decade.

Distributed signing is usually enforced by using multisignature or threshold signature schemes. Multisignature schemes [Oka88,MOR01] enable a subgroup of potential signers, each with a public/private key pair, to jointly generate a signature σ on message m so that σ convinces a verifier that all members of the corresponding subgroup have signed m. On the other hand, threshold signature schemes [GJKR96a,GJKR96b] distribute a private key, with or without a trusted dealer, among n potential signers according to some t-out-of-n access structure. Only signatures generated by the cooperation of at least t signers will be accepted by verifiers with access to a unique fixed public key of the scheme. Note that multisignature schemes strive to prove that each member of the stated subgroup signed the message while the size of this subgroup can be arbitrary. As for threshold signature schemes, they aim to prove that a subgroup of sufficient size (the minimum subgroup size is known in advance) signed the message.

In some applications, threshold signature schemes are preferred over multisignature schemes due to privacy and availability reasons. From the privacy perspective, contrary to multisignatures, threshold signatures do not reveal the identities of individual signers and guarantee their anonymity. Finally, generating a threshold signature does not require all signers of a specific subgroup to be present online: the signature will be generated if at least t of the signers are online. In the case of generating a multisignature, all members of the signing subgroup must be present online (maybe not simultaneously as described in [BDN18].

Threshold signature schemes are constructed from primary digital signatures including RSA signatures [Sho00,DK01], DSA signatures [Lan95,GJKR96b] and their variants — ECDSA [GGN16,BGG17,GG18] signatures, BLS signatures [Bol03], and Schnorr signatures [SS01,GJKR03].

The above protocols require that signers interact in order to generate the final signature. Yet, in many real-life situations, it is desirable to have a *non-interactive* signature generation scheme that allows each signer, who saw the message, to generate their own "signature share" without having to interact with any other signer. Non-interactive threshold RSA and BLS signature schemes are proposed in [JO08] and [BLS01], respectively. Preprocessing technique is used in ECDSA threshold schemes [CGG+20] and Schnorr-based schemes [KG20] to construct non-interactive protocols.

Robustness is another desirable property of multisignature and threshold signature schemes; it ensures proper execution of the protocol even if there are

cheating signers who deviate from the protocol. One way to achieve robustness is by *identifying cheaters* and excluding them from the protocol execution. In such environments as blockchains that allow financial punishment of cheating participants, when cheating identifiability combines with sufficient punishment, the resulting method can stop (rational) parties from cheating and thus decrease the failure probability of the protocol: i.e., enhance its robustness. Note that cheating identifiability does not necessarily result in robustness as some protocols will abort execution after identifying cheaters (identifiable aborts): e.g., threshold ECDSA protocols in [CGG+20,GG20]. The obtained guarantee in these cases is referred to as *weak robustness* [MOR01].

Cheating identifiability can be viewed as an extended notion of soundness where the validity of local, possibly secret, communication between two participants can be publicly verified by the rest. In this way, all honest participants can agree on the set of misbehaving participants and exclude them. This can be easily achieved through the use of zero-knowledge proofs and encryption by making participants encrypt their secret messages and show in zero-knowledge its correctness. Non-interactive versions of this mechanism typically require the use of pairings, or at least homomorphic encryption, so that it is possible to efficiently construct statements related to the encrypted communication. ElGamal-based encryption with non-interactive checks of correctness such as the one used in [Gro21] suffers from inefficient decryption requiring the computation of the discrete logarithm problem for bounded challenges.

The signature generated by a threshold scheme eventually becomes part of a certificate in protocols that require authentication. Before the signing and verification sub-protocols start, the group's public key (necessary to verify the generated signature) should be communicated reliably to the certificate authority. However, each public key is valid only for a certain time period and after that the update is necessary to minimize the assessed risk under the potential attacks. This highlights the challenge of frequent update and communication of temporary public keys that may affect the efficiency of the protocol, particularly while generating multiple threshold signatures for multiple messages. A solution to this is using *long-lived (i.e., static) public keys* that will remain unchanged for, possibly, the lifetime of the signing group. Protecting long-lived keys requires constructing proactive countermeasures by using variations of Shamir's secret sharing scheme. The application of long-lived keys for threshold signatures was first considered in [HJJ+97], where the proactive secret sharing (PSS) scheme of [HJKY95] is used to propose a framework for transferring a wide class of discrete log-based threshold signature schemes into the proactive ones with long-lived secure private/public keys.

1.1 Our Contributions

We propose a threshold signature protocol based on FROST, a Flexible Round-Optimized Schnorr Threshold signature scheme by Komlo and Goldberg [KG20]. The proposed signature protocol has the advantage of preprocessing computa-

tions, first considered in [GG18] for threshold signature schemes. The protocol has two main phases:

1. A *Key Generation* phase in which all participating parties take part in generating and distributing a joint secret key that will be used for signing messages, and
2. A *Signing* phase in which a subgroup of parties, satisfying the threshold, use their shares from the joint secret key to sign a message.

Our main contributions in this work are as follows:

i) *Achieving robustness in Key Generation phase of the threshold signature protocol.* We propose a cheating identification mechanism and then exclude cheating participants from the protocol, without aborting, to bring robustness to the Key Generation protocol. We consider a party to be cheating when it distributes inconsistent shares among other participants or accuses an honest participant of distributing inconsistent shares. To emphasize this contribution, we named our proposed signature protocol as "Identifiable Cheating Entity (ICE) FROST". The description of the protocol is given in Section 4. Due to the robustness of the Key Generation phase in ICE FROST and identification of cheaters during the Signing phase (that is inherent from FROST), ICE FROST is weakly robust in the sense of Micali et al. [MOR01]. Further, in Section 7, we show that with an appropriate choice of a subset of signers, the weak robustness property of ICE FROST scales well for systems with large numbers of participants. Improving the robustness of ICE FROST, by making Signing phase robust, remains an interesting future research question.

For achieving cheating identifiability, instead of using homomorphic encryption as in [Gro21], we use symmetric encryption at the cost of more interaction. In our construction, shares are encrypted using the the DH key between the sender and the receiver. When a receiver decrypts an invalid share, it can convince the rest of the parties that the sender cheated by revealing the DH key with a proof of its validity with respect to the sender and receiver key.

ii)*Designing the first proactively secure Schnorr-based signature scheme with static private/public keys.* We modify the Key Generation protocol of ICE FROST to maintain long-lived secure signing keys with the help of the PSS protocol [HJKY95] that allows for regular key redistribution between participants while keeping the resulting distributively generated key unchanged. Note that the set of participants can be updated as well, while the key would still remain the same.

According to [HJJ+97], "proactivization" of a threshold signature scheme is possible under satisfaction of certain conditions. Namely, if the signature scheme is a discrete log-based robust threshold signature scheme, whose threshold key generation protocol implements Shamir's secret sharing of the secret key x corresponding to the public key $y = g^x$ and outputs verification information $(g^{x_1}, \ldots, g^{x_n})$, where $(x_1, \ldots, x_n)$ are secret shares of the players and if the threshold signature protocol is can be simulated [GJKR96b, Definition 2]. While Schnorr-based threshold signature schemes are introduced as one of the potential candidates for proactivization using the framework of [HJJ+97], to the best of

our knowledge, an actual proactive Schnorr-based scheme has never been proposed in the literature. Proactivization of existing robust Schnorr-based threshold signatures of [SS01] and [GJKR03] using this the framework of [HJJ+97] is not possible because they do not satisfy all the mentioned conditions. In particular, they use additive shares rather than Shamir's (polynomial) shares for generating the threshold signature. Our scheme on the other hand, is the first robust Schnorr-based threshold signature scheme that satisfies all the conditions, noting that the proof of simulatability is implicit in our unforgeability proof (Theorem 1), where we use simulation technique to proceed with the proof. The details of our design is given in Section 5.

iii) *Mitigating the Key Bias Attack from [GJKR99] without increasing the number of rounds of the protocol.* We achieve this by using public verifiable randomness to choose and sacrifice one of the participants at random to ensure that an adversary cannot bias the generated key. In the environments where a distributed randomness is available to participants (e.g., in a blockchain), implementing the suggested mitigation is easily feasible.

So far, no explicit forgery attack associated to the key bias has been reported. However, according to [GJKR03], mitigating this attack reduces the required security parameters to achieve a desired level of security.

Organization In Section 2 we review the related backgrounds. In Section 3 we review the FROST signature scheme, which is the basis of the proposed work, and then discuss some suggestions for achieving (weak) robustness in FROST. In Section 4 we present our main contribution, a Schnorr-based threshold signature scheme that can identify cheating participants in its key generation protocol. In Section 5 we present a modification of ICE FROST that allows the use of static group public and private keys for signing, while the shares of signing participants are regularly updated. In Section 7 we present some practical considerations that can improve the efficiency of ICE FROST in practice, especially when dealing with large numbers of participants. Section 7 contains benchmarks of our implementations. In Section 8 we review the possible attacks against which ICE FROST is secure and propose mitigation for attacks against which ICE FROST is vulnerable. In Section 9 we compare the ICE FROST with FROST in terms of computational and communication costs, and finally, we conclude this work in Section 10.

1.2 Related Works

Threshold cryptography [Des87,DF89] aims to distribute secret information (i.e., private keys) and computation (i.e., signature generation or decryption) among n parties to avoid having a central point of trust, which may become a single point of failure. Threshold signature schemes lay under the umbrella of threshold cryptography. They are based on RSA signatures [Rab98,Sho00,FGMY97a], BLS signatures [BDN18,BLS01], and Schnorr signatures [AAM19,GG20,GJKR03,SS01].

Robust schemes ensure successful execution of the protocol if t participants follow the protocol, even if a subgroup of participants (at most $n-t$) contributes

malformed shares. On the other hand, non-robust schemes abort after detecting any misbehavior of a participant. The schemes of [FGMY97a,Rab98] are RSA-based robust threshold signature schemes and [SS01,GJKR03] are Schnorr-based robust threshold signature schemes. The latter two schemes require all n signers, of which no more than t are malicious, to participate in order to generate a signature. Both of these schemes consist of two main phases: Key Generation phase and Signing phase. Key Generation phase uses Pedersen's distributed key generation (DKG) algorithm [Ped91a], which allows for the joint generation of signing keys. This algorithm uses a variation of Shamir's secret sharing scheme (SSS) [Sha79], the Verifiable Secret Sharing (VSS) scheme, which allows verification (in terms of consistency) of the shared secrets during the secret distribution and verification of presented shares during the secret reconstruction steps [Fel87]. VSS schemes are examined in the presence of a trusted dealer [CGMA85,CDD+99,RBO89] as well as in the absence of a trusted dealer [CDF01]. Cheating in these schemes is detected, and robustness is achieved by avoiding incorrect input in some cases. However, it does not provide a strong deterrent against cheating because cheating entities can cause additional computations in the protocol without being identified. In identifiable secret sharing, a failure of the reconstruction algorithm results in identifying the participants who modified their shares. Computationally identifiable secret sharing schemes have been proposed in [KOO95,MS81,Cho11,IOS12].

Long-lived keys achieved with the aids of proactive secret sharing schemes are first considered in [HJJ+97] for discrete log-based threshold signature schemes. Proactive RSA based threshold schemes are considered in [FGMY97b] and [Rab98]. Static keys were used in [Gro21] for BLS signatures, but our work is the first to consider static keys for a Schnorr-based signature scheme.

2 Preliminaries

In this section, we first introduce the notations that will be used throughout the paper. Then we review the mathematical background needed to understand the contributions of this paper.

Notations. We use calligraphic letters to denote integer sets, or sets of participants in a protocol Π. By $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ we denote the set of n participants and can identify a participant by their index. If $\mathcal{S} \subset \mathcal{P}$ is a subset of participants, then $|\mathcal{S}|$ denotes its size.

All the logarithms are in base 2. We use $x \xleftarrow{\$} X$ to denote a uniformly random selection of an element x from a set X. The set X is usually instantiated by a finite field of prime order q, i.e., $\mathbb{Z}_q$.

A symmetric encryption scheme is denoted by $\mathcal{E}_{sym} = (Enc_{sym}, Dec_{sym})$ associated with key space $\mathcal{K}$, message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$, for $k \in \mathcal{K}$, encrypts the message $m \in \mathcal{M}$ to $c = Enc_{sym}(k, m)$, where $c \in \mathcal{C}$ so that $Dec_{sym}(k, Enc_{sym}(k, m)) = m$.

2.1 Distributed Secret Generation

Shamir's Secret Sharing (SSS) [Sha79]. Secret sharing is a cryptographic primitive that allows a trusted dealer to distribute a secret s among a set of parties by allocating a share of the secret to them. More precisely, to distribute a secret $s \in \mathcal{S}$ among a set of signers $\mathcal{P} = \{P_1, \ldots, P_n\}$ the trusted dealer computes the shares $f_1, \ldots, f_n \in \mathcal{F}$, where $\mathcal{S}$ is the space of secret keys and $\mathcal{F}$ is the space of all possible shares. The secret can be reconstructed only when an authorized subset of the parties $\mathcal{A} \subset \mathcal{P}$ of these signers pools their shares.

In threshold secret sharing schemes, authorized subsets of valid shares are those whose cardinality is greater than the threshold value denoted by t. Shamir's threshold secret sharing scheme [Sha79] is based on polynomial interpolation. This scheme is information-theoretically secure and does not reveal any information to an unauthorized subset of users. Shamir's (t, n)-threshold secret sharing scheme consists of the Share and Reconstruct algorithms defined as follows:

- Share takes a secret $s \in \mathbb{F}_q$ as input and constructs a polynomial of degree $t-1$ by randomly selecting $t-1$ coefficients $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$, and letting the polynomial be $f(x) = s + \sum_{i=1}^{t-1} a_i x^i$. It computes the share $f_i \in \mathbb{F}_q$ for the signer $s_i \in \mathcal{S}$ with a unique ID $id_i \in \mathbb{F}_q$ as a point on polynomial $f(x)$, i.e., $f_i := (id_i, f(id_i))$.
- Reconstruct takes as input a set of shares held by a subset $\mathcal{A} \subset \mathcal{S}$ of signers. If $|\mathcal{A}| \leq t$, then it outputs $\perp$. Otherwise, it reconstructs the polynomial $f(x)$ using Lagrange interpolation formula as follows:

$$f(x) = \sum_{P_i \in \mathcal{A}} f(id_i) \prod_{P_j \in \mathcal{A}, id_i \neq id_j} \frac{x - id_j}{id_i - id_j},$$

  and outputs the secret $s \in \mathbb{F}_q$ as $f(0) = s$.

Cheating (tampering) in secret sharing corresponds to a scenario in which a signer provides the reconstruction algorithm with a share different than the one assigned to them by the distribution algorithm. An *identifiable secret sharing* scheme, which was first considered by McEliece and Sarwate [MS81], is a secret sharing scheme in which the reconstruction algorithm can identify all cheaters with high probability.

Proactive Secret Sharing (PSS) [HJKY95]. Proactive secret sharing was introduced by Herzberg et al. in [HJKY95]. It addresses the problem of the long-term confidentiality of the shared secret by periodically *renewing* shares to prevent a mobile adversary from collecting enough shares over time to reconstruct the secret. For share renewal, each participant can generate (t, n) shares of the value of "0" each and send the shares to respective participants. Next, the participants add all the received shares to get their renewed secret share of the initial secret [HJKY95].

Refined constructions of PSS allow not only for renewal of existing shares, but also for distribution of new shares to different signers. Therefore, this procedure is often referred to as *redistribution* rather than *renewal* of shares [DJ97].

Suppose an underlying (t, n)-Shamir's secret sharing of a secret s, which should be converted into a (t', n')-Shamir's secret sharing of the same secret s. Let $\mathcal{P}$ be the set of current signers and $\mathcal{P}'$ the set of new signers to which share redistribution should happen, where $|\mathcal{P}| = n$ and $|\mathcal{P}'| = n'$. Each participant $P_i \in \mathcal{P}$ applies the (t', n') secret sharing procedure to their currently stored share s_i and sends (through a secure channel) the sub-shares to the respective participants. Next, the receiving participants agree on a subset $\mathcal{P}_c \subset \mathcal{P}$ of t participants and compute their new share s' by combining the t sub-shares received from the members of P_c. Once the new shares are computed and stored, the respective participants should delete the old shares. Note that, by creating sub-shares of the existing shares and recombining them, the original secret remains unchanged, and shares from different time periods cannot be combined to reconstruct the secret.

Verifiable Secret Sharing (VSS) [CGMA85]. Verifiable secret sharing was introduced by Chor et al. in [CGMA85] and allows signers to verify the validity of the received shares from the dealer. Verifiable secret sharing is usually provided by the application of a public *commitment*, which is assumed to be correctly visible to all participants [Fel87,Ped91a].

To share a secret s using a (t, n) VSS, the dealer (as in Shamir's secret sharing) generates $t-1$ random coefficients $a_1, \ldots, a_{t-1}$ and uses them to define a polynomial $f(\cdot)$ of degree $t-1$ so that $f(0) = s$ and $f_i = f(id_i)$ is the corresponding share for the party P_i with an ID id_i. However, to facilitate the verification of the distributed shares, the dealer also broadcasts a public commitment vector $\mathbf{C} = \langle \phi_0, \phi_1, \ldots, \phi_{t-1} \rangle$, where ϕ_0 is a commitment to the secret s and, for $1 \leq i \leq t-1$, ϕ_i is a commitment to the coefficient a_i used to define the polynomial $f(\cdot)$. Each party verifies the validity of their share using the commitment vector $\mathbf{C}$. If the verification fails, the signer can issue a *complaint* against the dealer and take actions such as broadcasting the complaint to all other participants.

Commitment schemes. Commitment schemes allow a committer to publish a value that binds them to a message (binding) without revealing it (hiding).

We review Feldman's commitment scheme [Fel87] here; the scheme is a triple of algorithms (Setup, Commit, Open):

- Setup takes a security parameter λ as input and outputs $\mathbb{G}$, a group of prime order q, and a generator $g \in \mathbb{G}$.
- Commit takes a message $m \in \mathbb{F}_q$ as input and outputs the commitment $\phi = g^m$.
- Open takes a commitment $\phi \in \mathbb{G}$ and a message $m \in \mathbb{F}_q$ as input and outputs 1 if $\phi = g^m$ and 0 otherwise.

The commitment scheme used in Feldman's VSS scheme [Fel87] is unconditionally binding and computationally hiding; there are other schemes, such as Pedersen's scheme [Ped91a], that are computationally binding and unconditionally hiding.

Distributed Key Generation (DKG) [Ped91b]. Distributed key generation allows a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ to jointly generate a public and private key pair without involving a trusted dealer. The public key is publicly known, while the private key is kept secret and shared via a (t, n) threshold scheme, where the shares belong to the parties in $\mathcal{P}$. More precisely, no attacker can learn anything about the key unless they obtain the corresponding shares from at least t parties in $\mathcal{P}$. For discrete log–based schemes, a distributed key generation scheme secretly shares a uniformly distributed value x and makes the value $y = g^x$ public as the public key, where $g \in \mathbb{G}$ is the generator of $\mathbb{G}$, a group of prime order q.

In the DKG scheme presented by Pedersen in [Ped91b], each participant acts as a dealer of Feldman's VSS protocol. Each participant selects a secret s_i and generates corresponding shares for all other parties. The protocol then requires two rounds of communication between all participants: a public communication round, where each party broadcasts a commitment to s_i and the coefficients of the corresponding polynomial, and a secure communication round, where each party P_i securely sends a secret share of s_i to all other participants. After receiving a share from P_i, each participant checks if the received share is consistent with the previously published commitment. If so, the received share is marked as *qualified*. Each participant derives their total share by adding up all the *qualified* shares they received. The secret shared value s itself is not computed by any party: however, it is equal to the sum of the shared s_is which passed commitment checks by all recipients of the shares.

2.2 Zero-knowledge Proofs

Zero-knowledge proofs were introduced in the seminal work of Goldwasser, Micali, and Rackoff [GMR89]. Informally, in a zero-knowledge proof system, the prover wants to convince the verifier that some statement is true, yet, the verifier should not learn anything from their interaction with the prover beyond the truth of the statement. In other words, a (potentially malicious) verifier V gains no new information from interacting with a prover P on a common input x, if everything that this verifier V can compute after interacting with P can be computed directly from the common input x by an efficient algorithm. Zero-knowledge proofs not only demonstrate to the verifier the *existence* of a witness w for the statement, but additionally prove that the prover *knows* that witness. We say that an efficient algorithm A *knows* a value w if we can construct another efficient algorithm that takes A as input (for example, by getting the code of A and its random coins) and outputs w. Such an algorithm is called an *extractor* for A.

A typical use case of zero-knowledge proof of knowledge is during authentication in a secure communication between the server and the client: the server publishes their public key pk and stores the corresponding secret key sk. The client can verify that it is talking to the correct server by asking the server to perform a zero-knowledge proof that it knows the secret key sk corresponding to pk.

Σ-Protocols. Σ-protocols, a special class of zero-knowledge proof systems, are the basis of many efficient zero-knowledge protocols.

Definition 1 (Σ-Protocol). *A Σ-protocol for a language $\mathcal{L}$ is a public-coin three-move honest-verifier zero-knowledge proof of knowledge, which has the following structure:*

1. *P sends to V some commitment value r,*
2. *V sends to P a uniformly random challenge e,*
3. *P sends to V an answer $f(w, r, e)$, where f is some public function, and w is the witness held by P.*

Fiat-Shamir Heuristic. The Fiat-Shamir heuristic [FS86] is a heuristic method to convert Σ-protocols into non-interactive zero-knowledge arguments. It proceeds as follows: let w be the witness. To prove that a word x belongs to a language $\mathcal{L}$, the prover P first computes the first flow (the commitment) of a Σ-protocol for this statement. Let c denote this first flow. Then, P sets $e = H(x, c)$, where H is some hash function, and computes the last flow of the Σ-protocol using e as the challenge. This approach has been proven to be secure in a random oracle model.

2.3 Schnorr Signature Scheme

A Schnorr signature [Sch89] is generated for a message m, under secret key $s \in \mathbb{Z}_q$ and public key $Y = g^s \in \mathbb{G}$ as follows:

1. Sample a random nonce $k \xleftarrow{\$} \mathbb{Z}_q$; compute the commitment $R = g^k \in \mathbb{G}$.
2. Compute the challenge $c = H(R, Y, m)$, where $H : \{0,1\}^* \to \mathbb{Z}_q$ is a cryptographic hash function.
3. Using the secret keys, compute the response $z = k + s \cdot c \in \mathbb{Z}_q$.
4. Define the signature over M to be $\sigma = (R, z)$.

Validating the integrity of m by an identified signer with the public key Y and the signature σ is performed as follows:

1. Parse σ as (R, z); derive $c = H(R, Y, m)$.
2. Compute $R' = g^z \cdot Y^{-c}$.
3. Output 1 if $R = R'$ to indicate success; otherwise, output 0.

In the random oracle model, the unforgeability of this scheme under chosen-message attack is reduced to the discrete logarithm problem according to [PS96].

Schnorr signatures can be considered just standard Σ-protocol proofs of knowledge of the discrete logarithm of Y, made non-interactive (and bound to the message m) with the Fiat-Shamir transform [FS86].

2.4 Threshold Signatures

The initial definition of a (t, n) threshold signature scheme introduced by Desmedt in [Des87] has two main properties; i) n participants with a common public key can issue a signature even if there are $t < n/2$ dishonest (cheating) participants and ii) any $t-1$ corrupted participants cannot forge a signature. These types of signature have been referred to as *robust* threshold signatures in [GJKR96a,GJKR96b] and *t-resilient* signatures in [PK96]. Frankel and Desmedt defined the concept of *threshold multisignature schemes* in [FD92] and proposed a threshold multisignature RSA scheme that is non-interactive. Threshold multisignatures allow any subset of participants of a size larger than the threshold to produce signatures over a message so that anyone can validate the signature using the unique public key assigned to the group of n parties. Threshold multisignatures are referred to as *threshold signatures* in [GJKR99,SS01,KG20]. We follow the latter and use a "(t, n)-threshold signature" to describe these schemes.

Shoup [Sho00] used the "dual-parameter" notion to describe a threshold signature scheme that allows any subset of sufficient participants to generate a signature, but that disallows the creation of a valid signature by insufficient (possibly corrupted) participants or the prevention of signature generation by uncorrupted participants. In this notion, there is one threshold t for the minimum quorum size and another threshold $k < t$ for the maximum number of cheating participants. A particular message is signed only if at least $t-k$ honest participants have authorized the signature. Shoup proposes an RSA threshold signature scheme for $k \leq t-1$.

The security notion for (t, n) threshold signature schemes requires *unforgeability* of the scheme. That is, after a distributed generation of the public key, no polynomial-time adversary who can access a polynomial number of threshold signatures on the messages of their choice and also $t-1$ corrupted participants, can produce, with a non-negligible probability, a valid signature under the generated public key on some new message. Unforgeability in this sense is referred to as *existential unforgeability* in the literature [GMR88]. A weaker notion is *universal unforgeability*, where an adversary is supposed to generate a valid signature for *any* message. We consider only *existential unforgeability* in this paper.

Definition 2. *Let* $\mathsf{TS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a threshold signature scheme with key generation, signing and verification algorithms* KeyGen, Sign *and* Verify, *respectively.* TS *is called a secure threshold signature scheme if the following conditions hold:*

1. ***Correctness.*** *Any subset $\mathcal{S}$ of participants with cardinality at least t can produce a valid signature on message M. A valid signature is a signature that will be verified by the* Verify *algorithm.*
2. ***Unforgeability.*** *Any polynomial-time adversary who sees the protocol's output (signature) on* $\mathsf{poly}(\lambda)$ *input messages of their choice and can corrupt up to $t-1$ players, cannot produce the valid signature σ for a message M, which has not been submitted to* Sign *before, with probability more than* $\mathsf{negl}(\lambda)$, *where* $\mathsf{negl}(\cdot)$ *is a negligible function.*

Robustness is a desirable property of a (t, n) threshold signature schemes. It ensures that a valid signature is generated in the presence of some malicious participants who seek to break the protocol. For the sake of clarity, further, we refer to participants who try to forge a signature as *corrupted* participants. Also, we refer to participants who try to prevent the generation of a valid signature as *cheating* participants. We use t to denote the number of trusted participants required to successfully run the signature scheme and k to denote the number of cheating participants. For an integer k, the robustness property is defined as follows.

($\mathbf{k}, \mathbf{n}$)-*Robustness [GJKR96b].* *Even k out of n cheating participants who deviate from the protocol cannot prevent honest participants from generating a valid signature. i.e., both Key Generation and Signing protocols will run successfully in the presence of k cheating participants.*

We also define (k, n) robustness for a standalone Key Generation and Signing protocols. The Key Generation protocol of a (t, n) threshold signature scheme is (k, n) robust if even in the presence of k cheating participants, the protocol outputs a group public key and associated signing and verification keys to each honest participant such that at least t honest participants can output a valid signature on a message m using their signing shares during the Signing protocol. The Signing protocol is (k, n) robust if even in the presence of k cheating adversaries, the protocol allows the maximal set of t honest participants who verify message m to outputs a valid signature on m using the generated signing keys during the Key Generation protocol.

3 FROST and Its Robustness

The FROST protocol focuses on efficiency rather than on robustness. In the absence of cheating participants, this approach works efficiently. However, cheating participants can delay or even stop the execution of FROST by distributing inconsistent shares among participants. In this section, we start with a review of the FROST protocol of [KG20] and then introduce mechanisms for dealing with cheating participants in FROST.

3.1 Flexible Round-optimized Schnorr Threshold Signature

FROST is an efficient Schnorr threshold signature scheme with security guarantees. Although the original design of FROST requires a two-round signing protocol in which each signer sends and receives two messages, it can be optimized to a (non-broadcast) single-round signing protocol with a preprocessing step. FROST has high efficiency in the absence of misbehaving parties. However, misbehaving participants can delay the FROST signing algorithm by forcing the honest parties to re-run the protocol after their misbehaviour is detected.

The FROST protocol consists of two sub-protocols: *(i) Key Generation*, in which a group of signers jointly generates a random key that will be used as

the group's signing key[1], and *(ii) Signing*, in which any subgroup of users of cardinality greater than or equal to the threshold can sign a message. The final signature is generated either by a semi-honest signature aggregator or by any of the active participants. We briefly review the (t, n) FROST signature scheme here for completeness. See [KG20, Section 5] for more information.

-FROST Key Generation Protocol. The key generation protocol of FROST runs as follows: each participant $P_i, 1 \leq i \leq n$ chooses a secret a_{i0} and generates a polynomial $f(\cdot)$ of degree $t-1$ by randomly choosing $t-1$ other coefficients $a_{i1}, \ldots, a_{i(t-1)}$. Next, they publish a zero-knowledge proof of knowledge of a_{i0} and a vector of commitments to every other coefficient of the polynomial (a_{i1} to $a_{i(t-1)}$). Publishing the zero-knowledge proof of knowledge of a_{i0} is required to prevent rogue-key attacks [BBS03] in the setting where $t \geq n/2$. The zero-knowledge proof published by P_i is checked by every other participant using the first component of the commitment vector - the commitment to the shared secret a_{i0}. If the proof verification fails, the protocol is aborted. Otherwise, P_i calculates the shares of the secret a_{i0} for all the other participants and securely sends each share to the corresponding participant. Each participant verifies the share received from P_i using the commitment vector and aborts the protocol if the verification fails. Otherwise, each participant calculates their long-lived private signing share s_i and the corresponding public verification share Y_i using the received shares from every other participant (including their own share). The group's public key is calculated using all the commitments to a_{i0} for $1 \leq i \leq n$.

-FROST signing protocol. The FROST signing protocol consists of two phases: a *preprocessing phase* and a *single-round signing phase*. In the *preprocessing phase*, each participant P_i generates a list of single-use private nonce pairs and corresponding public commitment shares $\langle((d_{ij}, D_{ij} = g^{d_{ij}}), (e_{ij}, E_{ij} = g^{e_{ij}}))\rangle_{j=1}^{\pi}$, where j is a counter that identifies the next nonce/commitment share pair available to use for signing and π determines the number of nonces that are generated and their corresponding commitments (D_{ij}, E_{ij}) in a single preprocessing step. P_i then publishes their ID id_i and the list of commitments.

To sign a message m in the *signing phase*, a set $\mathcal{S}$ of at least t signers is selected. After this, the signature aggregator (possibly included in $\mathcal{S}$) selects the next available commitment $(D_{ij} = g^{d_{ij}}, E_{ij} = g^{e_{ij}})$ for each signer and outputs $B_i = \langle(i, D_i, E_i)\rangle_{i \in \mathcal{S}}$. The signature aggregator then sends (m, B_i) to every signer in $\mathcal{S}$. Each participant checks M. If a participant agrees to sign M, they *(i)* calculate a binding value $\rho_i = H_1(i, m, B_i)$ using a hash function $H_1(\cdot)$ for every $i \in \mathcal{S}$, *(ii)* calculate the group commitment $R = \prod_{i \in \mathcal{S}} D_i \cdot (E_i)^{\rho_i}$ and the challenge $c = H_2(R, Y, m)$ using another hash function $H_2(\cdot)$, and finally, *(iii)* calculate the response $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i \cdot c$ to the challenge using their long-lived key s_i, where λ_i is the Lagrange coefficient for ID i in set $\mathcal{S}$. The signature aggregator then checks the consistency of z_i, reported by each participant, using (D_i, E_i) and their public verification share Y_i. If the check

[1] Note that the signing key is never reconstructed. Instead, at the end of the key generation protocol, each participant receives a long-lived private signing share and computes a public verification share.

passes, the group's response is $z = \sum_{i \in \mathcal{S}} z_i$ and the group signature on m is $\sigma = (R, z)$. This signature is verifiable to anyone performing a standard Schnorr verification operation with Y as the public key (Section 2.3).

-Robustness in the FROST protocol. To ensure successful execution of the FROST protocol without significant loss of efficiency, we define a relaxed version of robustness property called *weak robustness*, following the definition for multisignatures given in Micali et. al. [MOR01]. Let $0 < \alpha \leq 1$; we define α-weak robustness as follows.

Definition 3 (α-Weak Robustness.). *Let* $\mathsf{TS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a* (t, n) *threshold signature scheme with key generation, signing and verification algorithms* KeyGen, Sign *and* Verify, *respectively.* TS *is weakly robust if, after each failure in generating a valid signature on message* m *(either during KeyGen or Sign), the identity of at least one cheating participant is exposed with probability greater than* α.

Proposition 1. *An* α*-weak robust* (t, n) *threshold signature scheme run by* n *players, of which no more than* k *are malicious and* $k < n-t$*, will run successfully after at most* k *runs with probability at least* α^k.

Proof. In each run of TS, at least one cheating participant will be identified with probability at least α. Let the protocol re-run after excluding the cheating participant. After no more than k runs, every participant will be excluded, and the protocol will complete successfully. The probability of identifying all k cheating participants is bounded from above by the product of the probalility of detecting each participant in one run; according to the Bayes' theorem, that is α^k. □

In FROST, cheaters are identified during the signing phase. However, the scheme, as a whole, is neither robust nor weakly robust, because the key generation process may abort without exposing any cheating participants. To achieve weak robustness, we propose to make the key generation phase robust. In this case, the only event where the protocol fails is during the signing phase and, since cheating is identifiable in the signing phase, the whole scheme can become robust. We consider two approaches for making the key generation of FROST robust, namely, i) using Pedersen's distributed key generation protocol [Ped91b] for generating the key, and ii) identifying and excluding cheaters from the key generation process. We note that, unlike the signing phase, the key generation phase can continue after cheaters are identified and excluded; therefore, the guarantee that we get by enabling cheating identification in the key generation phase is robustness rather than weak robustness for the Key Generation protocol. However, the whole scheme is only weakly robust Further, we look into these two approaches.

3.2 First Approach: Robustness Using Pedersen's DKG

Weak robustness for FROST can be maintained by modifying the key generation phase using Pedersen's DKG in [Ped91b]. This approach for key generation of

a threshold signature is used in [SS01,GJKR99]. The modified KeyGen protocol of FROST proceeds as follows: after P_i verifies the zero-knowledge proof broadcasted by P_l for the secret a_{l0} and receives the corresponding share securely, instead of *aborting* the protocol (step 2 of Round 2 in the original FROST protocol [KG20, Figure 1]) because of the failed verification of the received share (using the vector of commitments to P_l's generated polynomial coefficients), P_i issues a *complaint* against P_l and sends the complaint message directly (but publicly) to P_l. After receiving the complaint, P_l, in defence, reveals $f_l(i)$: this is P_i's share of the secret a_{l0} generated by P_l. P_l will be excluded from the rest of the protocol if

- P_l fails to object, i.e., $f'_l(i)$ revealed by P_l is inconsistent with $f_l(i)$, the vector of commitments to the coefficients of the generated polynomial by P_l that is published at the beginning of the protocol).
- P_l receives more than t complaints, where t is the threshold of the scheme.

In the above procedure, please note that as long as fewer than t complaints are issued against P_l, revealing $f_l(i)$ does not affect the security of the protocol. Besides, P_i may receive an inconsistent share from P_l only because of the noisy communication channel. Allowing P_l to defend themselves adds a level of reliability to the system against noisy channels. Anyway, when the number of complaints against P_l reaches the critical threshold t, P_l will be excluded. And in order to intentionally exclude P_l from the protocol, at least t cheating participants have to collaborate.

To argue that the FROST protocol with the above modification will be weakly robust, we point at the following technicalities:

1. In the original FROST protocol, every party can check the zero-knowledge proof published by P_l during Step 2 of Round 1 in [KG20, Figure 1]). If the proof is not verified, every party will ignore both P_l and the information transmitted by P_l without needing to abort the protocol.
2. At the end of the KeyGen protocol, any participant can check the public verification share of any other participant by calculating Y_i. If the check does not pass, the public verification share calculated by the majority of the participants will be used.
3. During the signature generation, if the response by each participant is not verified, (step 7.b in [KG20, Figure 3]) the protocol will re-run after identifying and excluding the cheating participant.

Proposition 2. *The described modified* (t, n) *FROST protocol is a 1-weakly robust* (t, n) *threshold signature scheme if* k*, the number of cheating participants is less than* $min(n/2, t)$*.*

Proof. The robustness of the KeyGen protocol follows from the robustness of the Pedersen's DKG protocol as long as $k < n/2$ and the majority of the participants are honest. Moreover, this protocol is robust for $k < t$ because t cheating participants can intentionally make targeted participants excluded

from the protocol (by issuing t complaints) such that the number of remaining participants doesn't meet the required threshold (t) for generating the group's public/private key pair. The Sign protocol satisfies only weak robustness because each run of the Sign protocol detects a cheating participant; since there are no more than k cheating participants, after k runs of the Sign protocol, a valid signature is generated with probability 1.□

3.3 Second Approach: Robustness by Identifying Cheaters

The protocol described in the previous approach does not detect cheating participants. The participant who issues a complaint can never prove that they received an inconsistent share. Likewise, the accused participant cannot prove that they sent the correct share to the complaining participant. In this approach we design a protocol that allows us to identify and punish cheating participants in the key generation protocol. This property is especially important in a blockchain environment where participants may have economic incentives for acting maliciously, and a financial punishment may stop rational adversaries from cheating[2] Note that the key generation protocol will run much faster in the absence of corrupted participants.

Next, we define a threshold signature scheme with cheating identifiability property. The definition is inspired by the *unanimously identifiable secret sharing* notion proposed by Ishai et. al. [IOS12].

Definition 4. *Let* $\mathsf{TS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a* (t, n) *threshold signature scheme with key generation, signing, and verification algorithms* KeyGen, Sign, *and* Verify, *respectively, and* $0 < \beta \leq 1$. TS *provides*

- β-***Cheating identifiability for key generation*** *if any participant who deviates from (cheats during) the* KeyGen *protocol is detected by honest participants with probability greater than* β.

Remark 1. Cheating identifiability allows achieving (weak) robustness even when the majority of participants are cheating (i.e., as long as there are t honest participants in the group, it is possible to complete the protocol successfully). This breaks the $k < n/2$ bound presented in the previous approach (Proposition 2) and only requires $k < n - t$ to complete the protocol.

Proposition 3. *If a key generation algorithm with a* β*-cheating identifiability property substitutes the* KeyGen *protocol of FROST for a setting with no more than* k *cheating participants, the resulting protocol will be at least* β*-weakly robust against* k *cheating participants.*

Proof. By using a β-cheating identifiable key generation protocol, cheating will be detected with probability at least β in KeyGen and with probability 1 in Sign protocols. So every cheating participant will be detected with probability at least β and the protocol achieves β-weak robustness. □

[2] This can be formalized as a decision-making problem.

4 Identifiable Cheating Entity (ICE) FROST

In this section, we propose a variation of the FROST protocol that can identify cheating participants in KeyGen protocol. The protocol follows the steps of the original FROST protocol described in Section 3.1 except for the case when an invalid share[3] is received by an honest participants. In FROST, such a share will be detected, and the protocol will simply terminate. By enabling the cheating identifiability mechanism, each participant issues a complaint against a malicious action instead of aborting the protocol. However, the complaint must be verifiable by honest participants.

To enable cheating identifiability, we note that all shares in the scheme are communicated through a secure channel: otherwise, the unforgeability of the scheme is completely lost. In practice, the sender establishes a secure channel by encrypting shares and broadcasting the encrypted values (ciphertext). One way to prove that the sender of a share has cheated is for the receiver to reveal the received encrypted value to every participant through revealing the decryption key. In this case, the decryption key should have special properties. In Theorem 1, we use these properties to prove the unforgeability of the proposed signature scheme. These special properties are as follows.

P1) The decryption key corresponding to the secure communication between each pair of participants should be unique. This property ensures that revealing the decryption key by the receiver P_i, who issues a complaint against the sender P_l, only reveals the share transmitted from P_l to P_i, and every other communication stays as secure as before revealing the decryption key between P_i and P_l.

P2) The decryption key should be verifiable by other participants. A verified decryption key ensures that the decryption returns the exact encrypted share; the receiver cannot deceive other participants by revealing a well-formed but fake decryption key that decrypts the ciphertext into some malformed share (i.e., any share different from what was encrypted and transmitted in the beginning).

The first property indicates that the simple application of a public-key encryption scheme is insufficient for our purposes. In a simple public-key encryption scheme, the decryption key is the receiver's private key. The reveal of the P_i's private key will reveal all the encrypted shares that are transmitted to P_i while we expect a scheme that allows the reveal of *only one* encrypted share transmitted from P_l to P_i.

To satisfy the above requirements, we use a variant of the ElGamal public-key encryption scheme [ElG85] that incorporates a symmetric encryption scheme and allows the encryption of arbitrary bit strings (in contrast to the original ElGamal scheme that requires the message to be a group element). Each participant is assigned a pair of public and private keys. We set the key used between each

[3] A share that is inconsistent with the commitment vector to the polynomial's coefficients, which was previously published by the sender of the share.

two participants acting as sender and receiver to be the "implicit" Diffie-Hellman (DH) key constructed from the receiver's public key and the sender's *ephemeral* public key. The sender encrypts the share using a symmetric encryption scheme under the shared DH key and broadcasts it. To issue a complaint about a malformed received share, the receiver must reveal the shared DH key that is unique to the two sides of the communication. Furthermore, we make the DH key verifiable by including proof of the "correct decisional Diffie-Hellman (DDH) triple". Such statement can be proved straightforwardly with a slight modification of Schnorr identification protocol.

4.1 (t, n) ICE FROST Protocol

Initialization. For the initial setup of the protocol, we have the following conditions:

- $\mathcal{P}$ is the set of participants, where $|\mathcal{P}| = n$. The threshold value t and the n participants are determined prior to the start of the protocol.
- $\mathbb{G}$ is a group of prime order q in which the DDH problem is hard, and g is a generator of that group.
- Each participant P_i, $1 \leq i \leq n$, is assigned a private key $sk_i \xleftarrow{\$} \mathbb{Z}_q$ and a public key $pk_i \leftarrow g^{sk_i}$.
- Honest participants want to sign a message m agreed upon externally to the scheme.
- The encryption scheme $\mathcal{E}_{sym} = (Enc_{sym}, Dec_{sym})$ is associated with the key space $\mathbb{G}$, the message space $\mathbb{Z}_q$, and the ciphertext space C.

The protocol consists of two main subprotocols, namely KeyGen and Sign, given in Figures 1 and 3, respectively. Figure 2 is the complaint procedure that runs only if a malicious behaviour happens during the key generation protocol.

The proposed ICE protocol is a secure robust threshold signature scheme with cheating identifiability property. Further, we discuss the properties in Definition 2, which are provided by ICE FROST .

Assumptions. We analyze the security of ICE FROST by presenting the following assumptions. In Section 7, we discuss how our implementation supports these assumptions.

- *Message Validation.* We assume that every participant checks the validity of the message m to be signed before issuing their share of the signature.
- *Reliable Message Delivery.* We assume that participants use reliable broadcast channels to send messages between one another.
- *Participant Identification.* In order to report misbehaving participants, we require that values submitted by the participants should be identifiable within the signing group. In practice, this assumption will be realized by including a personal signature of each participant besides their transmitted messages,
- *Availability of a unique public keys.* Each party can register a unique public key that is communicated to any other party.

Key Generation protocol (KeyGen)

Let H_1 and H_2 be two collision-resistant hash functions whose output is in $\mathbb{Z}_q^*$, and $HKDF$ be the HMAC-based key derivation function proposed in [Kra10] with an output tailored to be used in the $\mathcal{E}_{sym} = (Enc_{sym}, Dec_{sym})$ encryption scheme.

Round 1

1. For $1 \leq i \leq n$, every P_i samples t random values $(a_{i0}, ..., a_{i(t-1)}) \xleftarrow{\$} \mathbb{Z}_q$ and uses them 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 $r \xleftarrow{\$} \mathbb{Z}_q$, $R_i = g^r$, $c_i = H_1(i, \Phi, g^{a_{i0}}, R_i)$ and $\mu_i = r + a_{i0}.c_i$, with Φ being a context string to prevent replay attacks.
3. Every P_i samples sk_i randomly and computes $pk_i = g^{sk_i}$.
4. Every P_i computes a proof of knowledge of the secret key sk_i by calculating $\tau_i = (S_i, \nu_i)$, such that $k \xleftarrow{\$} \mathbb{Z}_q, S_i = g^k, d_i = H_2(i, \Phi, pk_i, S_i), \nu_i = k + sk_i.d_i$, with Φ being a context string to prevent replay attacks.
5. Every participant P_i computes a public commitment $\mathbf{C_i} = \langle \phi_{i0}, ..., \phi_{i(t-1)} \rangle$, where $\phi_{ij} = g^{a_{ij}}$ for $0 \leq j \leq t-1$.
6. Every P_i broadcasts $\mathbf{C_i}$, σ_i, pk_i, and τ_i.
7. Participant P_j generates a set $\mathcal{QL}$ of qualified dealers. (In the analysis, we show that the set of qualified participants is the same for every honest party, and therefore the set $\mathcal{QL}$ is not specific to a particular participant). Initially $\mathcal{QL} = \emptyset$. Upon receiving $\mathbf{C_i}$, σ_i, pk_i, and τ_i from participant P_i , where $i \neq j$, participant P_j verifies i)$\sigma_i = (R_i, \mu_i)$ by checking $R_i \stackrel{?}{=} g^{\mu_i}.\phi_{i0}^{-c_i}$, where $c_i = H_1(i, \Phi, \phi_{i0}, R_i)$, and ii)$\tau_i = (S_i, \nu_i)$ by checking $S_i \stackrel{?}{=} g^{\nu_i}.pk_i^{-d_i}$, where $d_i = H_2(i, \Phi, pk_i, S_i)$. If the check passed, P_j adds P_i to the list of qualified dealers. That is, $\mathcal{QL} = \mathcal{QL} \cup \{P_i\}$. On failure, P_j broadcasts ("`malicious`", $\mathtt{P_i}, \mathtt{P_j}$).

Round 2

1. Let $\mathcal{QL}^* = \mathcal{QL}$. Each P_l does the following for each $P_i \in \mathsf{QL}^*$, $i \neq l$:
   - Computes DH key $k_{li}^{DH} = pk_i^{sk_l}$ and symmetric key $k_{li}^{sym} = HKDF(k_{li}^{DH})$.
   - Encrypts $e_{li} = Enc_{sym}(k_{li}^{sym}, f_l(i))$.
   - Broadcasts $((l, i), e_{li})$.
2. Upon receipt of $((l, i), e_{li})$ from P_l, $1 \leq l \leq n$, if $P_l \in \mathcal{QL}^*$, then P_i does the following:
   - Computes $k_{li}^{DH} = pk_l^{sk_i}$ and $k_{li}^{sym} = HKDF(k_{li}^{DH})$.
   - Decrypts $\delta = Dec_{sym}(k_{li}^{DH}, e_{li})$.
   - Verifies the share by checking $g^{\delta} \stackrel{?}{=} \prod_{k=0}^{t-1} \phi_{lk}^{i^k \bmod q}$. If the share is incorrect, initiates the procedure Complain.
3. Participants resolve all complaints with the procedure Exclude and output a new set of qualified participants $\mathcal{QL}^*$ (similar to the set $\mathcal{QL}$ in Round 1, $\mathcal{QL}^*$ is not specific to a particular participant). If $|\mathcal{QL}^*| < t$, then KeyGen is aborted.
4. Each P_i calculates their private signing share by computing $s_i = \sum_{P_l \in \mathcal{QL}^*} f_l(i)$, stores s_i securely, and deletes each $f_l(i)$.
5. Each P_i calculates their public verification share $Y_i = g^{s_i}$ and the group's public key $Y = \prod_{P_j \in \mathcal{QL}^*} \phi_{j0}$. Any participant can compute the verification share of any other participant by finding $Y_i = \prod_{j \in \mathcal{QL}^*} \prod_{k=0}^{t-1} \phi_{jk}^{i^k \bmod q}$. Each P_i then broadcasts Y.

Fig. 1: Key generation protocol in ICE FROST

Complaint Management Protocol (Complain/Exclude)

Complain(i, l)

1. P_i computes a proof that k_{il}^{DH} and respectively k_{il}^{sym} is well-formed, which is a proof of knowledge of sk_i such that $(pk_i, pk_l, k_{il}^{DH}, A_1, A_2)$ is of the form $k_{il}^{DH} = pk_l^{sk_i}$, and $pk_i = g^{sk_i}$. To do so, P_i proceeds as follows:
   - Computes $A_1 = g^{\alpha}$, $A_2 = pk_l^{\alpha}$, where $\alpha \xleftarrow{\$} \mathbb{Z}_q$ and $h = H(pk_i, pk_l, k_{il}^{DH})$.
   - Computes $z = \alpha + h \cdot sk_i$.
   - The proof is $\pi = (A_1, A_2, z)$.
2. P_i issues a complaint against P_l by broadcasting the message $(\text{“}\texttt{complaint}\text{”}, \mathtt{P_i}, \mathtt{P_l}, \mathtt{k_{il}^{DH}}, \pi)$

Exclude$(\text{“}\texttt{complaint}\text{”}, \mathtt{P_i}, \mathtt{P_l}, \mathtt{k_{il}^{DH}}, \pi)$

Every non-excluded party P_j, $1 \leq j \leq n$, $j \neq i$ and $j \neq \ell$,

1. Verifies the proof π by checking $A_1 \cdot pk_i^h \stackrel{?}{=} g^z$, $A_2 \cdot (k_{il}^{DH})^h \stackrel{?}{=} pk_l^z$, where $\pi = (A_1, A_2, z)$ and $h = H(pk_i, pk_l, k_{il}^{DH})$. If the proof is valid, proceeds to step 2. Else, broadcasts $(\text{“}\texttt{malicious}\text{”}, \mathtt{P_i}, \mathtt{P_j})$, excludes P_i from the list of participants, that is, $\mathcal{QL}^* = \mathcal{QL}^* \backslash \{P_i\}$, and terminates the procedure.
2. If there is an entry $((l, i), pk_l, e_{li})$ published by P_i, the party proceeds to step 3. Else, broadcasts $(\text{“}\texttt{malicious}\text{”}, \mathtt{P_i}, \mathtt{P_j})$, excludes P_i from the list of participants, and terminates the procedure.
3. Computes $k_{il}^{sym} = HKDF(k_{il}^{DH})$ and then $\delta = Dec_{sym}(k_{il}^{sym}, e_{li})$. Verifies the decrypted share by checking $g^{\delta} \stackrel{?}{=} \prod_{k=0}^{t-1} \phi_{lk}^{i^k \bmod q}$. If the share is correct, broadcasts $(\text{“}\texttt{malicious}\text{”}, \mathtt{P_i}, \mathtt{P_j})$, and excludes P_i from the list of participants, that is, $\mathcal{QL}^* = \mathcal{QL}^* \backslash \{P_i\}$. Else, broadcasts $(\text{“}\texttt{malicious}\text{”}, \mathtt{P_l}, \mathtt{P_j})$ and excludes P_l, that is, $\mathcal{QL}^* = \mathcal{QL}^* \backslash \{P_l\}$.

Fig. 2: Complaint management procedure in ICE FROST

Signing protocol (Sign)

We assume that a key generation protocol has been successfully completed. Each of the n remaining participants now holds a secret share, and the group's public key is Y. Let H_1, H_2 be collision-resistant hash functions whose outputs are in $\mathbb{Z}_q^*$.

Round 1

1. The participants randomly select $\mathcal{S} \subset \mathcal{QL}^*$ such that $|\mathcal{S}| \geq t$. The signing participants are $P_i \in \mathcal{S}$.
2. Each P_i samples single-use nonces $(d_i, e_i) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.
3. Each P_i broadcasts (D_i, E_i), where $D_i = g^{d_i}$ and $E_i = g^{e_i}$. They can broadcast a list of $(D_i^{(r)}, E_i^{(r)})_{r \in [1,n]}$ for ν rounds of the protocol, where the rth element of the list will be used in the rth run of the protocol (given that no participant cheats during the signing).

Round 2

1. Each P_i constructs $B = \langle (l, D_l, E_l) \rangle_{l \in \mathcal{S}}$, computes the binding values $\rho_l = H_1(l, m, B)$, $l \in \mathcal{S}$, and then derives the group commitment $R = \prod_{l \in \mathcal{S}} D_l.(E_l)^{\rho_l}$ and the challenge $c = H_2(R, Y, m)$.
2. Each P_i computes their response using their long-lived secret share s_i by computing $z_i = d_i + (e_i \cdot \rho_i) + \lambda_i \cdot s_i.c$ using $\mathcal{S}$ to determine the ith Lagrange coefficient λ_i.
3. Each P_i deletes (d_i, D_i, e_i, E_i) from their local storage and then broadcasts z_i.
4. Each P_i does the following:
   - Upon receiving z_l from participant P_l, $l \in \mathcal{S}$, $l \neq i$, verifies the validity of the response by checking $g^{z_l} \stackrel{?}{=} R_l.Y_l^{c.\lambda_l}$. On failure, broadcasts ("`malicious`", $\mathtt{P_1}, \mathtt{P_i}$), excludes P_l from the list of participants that is, $\mathcal{S} = \mathcal{S} \backslash \{P_l\}$, and proceeds to step 5.
   - If all responses are correct, computes the group's response $z = \sum z_i$.
   - Broadcasts the signature $\sigma = (R, z)$ along with m and terminates the procedure.
5. If no signature has been generated and some participants have been excluded, go back to Round 1, step 2 and use the updated $\mathcal{S}$. If the resulting set has fewer than t members, abort the signature generation.

Fig. 3: Signing protocol in ICE FROST

Preprocessing. Each participant P_i begins by generating a list of single-use private nonce pairs and corresponding public commitment shares $\langle((d_{ij}, D_{ij} = g^{d_{ij}}), (e_{ij}, E_{ij} = g^{e_{ij}}))\rangle_{j=1}^{\pi}$, where j is a counter that identifies the next nonce or commitment share pair available to use for signing. Each P_i then publishes (i, L_i), where L_i is their list of commitment shares $L_i = \langle(D_{ij}, E_{ij})\rangle_{j=1}^{\pi}$. The location where participants publish these values can depend on the implementation. The set of (i, L_i) tuples is then stored by any entity that might perform the signature aggregator role during the signing process.

4.2 Existential Unforgeability of ICE FROST

In this section, we show the security for the standard notion of existential unforgeability against chosen message attacks (EUF-CMA) in the sense of [GMR88] for ICE FROST. The security argument works by demonstrating that for a forger F, the difficulty of forging a FROST signature by performing an adaptive chosen message attack in the random oracle model reduces to the difficulty of computing the discrete logarithm of an arbitrary challenge value ω in the underlying group for an adversary A who controls a number of participants less than the threshold t.

Let q_{h1} and q_{h2} be the number of queries made to the random oracle in the key generation and signing algorithms, respectively, π be the batch size in preprocessing protocol, q_p be the number of allowed preprocess queries, and q_s be the number of signing queries. The following is the *unfogeability* theorem of ICE FROST.

Theorem 1. *If the discrete logarithm problem in $\mathbb{G}$ is (τ', ϵ')-hard,then ICE FROST signature scheme over $\mathbb{G}$ with n signing participants, a threshold of t, and a preprocess batch size of π is $(\tau, q_{h1} + q_{h2}, q_p, q_s, \epsilon)$-secure whenever*

$$\epsilon' \leq \frac{\Big(\epsilon + \mathsf{Adv}_{\mathsf{DH}}(\mathsf{B}_1) + (n-t)\mathsf{Adv}_{\mathsf{CPA}}{}^{Enc_{sym}}(\mathsf{B}_2)\Big)^2}{q_{h1}\big(2(q_{h1}+q_{h2}) + (\pi+1)q_p + 1\big)},$$

where $\mathsf{Adv}_{\mathsf{DH}}\mathsf{B}_1$ is B_1's advantage of distinguishing a DH key from a uniformly random key under the DDH assumption $\mathsf{Adv}_{\mathsf{CPA}}{}^{Enc_{sym}}(\mathsf{B}_2)$ is B_2's CPA advantage in distinguishing the encryption of two distinct messages encrypted by $Enc_{sym}(\cdot)$ scheme.

Proof Sketch. The main difference in the unforgeability proof of ICE FROST is that, in contrast to FROST, ideal secure communication channels are not assumed in our case. Instead, we directly implement the secure channels by proper application of an encryption scheme. Secure communication channels are required to ensure the security of the shared secret by each individual participant. Note that when $t = n$ and $n - 1$ participants are corrupted, using encryption in our scheme is not necessary and the (information-theoretic) security is guaranteed because the number of shares is not enough for reconstructing the secret. In

this case, the simulation is straightforward and the security proof of ICE FROST is the same as the security proof of FROST.

The interesting case is when $t < n$ and the n encrypted shares published by P_i, $i = 1, \ldots, n$ enable reconstruction of the honest participant's secret. Therefore, we need to revolve to the semantic security of a public key encryption scheme. For this, we use public key encryption to send the encrypted shares over an *authenticated broadcast channel*. Each participant P_i should broadcast an encryption of its j-th share $f_i(j)$ under the receiver's public key, party P_j.

Ideal secure channels in ICE FROST are realized by encrypting and then broadcasting shares of the secret generated by each (honest) participant. The participant P_i broadcasts the encryption of the generated share for participant P_j, denoted by $f_i(j)$, under P_j's public key. Note that in the reduction to the discrete logarithm problem, the simulator cannot know all shares of the honest participants' secret, as it amounts to knowing the discrete logarithm challenge whenever the number of honest participants is greater than t. Specifically, when simulating honest participants to the adversary, the simulator cannot send the encryption of the right shares to honest participants but only inconsistent values. However, plaintext indistinguishability of the encryption scheme implies that this change is unnoticed from the adversary's viewpoint.

The Role of Cheating Identifiability. The instantiation of cheating identifiability according to the complaint management protocol in ICE FROST (Figure 2 has negative impact on the indistinguishability of incorrectly encrypted shares. Indeed, after an honest party identifies a cheater, it reveals non-trivial information of the share: (at least) a bit saying if the decrypted share is valid or not. This has two implications on the simulation. First, the simulator should be able to decrypt received encrypted messages, using the honest participant's secret key, to be able to issue a complaint whenever the received share is inconsistent with the corresponding initially published commitment vector. And second, at the same time keep secret the inconsistent shares[4]. We can solve this tension by revolving to CCA security of the encryption scheme, rather that CPA security. In this way the reduction can use the decryption oracle to check whether the encrypted shares of the adversary contain invalid shares or not.

We note that in our specific construction, a public key encryption scheme is not explicitly used, but instead, each pair of participants derive their DH key and use that key to encrypt/decrypt the share using the DH key as a symmetric encryption key. We require the sender to prove knowledge of its secret key sk_S such that $pk_S = g^{sk_S}$, which in turn implies that it also knows the secret key $K = pk_R^{sk_S}$. Note that this implicitly proves that the symmetric encryption scheme where the key is the DH key is CCA secure. To avoid to formally define this "tweaked" encryption scheme, where both the sender an receiver have pub-

[4] There are other ways in which this inconsistent simulation might become problematic. If, for example, the adversary demands to corrupt an honest party after a simulated inconsistent share addressed to the same party was sent, we are forced to stop the simulation as it amounts to reveal the inconsistency. Nevertheless, if we limit the attacks only to static adversaries, this is no longer a problem.

lic/secret key pairs, in the proof we leave this implicit. Our reduction constructs the decryption oracle itself using the PoK of the secret key to recompute the DH key and decrypt.

After having dealt with inconsistent shares and cheating identifiability, we may proceed simulation as in the original proof. According to Lemma 1, the generalized forking algorithm $\mathsf{GF_A}$ generates a random tape β and q_r random values $h_1, \ldots, h_{q_r} \xleftarrow{\$} H$, where $q_r = 2q_{h_2} + (\pi+1)q_p + 1$, and runs the simulator $\mathsf{A}(Y, \{h_1, \ldots, h_{q_r}\}, \beta)$ who invokes the forger F by simulating the responses to its random oracle queries using $\{h_1, \ldots, h_{q_r}\}$, and also simulates the honest party P_t in KeyGen, Preprocess, and Sign. To carry on simulation, we need the secret key of the adversary in order to derive the DH key shared between a malicious and honest party. Only in this way we can simulate encrypted messages for the adversary. We use the general Forking Lemma here again to extract the adversary's secret key and encrypt secret shares by the simulator.

The forger F uses h_J to generate a forged signature σ and outputs (σ, J) or a special symbol $\perp$ indicating a failure to output a forgery. The forking algorithm $\mathsf{GF_A}$ then generates fresh random values $h'_1, \ldots, h'_{n_r} \xleftarrow{\$} H$ and forks from h_J, i.e., runs the simulator $A(Y, \{h_1, \ldots, h_{j-1}, h'_J, \ldots, h'_{n_r}\}, \beta)$, which invokes the forger F as before and outputs (σ', J') or a special symbol σ'. The outputs (σ, σ') and $(h_J, h'_{J'})$ are used to solve the DLP for $\omega \in \mathbb{G}$. We explain the details of the proof in Appendix A.

4.3 Cheating Identifiablity

A participant can deviate from the KeyGen protocol by: i) not committing to an initial secret by each participant, ii) sending non-decryptable secret shares to at least one participant, iii) dealing inconsistency to at least one participant, or iv) accusing an honest participant of malicious behaviour. The `complain/exclude` procedure is proposed to identify the cheating participant in all cases. This procedure should satisfy two conditions:

C1) Imply that the participant accused of malicious behaviour did not follow the protocol.

C2) The procedure should be publicly verifiable.

The first condition guarantees that we cannot accuse honest participants, while the second condition guarantees that honest participants can remain in a consistent consensus about which participant was dishonest. The combination of these two conditions allows *all* honest parties to maintain the same set of qualified participants $\mathcal{QL}^*$.

Satisfaction of C1. A participant will be marked as "malicious" in the KeyGen protocol in three cases:

- *Round 1 - step 5:* The failure of a participant at this stage means they are not committed to an initial secret. We can easily identify the misbehaving party thanks to the authenticated communication and the fact that the verification failure implies proofs were not computed correctly.

- *Procedure* `Exclude` - *step 1.* This step is to check if the shared secret sent from P_i to P_l is encrypted under a proper key that can be used by P_l to decrypt. This is exactly as the previous check where the misbehaving party computes an invalid proof.
- Procedure `Exclude` - step 3. Due to the authenticated communication, we can always identify P_l, the author of $((i,l), ve_{i,l})$. For a correctly accused P_l, it must hold that $g^{\mathsf{Dec}_{sym}(k_{i,l}, e_{i,l})} \neq \prod_{k=0}^{t-1} \phi_{l,k}^{i^k}$, where $k_{i,l} = g^{u \cdot sk_i}$. If P_i issues an incorrect complaint against P_l, we have $g^{\mathsf{Dec}_{sym}(k_{i,l}, e_{i,l})} = \prod_{k=0}^{t-1} \phi_{l,k}^{i^k}$ and P_i will be detected as a malicious participant.

As noted, each "malicious" message generated by the KeyGen protocol corresponds to one of the possible misbehaviours of a participant. As a result, condition C1 is satisfied.

Satisfaction of C2. All checks in the steps described above that cause a participant to be marked as "malicious" can be publicly verified by every other participant. This satisfies the condition C2 and moreover, lets all honest participants generate the same set of qualified users.

4.4 Robustness of ICE FROST

ICE FROST provides robustness in KeyGen, as the protocol will continue after identifying cheaters. On the other hand, the Sign protocol is weakly robust; therefore, the complete protocol provides weak robustness. Cheating identifiability in both protocols is with probability 1.

Robustness of KeyGen. Let n be the total number of participants with at least t honest participants. The robustness of the key generation protocol in ICE FROST is concluded from the robustness of Pedersen's DKG protocol. Note that the difference between the KeyGen in ICE FROST and Pedersen's DKG protocol is in the way they handle complaints. In KeyGen, once a party receives a complaint, instead of defending itself by revealing the correct secret share in plain text (similar to Pedersen's DKG), it reveals the secret key established corresponding to the encrypted share that has been broadcasted previously. This, in effect, reveals the secret share that is allegedly sent to the complainer and identifies the participant who has cheated either by distributing an inconsistent share or by maliciously issuing a complaint against an honest party. Moreover, in Pedersen's DKG, once a party receives more than t complaints, it is marked malicious and then removed from the list of qualified participants. Such a cheating participant will be identified by ICE FROST immediately after the first complaint. As a result, the outcomes of the KeyGen protocol in ICE FROST and Pedersen's DKG protocol will be exactly the same, and the robustness will follow from the robustness of Pedersen's DKG as long as enough honest participants remain in the scheme after excluding the cheaters. Suppose k is the number of cheating participants. The KeyGen is robust if $n - t \geq k$.

Weak robustness of Sign. The Sign protocol is able to identify the cheater whenever it fails and therefore provides weak robustness. Each run of the protocol identifies and removes one malicious participant. Suppose $\mathcal{S}$ is the set of signers consisting of at least t honest participants. According to Proposition 3, after at most k runs of the Sign protocol, it will output a valid signature (because in each run one malicious participant will be detected and removed).

In (t, n) ICE FROST, since KeyGen is robust while Sign in weakly robust, the maximum number of runs that will guarantee a successful execution of the signature scheme depends only on the number of cheating participants in the Sign protocol. We present this as the following proposition for easier reference.

Proposition 4. *For (t, n) ICE FROST protocol, let k_1 be the number of cheating participants in* KeyGen *and k_2 be the number of cheating participants in* Sign. *Then ICE FROST will run successfully after at most k_2 runs as long as $k_1 \leq n - t$.*

5 ICE FROST with Static Public/Private Keys

One of the primary motivations for developing the ICE FROST signature scheme is using it in blockchain environments. In particular, we consider a blockchain environment consisting of the main chain and an arbitrary number of side-chains. Each side-chain will use the ICE FROST scheme to sign messages that need to be verified by the main chain. In such an environment, it is desirable to assign static public and private keys to each side-chain to avoid transmitting updated public keys after each round of key generation.

In this section, we use a Proactive Secret Sharing (PSS) scheme to modify the ICE FROST scheme into a signature scheme that uses static group public and private keys to sign messages. To guarantee the security of the group's long-lived keys, we update the long-lived signing keys of each participant every so often, while keeping the group's signing keys unchanged. The key update protocol will replace the key generation protocol in Section 4.1. Moreover, the key update protocol allows changing the setup of the protocol into a scheme with a different set of participants (probably with different cardinality) and a different threshold. This flexibility is especially important after identifying the cheating participant in ICE FROST because the cheater will be excluded from the protocol in subsequent rounds.

5.1 Key Update Protocol for ICE FROST with Static Keys

The key update protocol is a redistribution of secret shares that provides each share-holding participant with a fresh signing share while allowing to adjust the total number of participants, the signature's threshold, and the set of share-holding participants to be updated at each round of execution. More precisely, the key update protocol allows an (t, n) ICE FROST protocol executed by a set of participants $\mathcal{P}$ to be updated into an (t', n') ICE FROST protocol executed by a possibly different set of participants $\mathcal{P}'$.

The uth execution of the key update protocol is specified by $\mathsf{KeyUpd}^{(u)}$, where $0 \leq u \leq T$, and T is the maximum number of key update executions that subsequently specifies the life-time of the group's long-lived private and public keys, and $\mathsf{KeyUpd}^{(0)}$ is the KeyGen protocol described in Fig 1. We denote the set of participants in $\mathsf{KeyUpd}^{(u)}$ by $\mathcal{P}^{(u)}$, where $|\mathcal{P}^{(u)}| = n^{(u)}$ and the desired threshold for the signature is $t^{(u)}$. The flow of $\mathsf{KeyUpd}^{(u)}$ for $0 < u$ is described in Fig. 4. The key update protocol uses the PSS scheme of [DJ97] to redistribute shares.

Key Update protocol ($\mathsf{KeyUpd}^{(u)}$)

Let $s_i^{(u-1)}$ be the private signing share of $P_i \in \mathcal{P}^{(u-1)}$ that is obtained by running $\mathsf{KeyUpd}^{(u-1)}$. The rest of the setting is the same as the KeyGen setting in Fig. 1.

Round 1

- For $1 \leq i \leq n^{(u-1)}$, every P_i secretly shares $s_i^{(u-1)}$ by sampling $t^{(u)} - 1$ random values $(a_{i1}, ..., a_{i(t-1)}) \xleftarrow{\$} \mathbb{Z}_q$ and uses them as coefficients to define a degree $t^{(u)}-1$ polynomial $f_i(x) = s_i^{(u-1)} + \sum_{j=1}^{t^{(u)}-1} a_{ij}x^j$.
- Follow Steps 2 to 7 of Round 1 in KeyGen.

Round 2

- Follow Steps 1 to 3 of Round 2 in KeyGen.
- Each P_i calculates their private signing share by computing $s_i^{(u)} = \sum_{P_l \in \mathcal{QL}^*} f_l(i).\lambda_l$, where λ_l is the Lagrange coefficient corresponding to P_l with an ID id_l. Then P_i stores $s_i^{(u)}$ securely, and deletes each $f_l(i)$.
- Each P_i calculates their public verification share $Y_i = g^{s_i^{(u)}}$ and the group's public key $Y = \prod_{P_j \in \mathcal{QL}^*} Y_j^{\lambda_j}$. Any participant can compute the verification share of any other participant by finding $Y_i = \prod_{P_j \in \mathcal{QL}^*} (\prod_{k=0}^{t^{(u)}-1} \phi_{jk}^{i^k \bmod q})^{\lambda_j}$. Each P_i then broadcasts Y.

Fig. 4: Key update protocol in ICE FROST

A note for share renewal. If the set of participants and the threshold remain the same, the purpose of executing the key update protocol is only to refresh the shares in order to remove possible leaks. In this case, it is sufficient for every participant to execute the KeyGen protocol but instead of dealing the secret, pass the value "0" and distribute the corresponding shares to other participants.

Each participant obtains their renewed share by adding up the received shares and adding the result to their previous secret share value.

5.2 Forward Secrecy

Protecting long-lived signing keys of a threshold signature scheme requires a guarantee of the long-term security of the encrypted communication during each session. ICE FROST provides this guarantee by establishing and exploiting independent DH keys for each session. In each session, the DH keys exchanged between each pair of participants depend only on the (ephemeral) public and private keys used by that pair. Therefore, if a single session is compromised, it affects only the security of the current session, and an adversary cannot use the compromised secret keying material to extract information from previous sessions. This property ensures that as long as an adversary cannot compromise more than $t-1$ participants in each session, the signature scheme remains unforgeable (according to Theorem 1 and the adversary cannot combine their knowledge from attacks in multiple sessions to recover the long-lived signing keys.

The described property is called *forward secrecy* in literature and was first defined by Diffie et al. in [DVOW92]. According to them, "an authenticated key exchange protocol provides perfect forward secrecy (PFS) if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs." In ICE FROST, forward secrecy is offered to securely communicate shares between participants because the only long-term secret keying material stored by users is what they use for authentication. If this secret key is compromised, the security of the exchanged keys from earlier runs is not affected because an independent Diffie-Hellman key exchange is used in each session; Diffie-Hellman key exchange has no long-term keying material.

6 ICE FROST with Large Number of Participants

In the following, we elaborate on a few operational considerations that can improve the efficiency of ICE FROST when the total number of participants n is relatively large. Let γ be the ratio of cheating participants and $\nu < 1-\gamma$ be the ratio of honest participants required to generate a signature.

6.1 Improving Weak Robustness of ICE FROST

As noted in Proposition 4, the maximum number of re-runs to guarantee the successful signature generation by ICE FROST depends only on the number of cheating participants in the Sign protocol. When n is too large, the number of cheating participants will also be too large ($k = \gamma.n$); it means that weak robustness, in this case, may not provide sufficient guarantee of the signature generation in a relatively short time. An efficient solution to decrease k in this situation is to randomly choose $\mathcal{S}$, the set of signers and, instead of choosing

another set $\mathcal{S}'$ after identifying a cheating participant P_c, update $\mathcal{S}$ by eliminating only P_c from it; that is $\mathcal{S}' = \mathcal{S} \backslash \{P_c\}$. Let s be the size of $\mathcal{S}$, then with a random choice of $\mathcal{S}$ and eliminating cheaters from it, assuming a large n, the protocol will run successfully after almost $\gamma.s < \gamma.n$ runs as long as $\mathcal{S}$ includes enough ($t = \nu.n$) honest participants. The random choice of signers is especially practical in blockchain environments where a public verifiable randomness for the choice of $\mathcal{S}$ is available and $\mathcal{S}$ can be chosen from the set of online validators.

6.2 Choice of the Size of the Group of Signers

Ideally, the size of the set of signers $\mathcal{S}$ should be as small as possible because i)smaller $\mathcal{S}$ means fewer participants need to be online for signing messages ii) the signature generation will proceed faster (more efficiently) with a smaller number of participants, and iii) smaller $\mathcal{S}$ contains fewer cheating participants, which means that the protocol is gauranteed to generate a valid signatures after fewer re-runs. The only restriction is that $\mathcal{S}$ should contain at least t honest participants.

To relate s, the size of randomly chosen set of signing participants, to the probability of having at least t honest participants in $\mathcal{S}$, let's index participants in $\mathcal{S}$ from $i = 1, \ldots, s$. Next, consider the following random variable: $X_i = 1$ when the party P_i is "honest" and $X_i = 0$ otherwise. Then $X = \sum_{i=1}^{n} X_i$ is the number of honest parties. The question is, "What is the minimum amount of s that ensures $\Pr[X \geq t+1] \geq \alpha$, where α is the probability of containing t honest participants in $\mathcal{S}$?"

We use the Chernoff bounds to s, the size of the group of signers who perform a (t, n) ICE FROST protocol. First, we restate these bounds:

Chernoff bounds[C$^+$52]. Let $X = \sum_{i=1}^{n} X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_is are independent. Let $\mu = \mathbb{E}(X) = \sum_{i=1}^{n} p_i$. Then

(i) Upper Tail: $\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu}$

(ii) Lower Tail: $\Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$

We use the lower tail of the Chernoff bound to get an upper bound on the failure probability, that is $\Pr[X \leq t+1]$. Suppose n is the total number of parties, h is the number of honest parties, and k is the number of cheating participants. We have

$$\mu = s\frac{h}{n} = s(1-\gamma)$$
$$(1-\delta)\mu = t = \nu.n \Rightarrow \delta = 1 - \frac{\nu.n}{\mu} = 1 - \frac{\nu.n}{s(1-\gamma)}$$
$$\Rightarrow \Pr[X \leq \nu.n] \leq e^{-(s(1-\gamma)-\nu.n)^2/2s(1-\gamma)}$$

To guarantee that the failure probability of the protocol is less than a small probability $1 - \alpha$, we need to have

$$e^{-(s(1-\gamma)-\nu.n)^2/2s(1-\gamma)} \leq 1-\alpha \Rightarrow (s(1-\gamma)-\nu.n)^2/2s(1-\gamma) \geq -ln(1-\alpha)$$

By finding the value of s according to the above inequality we can make sure the success probability is always more than α.

Approximation: Suppose $1-\alpha = e^{-x}$ and $x = \frac{s(1-\gamma)}{x'}$. Then we have

$$s(1-\gamma) - nu.n \geq \sqrt{2/x'}s(1-\gamma) \Rightarrow s(1-\gamma)(1-\sqrt{2/x'}) \geq \nu.n$$

$$\Rightarrow s \geq \frac{\nu.n}{(1-\gamma)(1-\sqrt{2/x'})}.$$

Now note that the value of x is conventionally in the [1-4] interval (to having an error rate between 50% to 0.1%). Therefore, for a large enough n and not too small γ, the value of x' is relatively large and $1-\sqrt{2/x'}$ is close to 1. That is,

$$s \approx \frac{\nu.n}{1-\gamma}$$

For a more accurate approximation, we use $s(1-\gamma) \geq \gamma.n$ and thus $x' \approx \gamma n/x$. Therefore,

$$s \approx \frac{\nu.n}{1-\gamma}/(1-\sqrt{2x/\gamma n})$$

After choosing appropriate s, the number of cheating participants in $\mathcal{S}$ denoted by k' determines the maximum number of runs of the Sign protocol in order to guarantee that a valid signature will be generated. If n is large enough, then $k' = \gamma.s \approx \frac{\gamma.\nu.n}{(1-\gamma)}$.

For the case where $\gamma.n = \nu.n - 1$, Table 1 gives numerical values for the probability of containing at least t honest participants in $\mathcal{S}$ depending on the number of selected participants for generating the signature.

	Minimum s value		
n	Pr	$\gamma = 0.2$	$\gamma = 0.33$
100	90%	30	56
	99.9%	35	62
	100%	42	68
500	90%	133	259
	99.9%	143	273
	100%	202	332

Table 1: Minimum size of the group of signers for a (probabilistically) successful generation of a (t, n) signature

7 Implementation

We have implemented the protocols for key generation, complaint creation and management, and signing for the proposed scheme. The results confirm that the signature scheme is efficient and fast enough for practical applications. We run benchmarks on the ICE FROST version of protocol from Section 4 (which uses non-static public/private keys for participants). We compare the performance of our protocol to the runtimes of FROST in [KG20]. The complaint management sub-protocol is specific to ICE FROST, so for a fair comparison to FROST, we consider ICE FROST without any issued complaints.

All benchmarks were performed on Ubuntu 18.04 LTS running on an Azure Esv3 machine with 32 CPUs and 256GB of RAM. Each benchmark is associated with the computation time related to a given task performed by one participant. Therefore the communication time is not considered in our benchmarks. In general, communication time depends mainly on the infrastructure where the protocol is executed and may vary for different infrastructures. The communication complexity of our protocol can be estimated theoretically (see Section 9 for details).

Benchmark data are averaged over 1000 consecutive runs of the protocol to eliminate possible inaccuracies in individual runs. Our implementation differs slightly from the description of ICE FROST and FROST protocols in Section 4 and [KG20, Section 5], respectively: the first step in Round 2 of both protocols (where each participant calculates shares for other participants) is included in Round 1. Also, to implement the required "secure communication" in FROST, we encrypt shares during Round 1 before sending. For a fair comparison, we used the same encryption scheme, AES encryption in Counter Mode of operation, to encrypt secret shares in FROST and ICE FROST.

It should be noted that we implemented our scheme in Rust. Finite field computations of our benchmarks were performed on the elliptic curve `Curve25519` from [Lov], and by default, we used a `u64` backend type for this curve. For the benchmarks, we used the criterion tool [Too].

7.1 Benchmarking Key Generation Time

Key generation in FROST and ICE FROST consists of two rounds. In both protocols, Round 2 takes longer than Round 1, mainly because of the share verification step in Round 2.

We benchmark the protocols for the number of participants $n = 100, 200, 300,$ and 500 and the threshold $t = n/3$ and $2n/3$. As one would expect, the time cost of the protocol increases as the number of participants or the threshold grows.

We expect the running time of Round 1 to consist of the following elements:

- $C_{cg}(t)$: the execution time for generating polynomial coefficients and then committing to them. This time is a linear function of the threshold and can be written as $C_{cg}(t) = t \times x_{cg}$, where t is the threshold and x_{cg} is the time for generating one single coefficient and committing to it.

- C_{pg} : the execution time for generating proof of knowledge by each participant. This step is independent of the number of players or the threshold. This time in ICE FROST is twice the time in FROST because each participant in ICE FROST generates two proofs of knowledge instead of one in FROST.
- $C_{pv}(n)$: the execution time for verifying other participants' proofs. This time in ICE FROST is twice the time in FROST because two proofs are verified in ICE FROST, while only one proof is verified in FROST. The verification time is a linear function of the total number of participants and can be written as $C_{pv}(n) = n \times x_{pv}$ for FROST and $C_{pv}(n) = 2n \times x_{pv}$, where n is the total number of participants and x_{pv} is the verification time for one proof.
- $C_{sh}(t, n)$: the execution time for calculating each participant's share and the mutual DH key, and then encrypting the share under the DH key. This step is a function of t, the degree of the share calculation polynomial, and n, the total number of participants; it can be expressed as $O(tn)$.

Using our benchmark data, we estimated the above constants in milliseconds as follows: $x_{cg} = 0.035$ ms, $C_{pg} = 1.2$ ms, and $x_{pv} = 0.094$ ms. Therefore, for running time of the first round in the key generation phase of a (t, n) FROST/ICE FROST scheme, we have:

$$\begin{aligned} Time^{\text{KGen-1}}_{FROST}(t, n) &= C_{pg} + t \times x_{cg} + (n-1) \times x_{pv} + C_{sh} \\ &= 1.2 + t \times 0.035 + (n-1) \times 0.094 + O(nt) \end{aligned}$$

$$\begin{aligned} Time^{\text{KGen-1}}_{ICE\text{-}FROST}(t, n) &= 2C_{pg} + t \times x_{cg} + 2(n-1) \times x_{pv} + C_{sh} \\ &= 2.4 + t \times 0.035 + 2(n-1) \times 0.094 + O(nt) \end{aligned}$$

The running time of `Round 2` in the key generation protocol (excluding the first step) consists of the following three components:

- $C_{shv}(t, n)$: the execution time for decrypting and then verifying the received shares. This step's running time depends on the degree of the secret-sharing polynomial (t) and the number of received shares n.
- $C_{kc}(n)$: the execution time for calculating the long-lived key, i.e., the sum of received shares. This step's running time depends only on n, the number of participants.
- $C_{gpk}(n)$: the execution time for calculating the group's public key that will be used for signing. This step's running time depends only on n, the number of participants.

Note that we do not consider the time required to verify public verification shares of participants, but depending on the level of trust in the system, each participant can verify public verification shares of all other participants or only some of them. The total running time of `Round 2` in both FROST and ICE FROST is

$$Time^{KGen\text{-}2}_{(ICE)FROST}(t,n) = C_{shv}(t,n) + C_{kc}(n) + C_{gpk}(n)$$

The raw benchmark data for key generation of FROST and ICE FROST are given in Tables 2 and 3, respectively, in Appendix B.

7.2 Benchmarking Signing

We benchmark the signing time for a single participant and the signature aggregation time for t participants. The signing time depends only on the number of active participants (t) but does not depend on n, the total number of participants. This benchmark is the same for FROST and ICE FROST since the signing protocol is the same in both protocols:

The output of the signing protocol is a regular Schnorr signature that will be verified by a Schnorr signature verification algorithm. The verification time is independent of t and n and, in our setting, is 0.097 milliseconds.

The raw benchmark data for signing are given in Table 4 in Appendix B.

7.3 Benchmarking Complaint Management

We benchmark complaint generation and verification time in our protocol:

$$\text{complaint generation} : 0.11 \text{ ms}$$

$$\text{complaint verification} : 0.27 \text{ ms}$$

Note that for the rest of the benchmarks, we assumed that none of the participants would cheat, therefore, the complaint management protocol would not be triggered. Depending on the number of cheating participants (k), the running time of the key generation protocol may increase by another $k \times (0.11 + 0.27)$ milliseconds.

7.4 Benchmarking Data Complexity

The signature consists of a point in the Ristretto group (4 FieldElement, by default with `u64` backend for a total of 4×5 `u64` $= 160$ bytes) + a scalar of 32 bytes. Total signature size $= 192$ bytes. For a threshold t, the total amount of data d in bytes that the signature aggregator will receive, is

$$d_{\text{ICE-FROST}}(t) = t \times 192 \text{ Bytes}$$

8 Security Against Known Attacks and Vulnerabilities

Security of ICE FROST covers not only the unforgeability of the signature. It provides security against a number of known attacks that we list and describe in the following.

Rogue-key attack. Security against rogue-key attacks is achieved through Step 2 of Round 1 in ICE FROST by requiring each participant to prove knowledge of their secret value.

Malicious signature aggregator. A malicious aggregator is not able to forge a valid signature. This follows from the unforgeability of the signature scheme and the fact that an aggregator does not have any extra knowledge compared to the system adversary. However, a malicious signature aggregator does have the power to perform denial-of-service attacks.

Replay attack. The replay attack in ICE FROST is prevented by using the context string Φ.

Attack via Wagner's Algorithm. Drijvers et. al. [DEF+19] found an attack against some two-round Schnorr multisignature schemes. This attack can be performed when the adversary has control over either choosing the message m to be signed or the ability to adaptively choose their own individual commitments used to determine the group commitment ϕ after seeing commitments from all other signing parties. This attack in FROST and ICE FROST is prevented by using a binding technique made effective in step 4 of the signing algorithm.

Known $\mathbf{k_{i,l}}$ in consecutive rounds. After the execution of `complain/exclude` between the participants P_i and P_l, the secret key $k_{i,l}$ is revealed and cannot be used in future rounds of the protocol. To avoid this attack, the party who is marked as malicious at the end of `complain/exclude` is only allowed to take part in the next rounds of the protocol if it updates its key pair (the previous public key will be blacklisted).

Next, we discuss the potential security vulnerabilities of ICE FROST. Although these vulnerabilities will not necessarily lead to a major attack on the security of the scheme, we consider them and discuss potential mitigations.

Public/Private Key Distribution Bias. Gennaro et al. [GJKR99] showed that an adversary with control of more than 2 out of t participants can bias the generated public key during Pedersen's DKG algorithm [Ped91b] (see the description of the protocol in the beginning of Section 3). Let's assume that the adversary wants to bias the distribution toward keys whose last bit is 0. The adversary uses two corrupted parties P_1 and P_2 to conduct this attack. Initially, P_1 sends inconsistent shares to exactly $t-1$ parties (causing $t-1$ complaints, which is not sufficient for making P_1 disqualified). The adversary calculates $\alpha = \prod_{i=1}^{n} A_{i0}$, where $A_{i0} = g^{a_{i0}}$ and a_{i0} is the secret shared by P_i. If the last bit of $\alpha = 0$, the adversary does nothing, otherwise ($\alpha = 1$), P_2 issues a complaint against P_1 and makes P_1 disqualified. This attack produces a bias on the distribution of the generated key. As a mitigation, they proposed an alternative three-round protocol that doesn't allow the adversary to bias the

key distribution. In a more recent paper, Gennaro et al. [GJKR03] proved the security of a threshold signature scheme based on Pedersen's DKG algorithm without requiring the shared key to be uniformly random. However, achieving the security of the three-round protocol with the old approach requires using larger groups.

A similar attack can also affect the ICE FROST protocol. After seeing α, a malicious party can issue a fake complaint and eliminate itself from the protocol to bias the *public* key toward its will. By choosing large enough groups for the computations, this attack will not affect the security of the system according to [GJKR03]. However, we can improve the efficiency of the protocol by mitigating this attack and staying with smaller-size groups for the computations.

Mitigation for the public key bias attack. At the end of the KeyGen protocol, one of the remaining qualified participants is randomly selected (using a public verifiable randomness) and removed, causing their share to be eliminated from the calculation of the public key to ensure the generated public key is uniformly random.

Security implication of the attack mitigation. As mentioned before, Gennaro et al. [GJKR03] proved the security of the threshold Schnorr signature scheme implemented with Pedersen's DKG in spite of the attack corresponding to biasing the public key. However, the security reduction to the underlying hard problem is less efficient than the security reduction that exist for the original Schnorr scheme. Suppose q_h is the number of random oracle queries made by the adversary who breaks the unforgeability of a Schnorr signature scheme with probability ϵ. Then, according to the results of [PS96], this forgery implies the computation of the discrete logarithm with probability $\frac{\epsilon^2}{q_h}$ in comparable time with the forgery computation time. This means that by managing to remove the key bias attack and reducing the scheme of [GJKR03] to the original Schnorr signature, the forgery with probability ϵ corresponds to breaking the hardness of discrete logarithm with probability $\frac{\epsilon^2}{q_h}$. This is while the unforgeability proof of [GJKR03] (without reducing to the original Schnorr protocol and therefore without mitigating the key bias attack) shows that a forgery with probability ϵ implies solving the discrete logarithm problem with probability $\frac{\epsilon^2}{q_h^2}$ in comparable time with the forgery computation time. This is a factor of q_h degradation of security compared to the centralized version of this scheme.

Similar implication follows from our unforgeability proof. We show that if q_{h1} is the number of random oracle queries during Key Generation protocol and q_{h2} is the number of random oracle queries during Signing protocol, then a signature forgery that happens with probability ϵ, implies breaking the discrete logarithm with probability proportional to $\frac{\epsilon^2}{(q_{h1}+q_{h2}).q_{h1}}$. This is a factor of q_{h1} degradation of security compared to the centralized version as well as the FROST scheme. We note that in FROST, due to the abortion of protocol after detecting a malicious action, the key bias attack is naturally resolved at the cost of loosing robustness. Our mitigation of the key bias attack on the other hand allows achieving the

same level of security as the original Schnorr (and FROST) scheme while having a robust threshold signature scheme.

Proving the randomness of the generated key as the result of our mitigation and then proving the unforgeability of ICE FROST by reducing it to the original Schnorr signature with random keys remains as future work.

9 Computation and Communication Comparison

In this section we compare the computation and communication complexity of the proposed protocol with two other protocols; The first protocol is the original FROST protocol, where the extra computation and communication brings the cheating identifiability property to us. The second protocol is the DKG protocol in [NBBR16], which to the best of our knowledge, is the only other DKG algorithm that allows precise identification of malicious participants.

Comparison with FROST. In the absence of a malicious participant ICE FROST will run similar to FROST except for transmitting the shares. While in FROST each participant sends secret shares *securely* to the corresponding receivers, in ICE FROST shares are *encrypted* and then *broadcast*. As a result, during the KeyGen phase, each participant in FROST will receive $n-1$ secret shares, while in ICE FROST each participant will receive $\frac{n(n-1)}{2}$ messages related to secret shares, from which $n-1$ messages will be decrypted and the rest will be used if a related complaint is issued. The computation complexity of the `complain/exclude` protocol is the same as non-interactive zero-knowledge proofs; that is, $O(q\gamma)$, where 2^γ is the error probability of the zero-knowledge protocol. Since every party has to verify the proof, the total complexity related to this part will be $O(nq\gamma)$, where n is the maximum number of participants. This is an extra complexity of ICE FROST that brings cheating identifiability property in exchange.

Comparison with DKG of [NBBR16]. The common idea in both schemes is to encrypt and then broadcast the secret shares (instead of securely sending to the recipient). However, in ICE FROST we use a symmetric encryption scheme, while in [NBBR16] a public-key encryption scheme is used. As a result ICE FROST will run much faster. The other advantage of the DKG protocol of ICE FROST compared to [NBBR16] is that in ICE FROST, when a complaint is issued by P_i against P_j, the symmetric key k_{ij} will be revealed. After detecting the cheater and removing it from the protocol, the honest participant can still use its previous private key securely but in [NBBR16], the same situation requires both P_i and P_j to reveal their private keys and as a result the private key of both of them (even though one of them is honest) will be expired and need to be updated.

The last advantage of ICE FROST compared to [NBBR16] is that it is secure against rogue key attacks because we require the participant to show the knowledge of the private key at the beginning of the protocol while this is not the case in [NBBR16].

10 Conclusion

In this work, we introduced ICE FROST, an improvement over FROST Schnorr based threshold signature scheme that allows identification of cheating entities in KeyGen protocol. The cheating identifiability property provides a weak robustness guarantee for the threshold signature scheme that can be improved when the set of signers is randomly chosen and is sufficiently large. Our cheating identifiability mechanism allows every participants to individually check the validity of issued complaints against possibly cheating participants. This is done by revealing the DH key that is used for the encryption of secret shares between two particular participants who are involved in the complaint procedure. The participant who receives the complaint is responsible for defending itself by revealing the DH key and the prove to show the validity of the revealed key.

We implement our proposed scheme. Our implementation benchmarks show that the run time of the protocol is feasible for real world applications and in comparison to FROST doesn't have that much over head.

Our final contribution in this work is the introduction of static public keys for ICE FROST. With static public keys, the group's established public and private keys stays constant for the life-time of signers but the signing shares of each participant will be updated over time that ensures long-term security of the static keys. This contribution eases the verification process of the generated threshold signature as the group of signers only communicate their public key with the verifier once in their whole life.

There are different directions for future works. Achieving robustness in signing protocol of ICE FROST is an interesting question that remains to be answered in future. Implementation of ICE FROST in real blockchain systems is also a practical direction for future works. The proposed scheme is not optimized in terms of public communications that happen during the key generation phase (quadratic in terms of the number of participants). Optimizing the communication cost of ICE FROST is another direction for future works.

Acknowledgements. We thank Theo Gauthier, Travis Alan Baumbaugh, and Vijay Singh, our colleagues at ToposWare corporation, for helpful discussions on the robustness of ICE FROST and careful review and suggestions for improving the quality of our work.

References

AAM19. Aysajan Abidin, Abdelrahaman Aly, and Mustafa A Mustafa. Collaborative authentication using threshold cryptography. In *International Workshop on Emerging Technologies for Authorization and Authentication*, pages 122–137. Springer, 2019.

BBS03. Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemeas. In *International Workshop on Public Key Cryptography*, pages 85–99. Springer, 2003.

BDN18. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.

BGG17. Dan Boneh, Rosario Gennaro, and Steven Goldfeder. Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security. In *International Conference on Cryptology and Information Security in Latin America*, pages 352–377. Springer, 2017.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, 2006.

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2003.

C+52. Herman Chernoff et al. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.

CDD+99. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 311–326. Springer, 1999.

CDF01. Ronald Cramer, Ivan Damgård, and Serge Fehr. On the cost of reconstructing a secret, or vss with optimal reconstruction phase. In *Annual International Cryptology Conference*, pages 503–523. Springer, 2001.

CGG+20. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1769–1787, 2020.

CGMA85. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395. IEEE, 1985.

Cho11. Ashish Choudhury. Simple and asymptotically optimal t-cheater identifiable secret sharing scheme. *IACR Cryptol. ePrint Arch.*, 2011:330, 2011.

DEF+19. Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1084–1101. IEEE, 2019.

Des87. Yvo Desmedt. Society and group oriented cryptography: A new concept. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 120–127. Springer, 1987.

DF89. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Conference on the Theory and Application of Cryptology*, pages 307–315. Springer, 1989.

DJ97. Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Citeseer, 1997.

DK01. Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 152–165. Springer, 2001.

DVOW92. Whitfield Diffie, Paul C Van Oorschot, and Michael J Wiener. Authentication and authenticated key exchanges. *Designs, Codes and cryptography*, 2(2):107–125, 1992.

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

FD92. Yair Frankel and Yvo Desmedt. Parallel reliable threshold multisignature. Technical report, Citeseer, 1992.

Fel87. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.

FGMY97a. Yair Frankel, Peter Gemmell, Philip D MacKenzie, and Moti Yung. Proactive rsa. In *Annual International Cryptology Conference*, pages 440–454. Springer, 1997.

FGMY97b. Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive rsa. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 440–454, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

GG18. Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194, 2018.

GG20. Rosario Gennaro and Steven Goldfeder. One round threshold ecdsa with identifiable abort. *IACR Cryptol. ePrint Arch.*, 2020:540, 2020.

GGN16. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156–174. Springer, 2016.

GJKR96a. Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of rsa functions. In *Annual International Cryptology Conference*, pages 157–172. Springer, 1996.

GJKR96b. Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 354–371. Springer, 1996.

GJKR99. Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 295–310. Springer, 1999.

GJKR03. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of pedersen's distributed key generation protocol. In *Cryptographers' Track at the RSA Conference*, pages 373–390. Springer, 2003.

GMR88. Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing*, 17(2):281–308, 1988.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

Gro21. Jens Groth. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.*, 2021:339, 2021.

HJJ+97. Amir Herzberg, Markus Jakobsson, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 100–110, 1997.

HJKY95. Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *annual international cryptology conference*, pages 339–352. Springer, 1995.

IOS12. Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In *Theory of Cryptography Conference*, pages 21–38. Springer, 2012.

JO08. Stanisław Jarecki and Josh Olsen. Proactive rsa with non-interactive signing. In *International Conference on Financial Cryptography and Data Security*, pages 215–230. Springer, 2008.

KG20. Chelsea Komlo and Ian Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. *IACR Cryptol. ePrint Arch*, 852:2020, 2020.

KOO95. Kaoru Kurosawa, Satoshi Obana, and Wakaha Ogata. t-cheater identifiable (k, n) threshold secret sharing schemes. In *Annual International Cryptology Conference*, pages 410–423. Springer, 1995.

Kra10. Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

Lan95. Susan K Langford. Threshold dss signatures without a trusted party. In *Annual International Cryptology Conference*, pages 397–409. Springer, 1995.

Lov. Isis Agora Lovecruft. dalek-cryptography/curve25519-dalek. `https://github.com/dalek-cryptography/curve25519-dalek`.

MOR01. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 245–254, 2001.

MS81. Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

NBBR16. Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and communication networks*, 9(17):4585–4595, 2016.

Oka88. Tatsuaki Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Comput. Syst.*, 6(4):432–441, nov 1988.

Ped91a. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

Ped91b. Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 522–526. Springer, 1991.

PK96. Choonsik Park and Kaoru Kurosawa. New eigamal type threshold digital signature scheme. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 79(1):86–93, 1996.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 387–398. Springer, 1996.

Rab98. Tal Rabin. A simplified approach to threshold and proactive rsa. In *Annual International Cryptology Conference*, pages 89–104. Springer, 1998.

RBO89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, 1989.

Sch89. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Sho00. Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.

SS01. Douglas R Stinson and Reto Strobl. Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In *Australasian Conference on Information Security and Privacy*, pages 417–434. Springer, 2001.

Too. Tool. Crate criterion. `https://docs.rs/criterion/0.3.5/criterion/`.

Appendices

A Proof of the Main Theorem

To prove the unforgeability of ICE FROST we use the simulation technique. We use the general forking lemma [BN06, lemma 1] in our proof. First, we review this lemma.

Lemma 1 (General forking lemma [BN06]). *For a given integer $q \geq 1$ and a set H of size $h \geq 2$, let A be a randomized algorithm that on input $x, h_1, \ldots, h_q$ returns a pair (J, σ), where $J \in \{0, 1, \ldots, q\}$, and σ is over an arbitrary alphabet. Let IG be a randomized algorithm called the input generator. The accepting probability of A , denoted acc, is defined as the probability that $J \geq 1$ in the following experiment*

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

The forking algorithm F_A associated to A is the randomized algorithm that takes input x proceeds as follows:

Algorithm $\mathsf{F}_A(x)$

- Pick coins ρ for A at random
- $h_1, \ldots, h_q \xleftarrow{\$} H$
- $(I, \sigma) \leftarrow \mathsf{A}(x, h_1, \ldots, h_q; \rho)$
- If $I = 0$ then return $(0, \epsilon, \epsilon)$
- $h'_I, \ldots, h'_q \xleftarrow{\$} H$
- $(I', \sigma') \leftarrow \mathsf{A}(x, h_1, ..., h_{I-1}, h'_I, ..., h'_q; \rho)$
- If $(I = I'$ and $h_I \neq h'_I)$ then return $(1, \sigma, \sigma')$ Else return $(0, \epsilon, \epsilon)$.

Let

$$frk \triangleq \Pr\Big[b = 1 : x \xleftarrow{\$} \mathsf{IG}; (b, \sigma, \sigma') \xleftarrow{\$} \mathsf{F}_A(x)\Big].$$

Then

$$frk \geq \mathsf{acc} \cdot \Big(\frac{\mathsf{acc}}{q} - \frac{1}{h}\Big)$$

A.1 Simulating KeyGen in ICE FROST

According to the described proof sketch in Section 4.2, ICE FROST is unforgeable, because otherwise there exists an algorithm C that uses the forger adversary and solves the discrete logarithm problem for an arbitrary challenge value $\omega \in \mathbb{G}$.

For this, C embeds the challenge value ω into the group public key Y by simulating honest parties and runs the simulator algorithm $\mathsf{A}(Y, \{h_1, \ldots, h_{n_r}\}, \beta)$ who invokes the forger F by simulating the responses to its random oracle queries using $\{h_1, \ldots, h_{n_r}\}$. The simulation is successful if the adversary cannot distinguish the real world from the simulated world.

Without loss of generality, suppose the forger adversary F controls $t-1$ participants P_1 to P_{t-1}. The algorithm C simulates the (honest) participants P_t to P_n. Simulation effectively is generating valid messages on be half of honest participants. However, the simulator does not know the secret key corresponding to honest participants.

To simulate the public communication between two honest participants P_h and P_j the simulator C samples a DH key key at random from $\mathbb{Z}_q$ and encrypts "0" under this key, that is, $e_{hj} = Enc_{Sym}(key, 0)$.

To simulate messages from honest participants to those participants controled by the adversary, the simulator needs k_{ha}^{Sym}, the mutual DH key between the honest participant P_h and the adversarial participant P_a. Then P_h can encrypt the secret share $f_h(a)$ under k_{ha}^{Sym} or decrypt received messages from P_a and issue a complaint message if inconsistency is detected. Note that since the simulator C doesn't know the honest participant's secret key, it can only derive the DH key using P_a's secret key sk_a. The simulator uses the generalized forking algorithm to get two valid proofs of knowledge of the secret key $\tau_a = (S_a, \nu_a)$ and $\tau'_a = (S'_a, \nu'_a)$ from the adversary and extract sk_a from the two proofs that is possible when $S_a = S'_a$. Suppose q_{h1} is the number of random oracle queries made during the Key generation protocol. According to Lemma 1, the simulator derives sk_a successfully with probability at least $\frac{q_{h1}-1}{q_{h1}}(\frac{q_{h1}-1}{q_{h1}^2} - \frac{1}{h})$, where h is the size of the random oracle outputs that is very large. Therefore, the success probability of the simulator in deriving sk_a and hence successfully proceed with the simulation is almost $\frac{1}{q_{h1}}$.

We consider the following sequence of games to model the adversary's interaction with the simulator algorithm:

Game_0**:** This is the real unforgeability experiment. The game outputs 1 if the adversary produces a forgery by corrupting less than t participants. For simplicity, we assume that participants P_1 to P_{t-1} are corrupted while $P_t, \ldots, P_n$ are honest.

Game_1**:** Exactly as Game_0 but the DH keys between an honest participant P_n and other honest participants $P_t, \ldots, P_{n-1}$, i.e., $k_{n,t}, \ldots, k_{n,n-1}$ are changed into random group elements[5].

Game_2**:** Exactly as Game_1 but the ciphertexts $e_{n,t}, \ldots, e_{n,n-1}$, containing the shares $f_n(t), \ldots, f_n(n-1)$, are computed as $e_{n,j} = Enc_{sym}(k_{n,j}^{DH}, 0)$ for $t \leq j < n$.

In Game_2 the adversary's advantage of successfully forging a signature can be bounded by the advantage of breaking the discrete logarithm in almost exactly

[5] We can pick any arbitrary honest participant instead of P_n and change its mutual DH key with other honest participants to random group elements to define Game_1. We picked P_n for the simplicity of presentation.

the same ways as it is proven for the original FROST. This follows from the fact that in Game_2 it is possible to randomly sample the $t-1$ honest-to-malicious shares, while the honest-to-honest shares are all set to 0. For completeness, in the final part of this proof we bound the probability of forging in Game_2 as in the original proof.

To conclude our final claim, we show two lemmas stating the indistinguishability of the different games.

Lemma 2. *For any forger* F *there exists an adversary* B_1 *against the DDH assumptions such that* $|\Pr[\mathsf{Game}_0(\mathsf{F}) = 1] - \Pr[\mathsf{Game}_1(\mathsf{F}) = 1]| \leq \mathsf{Adv}_{\mathsf{DH}}(\mathsf{B}_1)$*, where* $\mathsf{Adv}_{\mathsf{DH}}\mathsf{B}_1$ *is* B_1*'s advantage of distinguishing a DH key from a uniform key under the DDH assumption.*

Proof. We construct an adversary B_1 against the DDH assumption that receives triples of the form $(X, Y, Z) \in \mathbb{G}^3$ and decides if Z is the DH key or a random string. B simulates $\mathsf{Game}_0(\mathsf{F})$ as follows: it sets P_n's public key to X and samples $r_t, \ldots, r_{n-1} \leftarrow \mathbb{Z}_q$, and sets $Y^{r_t}, \ldots Y^{r_{n-1}}$ as the public keys of $P_t, \ldots, P_{n-1}$, respectively. The key between participants P_n and P_j is computed as Z^{r_j} for $t \leq j < n$. It is direct to conclude that, when Z is uniform, B_1's output follows exactly the same distribution as in Game_1. On the other hand, when $Z = Y^{\log X}$, the output of B_1 follows exactly the same distribution as in Game_0. □

Lemma 3. *For any forger* F *there exists an adversary* B_2 *such that* $|\Pr[\mathsf{Game}_1(\mathsf{F}) = 1] - \Pr[\mathsf{Game}_2(\mathsf{F}) = 1]| \leq (n-t) \cdot \mathsf{Adv}_{\mathsf{CPA}}{}^{Enc_{sym}}(\mathsf{B}_2)$*, where* $\mathsf{Adv}_{\mathsf{CPA}}{}^{Enc_{sym}}(\mathsf{B}_2)$ *is* B_2*'s CPA advantage in distinguishing the encryption of two distinct messages encrypted by* $Enc_{sym}(\cdot)$ *scheme.*

Proof. The result follows from a simple hybrid argument where at each step we change an encrypted share. At the j-th step of the hybrid argument we construct a CPA adversary which samples P_n's random polynomial itself, and uses its encryption oracle to encrypt $f_n(t), \ldots, f_n(j-1)$ and $n-j$ zeros. For the j-th share it uses its LoR oracle to encrypt either $f_n(j)$ or 0. □

Lemma 4. *Consider* n *signing participants and a threshold of* t*;* q_{h1} *and* q_{h2} *be the number of queries made to the random oracle in key generation and signing algorithms, respectively;* π *be the batch size in preprocessing protocol,* q_p *be the number of allowed preprocess queries, and* q_s *be the number of signing queries. If the discrete logarithm problem in* $\mathbb{G}$ *is* (τ', ϵ')*-hard then*

$$\epsilon' \leq \frac{\Pr[\mathsf{Game}_2(\mathsf{F}) = 1]^2}{q_{h1} \cdot \big(2(q_{h1} + q_{h2}) + (\pi+1)q_p + 1\big)}$$

Proof. Suppose the forger adversary F controls $t-1$ participants P_1 to P_{t-1}. We construct an algorithm C that starts a perfect simulation of $\mathsf{Game}_2(\mathsf{F})$ embedding discrete log challenge in the global public key. As described, the simulator needs a PoK extractor to extract the secret keys of the participants controlled by the adversary and then derive the corresponding mutual DH keys and use them for encryption and decryption of messages communicated between adverserial and

honest participants. The extractor will derive the secret key with probability $\frac{1}{q_{h1}}$ according to our argument. Then, in the first round of Key Generation $\mathbf{C}_i = \langle \phi_{i0}, \ldots, \phi_{i(t-1)} \rangle$ is the set of public commitments for participant P_i. The simulation of a participant P_n is as follows;

1. Randomly generates values $x_1, \ldots, x_{t-1}$ to serve as secret shares $f_t(1), \ldots, f_n(t-1)$, respectively.
2. Set ϕ_{n0} to be the the challenge value ω.
3. Calculates $\phi_{n1}, \ldots, \phi_{n(t-1)}$ by performing Lagrange interpolation in the exponent. That is, $\phi_{nk} = \omega^{\lambda_{n0}} . g^{\sum_{i=1}^{t-1} \lambda_{ki} . x_i}$.

Then $\mathbf{C}_t$ is broadcasted and in the second round $((n,1), Enc_{sym}(k_{t,1}^{sym}, x_1), \ldots, ((n,t-1), Enc_{sym}(k_{n,t-1}^{sym}, x_{t-1})$ is sent to the malicious participants $P_1, \ldots, P_{t-1}$. Further, C simulates the proof of knowledge for $a_{t1}, \ldots, a_{t(t-1)}$. In addition, C derives the signing public key for P_t by following the same steps they would use to calculate the public key for their peers (as the discrete log of the challenge value ω is unknown), by performing:

$$Y_t = \prod_{j=1}^{n} \prod_{k=0}^{t-1} \phi_{jk}^{t^k \ mod \ q}$$

The participants controlled by F can derive their secret key shares s_i by directly following the KeyGen protocol, then deriving $Y_i = g^{s_i}$. Each party (honest or corrupted by F) can follow the KeyGen protocol to derive the group's long-lived public key, by calculating $Y = \prod_{j=1}^{n} \phi_{j0}$.

In addition, C must obtain F's secret values $a_{10}, ..., a_{(t-1)0}$ using the extractor for the zero-knowledge proofs that F generates, while $a_{t0}, \ldots, a_{(n-1)0}$ are directly computed by the simulation. C will use these values next in order to convert the discrete logarithm for the group public key Y into the discrete logarithm for the challenge value ω.

Finding the Discrete Logarithm of the Challenge Input. As described in [KG20, Section A.1.3], using the two forgeries (σ, σ'), the discrete logarithm of ω can be derived. Then according to Forking Lemma (Lemma 1) the probability of generating two valid forgeries and therefore returning the correct discrete logarithm of ω by C is at least $\frac{\epsilon^2}{n_r}$ and therefore the total success probability is at least $\frac{\Pr[\mathsf{Game}_2(\mathsf{F})=1]^2}{n_r}$, where n_r is the total number of queries made to the random oracle during all stages of the forgery. That is $2(q_{h1} + q_{h2}) + (\pi + 1)q_p + 1$. We note that the success probability of the simulation is $\frac{1}{q_{h1}}$ and therefore, $\epsilon' \leq \frac{\Pr[\mathsf{Game}_2(\mathsf{F})=1]^2}{q_{h1}\left(2(q_{h1}+q_{h2})+(\pi+1)q_p+1\right)}$. □

Proof of Theorem 1:

Proof. According to Lemma 2 and Lemma 3,

$$\Pr[\mathsf{Game}_0(\mathsf{F}) = 1] = \Pr[\mathsf{Game}_0(\mathsf{F}) = 1] \pm \Pr[\mathsf{Game}_1(\mathsf{F}) = 1] \pm \Pr[\mathsf{Game}_2(\mathsf{F}) = 1]$$
$$\Rightarrow \Pr[\mathsf{Game}_2(\mathsf{F}) = 1] \leq \mathsf{Adv}_{\mathsf{DH}}(\mathsf{B}_1) + (n-t)\mathsf{Adv}_{\mathsf{CPA}}^{Enc_{sym}}(\mathsf{B}_2) + \Pr[\mathsf{Game}_0(\mathsf{F}) = 1]$$

Since the probability of generating a valid forgery is ϵ, we have $\Pr[\mathsf{Game}_0(\mathsf{F}) = 1] = \epsilon$. Therefore, from Lemma 4:

$$\epsilon' \leq \frac{\Pr[\mathsf{Game}_2(\mathsf{F}) = 1]^2}{q_{h1}\big(2(q_{h1}+q_{h2})+(\pi+1)q_p+1\big)} \leq \frac{\Big(\epsilon + \mathsf{Adv}_{\mathsf{DH}}(\mathsf{B}_1) + (n-t)\mathsf{Adv}_{\mathsf{CPA}}{}^{Enc_{sym}}(\mathsf{B}_2)\Big)^2}{q_{h1}\big(2(q_{h1}+q_{h2})+(\pi+1)q_p+1\big)}$$

Since in above expression the numerator is of order $O(n^2)$, we conclude that unforgeability is achieved given that $n << q_{h1}, q_{h2}$. This is usually the case in settings that we consider. □

B Raw Benchmarking Data

In the following, we present our raw benchmark data from the implementation described in Section 7.

In Table 2 and Table 3, we present benchmarks for running the key generation protocols of FROST and ICE FROST, respectively. Each table contains running times for four parts of the protocol as follows:

- *Participant creation,* consists in Round 1, steps 1 to 3 of [KG20, Figure 1(KeyGen)]. For ICE FROST, it includes NIZK proof generation for the DH key pair.
- *Round One,* consists of Round 1, step 5 and Round 2, step 1 of [KG20, Figure 1(KeyGen)]. In both original FROST and ICE FROST, it includes encryption of shares and NIZK proofs verification (one in FROST for participant secret key, two in ICE FROST for the previous and DH key pair).
- *Round Two* consists of Round 2, step 2 of [KG20, Figure 1(KeyGen)] plus the decryption step for the received shares, for both FROST and ICE FROST.
- *Finish* consists of Round 2, steps 3-4 of the original paper.

t-out-of-n	Participant creation	Round 1	Round 2	Finish
34-out-of-100	0.696	17.890	227.10	0.074
67-out-of-100	1.325	18.425	439.99	0.129
67-out-of-200	1.323	37.042	885.97	1.216
134-out-of-200	2.569	41.556	1764.30	2.446
101-out-of-300	1.970	58.188	2001.00	2.769
201-out-of-300	3.885	69.291	3980.63	6.023
167-out-of-500	3.226	110.630	5507.92	9.472
334-out-of-500	6.386	141.290	11151.00	20.561

Table 2: FROST key generation running time (in ms)

t-out-of-n	Participant creation	Round 1	Round 2	Finish
34-out-of-100	0.727	27.866	232.14	0.076
67-out-of-100	1.356	28.603	447.45	0.135
67-out-of-200	1.356	56.768	893.73	1.190
134-out-of-200	2.632	61.830	1773.30	2.373
101-out-of-300	2.005	88.804	2009.66	2.880
201-out-of-300	4.007	102.087	3986.53	5.761
167-out-of-500	3.361	160.244	5522.80	9.650
334-out-of-500	6.636	199.984	11537.22	19.978

Table 3: ICE FROST key generation running time (in ms)

t-out-of-n	Single signature generation	Signature aggregation	Signature verification
34-out-of-100	3.0559	8.304	0.0974
67-out-of-100	5.9797	17.206	0.0985
67-out-of-200	5.9541	17.081	0.0959
134-out-of-200	11.898	38.441	0.0985
101-out-of-300	8.9412	27.453	0.0979
201-out-of-300	17.828	64.057	0.0982

Table 4: FROST/ICE FROST signing protocol running time (in ms)