Large-Scale Non-Interactive Threshold Cryptosystems Through Anonymity

Andreas Erwig, Sebastian Faust, and Siavash Riahi

Technische Universität Darmstadt, Germany
`firstname.lastname@tu-darmstadt.de`

Abstract. A (t, n)-public key threshold cryptosystem allows distributing the execution of a cryptographic task among a set of n parties by splitting the secret key required for the computation into n shares. A subset of at least $t + 1$ honest parties is required to execute the task of the cryptosystem correctly, while security is guaranteed as long as at most $t < \frac{n}{2}$ parties are corrupted. Unfortunately, traditional threshold cryptosystems do not scale well, when executed at large-scale (e.g., in the Internet-environment). In such settings, a possible approach is to select a subset of n players (called a committee) out of the entire universe of $N \gg n$ parties to run the protocol. If done naively, however, this means that the adversary's corruption power does not scale with N as otherwise, the adversary would be able to corrupt the entire committee. A beautiful solution for this problem is given by Benhamouda et al. (TCC 2020) who present a novel form of secret sharing, where the efficiency of the protocol is *independent* of N, but the adversarial corruption power *scales* with N (a.k.a. fully mobile adversary). They achieve this through a novel mechanism that guarantees that parties in a committee stay anonymous until they start to interact within the protocol.
In this work, we initiate the study of large-scale threshold cryptosystems. We present novel protocols for distributed key generation, threshold encryption, and signature schemes that guarantee security in large-scale environments with complexity independent of N. One of our key contributions is to show how one can transform a scheme which is only secure against static adversaries to a large-scale threshold cryptosystem via anonymity and prove that it is secure against a fully mobile adversary. We believe that our framework and proof techniques can be used in the future to design and prove schemes secure in the large-scale setting.

1 Introduction

In a threshold cryptosystem [22, 26, 21], a secret key sk is distributed among a set of n parties, where each party holds a share sk_i of the secret key. A subset of $t + 1$ parties is needed to re-construct the secret key (or carry out the cryptographic task such as signing), while $\leq t$ parties learn nothing about the sensitive information. A threshold cryptosystem consists of two components. First, a protocol for securely generating the secret key – so-called *distributed key generation (DKG)* [46] – that enables the parties to securely generate a shared sk and the corresponding public key pk. At the end of this protocol each party holds its secret key share sk_i and is aware of the public key pk. Second, a distributed version of the cryptosystem, where the parties can use their shares to perform the cryptographic tasks at hand. Two important examples of threshold cryptosystems are threshold signatures for signing messages in a distributed fashion (e.g., [12, 42]), and threshold public key encryption for distributed decryption of ciphertexts (e.g., [48, 17, 41]).

Traditionally, threshold cryptographic schemes have been considered in a setting where the number of parties n is relatively small and the adversary is restricted to corrupt at most $t < n/2$ parties. This upper bound is required to achieve guaranteed output delivery [20], i.e., the adversary cannot stall the system even when behaving in an arbitrary malicious way. When we move to a large-scale Internet setting with a large user base (e.g., as prominently considered in the blockchain setting), several new challenges arise. In particular, we aim for a solution that is *scalable* when the number of users in the universe – denoted by N – drastically increases. On the one hand, we aim at a protocol where only a small subset of the entire population carry out the cryptographic computation (e.g., running DKG with thousands of users is practically infeasible). On the other hand, we want that the adversary can corrupt a fraction of all parties N, such that its corruption power is not bounded by a small value t.

To address these two challenges, the recent work of Benhamouda et al. [9] introduces the concept of *evolving-committee proactive secret sharing* (ECPSS). In a nutshell, to achieve an efficient solution, ECPSS considers a committee of n parties (where n is independent of the number N of parties in the universe) that hold a shared secret. In addition, to ensure that the corruption power of the adversary is linear in N, Benhamouda et al. combine two ideas, namely (1) using dynamic proactive secret sharing and (2) anonymizing the identity of the secret shareholders in this protocol. Let us provide some more details on the solution of [9]. Dynamic proactive secret sharing is a secret sharing protocol that proceeds in epochs, where the adversary is allowed to corrupt at most t parties per epoch. Such an adversary is often also referred to as a *mobile adversary* [45]. To ensure security in this setting dynamic proactive secret sharing schemes deploy a so-called *handover* protocol, where the shared secret is re-shared to a new committee at the end of each epoch. While a mobile adversary can corrupt over time $\gg N$ users, the naive application of dynamic proactive secret sharing only tolerates $t < n/2$ corruptions. To circumvent this, inspired by recent advances in blockchain consensus, Benhamouda et al. introduce a novel concept of anonymity. More precisely, after a party in a committee is activated and communicates (e.g., for reconstructing the shared secret), a new committee is selected in such a way that the members of the new committee stay anonymous. This feature guarantees that an adversary cannot target the members of the small-sized committee, even for a corruption power of $\gg n$ corruptions per epoch.

The original work of Benhamouda et al. [9] considers only the question of how to store a secret in a large scale environment. This work has recently been extended in the so-called YOSO model of computation ("You only speak once") [30], which presents a general framework for committee-style secure computation with anonymity. The result of [30] is however mainly a feasibility result (we will discuss it – and in particular its limitations – in more detail in Sec. 1.2), and relies on techniques from general purpose secure multiparty computation. In this work we initiate the study of scalable threshold cryptography for large scale environments, where the adversary's corruption power per epoch can grow with N. We discuss our main contributions in more detail below.

1.1 Our Contribution

The goal of this work is to study threshold public key cryptosystems that are executed by a small set of n parties among a large universe of N parties. We call such schemes *large-scale threshold public key cryptosystems* and require that these schemes must be secure in presence of an adversary whose corruption power is proportional to N, and in particular $\gg n$. We call such an adversary *fully mobile adversary.* Traditionally, threshold cryptography considered three different corruption models, namely (1) static, (2) adaptive and (3) mobile adversaries, where a static adversary must choose the set of corrupted parties at the beginning of a protocol execution while an adaptive adversary is allowed to corrupt parties at any time. Finally, a mobile adversary has been considered in the proactive setting, where a protocol proceeds in epochs and the adversary is not only allowed to corrupt parties during an epoch but to also "uncorrupt" parties at the end of an epoch. All of these adversarial models are restricted in that the corruption power of the adversary is a fraction (typically $< 1/2$) of the size of protocol participants n. In contrast, in this work we consider *fully mobile adversaries*, whose corruption power is proportional to the universe size N thereby allowing to corrupt $> n$ parties.

In our work, we first formalize the concept of discrete-log-based large-scale distributed key generation schemes and show a concrete instantiation. We follow the idea of Benhamouda et al. [9] to achieve security against a fully mobile adversary through anonymization. This, however, complicates the construction and security proof as we have to ensure that parties stay anonymous as long as they are involved in the protocol execution. The main challenge arises from the fact that distributed key generation protocols are typically highly interactive which poses a problem in our full security setting since parties can at most speak once in order to preserve their anonymity. Surprisingly, we cannot use the ECPSS construction from Benhamouda et al. entirely in black-box for our instantiation, as we require anonymous parties to broadcast DKG-specific values which are not needed (and thus not supported) in the ECPSS construction itself. In a bit more detail, since parties can at most speak once, DKG-specific values have to be broadcast during a state handover from one anonymous committee to another. To tackle this issue we provide a *generalized* handover procedure which allows parties to broadcast auxiliary information while handing over their internal state. We believe that our formalization of large-scale DKG protocols and the corresponding discrete-log-based protocol can pave the way for designing DKG protocols under different assumptions such as the RSA assumption which has been mentioned as an open problem in [30].

We next consider the setting of large-scale non-interactive threshold public key encryption and signature schemes by first providing formal definitions of such primitives and then showing concrete instantiations. To this end, we show how the statically-secure non-interactive threshold public key encryption scheme from Shoup and Gennaro [48] can be transformed to the large-scale setting with security against fully mobile adversaries. Similarly, in Appendix E we show how the statically-secure threshold signature scheme from Boldyreva [12] can be transformed to the large-scale setting. The main challenge for both of these transformations is to prove security against a powerful fully mobile adversary. In general, the issue when proving security of threshold schemes with adaptive or mobile security is that one must exhibit a simulator that simulates the view of the adversary without knowing the secret key of the scheme. That is, the simulator does not know the secret key shares of some honest parties and thereby, upon corruption of these parties, cannot provide an internal state that is consistent with previous information that the adversary has seen.

We circumvent this issue by designing our protocols carefully in such a way that allows us to construct a simulator whose answers to the adversary's corruption queries are consistent with the view of the adversary. At a high level, we achieve this by postponing the publication of secret key share dependent values to the end of an epoch while still guaranteeing correctness and security of our scheme. As such, the simulator can maintain a set of secret key shares which *look* consistent from the adversary's view of the protocol execution, as long as the adversary does not corrupt more than t secret key shareholders per epoch. We ensure this corruption upper bound by leveraging the idea of Benhamouda et al. [9] of keeping the identities of secret key shareholders anonymous until the end of an epoch.

As a next step, we argue that the two transformations of statically-secure threshold public key encryption and signature schemes to the large-scale setting can be generalized to any discrete-log-based threshold encryption/signature scheme which satisfies certain properties.

Finally, we provide various applications of our schemes in the blockchain setting, including the fair exchange of secret values and the checkpointing of individual blocks in a blockchain to decrease computational effort for new parties in the network. The latter further has applications in blockchain interoperability, where parties in a blockchain network have to prove to parties outside of the network that a specific block is indeed included in the blockchain.

1.2 Related Work

Anonymity and You Only Speak Once Paradigm. The most relevant previous work for us is by Benhamouda et al. [9] who introduce the notion of evolving-committee proactive secret sharing (ECPSS) which extends previous secret sharing notions in the following ways: (1) an ECPSS scheme includes a procedure to select the committee of secret shareholders, and (2) an ECPSS scheme does not *assume* that an adversary corrupts at most a minority of parties in a committee but rather provides a mechanism for *provably* achieving this. Benhamouda et al. present an instantiation of an ECPSS scheme which they prove secure against a fully mobile adversary by keeping the identities of the secret shareholders *anonymous* among a large set of parties. That is, they prove that if the adversary corrupts at most 25% of all parties in the universe, their ECPSS scheme remains secure. A recent work by Gentry et al. [29] improves Benhamouda et al.'s solution by allowing for a more powerful adversary that can corrupt up to less than 50% of all parties. We recall the definition of ECPSS schemes in Sec. 2.6 and we provide a more detailed description of the ECPSS scheme of Benhamouda et al. in Sec. 3.

Another recent work of Gentry et al. [30] generalizes the concept of computing on secrets among anonymous parties by introducing the *you only speak once* (YOSO) model and showing how to realize information theoretical and computational secure multi-party computation in this model. However, this work is mainly a feasibility result and it relies on certain idealized functionalities that currently cannot be instantiated. In a similar spirit, a recent work by Choudhuri et al. [19] presents general-purpose multi-party computation in the so-called fluid model, where parties can dynamically join and leave the protocol execution. However, the authors of [19] do not analyze their solution w.r.t. a fully mobile adversary who has sufficient corruption power to potentially corrupt a majority of the universe's participants. Finally, Campanelli et al. [16] analyze the notion of *encryption to the future* which generally allows parties to send messages to an anonymous and yet to be selected committee.

Threshold Cryptosystems. There has been extensive work in the field of threshold cryptosystems. Distributed key generation (DKG) protocols have been studied in the past mostly in the static corruption setting (e.g., [46, 28, 38, 18]). Recently, Gurkan et al. [33] presented a DKG protocol with aggregatable and publicly-verifiable transcripts based on a gossip network which reduces communication complexity and verifcation time but is secure only against static adversaries. Likewise, Abraham et al. [2] recently presented an asynchronous DKG protocol and Shrestha et al. [49] presented a synchronous DKG protocol that does not require broadcasts. Both these works are in the static security setting. Abe and Fehr [1] and Canetti et al. [17] proposed DKG protocols which are secure against adaptive adversaries. Most related to our work is the recent work by Groth [32] which introduces a non-interactive distributed key generation protocol, which is secure against a mobile adversary who corrupts at most a minority of parties at a time. However, this work does not consider the fully mobile setting that we consider in our work.

Threshold public key cryptosystems have been extensively studied with security against static adversaries (e.g., [15, 13, 50, 48, 12]) and adaptive adversaries (e.g., [27, 17, 37, 40, 41, 42, 23]). Herzberg et al. [36] proposed a solution how to generically proactivize discrete-log-based public key threshold cryptosystems. However, their generic construction is only secure in the static proactive setting, i.e., the adversary has to decide which parties to corrupt at the beginning of each epoch. Finally, there have been works in the adaptive proactive adversarial setting (e.g., [25, 17, 3]) which is the setting that is most similar to the setting we consider in this work. However, all of the above mentioned works focus on an adversary (static, adaptive or mobile) that is restricted to only corrupt at most a minority of the participants in the universe, whereas we consider a fully mobile adversary that has sufficient corruption power to corrupt a large fraction of all parties.

We provide discussion on additional related work in Appendix A.

2 Preliminaries

In this section, we provide required notation and discussion on our communication and adversarial model as well as building blocks that we require for our work.

2.1 Notation

We use the notation $s \leftarrow_\$ H$ to denote that a variable s is sampled uniformly at random from a set H. For an integer i, we use $[i]$ to denote the set $\{1, \cdots, i\}$. For a probabilistic algorithm A, we use $s \leftarrow_\$ A(x)$ to denote that s is the output of an execution of A on input x. For a deterministic algorithm B, we use $s \leftarrow B(x, r)$ to denote that s is the output of an execution of B on input x and randomness r. We use the notation $s \in A(x)$ to denote that s is in the set of possible outputs of A on input x.

For a set of parties C and a protocol Π, we use the notation $\Pi[C_{\langle x_1, \cdots, x_{|C|} \rangle}]$ to denote that Π is jointly executed by all parties $P_i \in C$ with respective secret inputs x_i for $i \in [|C|]$. Furthermore, we use the notation $\Pi[C_{\langle x_1, \cdots, x_{|C|} \rangle}](y)$ if all $P_i \in C$ receive a common public input y. Finally, for a set of parties U s.t. $C \subset U$ and a protocol Π' we use the notation $\Pi'[C_{\langle x_1, \cdots, x_{|C|} \rangle}, U](y)$ to denote the joint execution of Π' by all parties in U with common public input y where party $P_i \in C$ has secret input x_i with $i \in |C|$.

2.2 Communication and Adversarial Model

Our communication and adversarial model follows the model of Benhamouda et al. [9]. We assume that parties have access to an authenticated broadcast channel and a public key infrastructure (PKI). Furthermore, we consider a synchronous communication model where messages broadcasted in some round i are received by all other parties in round $i+\delta$ where δ is a fixed upper bound. A blockchain network satisfies this communication model.

The authenticated broadcast channel is the only means of communication in our model. In particular, we do not consider sender-anonymous channels which are inherently difficult to construct in practice and significantly simplify the problem of keeping the identity of parties anonymous.

We further assume that communication between parties during the lifetime of the system can be divided into *epochs*. At the beginning of each epoch, all parties broadcast a new public key via the PKI.

We consider a fully mobile adversary who can monitor the broadcast channel and corrupt parties at any point in time. Corrupted parties are controlled by the adversary and can deviate arbitrarily from the protocol execution. A fully mobile adversary can corrupt a certain fraction p of *all* parties in the system at any point in time. The fraction p is called the adversary's *corruption power*. The adversary can further decide to "uncorrupt" a corrupted party, i.e., once an uncorrupted party broadcasts a new public key to the PKI it is no longer controlled by the adversary.

We also assume that parties can erase their internal states such that upon their corruption, the adversary would be oblivious to the secret values that a party had previously stored and erased. Note that this is an inherent requirement in all proactive protocols.

2.3 Public Key Encryption

Throughout this work, we use different notions of public key encryption (PKE) which we briefly recall here. We first give the definition of a PKE scheme and then provide the security notion of secrecy under selective opening attacks. We then recall the definition of anonymous PKE before providing the definition of a non-interactive threshold PKE scheme.

Definition 1. *A public key encryption scheme* PKE *consists of a tuple* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *of efficient algorithms which are defined as follows:*

$\mathsf{KeyGen}(1^\lambda)$*: This probabilistic algorithm takes as input a security parameter* λ *and outputs a public key* pk *and a secret key* sk*.*

$\mathsf{Enc}(pk, m)$*: This probabilistic algorithm takes as input a public key* pk *and a message* m *and outputs a ciphertext* ct*.*

$\mathsf{Dec}(sk, ct)$*: This algorithm takes as input a secret key* sk *and a ciphertext* ct *and it outputs either* $\perp$ *or a message* m*.*

Where necessary, we will explicitly mention the random coins used during the encryption procedure as $\mathsf{Enc}(pk, m; r)$ where r is sampled from the randomness space $\mathcal{R}$ used in the encryption procedure. Note that when using this notation the encryption algorithm itself is deterministic.

Furthermore, we will later in this paper use the assumption that given a secret key sk generated by the KeyGen algorithm, it is possible to derive the corresponding public key pk via a deterministic function SkToPk.

Secrecy under selective opening attacks (RIND-SO). We now recall the indistinguishability-based notion of receiver selective opening security (RIND-SO) from Hazay et al. [35] for public key encryption schemes, which in turn is based on [24] and [8]. The RIND-SO notion defines a security game between a challenger and an adversary in which the challenger first samples a set of key pairs and then sends the public keys to the adversary. The adversary then chooses a distribution $\mathcal{D}$ and receives a vector of ciphertexts, which encrypt messages that are sampled from $\mathcal{D}$. The adversary can then choose some of the ciphertexts and receives the corresponding secret keys which can be used to decrypt them. Finally, for the remaining ciphertexts, the adversary either receives the correct plaintext or randomly sampled messages from $\mathcal{D}$, conditioned on the opened plaintext[1].

[1] Note that $\mathcal{D}$ must be efficiently *resamplable*, namely it should be possible to draw new elements from $\mathcal{D}$ conditioned on the opened plaintexts.

Definition 2 (Efficiently Resamplable Distribution). *Let $k, n > 0$. A distribution $\mathcal{D}$ over $(\{0,1\}^k)^n$ is efficiently resamplable if there is a PPT algorithm $\mathsf{Resamp}_{\mathcal{D}}$ such that for any $\mathcal{I} \subset [n]$ and any partial vector $\mathbf{m}'_{\mathcal{I}}$ consisting of $|\mathcal{I}|$ k-bit strings, $\mathsf{Resamp}_{\mathcal{D}}(\mathbf{m}'_{\mathcal{I}})$ returns a vector $\mathbf{m}$ sampled from $D|_{\mathbf{m}'_{\mathcal{I}}}$ i.e., $\mathbf{m}$ is sampled from $\mathcal{D}$ conditioned on $\mathbf{m}_{\mathcal{I}} = \mathbf{m}'_{\mathcal{I}}$.*

Definition 3 (RIND-SO Security). *For a PKE scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, security parameter $\lambda \in \mathbb{N}$, and a stateful PPT adversary $\mathcal{A}$, the RIND-SO game $\mathsf{Rind\text{-}SO}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda)$ is defined as follows:*

1. $(\mathbf{sk}, \mathbf{pk}) \coloneqq (sk_i, pk_i)_{i\in[n]} \leftarrow (\mathsf{Gen}(1^\lambda))_{i\in[n]}$
2. $(\mathcal{D}, \mathsf{Resamp}_{\mathcal{D}}, state_1) \leftarrow \mathcal{A}(\mathbf{pk})$
3. $\mathbf{m} \coloneqq (m_i)_{i\in[n]} \leftarrow \mathcal{D}$
4. $\mathbf{c} \coloneqq (c_i)_{i\in[n]} \leftarrow (\mathsf{Enc}(pk_i, m_i; \$))_{i\in[n]}$
5. $(\mathcal{I}, state_2) \leftarrow \mathcal{A}(\mathbf{c}, state_1)$
6. $\mathbf{m}' \leftarrow \mathsf{Resamp}_{\mathcal{D}}(\mathbf{m}_{\mathcal{I}})$
7. $b \leftarrow \{0,1\}, \quad \mathbf{m}^* \leftarrow \begin{cases} \mathbf{m}' & \text{if } b = 0 \\ \mathbf{m} & \text{if } b = 1 \end{cases}$
8. $b' \leftarrow \mathcal{A}(\mathbf{sk}_{\mathcal{I}}, \mathbf{m}^*, state_2)$

The advantage of the adversary $\mathcal{A}$ is defined as $2 \cdot |\Pr[b = b'] - \frac{1}{2}|$. A PKE scheme is RIND-SO secure, if every PPT $\mathcal{A}$ only has negligible advantage (in λ) in winning the above game.

We note that the RIND-SO definition given here is "semi-adaptive", i.e., the adversary decides in *one shot* which secret keys are revealed. However, the ECPSS construction by Benhamouda et al. [9] (and consequently our constructions) relies on public key encryption schemes that are secure against a "fully-adaptive" adversary who can choose the keys that are revealed adaptively [11]. Throughout this paper, we use this adaptive RIND-SO setting.

Anonymous PKE We now briefly recall the definition of an anonymous PKE scheme as introduced by Bellare et al. [7].

Definition 4. *A public key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is anonymous if for every PPT adversary $\mathcal{A}$ there exists a negligible function ν in the security parameter λ such that $\Pr[\mathsf{Anon}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda) = 1] \leq 1/2 + \nu(\lambda)$ where the game $\mathsf{Anon}^{\mathcal{A}}_{\Sigma_{\mathsf{APKE}}}(\lambda)$ is defined as follows:*

1. *The game executes the key generation procedure twice to obtain key pairs $(pk_i, sk_i) \leftarrow_\$ \mathsf{KeyGen}(1^\lambda)$ for $i \in \{0,1\}$ and forwards pk_0, pk_1 to the adversary.*
2. *The game receives a message m from the adversary.*
3. *The game chooses at random a bit $b \leftarrow_\$ \{0,1\}$ and executes $ct_b \leftarrow \mathsf{Enc}(pk_b, m)$. The game sends ct_b to the adversary.*
4. *The adversary outputs a bit b' and wins the game if $b' = b$.*

We define the advantage of the adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Anon},\mathsf{PKE}}(\lambda) = 2 \cdot \Pr[\mathsf{Anon}^{\mathcal{A}}_{\Sigma_{\mathsf{APKE}}}(\lambda) = 1)] - \frac{1}{2}.$$

Threshold Public Key Encryption

Definition 5. *A non-interactive (t, n)-threshold public key encryption scheme TPKE consists of a tuple of efficient algorithms and protocols $\mathsf{TPKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{TEnc}, \mathsf{TDec}, \mathsf{TShareVrfy}, \mathsf{TCombine})$ which are defined as follows:*

$\mathsf{Setup}(1^\lambda)$*: This probabilistic algorithm takes a security parameter $\lambda \in \mathbb{N}$ as input and output public parameters pp.*

$\mathsf{KeyGen}(pp, t, n)$*: This probabilistic algorithm takes as input public parameters* pp *and two integers* $t, n \in \mathbb{N}$*. It outputs a public key* pk*, a set of verification keys* $\{vk_i\}_{i\in[n]}$ *and a set of secret key shares* $\{sk_i\}_{i\in[n]}$ *.*

$\mathsf{TEnc}(pk, m, L)$*: This probabilistic algorithm takes a public key* pk*, a message* m *and a label* L *as input and outputs a ciphertext* ct*.*

$\mathsf{TDec}(sk_i, ct, L)$*: This algorithm takes as input a secret key share* sk_i*, a ciphertext* ct *and a label* L *and it outputs either* $\perp$ *or a decryption share* ct_i *of the ciphertext* ct*.*

$\mathsf{TShareVrfy}(ct, vk_i, ct_i)$*: This deterministic algorithm takes as input a ciphertext* ct*, a verification key* vk_i *and a decryption share* ct_i *and it outputs either* 1 *or* 0*. If the output is* 1*,* ct_i *is called a valid decryption share.*

$\mathsf{TCombine}(T, ct)$*: This deterministic algorithm takes as input a set of valid decryption shares* T *such that* $|T| = t$ *and a ciphertext* ct *and it outputs a message* m*.*

Correctness A $(t, n) - \mathsf{TPKE}$ scheme must fulfill the following two requirements.
Let $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(pk, \{vk_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow_\$ \mathsf{KeyGen}(pp, t, n)$.

1. For any message m, any label L and any ciphertext $ct \leftarrow_\$ \mathsf{TEnc}(pk, m, L)$, it must hold that
$$\mathsf{TShareVrfy}(ct, vk_i, \mathsf{TDec}(sk_i, ct, L)) = 1.$$
2. For any message m, any label L, any ciphertext $ct \leftarrow_\$ \mathsf{TEnc}(pk, m, L)$ and any set $T = \{ct_1, \cdots, ct_t\}$ of valid decryption shares $ct_i \leftarrow \mathsf{TDec}(sk_i, ct, L)$ with sk_i being t distinct secret key shares, it must hold that $\mathsf{TCombine}(T, ct) = m$.

CCA-Security We recall the definition of chosen-ciphertext security for a $(t, n) - \mathsf{TPKE}$ scheme with static corruptions. Consider a PPT adversary $\mathcal{A}$ playing in the following game $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$:

1. The adversary outputs a set $B \subset \{1, \cdots, n\}$ with $|B| = t$ to indicate its corruption choice. Let $H := \{1, \cdots, n\} \setminus B$.
2. The game executes
$$pp \leftarrow \mathsf{Setup}(1^\lambda)$$
and
$$(pk, \{vk_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(pp, t, n).$$
It sends pp, pk and $\{vk_i\}_{i\in[n]}$ as well as $\{sk_j\}_{j\in B}$ to the adversary.
3. The adversary $\mathcal{A}$ is allowed to adaptively query a decryption oracle, i.e., on input (ct, L, i) with $ct \in \{0,1\}^*, L \in \{0,1\}^*$ and $i \in H$, the decryption oracle outputs $\mathsf{TDec}(sk_i, ct_1, L)$.
4. Eventually, $\mathcal{A}$ chooses two messages m_0, m_1 with $|m_0| = |m_1|$ and a label L and sends them to the game. The game chooses a random bit $b \leftarrow_\$ \{0,1\}$ and sends $ct^* \leftarrow_\$ \mathsf{TEnc}(pk, m_b, L)$ to $\mathcal{A}$.
5. $\mathcal{A}$ is allowed to make decryption queries with the exception that it cannot make a query on ciphertext ct^*.
6. Eventually, $\mathcal{A}$ outputs a bit b'. The game outputs 1 if $b' = b$ and 0 otherwise.

Definition 6. *A non-interactive* (t, n)*-threshold public key encryption scheme* TPKE *is secure against chosen-ciphertext attacks with static corruptions if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* ν *in the security parameter* λ*, such that* $\Pr[\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}(\lambda) = 1] \leq 1/2 + \nu(\lambda)$*. We define the advantage of* $\mathcal{A}$ *in game* $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ *as* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{TPKE{-}CCA_{TPKE}}} = |\Pr[\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}} = 1] - 1/2|$*.*

Definition 7 (Decryption Consistency). *A* TPKE *scheme satisfies decryption consistency if for all PPT adversaries* $\mathcal{A}$ *it must hold that:*

$$\Pr\left[\begin{array}{c|l} \forall i \in [t+1] : \mathsf{TShareVrfy}(ct^*, vk_i, ct_i) = 1 & \mathsf{K} := (pk, \{vk_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(pp, t, n) \\ \wedge \mathsf{TShareVrfy}(ct^*, vk_i, \tilde{ct_i}) = 1 & (ct^*, T, \tilde{T}) \leftarrow \mathcal{A}(\mathsf{K})\ s.t., \\ \wedge \mathsf{TCombine}(T, ct^*) \neq \mathsf{TCombine}(\tilde{T}, ct^*) & T := \{ct_1, \cdots, ct_{t+1}\} \\ & \tilde{T} := \{\tilde{ct}_1, \cdots, \tilde{ct}_{t+1}\} \end{array}\right] \leq \nu(\lambda).$$

where ν *is a negligible function in the security parameter* λ*.*

2.4 Non-Interactive Zero-Knowledge

We now recall the definition of a non-interactive zero-knowledge (NIZK) proof of knowledge which has first been introduced in [10].

Definition 8. *A NIZK proof of knowledge for a language L with a polynomial-time recognizable binary relation R is given by the following tuple of PPT algorithms $\mathsf{NIZK} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, where (i) $\mathsf{Setup}(1^\lambda)$ outputs a common reference string crs; (ii) $\mathsf{Prove}(\mathsf{crs}, (Y, y))$ outputs a proof π for $(Y, y) \in \mathsf{R}$; (iii) $\mathsf{Verify}(\mathsf{crs}, Y, \pi)$ outputs a bit $b \in \{0, 1\}$. Further, the NIZK proof of knowledge w.r.t. R should satisfy the following properties:*

(i) Completeness*: For all $(Y, y) \in \mathsf{R}$ and $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, it holds that $\mathsf{Verify}(\mathsf{crs}, Y, \mathsf{Prove}(\mathsf{crs}, (Y, y))) = 1$ except with negligible probability;*

(ii) Soundness*: For any $(Y, y) \notin \mathsf{R}$ and $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, it holds that $\mathsf{Verify}(\mathsf{crs}, Y, \mathsf{Prove}(\mathsf{crs}, (Y, y))) = 0$ except with negligible probability;*

(iii) Zero knowledge*: For any PPT adversary $\mathcal{A}$, there exist PPT algorithms Setup' and S, where $\mathsf{Setup}'(1^\lambda)$ on input the security parameter, outputs a pair $(\widetilde{\mathsf{crs}}, \tau)$ with τ being a trapdoor and $\mathsf{S}(\widetilde{\mathsf{crs}}, \tau, Y)$ which on input $\widetilde{\mathsf{crs}}, \tau$ and a statement Y, outputs a simulated proof $\tilde{\pi}$ for any $(Y, y) \in R$. It must hold that (1) the distributions $\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)\}$ and $\{\widetilde{\mathsf{crs}} : (\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{Setup}'(1^\lambda)\}$ are indistinguishable to $\mathcal{A}$ except with negligible probability; (2) for any $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{Setup}'(1^\lambda)$ and any $(Y, y) \in \mathsf{R}$, the distributions $\{\pi : \pi \leftarrow \mathsf{Prove}(\widetilde{\mathsf{crs}}, Y, y)\}$ and $\{\tilde{\pi} : \tilde{\pi} \leftarrow \mathsf{S}(\widetilde{\mathsf{crs}}, \tau, Y)\}$ are indistinguishable to $\mathcal{A}$ except with negligible probability.*

For simplicity, we use throughout our paper a single NIZK proof system for multiple languages. We emphasize that we do so only to improve readability. Naturally, this more general NIZK proof system can be replaced by concrete NIZK proof systems for each language, thereby improving efficiency.

2.5 Secret Sharing

We briefly recall the notions of (robust) secret sharing and evolving-committee proactive secret sharing. For completeness, we provide descriptions of the notions of proactive secret sharing and dynamic proactive secret sharing in Appendix A.

Secret Sharing A (t, n)-secret sharing scheme consists of sharing and reconstruction procedures, where the sharing procedure allows a dealer to share a secret s to a committee of n parties and the reconstruction procedure allows a subset of this committee of size $\geq t$ to reconstruct the original secret s. A (t, n)-secret sharing scheme must fulfill the following two properties against an efficient adversary $\mathcal{A}$ that corrupts at most $t - 1$ parties.

1. Secrecy: $\mathcal{A}$ samples two secrets s_0 and s_1, one of which is shared by an honest dealer to a committee of n parties. $\mathcal{A}$ must be able to distinguish which secret was shared at most with negligible probability.
2. Reconstruction: Any set of honest secret shareholder of size $\geq t$ can reconstruct the original secret s.

Shamir's secret sharing [47] is the most prominent (t, n)-secret sharing scheme and we will recall it here briefly. Let q be a prime and let $1 \leq t \leq n < q$. The dealer chooses a secret $s \in \mathbb{Z}_q$ and a random polynomial $F(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ where $a_0 = s$. For $1 \leq i \leq n$, the dealer computes $s_i = F(i)$ and sends s_i to secret shareholder P_i. A set S of honest shareholders with $|S| \geq t$ can reconstruct s via interpolation. More concretely, for any $i \in \mathbb{Z}_q$ and any $j \in S$ there exist lagrange coefficients $l_{i,j}$ such that $F(i) = \sum_{j \in S} l_{i,j} s_j$.

Robust Secret Sharing. Robust secret sharing extends the reconstruction property of secret sharing schemes in the sense that a secret s can be successfully reconstructed from any set of secret shares as long as the set includes at least t correct shares.

2.6 Evolving-Committee Proactive Secret Sharing

Recently, Benhamouda et al. [9] introduced the notion of evolving-committee proactive secret sharing (ECPSS), which is defined w.r.t. a universe of N parties and parameters $t \leq n < N$. Similar to dynamic proactive secret sharing (see Appendix A), ECPSS allows to share a secret to a committee of parties and to periodically exchange the secret shareholders of the committee. ECPSS further extends previous secret sharing notions by providing a procedure that selects a size n committee from all N parties and by proving that a fully mobile adversary with corruption power p s.t. $p \cdot N > t - 1$ can at most corrupt $t - 1$ shareholders at a time.

We now recall the definition of an ECPSS scheme as given in [9].

Definition 9 (ECPSS). *An evolving-committee proactive secret sharing scheme with parameters $t \leq n < N$ consists of the following procedures:*

Setup: *Optional procedure that provides the initial state for a universe of N parties.*
Sharing: *Shares a secret s among an initial committee of size n.*
Committee-Selection: *This procedure is executed among all N parties and selects the next n-party committee.*
Handover: *This procedure is executed among n parties, takes the output of committee-selection and the current shares and re-shares them among the next committee.*
Reconstruction: *Takes t or more shares from the current committee and reconstructs the secret s or outputs $\perp$ on failure.*

An ECPSS protocol is scalable if the messages sent during committee-selection and handover are bounded in total by some fixed poly(n, λ), independent of N.

An ECPSS scheme must fulfill the same secrecy and (robust) reconstruction properties as a (robust) secret sharing scheme.

We call an ECPSS scheme $(\lambda, n, t-1, p)$-secure, if it satisfies the secrecy and reconstruction property w.r.t. a security parameter λ, committee size n, upper bound $t-1$ of corrupted parties in the committee and adversarial corruption power p.

3 ECPSS Construction from Benhamouda et al. [9]

In this section, we recall the scalable ECPSS scheme with security against a fully mobile adversary as presented by Benhamouda et al. [9] as it constitutes an important building block of our work. We denote the scheme by Σ_{ECPSS}. The main idea behind the scheme is to achieve scalability by choosing small committees of secret shareholders whose size n does not depend on the total set of parties N. However, due to the small committee size, a fully mobile adversary, whose corruption power is proportional to N instead of n, is able to corrupt all members of the committee, thereby compromising security. Benhamouda et al.'s idea to tackle this issue is to keep the identity of the committee members hidden from the adversary, i.e., the committee members should be anonymous until they have to communicate for the first time.

We now provide an overview of the Σ_{ECPSS} scheme to show how anonymity is achieved. The scheme proceeds in epochs, at the beginning of which all parties in the system generate a key pair for an anonymous public key encryption scheme. This key pair is denoted to as the *long-term keys* of a party. All parties broadcast their long-term public key to the PKI.

In each epoch two committees are selected, a nominating and a holding committee. The latter is responsible for maintaining the secret shares of the original secret while the former is responsible for selecting the members of the holding committee. The nominating committee self-selects, for instance by the use of verifiable random functions. This self-selection process ensures that members of the nominating committee remain anonymous until they send a message via the broadcast channel for the first time in the current epoch. After self-selecting, each member of the nominating committee randomly selects a member of the holding committee, generates a fresh session key pair (also referred to as ephemeral key) and encrypts the ephemeral secret key under the long-term public key of the selected holding committee member. The resulting ciphertext is then broadcast along with the ephemeral public key.

Upon broadcasting these values the members of the nominating committee erase their internal state as their identity is now known to the adversary. All N parties can now check if they were selected to the next holding committee by trying to decrypt the published ciphertexts. At this point, the previous-epoch holding committee (which holds the shares of the secret) encrypts (a sharing of) the secret shares under the ephemeral public keys and broadcasts the resulting ciphertexts. Again, as broadcasting compromises anonymity, the parties in the previous holding committee first erase their internal states. We refer the reader to [9] for the full description of the Σ_{ECPSS} scheme.

We denote the general idea of selecting an anonymous committee (holding committee) through another committee (nominating committee) as the *two committee framework* and we will later use this framework to build schemes that are secure against fully mobile adversaries.

We will now describe the Setup procedure as well as the Select and Handover procedures in more detail as these are most relevant for our work.

The Setup procedure works as follows.

$\mathsf{Setup}(1^\lambda)$**:** On input a security parameter λ, this procedure chooses a λ-bit prime q and executes $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ to generate the common reference string crs of a NIZK proof system (cf. Def. 8). The procedure outputs public parameters $pp := (\mathsf{crs}, q)$.[2]

During the Select procedure, a nominating committee first self-selects and then in turn selects the next n-party holding committee. We omit the details on the self-selection of the nominating committee and just describe the selection of the holding committee. In the following, we denote by APKE the anonymous public key encryption scheme and by PKE the public key encryption scheme that generates the ephemeral keys.

Select procedure:

Let C_{nom} be the self selected committee (nominating committee) which selects the next n-party holding committee. Each party $P_i \in C_{\mathsf{nom}}$ does the following:

1. Choose a nominee for the next holding committee $p \in [N]$ and let pk_p be this selected party's long-term public key which was broadcast to the PKI at the beginning of the current epoch.
2. Generate a new ephemeral key pair $(\mathsf{esk}_i, \mathsf{epk}_i) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, and compute $c_i \leftarrow \mathsf{APKE.Enc}(pk_p, \mathsf{esk}_i)$.
3. Erase esk_i, and broadcast (epk_i, c_i).

Upon receiving pairs $((\mathsf{epk}_1, c_1), \cdots, (\mathsf{epk}_n, c_n))$, all parties P_j with $j \in [N]$ do the following:

4. Verify that the broadcasters were indeed in the self-selected committee C_{nom}. Otherwise, ignore the tuple sent by a party not in the committee C_{nom}.
5. For each tuple (epk_i, c_i) try to decrypt c_i using your long-term secret key sk_j. If successful, P_j is in the next holding committee and stores the decrypted value esk_i.

Due to the self-selection of the nominating committee and the use of the anonymous public key encryption scheme APKE, the members of the holding committee are anonymous from the adversary's point of view (except for members already corrupted before being selected). Indeed, Benhamouda et al. show that for specific holding committee sizes and adversarial corruption power, the adversary cannot corrupt more than $t \approx n/2$ members of the holding committee except with negligible probability in the security parameter. We note that there is no guarantee that the Select procedure will indeed select a holding committee of size n, since malicious parties in the nominating committee can refuse to nominate a party to the holding committee. However, this does not affect the functionality or security of the Σ'_{ECPSS} scheme, since a malicious party in the nominating committee can always nominate a malicious party to the holding committee. Therefore, refusing to nominate any party to the holding committee decreases the number of total parties n but likewise, the number of possibly corrupted parties t. For simplicity, we assume for the rest of this work that Select chooses holding committees of size exactly n.

[2] For simplicity, we omit the setup of the PKI here.

We now describe the Handover procedure. This procedure enables the current holding committee to "pass" its shares of a secret to the next-epoch holding committee (selected via the Select procedure). In order to do so, each member of the current holding committee re-shares their secret share and sends the resulting shares to the new holding committee by encrypting them under the ephemeral public keys broadcast at the end of the Select procedure.

Handover procedure:

Let C be a holding committee such that each party $P_i \in C$ for $i \in [n]$ knows a secret share $s_i \in \mathbb{Z}_q$. In this procedure, the entire universe of parties U first runs a self-selection process to select a nominating committee C_{nom} and then executes Select to select the next-epoch holding committee C' such that each $P_j{}' \in C'$ is associated with an ephemeral public key epk_j.[a]
Each party $P_i \in C$ does the following:

1. Choose a random degree-t polynomial $F_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t}x^t \in \mathbb{Z}_q[x]$ with $a_{i,0} = s_i$ and compute the shares $s_{i,j} := F_i(j)$ for $j \in [n]$.
2. For each $j \in [n]$ compute $c_{i,j} \leftarrow \mathsf{PKE.Enc}(\mathsf{epk}_j, s_{i,j})$.
3. Let com_i be a commitment to s_i from the Handover procedure of the previous epoch. Compute a NIZK proof $\pi_{i,\mathsf{Handover}}$ for the statement that $(\mathsf{com}_i, \{c_{i,j}\}_{j\in[n]})$ are a commitment and encryptions of values on a degree-t polynomial w.r.t. evaluation points $j \in [n]$.
4. Choose a long-term key pair $(sk_i', pk_i') \leftarrow \mathsf{APKE.KeyGen}(1^\lambda)$ and erase sk_i and all protocol secrets.
5. Broadcast $(pk_i', \pi_{i,\mathsf{Handover}}, \{c_{i,j}\}_{j\in[n]})$.

Upon receiving $(pk_i', \pi_{i,\mathsf{Handover}}, \{c_{i,j}\}_{j\in[n]})$ for $i \in [n]$, all parties $P_j' \in C'$ do the following:

6. Verify the NIZKs $\pi_{i,\mathsf{Handover}}$ and for the first $t+1$ valid proofs π_i store i in a set $Qual$.
7. Compute $s_{i,j} \leftarrow \mathsf{PKE.Dec}(\mathsf{esk}_j, c_{i,j})$ for all $i \in Qual$.
8. Compute the secret share $s_j' = \sum_{i\in Qual} l_i \cdot s_{i,j} \in \mathbb{Z}_q$, where l_i are the corresponding Lagrange coefficients.

[a] For simplicity, we assume that Select is executed as part of the Handover procedure. This is different from the original protocol description in [9], but does not affect the functionality or security of the protocol.

We would like to point out that the encryption schemes PKE and APKE are used in the spirit of a hybrid encryption scheme, i.e., secret keys from PKE are encrypted under public keys of APKE and messages are encrypted under the public keys of PKE. Benhamouda et al. call this a "combined" encryption scheme and we will denote it by CPKE. This combined scheme must be RIND-SO secure. We recall this scheme more formally in Appendix A.3.

Finally, we note that Benhamouda et al.'s solution can be instantiated with different parameters. For instance, it has been shown to be $(128, 889, 425, 0.05)$-secure or $(128, 38557, 19727, 0.25)$-secure. A recent work by Gentry et al. [29] showed how to improve these parameters by introducing a new Select mechanism that allows Σ_{ECPSS} to remain secure for any $p < \frac{1}{2}$. More concretely, for $p = 0.25$ they only require a holding committee of size 680 and for $p = 0.40$ a committee of size 3500, which are significant improvements compared to the original parameters from [9].

Generalized Handover *procedure.* As can be seen from the above description, the Handover procedure presented by Benhamouda et al. is tailor-made for the ECPSS construction, i.e., parties in the holding committee cannot broadcast any additional values during the execution of the procedure. This makes it difficult to use the Σ_{ECPSS} scheme in black-box in other protocols. As such, we define a *generalized* Handover procedure, denoted by G–Handover, where parties can also broadcast additional values depending on the protocol building on the Σ_{ECPSS} scheme. More precisely, we write $\mathsf{G\text{–}Handover}[C^j_{\langle(s_1^j, aux_1^j),\cdots,(s_n^j, aux_n^j)\rangle}, U](pp)$ to state that the Handover procedure is executed in epoch j between the current holding committee members C^j and the parties in the universe U where each committee member $P_i^j \in C^j$ uses its secret input s_i^j as in the original Handover procedure and additionally broadcasts its auxiliary input aux_i^j alongside the other values in step 5 of the Σ_{ECPSS}.Handover procedure. Note that this change does not affect the correctness of the Σ_{ECPSS}.Handover procedure.

We will later see how protocols can be proven secure when using $\mathsf{G{-}Handover}$ as a building block. In a nutshell, it must be proven that the auxiliary information does not leak any information about the secret value that is being passed on to the next committee. We believe that, by defining $\mathsf{G{-}Handover}$ and later showing how security proofs can be written when $\mathsf{G{-}Handover}$ is used, we have paved the way for future works to build upon the Σ_{ECPSS} scheme and prove their protocols to be secure in the YOSO model.

4 Large-Scale Distributed Key Generation

In this section, we first formally define the notion of large-scale distributed key generation ($\mathsf{LS{-}DKG}$) and then present a construction in our model.

4.1 Model

A (t, n)-distributed key generation protocol (DKG) allows a set of n parties to generate a public/secret key pair (pk, sk) such that all n parties learn pk, but no single party learns sk. Instead each party learns a *share* of the secret key s.t. any subset of $t + 1$ parties can reconstruct sk. A DKG protocol is considered secure if an adversary that corrupts at most t parties learns no information about sk.

A large-scale (t, n)-distributed key generation protocol ($\mathsf{LS{-}DKG}$) differs from the above notion of distributed key generation protocols in the sense that it is defined w.r.t. a universe of parties U from which a committee of parties C of size n with $n < |U|$ is selected. This committee can then execute the key generation protocol. In terms of security, an $\mathsf{LS{-}DKG}$ protocol does not rely on the assumption that an adversary can corrupt at most t parties in C (as previous notions of DKG do), but rather assumes a fully mobile adversary with corruption power p that can corrupt up to $p \cdot |U| > t$ parties.

We now present the formal definition of a $\mathsf{LS{-}DKG}$ protocol.

Definition 10. *A large-scale (t, n)-distributed key generation protocol ($\mathsf{LS{-}DKG}$) is run among a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple $\mathsf{LS{-}DKG} = (\mathsf{Setup}, \mathsf{TKeyGen})$ of an efficient algorithm and a protocol which are defined as follows:*

$\mathsf{Setup}(1^\lambda)$*: This probabilistic algorithm takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs public parameters pp.*

$\mathsf{TKeyGen}[U](pp, t, n)$*: This is a protocol involving all parties $P_j \in U$, where each P_j receives as input public parameters pp and two integers $t, n \in \mathbb{N}$ such that $1 \leq t \leq n$. The protocol selects a committee of parties C with $|C| = n$ and outputs to all parties $P_j \in U$ a public key pk and to each party $P_i \in C$ a secret key share sk_i.*

In this work, we focus on discrete-log-based threshold cryptosystems, i.e., threshold schemes that operate over a cyclic group $\mathbb{G}$ of prime order q and output secret/public key pairs of the form (x, g^x), where $x \in \mathbb{Z}_q$ and g is a generator of $\mathbb{G}$. We now present the correctness properties of an $\mathsf{LS{-}DKG}$ scheme, which are similar to the correctness properties for DKG schemes in the discrete-log setting as introduced by Gennaro et al. [28].

Correctness: An $\mathsf{LS{-}DKG}$ protocol must satisfy the following three correctness properties.

1. All subsets of $t + 1$ secret key shares provided by honest parties in C define the same unique secret key sk.
2. After the execution of $\mathsf{TKeyGen}$, all parties $P_j \in U$ know the same public key pk which corresponds to the secret key sk.
3. sk and pk are uniformly distributed in $\mathbb{Z}_q$ and $\mathbb{G}$, respectively.

In addition to correctness, an $\mathsf{LS{-}DKG}$ scheme must satisfy the following secrecy property similar to the definition of Gennaro et al. [28].

Secrecy: An $\mathsf{LS\text{–}DKG}$ scheme is (λ, n, t, p)-secret if for every fully mobile adversary $\mathcal{A}$ with corruption power p s.t. $p \cdot |U| > t$, there exists an efficient algorithm $\mathcal{S}$, which on input a uniformly random element $pk \in \mathbb{G}$, generates an output distribution which is computationally indistinguishable from $\mathcal{A}$'s view of the output distribution of a real execution of the $\mathsf{LS\text{–}DKG}$ scheme that outputs pk.

We call a large-scale distributed key generation protocol $\mathsf{LS\text{–}DKG}$ (λ, n, t, p)-*secure*, if it is (λ, n, t, p)-secret and satisfies the correctness property.

4.2 Construction

We are now ready to present our construction of a large-scale distributed key generation ($\mathsf{LS\text{–}DKG}$) protocol. Before we explain our construction in detail, we first give an overview about the challenges that arise when designing an $\mathsf{LS\text{–}DKG}$ protocol and how we solve these challenges.

Technical challenges. Typically, discrete-log based DKG protocols are executed among a fixed set of parties where the execution can be divided into three phases: (1) share distribution, (2) qualification and (3) public key reconstruction phase. In the first phase, i.e., the share distribution phase, each party P_i chooses a random value s_i and distributes shares of this value to the other parties via a verifiable secret sharing protocol. Additionally, party P_i broadcasts a commitment to the group element g^{s_i}. The verifiability of the sharing is crucial as it allows to identify misbehaving parties and consequently to exclude those parties from the further execution of the protocol. In other words, the verifiability allows to identify a set of parties, which behaved honestly in the first phase and therefore "qualify" to participate in the further execution of the protocol. As such, the phase of identifying the set of qualified parties is called the qualification phase. At this point, all qualified parties can reconstruct their respective secret key share which is typically done by summing up the secret shares each party received from all qualified parties. In the final phase of the protocol, the public key reconstruction phase, each qualified party P_i opens its commitment to g^{s_i}, which enables the parties to reconstruct the public key from all the opened commitments of qualified parties (which is typically done by taking the product of the opened elements).

In our setting, we need to design a scheme that is secure against fully mobile adversaries. Note that such an adversary has sufficient corruption power p to corrupt $p \cdot |U| > t$ parties, which would trivially break the security of the scheme. To tackle this issue, we resort to anonymity, i.e., our protocol keeps the identity of parties anonymous by using the ECPSS scheme Σ_{ECPSS} as described in Sec. 3. In order to achieve anonymity using Σ_{ECPSS}, we must instantiate our DKG protocol in the YOSO (you only speak once) model, where each party is allowed to communicate at most *once* per epoch with other parties[3]. This, however, raises two issues as compared to previous DKG schemes since (1) parties cannot interactively identify misbehaving parties as is typically done in verifiable secret sharing protocols and (2) parties cannot first commit to a value and later send the opening of the commitment. To overcome these restrictions imposed by the YOSO model, we must design a protocol which can be executed in the span of multiple committees such that parties in one committee hand-over their state to the parties of the subsequent committee. Note however, that parties cannot simply forward their internal state to the next committee as this would allow the adversary to learn up to $2t$ shares and therefore compromise security.

Overview of our construction. Our $\mathsf{LS\text{–}DKG}$ protocol (which we denote throughout this paper by $\Pi_{\mathsf{LS\text{–}DKG}}$) follows the same three-phase framework of previous DKG protocols, while addressing the challenges of being executed in the YOSO model. At the core of our protocol lies the two committee framework of Benhamouda et al. [9] which allows selecting a committee of anonymous parties. Note that the YOSO model requires parties to perform any communication while handing over their secret states to the next anonymous committee. Therefore, it is not sufficient to simply employ the Σ_{ECPSS} scheme in black-box, since the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure does not allow parties to broadcast additional DKG-specific values. Instead we have to use the generalized handover procedure $\mathsf{G\text{–}Handover}$ as described in Sec. 3 which allows parties

[3] Recall that parties are no longer anonymous upon sending a message.

to broadcast additional values while handing over their state to the next committee and simultaneously ensuring that parties speak at most once. Note, however, that we still use the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure in black-box to select anonymous committees. Further, we make use of NIZK proofs to tackle the challenges mentioned above, i.e, to remove the need to interactively identify maliciously behaving parties and to avoid committing to a value and opening the commitment at a later time. In the following, we provide a more detailed description of our solution.

As the starting point of our protocol we assume that an anonymous committee C has been previously selected via the execution of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure. In the first phase of our protocol, the *committee selection phase*, another anonymous committee C' of size n is selected. The protocol then proceeds to the *share distribution phase*, during which each party $P_i \in C$ chooses a random value s_i which it shares to committee C' via the techniques of the Σ_{ECPSS} scheme. In order to make the sharing publicly verifiable without the need for any interaction, P_i broadcasts a NIZK proof that proves honest behavior of P_i during the share distribution phase.

In the next phase of our protocol, the *qualification phase*, each party ${P_j}' \in C'$ creates a set of qualified parties $Qual$, which consists of the first $t+1$ parties from committee C who gave a correct NIZK proof. We note that at this point, the public and secret key of the protocol are fixed as $pk = \prod_{i \in Qual} g^{s_i}$ and $sk = \sum_{i \in Qual} s_i$. Each party ${P_j}'$ can now reconstruct a secret key share sk'_j from the shares of s_i it received from parties $P_i \in Qual$. The only missing piece is to reconstruct and publish the corresponding public key pk. In order to do so, party ${P_j}'$ broadcasts $g^{sk'_j}$ along with a NIZK proof that proves that $g^{sk'_j}$ was computed correctly.

In the final phase of the protocol, the *public key reconstruction phase*, all parties in U can use the elements $g^{sk'_i}$ to compute the public key via lagrange interpolation in the exponent.

We now give a formal description of our LS–DKG protocol $\Pi_{\mathsf{LS-DKG}}$ using the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ and G–Handover procedures and a NIZK proof system NIZK.

$\mathsf{Setup}(1^\lambda)$**:** On input a security parameter λ, execute $pp^{\mathsf{ECPSS}} \leftarrow \Sigma_{\mathsf{ECPSS}}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{crs} \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$. Parse $pp^{\mathsf{ECPSS}} := (\mathsf{crs}', q)$. Choose a group $\mathbb{G}$ of prime order q with generator g such that the dlog problem is hard in $\mathbb{G}$. Output public parameters $pp^{\mathsf{LS-DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$.

$\mathsf{TKeyGen}(pp^{\mathsf{LS-DKG}}, t, n)$ **procedure:**

In the following, we denote by PKE the public key encryption scheme that generates the ephemeral keys of the Σ_{ECPSS} scheme. Note that PKE together with the anonymous public key encryption scheme APKE as used for the long-term keys in Σ_{ECPSS} constitute the combined public key encryption scheme CPKE (cf. Sec. 3).

Input: $pp^{\mathsf{LS-DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$, integers $t, n \in \mathbb{N}$, s.t. $n \geq 2t+1$ and an anonymous committee C where $|C| = n$ selected via the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure in the previous epoch.

Committee Selection Phase:

1. During the committee selection phase, the procedure $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ is executed to select a new committee C' where $|C| = |C'| = n$. Note that after the execution of $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ each party ${P_j}' \in C'$ is associated to an ephemeral public key epk_j which is known to all parties in C.

Share Distribution Phase:

2. Each party $P_i \in C$ does the following:
   (a) Choose $s_i \leftarrow_{\$} \mathbb{Z}_q$.
   (b) Choose a random degree-t polynomial $F_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t}x^t \in \mathbb{Z}_q[x]$ with $a_{i,0} = s_i$.
   (c) Compute shares $s_{i,j} := F_i(j)$ for $j \in [n]$.
   (d) For all $j \in [n]$ compute $c_{i,j} \leftarrow \mathsf{PKE}.\mathsf{Enc}(\mathsf{epk}_j, s_{i,j})$.
   (e) Compute a NIZK proof π_i for the following language:

$$\begin{aligned} L := \{((c_{i,1}, \cdots, c_{i,n}), (\mathsf{epk}_1, \cdots, \mathsf{epk}_n)) | \\ \exists (s_{i,1}, \cdots, s_{i,n}), (r_{i,1}, \cdots, r_{i,n}) \text{ s.t. } c_{i,j} \leftarrow \mathsf{PKE}.\mathsf{Enc}(\mathsf{epk}_j, s_{i,j}; r_{i,j}) \\ \wedge\, s_{i,j} \in \mathbb{Z}_q, r_{i,j} \in \mathcal{R} \text{ for } j \in [n]\}. \end{aligned}$$

Informally, π_i proves for a statement consisting of ciphertexts $(c_{i,1}, \cdots, c_{i,n})$ and ephemeral public keys $(\mathsf{epk}_1, \cdots, \mathsf{epk}_n)$ that each $c_{i,j}$ is a ciphertext encrypted under public key epk_j and $c_{i,j}$ encrypts a value in $\mathbb{Z}_q$.

(f) Erase all secret values, i.e., shares $s_{i,j}$, polynomial F_i and the value s_i.

(g) Broadcast $(\pi_i, \{c_{i,j}\}_{j\in[n]})$[a].

Qualification Phase:

3. Let $Qual = \emptyset$. Upon receiving the tuples $(\pi_i, \{c_{i,j}\}_{j\in[n]})$ for $i \in [n]$, all parties $P_j{}' \in C'$ check if π_i is valid and if so, store i in $Qual$ until $|Qual| = t + 1$.[b]
4. For all $i \in Qual$ compute $s_{i,j} \leftarrow \mathsf{PKE.Dec}(\mathsf{esk}_j, c_{i,j})$.
5. Compute the secret key share $sk'_j \in \mathbb{Z}_q$ as $sk'_j = \sum_{i\in Qual} s_{i,j}$.
6. Compute $S'_j = g^{sk'_j}$.
7. Compute a NIZK proof π'_j for the following language:

$$L' := \{\{c_{i,j}\}_{i\in Qual}, \mathsf{epk}_j, S'_j) | \exists \mathsf{esk}_j \text{ s.t. } g^{\sum_{i\in Qual} \mathsf{PKE.Dec}(\mathsf{esk}_j, c_{i,j})} = S'_j \\ \wedge \mathsf{epk}_j = \mathsf{SkToPk}(\mathsf{esk}_j)\}.$$

Informally, π'_j proves that the dlog of S'_j is the sum of the decryptions of $c_{i,j}$ for $i \in Qual$ under esk_j.

8. Execute the $\mathsf{G{-}Handover}[C'_{\langle(sk'_1,(S'_1,\pi'_1)),\cdots,(sk'_n,(S'_n,\pi'_n))\rangle}, U](pp)$ procedure. This execution selects a committee C'' with $|C''| = n$ s.t. each party $P_i{}'' \in C''$ learns a refreshed secret key share sk''_i.

Public Key Reconstruction Phase:

9. Let $PK = \emptyset$. All parties in U compute the set $Qual$ as above and check for all $j \in [n]$ if π'_j is valid and if so store S'_j in PK until $|PK| = t + 1$.
10. The public key $pk \in \mathbb{G}$ can then be computed as $pk = \prod_{k\in PK} S'^{l_k}_k$ where l_k are the corresponding lagrange coefficients.

[a] To be precise, parties must also provide a proof that they were indeed selected as members of the holding committee C in the previous epoch as in the $\mathsf{Handover}$ procedure of the Σ_{ECPSS} scheme. We omit this here for the sake of brevity.

[b] We are implicitly assuming that there is an order on these tuples.

Theorem 1. *Let the discrete-log assumption hold in $\mathbb{G}$, let NIZK be a non-interactive zero-knowledge proof system as per Def. 8, Σ_{ECPSS} be a (λ, n, t, p)-secure instantiation of the evolving-committee proactive secret sharing scheme as presented in Sec. 2.6 and CPKE be a RIND-SO secure public key encryption scheme. Then the protocol $\Pi_{\mathsf{LS-DKG}}$ from Sec. 4.2 is a (λ, n, t, p)-secure large-scale (t, n)-distributed key generation scheme.*

In order to prove Theorem 1, we have to show that $\Pi_{\mathsf{LS-DKG}}$ satisfies the correctness and secrecy property w.r.t. to a fully mobile adversary with corruption power p. We therefore state and prove the following lemmas.

Lemma 1. *The large-scale (t, n)-distributed key generation scheme $\Pi_{\mathsf{LS-DKG}}$ as presented in Sec. 4.2 the correctness property.*

We provide the proof of Lemma 1 in Appendix B.

Lemma 2. *The large-scale (t, n)-distributed key generation scheme $\Pi_{\mathsf{LS-DKG}}$ as presented in Sec. 4.2 is (λ, n, t, p)-secret.*

Proof Sketch: We provide the full proof of Lemma 2 in Appendix B, and only give the main ideas here. To prove that our scheme is (λ, n, t, p)-secret, we need to construct a simulator which on input a public key pk, can simulate an execution of the $\Pi_{\mathsf{LS-DKG}}$ protocol to a fully mobile adversary $\mathcal{A}$ in such a way that (1) the simulated execution is computationally indistinguishable from a real execution of $\Pi_{\mathsf{LS-DKG}}$ from $\mathcal{A}$'s point of view and (2) the public key that is output by the simulated execution is pk.

At a high level, the main challenge in the simulation is the following. The simulator, on behalf of honest parties P_i in committee C, has to first share the values s_i to committee C' and later adjust these values depending on the set of qualified parties and the values chosen by the adversary such that the following condition holds

$$pk = \prod_{k \in PK} g^{sk'^{l_k}_k} = \prod_{k \in PK} g^{(\sum_{j \in Qual} s_{k,j})^{l_k}}. \tag{1}$$

As a first step of our proof, we show that a fully mobile adversary $\mathcal{A}$ can corrupt at most t parties in each committee (i.e., in C, C' and C'' respectively) by exhibiting a reduction to the secrecy property of the Σ_{ECPSS} scheme. That is, we show that if $\mathcal{A}$ is able to corrupt more than t parties in either of C, C' or C'' with non-negligible probability, then we can construct an adversary that corrupts more than t parties of a holding committee in Σ_{ECPSS} with non-negligible probability, thereby breaking the secrecy property of Σ_{ECPSS}. From this follows that there is an honest majority in each of the committees C, C' and C''.

The main idea behind our simulation is to use the fact that the adversary corrupts at most a minority of parties in each committee. More concretely, this fact allows to make the following two observations: (1) there must exist at least one honest party in $Qual$ since the set of qualified parties $Qual$ consists of $t+1$ parties, and (2) the simulator has sufficient information to reconstruct all secret key shares sk'_k for all parties $P_k{}' \in C'$ and thereby to learn all elements S'_k. The simulator then "adjusts" the elements S'_j for some honest parties $P_j{}'$ in C' such that for any subset $T \subset \{S'_1, \cdots, S'_n\}$ with $|T| = t+1$ it is possible to reconstruct pk via interpolation in the exponent. This ensures that Eq. (1) holds. The simulator can then broadcast the adjusted elements S'_j along with a simulated NIZK proof. In order to prove that the adversary cannot distinguish the simulated values S'_j from the real ones, we exhibit reductions to the RIND-SO security of the CPKE scheme.

5 Large-Scale Threshold Public Key Encryption

In this section, we first introduce the notion of large-scale non-interactive threshold public key encryption (LS–TPKE) and then show a construction based on the threshold public key encryption scheme by Shoup and Gennaro [48], the evolving-committee proactive secret sharing solution Σ_{ECPSS} from Section 3, the large-scale distributed key generation scheme $\Pi_{\mathsf{LS-DKG}}$ from the previous section and a NIZK proof system.

5.1 Model

We now formally define the notion of a large-scale non-interactive threshold public key encryption scheme LS–TPKE. Let us start by pointing out the main differences between an LS–TPKE scheme and a TPKE scheme.

First, in contrast to a TPKE scheme, an LS–TPKE scheme does not consider the committee of secret key shareholders as an external input to the protocol, but rather includes the committee selection procedure as part of the scheme. More precisely, we define an LS–TPKE scheme w.r.t. to a universe U of parties from which committees are selected. Second, the execution of an LS–TPKE scheme proceeds in epochs, i.e., its execution is divided into time intervals at the beginning of which a new committee of secret key shareholders is selected. In order to transition from one epoch to the next, the previous-epoch committee must pass the secret key shares on to the next-epoch committee. We therefore add a refresh procedure to an LS–TPKE scheme which allows for such a transition between epochs. Finally, and most importantly, we define the security of an LS–TPKE scheme w.r.t. to a fully mobile adversary whose corruption power grows in the universe size $|U|$ instead of the size of the committee of secret key shareholders. Hence, the adversarial corruption power suffices to possibly corrupt all secret key shareholders in one epoch.

We now provide the formal definition of an LS–TPKE scheme.

Definition 11. *A large-scale non-interactive (t, n)-threshold public key encryption scheme* (LS–TPKE) *is defined w.r.t. a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple* LS–TPKE = (Setup, TKeyGen, TEnc, TDec, TShareVrfy, TCombine, Refresh) *of efficient algorithms and protocols which are defined as follows:*

$\mathsf{Setup}(1^\lambda)$*: This probabilistic algorithm takes a security parameter* $\lambda \in \mathbb{N}$ *as input and outputs public parameters* pp*.*

$\mathsf{TKeyGen}[U](pp, t, n)$*: This is a protocol involving all parties* $P_j \in U$*, where each* P_j *receives as input public parameters* pp *and two integers* $t, n \in \mathbb{N}$ *such that* $1 \leq t \leq n$*. The protocol selects a committee of parties* C *with* $|C| = n$ *and outputs to all parties* $P_j \in U$ *a public key* pk *and to each party* $P_i \in C$ *a verification key* vk_i *and a secret key share* sk_i*.*

$\mathsf{TEnc}(pk, m, L)$*: This probabilistic algorithm takes a public key* pk*, a message* m *and a label* L *as input and outputs a ciphertext* ct*.*

$\mathsf{TDec}(sk_i, ct, L)$*: This algorithm takes as input a secret key share* sk_i*, a ciphertext* ct *and a label* L *and it outputs either* $\perp$ *or a decryption share* ct_i *of the ciphertext* ct*.*

$\mathsf{TShareVrfy}(ct, vk_i, ct_i)$*: This deterministic algorithm takes as input a ciphertext* ct*, a verification key* vk_i *and a decryption share* ct_i *and outputs either* 1 *or* 0*. If the output is* 1*,* ct_i *is called a valid decryption share.*

$\mathsf{TCombine}(T, ct)$*: This deterministic algorithm takes as input a set of valid decryption shares* T*, s.t.* $|T| = t + 1$ *and a ciphertext* ct *and it outputs a message* m*.*

$\mathsf{Refresh}[C_{\langle (sk_1, vk_1, dl_1), \cdots, (sk_n, vk_n, dl_n) \rangle}, U](pp)$*: This is a protocol involving a committee* C *with* $|C| = n$ *and the universe of parties* U*. Each* $P_i \in C$ *takes as secret input a secret key share* sk_i*, verification key* vk_i *and decryption share list* dl_i*, and all parties* $P_j \in U$ *take as input public parameters* pp*. The protocol selects a committee of parties* C' *with* $|C'| = n$ *and outputs to each party* ${P_i}' \in C'$ *a verification key* vk'_i *and a secret key share* sk'_i*. Furthermore, all parties in the universe receive* vk_i *and* dl_i *for* $i \in [n]$*.*

We will now define the properties that an LS–TPKE must satisfy, namely *Correctness*, *CCA-Security* and *Decryption Consistency*. In these definitions, we denote by C^j the committee in the j-th epoch (similarly for a party ${P_i}^j \in C^j$, we denote verification keys as vk_i^j, secret key shares as sk_i^j, decryption shares as ct_i^j and decryption share lists as dl_i^j).

Correctness A $(t, n) - \mathsf{LS–TPKE}$ scheme must fulfill the following two requirements. For any $\lambda \in \mathbb{N}$, any $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and any $(pk, \{vk_i^1\}_{i \in [n]}, \{sk_i^1\}_{i \in [n]}) \leftarrow \mathsf{TKeyGen}[U](pp, t, n)$ with selected committee C^1, for $j > 1$ we define $(\{vk_i^j\}_{i \in [n]}, \{sk_i^j\}_{i \in [n]})$ recursively as

$$(\{vk_i^j\}_{i \in [n]}, \{sk_i^j\}_{i \in [n]}) \leftarrow \mathsf{Refresh}[C^{j-1}_{\langle (sk_1^{j-1}, vk_1^{j-1}, \cdot), \cdots, (sk_n^{j-1}, vk_n^{j-1}, \cdot) \rangle}, U](pp)$$

Recall that during these executions verification keys vk_i^{j-1} and decryption share lists dl_i^{j-1} for $i \in [n]$ are broadcasted.

1. For any message m, any label L and any ciphertext $ct \leftarrow \mathsf{TEnc}(pk, m, L)$, it must hold that

$$\mathsf{TShareVrfy}(ct, vk_i^j, \mathsf{TDec}(sk_i^j, ct, L)) = 1$$

2. For all decryption share lists dl_i^{j-1} where $i \in [n]$, each element in the list is computed as $ct_{i,k}^{j-1} \leftarrow \mathsf{TDec}(sk_i^{j-1}, ct_k, L)$, where $ct_k \leftarrow \mathsf{TEnc}(pk, m_k, L_k)$ for a message m_k and a label L_k. Further, for any set $T_k = \{ct_{1,k}^{j-1}, \cdots, ct_{t+1,k}^{j-1}\}$, it holds that $\mathsf{TCombine}(T_k, ct_k) = m_k$.

CCA-Security In the following, we give the definition of chosen-ciphertext security for a $(t, n) - \mathsf{LS–TPKE}$ scheme considering an efficient fully mobile adversary $\mathcal{A}$ with corruption power p s.t. $p \cdot |U| > t$. The state-of-the-art communication model for designing and analyzing protocols that are secure against such a strong adversary is the YOSO model, in which committee members can speak at most once before having to hand over their secret key share to the next committee. This has the following interesting implications on the definition of the security game as compared to the notion of CCA-security for a TPKE scheme (cf. Appendix A). First, upon a decryption oracle query, the game has to output decryption shares on behalf of honest secret key shareholders. However, this requires these shareholders to "speak". As each party should speak only once, decryption shares must be computed locally without immediately outputting them. Instead, only upon refreshing the secret key shares to the next committee can all previously computed decryption shares be output. Second, in contrast to traditional threshold public key encryption schemes, the verification key of each honest committee member remains private until decryption shares are output. Note that (1) the verification keys are required only to check the validity of decryption shares and (2) verification keys depend on the secret key shares, i.e., they are refreshed in each epoch. Therefore, it is sufficient to output verification keys simultaneously with the decryption shares at the end of an epoch.

We formally define the following game $\mathsf{LSTPKE{-}CCA}^{\mathcal{A}}_{\mathsf{LS{-}TPKE}}(\lambda)$ which is initialized with a security parameter λ:

1. The game executes $\mathsf{Setup}(1^\lambda)$ and obtains public parameters pp, which it forwards to the adversary $\mathcal{A}$. For each epoch $j \geq 0$, the game maintains a set of corrupted parties B^j which is initialized as $B^j := \emptyset$.
2. The adversary $\mathcal{A}$ is given access to the following corruption oracle:
   – **Corruption oracle**: On input an index $i \in [N]$, the game checks if $\left\lfloor \frac{|B^j|+1}{|U|} \right\rfloor \leq p$. If so, $\mathcal{A}$ receives the internal state of party $P_i{}^j$ and the game sets $B^j \leftarrow B^j \cup \{P_i{}^j\}$.
3. The protocol $\mathsf{TKeyGen}[U](pp, t, n)$ is executed. The protocol selects a committee C^1 with $|C^1| = n$ and outputs a public key pk, a set of verification keys $\{vk^1_1, \cdots, vk^1_n\}$ and a set of secret key shares $\{sk^1_1, \cdots, sk^1_n\}$, such that $P_i{}^1 \in C^1$ learns vk^1_i and sk^1_i.
4. At this point, $\mathcal{A}$ additionally obtains access to the following two oracles. Let $dl^1_i := \emptyset$ for $i \in [n]$:
   – **Refresh oracle**: On input a set $NB^j \subseteq B^j$, the protocol $\mathsf{Refresh}[C^j_{\langle (sk^j_1, vk^j_1, dl^j_1), \cdots, (sk^j_n, vk^j_n, dl^j_n) \rangle}, U](pp)$ is executed and the game sets $B^{j+1} \leftarrow B^j \setminus NB^j$. It further initializes lists $dl^{j+1}_i := \emptyset$ for $i \in [n]$.
   – **Decryption oracle**: On input a set of ciphertexts CT^j with a set of associated labels AL^j, the game computes $ct^j_{i,k} \leftarrow \mathsf{TDec}(sk^j_i, ct^j_k, L^j_k)$ for all $ct^j_k \in CT^j$ and $L^j_k \in AL^j$ and for all parties $P_i{}^j \in C^j \setminus B^j$. Then, the oracle adds all $ct^j_{i,k}$ to the list dl^j_i.
5. Eventually, $\mathcal{A}$ chooses two messages m_0, m_1 with $|m_0| = |m_1|$ and a label L and sends them to the game. The game chooses a random bit $b \leftarrow_\$ \{0,1\}$ and sends $ct' \leftarrow_\$ \mathsf{TEnc}(pk, m_b, L)$ to $\mathcal{A}$.
6. $\mathcal{A}$ is allowed to make queries as described in steps 2. and 4. with the exception that it cannot make a decryption query on ciphertext ct'.
7. Eventually, $\mathcal{A}$ outputs a bit b'. The game outputs 1 if $b' = b$ and 0 otherwise.

Definition 12. *A large-scale non-interactive (t,n)-threshold public key encryption scheme* $\mathsf{LS{-}TPKE}$ *with a universe of parties U is secure against chosen-ciphertext attacks w.r.t. parameters (λ, n, t, p) s.t. $p \cdot |U| > t$ if for every fully mobile PPT adversary $\mathcal{A}$ with corruption power p there exists a negligible function ν in the security parameter λ, such that*

$$\Pr[\mathsf{LSTPKE{-}CCA}^{\mathcal{A}}_{\mathsf{LS{-}TPKE}}(\lambda) = 1] \leq 1/2 + \nu(\lambda).$$

We define the advantage of $\mathcal{A}$ in game $\mathsf{LSTPKE{-}CCA}^{\mathcal{A}}_{\mathsf{LS{-}TPKE}}$ *as*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{LSTPKE{-}CCA},\mathsf{LS{-}TPKE}}(\lambda) = |\Pr[\mathsf{LSTPKE{-}CCA}^{\mathcal{A}}_{\mathsf{LS{-}TPKE}}(\lambda) = 1] - 1/2|.$$

Definition 13 (Decryption Consistency). *An* $\mathsf{LS{-}TPKE}$ *scheme satisfies decryption consistency w.r.t. parameters (λ, n, t, p) if there exists no fully mobile PPT adversary $\mathcal{A}$ with corruption power p that wins the game* $\mathsf{LSTPKE{-}DC}$ *described below with non-negligible probability:*

$\mathsf{LSTPKE{-}DC}$*: The game begins with steps 1.-4. as in game* $\mathsf{LSTPKE{-}CCA}$ *with the difference that the adversary is allowed to learn all secret key shares in each epoch j*[4]*. The adversary eventually outputs a ciphertext ct^*, two sets of verification keys $VK = \{vk^j_1, \cdots, vk^j_{t+1}\}$ and $\tilde{VK} = \{\tilde{vk}^j_1, \cdots, \tilde{vk}^j_{t+1}\}$ and two sets of decryption shares $T = \{ct^j_1, \cdots, ct^j_{t+1}\}$ and $\tilde{T} = \{\tilde{ct}^j_1, \cdots, \tilde{ct}^j_{t+1}\}$ and wins the game if the following conditions hold:*

[4] Note however that the adversary is not controlling these parties, i.e., not all parties are corrupted.

1. *For all* $i \in [t+1]$ *it holds that* $\mathsf{TShareVrfy}(ct^*, vk_i^j, ct_i^j) = 1$ *and* $\mathsf{TShareVrfy}(ct^*, \tilde{vk}_i^j, \tilde{ct}_i^j) = 1$.
2. $\mathsf{TCombine}(T, ct^*) \neq \mathsf{TCombine}(\tilde{T}, ct^*)$

We call a large-scale non-interactive (t,n)-threshold public key encryption scheme $\mathsf{LS{-}TPKE}$ (λ, n, t, p)-secure, if it satisfies correctness, decryption consistency and CCA-security w.r.t. parameters (λ, n, t, p).

5.2 Construction

Shoup and Gennaro [48] introduced two threshold public key encryption schemes denoted as $\mathsf{TDH1}$ and $\mathsf{TDH2}$ respectively, which are both CCA-secure against static adversaries in the random oracle model [6]. In the following, we show how the scheme $\mathsf{TDH1} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{TEnc}, \mathsf{TShareVrfy}, \mathsf{TCombine})$ can be transformed into a secure large-scale threshold public key encryption scheme $\Pi_{\mathsf{LS{-}TPKE}} = (\mathsf{Setup}, \mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{TDec}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Refresh})$. For this transformation we make use of the large-scale distributed key generation scheme $\Pi_{\mathsf{LS{-}DKG}} = (\mathsf{Setup}, \mathsf{TKeyGen})$ as described in Sec. 4, the two committee framework of Benhamouda et al. as well as the $\mathsf{G{-}Handover}$ procedure as presented in Sec. 3 and a NIZK proof system $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as per Def. 8.

Note that for similar reasons as for our $\Pi_{\mathsf{LS{-}DKG}}$ protocol, we cannot use the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure in black-box. Instead we have to use the generalized handover procedure $\mathsf{G{-}Handover}$, which internally uses the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure and the combined public key encryption scheme CPKE (cf. Sec. 3).

We detail the construction of the $\Pi_{\mathsf{LS{-}TPKE}}$ scheme below and we recall the $\mathsf{TDH1}$ scheme in Appendix D.

$\Pi_{\mathsf{LS{-}TPKE}}.\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, execute

$$pp^{\mathsf{TDH1}} \leftarrow \mathsf{TDH1}.\mathsf{Setup}(1^\lambda),\ \tilde{pp}^{\mathsf{LS{-}DKG}} \leftarrow \Pi_{\mathsf{LS{-}DKG}}.\mathsf{Setup}(1^\lambda)$$
$$\mathsf{crs} \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$$

Recall that $pp^{\mathsf{LS{-}DKG}}$ can be parsed as $\tilde{pp}^{\mathsf{LS{-}DKG}} := (\mathsf{crs}', \mathbb{G}, q, g)$. Define $pp^{\mathsf{LS{-}DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$ and output public parameters $pp := (pp^{\mathsf{TDH1}}, pp^{\mathsf{LS{-}DKG}})$.

$\Pi_{\mathsf{LS{-}TPKE}}.\mathsf{TKeyGen}[U](pp, t, n)$: On input public parameters pp and two integers $t, n \in \mathbb{N}$ s.t. $n \geq 2t+1$, this protocol parses $pp := (pp^{\mathsf{TDH1}}, pp^{\mathsf{LS{-}DKG}})$ and calls the $\Pi_{\mathsf{LS{-}DKG}}.\mathsf{TKeyGen}(pp^{\mathsf{LS{-}DKG}}, t, n)$ procedure, which selects a committee C^1 with $|C^1| = n$ and outputs a public key pk to all parties in U and secret key shares sk_i^1 to each party $P_i{}^1 \in C^1$. Additionally, all $P_i{}^1$ compute $\widehat{vk}_i^1 := g^{sk_i^1}$ and a NIZK proof π_i^1 proving that $\widehat{vk}_i^1$ was computed correctly[5]. $P_i{}^1$ then sets the verification key $vk_i^1 := \{\widehat{vk}_i^1, \pi_i^1\}$ and initializes a decryption share list $dl_i^1 := \emptyset$.

$\Pi_{\mathsf{LS{-}TPKE}}.\mathsf{TEnc}(pk, m, L)$: This procedure executes $\mathsf{TDH1}.\mathsf{TEnc}$.

$\Pi_{\mathsf{LS{-}TPKE}}.\mathsf{TDec}(sk_i^j, ct, L)$: This procedure executes $\mathsf{TDH1}.\mathsf{TDec}$ and adds the resulting decryption share to a list dl_i.

$\Pi_{\mathsf{LS{-}TPKE}}.\mathsf{TShareVrfy}(ct, vk_i^j, ct_i^j)$: On input a ciphertext ct, a verification key $vk_i^j := \{vk_i^{j'}, \pi_i^j\}$ and a decryption share ct_i^j, this procedure checks if π_i^j is a valid proof w.r.t. $\widehat{vk}_i^j$ (i.e., it checks if $\widehat{vk}_i^j$ is indeed the correct verification key of party $P_i{}^j \in C^j$). If this check does not hold, the procedure outputs 0. Otherwise, it outputs $\mathsf{TDH1}.\mathsf{TShareVrfy}(ct, \widehat{vk}_i^j, ct_i^j)$.

$\Pi_{\mathsf{LS{-}TPKE}}.\mathsf{TCombine}(T, ct)$: This is the $\mathsf{TDH1}.\mathsf{TCombine}$ procedure.

[5] The language for this proof is the same as the language L' in the $\Pi_{\mathsf{LS{-}DKG}}$ protocol.

$\Pi_{\mathsf{LS-TPKE}}.\mathsf{Refresh}[C^j_{\langle (sk^j_1, vk^j_1, dl^j_1), \cdots, (sk^j_n, vk^j_n, dl^j_n)\rangle}, U](pp)$: This protocol is executed between a committee C^j in epoch j and the universe U, where each $P^j_i \in C^j$ receives as input a secret key share sk^j_i, the verification key vk^j_i and the decryption share list dl^j_i. Furthermore, each party $P_k \in U$ receives as input $pp := (pp^{\mathsf{TPKE}}, pp^{\mathsf{LS-DKG}})$.
The protocol first runs $\mathsf{G\text{-}Handover}[C^j_{\langle (sk^j_1, (vk^j_1, dl^j_1)), \cdots, (sk^j_n, (vk^j_n, dl^j_n))\rangle}, U](pp)$ which selects a committee C^{j+1} with $|C^{j+1}| = n$ and outputs refreshed secret key shares sk^{j+1}_i to each $P^{j+1}_i \in C^{j+1}$. Additionally, all $P^{j+1}_i \in C^{j+1}$ compute $\widehat{vk}^{j+1}_i := g^{sk^{j+1}_i}$, generate a NIZK proof π^{j+1}_i that the verification key was computed correctly [6] and set $vk^{j+1}_i := \{\widehat{vk}^{j+1}_i, \pi^{j+1}_i\}$. Finally, all P^{j+1}_i initialize a decryption share list $dl^{j+1}_i := \emptyset$.

Theorem 2. *Let $\Pi_{\mathsf{LS-DKG}}$ be a (λ, n, t, p)-secure instantiation of the large-scale (t, n)-distributed key generation protocol from Sec. 4,* TDH1 *be the non-interactive (t, n)-threshold public key encryption scheme as described in Appendix D which is secure against chosen-ciphertext attacks with static corruptions in the ROM according to Def. 6, Σ_{ECPSS} a (λ, n, t, p)-secure instantiation of the evolving-committee proactive secret sharing scheme as presented in Sec. 3,* NIZK *a non-interactive zero-knowledge proof system as per Def. 8 and* CPKE *a RIND-SO secure public key encryption scheme. Then $\Pi_{\mathsf{LS-TPKE}}$ is a (λ, n, t, p)-secure large-scale non-interactive (t, n)-threshold public key encryption scheme in the ROM.*

In order to prove Theorem 2, we have to show that $\Pi_{\mathsf{LS-TPKE}}$ satisfies correctness and decryption consistency as well as security against chosen-ciphertext attacks w.r.t. parameters (λ, n, t, p). We therefore state the following lemmas.

Lemma 3. *The large-scale non-interactive (t, n)-threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}}$ as described in Sec. 5.2 satisfies correctness.*

Proof. This lemma follows directly from the correctness property of the TDH1 scheme, the completeness property of the NIZK proof system and from the handover correctness [9] of G–Handover. We provide a proof outline for Lemma 3 in Appendix C.

Lemma 4. *The large-scale non-interactive (t, n)-threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}}$ as described in Sec. 5.2 satisfies decryption consistency w.r.t. parameters (λ, n, t, p).*

Proof. We provide a proof for Lemma 4 in Appendix C.

Lemma 5. *The large-scale non-interactive (t, n)-threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}}$ as described in Sec. 5.2 is secure against chosen-ciphertext attacks w.r.t. parameters (λ, n, t, p).*

Proof Sketch. We provide the full formal proof of Lemma 5 in Appendix C. We provide now a high level proof sketch that summarizes the main ideas of our proof. At a high level, we show that if there exists a fully mobile adversary $\mathcal{B}$ with corruption power p who can win game $\mathsf{LSTPKE\text{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ with non-negligible probability, then there exists an efficient static adversary $\mathcal{A}$ who can use $\mathcal{B}$ to win its own game $\mathsf{TPKE\text{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ (cf. Def. 6) with non-negligible probability. Therefore, we show how $\mathcal{A}$ can simulate game $\mathsf{LSTPKE\text{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ to $\mathcal{B}$ in such a way that the simulation is indistinguishable from a real execution to $\mathcal{B}$ and how $\mathcal{A}$ can use $\mathcal{B}$'s output bit b' to win its own game.

The first step of our proof, similar to the proof of Lemma 2, is to show via a reduction to the secrecy property of the Σ_{ECPSS} scheme that $\mathcal{B}$ corrupts at most t secret key shareholders (i.e., committee members) per epoch.

We then show how $\mathcal{A}$ simulates the game $\mathsf{LSTPKE\text{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ to $\mathcal{B}$ w.r.t. the public key pk that it receives from its own game $\mathsf{TPKE\text{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$. $\mathcal{A}$ embeds pk in game $\mathsf{LSTPKE\text{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ by executing the simulator code of the $\Pi_{\mathsf{LS-DKG}}$ scheme (cf. Fig. 1) on input pk. After this execution, $\mathcal{A}$ knows the secret key

[6] The language for this proof is the same as the language L' in the $\Pi_{\mathsf{LS-DKG}}$ protocol.

shares of all honest and malicious parties. Note however that, according to the simulation strategy for the $\Pi_{\mathsf{LS-DKG}}$ scheme, these secret key shares are merely random values in $\mathbb{Z}_q$ that are independent of pk. The main idea of the proof is now to show that $\mathcal{A}$ can simulate game $\mathsf{LSTPKE–CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ to the adversary $\mathcal{B}$, without $\mathcal{B}$ noticing that the committee members hold a sharing of a random value.

In order to show this, we make the following crucial observation that is unique to the YOSO model. To simulate the decryption and refresh oracles to $\mathcal{B}$, the adversary $\mathcal{A}$ has to simulate verification keys and decryption shares for honest committee members that are consistent with $\mathcal{B}$'s view. In particular, this means that these simulated verification keys and decryption shares are not consistent with the secret key shares of honest parties. However, as the YOSO model requires committee members to first erase their secret key shares before outputting their verification key and decryption shares, this inconsistency between public information and the internal state of honest parties remains undetected by $\mathcal{B}$. Said differently, if $\mathcal{B}$ corrupts a committee member *before* this member has output its verification key and decryption shares, there exists no inconsistent public information through which $\mathcal{B}$ could distinguish the simulation from a real execution (as long as $\mathcal{B}$ corrupts at most t committee members). On the other hand, if $\mathcal{B}$ corrupts a committee member *after* this member has output its verification key and decryption shares, then the secret key share has already been erased and there is again no inconsistency between public information and internal states.

Remark 1. We note that the scheme $\Pi_{\mathsf{LS-TPKE}}$ inherits the security guarantee of the underlying TDH1 scheme, i.e., since the TDH1 scheme is chosen-ciphertext secure, so is $\Pi_{\mathsf{LS-TPKE}}$.

6 Transformation Framework from TPKE to LS–TPKE

We now discuss how the transformation of the TDH1 scheme to an LS–TPKE scheme as shown in Sec. 5 can be generalized to a framework that transforms discrete-log-based non-interactive threshold encryption schemes secure against static adversaries to a large-scale non-interactive threshold encryption scheme. We will show in Appendix E how the same idea can be applied to non-interactive threshold signature schemes.

At a high level, we utilize our LS–DKG protocol $\Pi_{\mathsf{LS-DKG}}$ as presented in Sec. 4, the evolving-committee proactive secret sharing scheme Σ_{ECPSS} as described in Sec. 3, the G–Handover procedure and a NIZK proof system to transform a non-interactive (t, n)-threshold encryption scheme secure against static adversaries TPKE to a large-scale non-interactive (t, n)-threshold encryption scheme secure against fully mobile adversaries LS–TPKE. For the transformation to succeed the TPKE scheme must satisfy certain properties which we discuss in the following.

Properties of the TPKE scheme. We abstract the properties that the TDH1 scheme satisfies and which allowed us to transform it into a secure LS–TPKE scheme.

Compatibility with $\Pi_{\mathsf{LS-DKG}}$. The TDH1 scheme is compatible with our $\Pi_{\mathsf{LS-DKG}}$ protocol, i.e., the public key pk and secret key shares $(sk_1, \cdots, sk_n)$ as output by $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ can be used in TDH1. More concretely, it must hold that $(pk, \cdot, (sk_1, \cdots, sk_n)) \in \mathsf{TDH1.KeyGen}$.

Dlog-Based Verification Keys. Secret key shares and the corresponding verification keys form a discrete-log instance in TDH1, i.e., for a secret key share sk_i the corresponding verification key is of the form $vk_i = g^{sk_i}$.

Simulatability. There exists a simulator for the TDH1 scheme that simulates the TPKE–CCA game (cf. Sec. 2.3) to a static adversary on input a public key, verification keys and t secret key shares in such a way that the simulated execution of the TPKE–CCA game is computationally indistinguishable from a real execution to the adversary. Intuitively, we use such a simulator in our security proof to simulate decryption oracle (and possibly random oracle) queries to the fully mobile adversary playing in game $\mathsf{LSTPKE–CCA}_{\mathsf{LS-TPKE}}$. However, since an LS–TPKE scheme is executed in epochs, it must be ensured that the simulation of the oracles remains consistent across multiple consecutive executions of the protocol with differing secret key shares and verification keys while the public key stays the same.

Let us now show why the above properties are crucial to transform a TPKE scheme into a secure $\mathsf{LS{-}TPKE}$ scheme.

The first property is naturally required in order to use our $\Pi_{\mathsf{LS-DKG}}$ protocol in combination with the TPKE scheme to construct an $\mathsf{LS{-}TPKE}$ scheme. The second property might seem less obvious and requires some further intuition. In our security proof of the $\Pi_{\mathsf{LS-TPKE}}$ scheme from Sec. 5, the reduction has to output verification keys for honest parties in each epoch without knowing the corresponding secret key shares. However, the reduction knows the verification keys of corrupted parties (say w.l.o.g. $(vk_1, \cdots, vk_t)$) and the public key pk. We know that the public key is of the form $pk = g^{sk}$ and hence, if a verification key vk_i is of the form $vk_i = g^{sk_i}$, then the reduction can use pk and $(vk_1, \cdots, vk_t)$ to construct a degree-t polynomial F in the exponent such that $F(0) = sk$ and $F(i) = sk_i$ for $i \in [t]$. The reduction can then evaluate F in the exponent at points $j \in [t+1; n]$ to compute valid verification keys for honest parties without the knowledge of the corresponding secret key shares. For the third property, we already provided an intuition above. In a nutshell, we require a simulator which we can use in our security proof to simulate the responses to oracle queries from a fully mobile adversary in game $\mathsf{LSTPKE{-}CCA}_{\mathsf{LS-TPKE}}$.

This transformation framework can be used for multiple different discrete-log-based TPKE schemes such as [48], [5] and [43]).

In Appendix E, we present a model for large-scale non-interactive threshold signature schemes and argue that the framework presented in this section can be applied for threshold signature schemes as well.

7 Applications

In this section, we show several interesting applications of our $\mathsf{LS{-}TSIG}$ and $\mathsf{LS{-}TPKE}$ schemes. Our schemes are perfectly suited to be used in blockchain networks, which have increasingly gained attention in the cryptography community as they have proven to be surprisingly versatile for the realization of cryptographic primitives and protocols. We split applications of our solutions into two categories, (1) storage of blockchain-backed secrets and (2) adding signing functionality to a blockchain.

7.1 Storage of Blockchain-Backed Secrets

Any information stored on a blockchain is publicly available to all users which severely restricts the usefulness of a blockchain and the class of applications it supports. Recently, Benhamouda et al. [9] and Goyal et al. [31] presented solutions based on secret sharing to allow the storage of secret values on a blockchain[7]. At a high level, these solutions allow a client to secret share a value to a committee, which then stores the secret and periodically refreshes the shares to a new committee. However, for many applications it is not necessary to store the secret on the entire blockchain, but it rather suffices to have a functionality that allows to commit to a secret and have the blockchain open the commitment in case of malicious behavior during the execution of the application.

Consider the example of a fair exchange protocol. Assume two parties, say Alice and Bob, wish to exchange secrets a and b, where Alice initially owns a and Bob owns b. Alice and Bob could now use either solution of Benhamouda et al. or Goyal et al. to share a and b to the committee and once both parties have done so, the committee could send the shares of b to Alice and vice versa. There are, however, several issues with this solution: (1) Alice and Bob have to interact with the committee, (2) the committee has to store shares of a and b and has to possibly refresh the shares to a new committee and (3) the committee members learn that Alice and Bob exchange secrets, thereby compromising the two parties' privacy. Instead, assume that each committee member holds a secret key share of an $\mathsf{LS{-}TPKE}$ scheme and that the corresponding public key pk is stored on the blockchain. In this case, Alice and Bob could just encrypt their secrets under pk

[7] Likewise Kokoris-Kogias et al. [39] presented a solution for auditable data-management based on blockchain and threshold encryption which allows for storage of secrets on a blockchain. We refer to Appendix A for further discussion.

and exchange the ciphertexts[8]. Once each of them have received the respective ciphertext, they can reveal their secrets to each other. If one party misbehaves, say Alice, by not revealing her secret, Bob can let the committee decrypt Alice's ciphertext[9]. Note that, in the optimistic case, i.e., when no party misbehaves, then we have that (1) there is no interaction with the committee required, (2) the committee does not have to store and refresh the secrets a and b and (3) the committee does not learn which parties interact with each other.

Naturally, the same idea can be used to store secrets on a blockchain if necessary, i.e., if Alice wants to store a secret on the blockchain, she can simply encrypt the secret under the committee's public key and publish the ciphertext to the blockchain. The advantage of this solution compared to the secret sharing based solutions of Benhamouda et al. and Goyal et al. is that the committee has to only refresh its secret key shares (instead of all secrets that are stored on the blockchain) and therefore the communication complexity of replacing a committee by a new committee is independent of the number of stored secrets.

7.2 Adding Signing Functionality to a Blockchain

Our $\mathsf{LS{-}TSIG}$ scheme can be used to generate signatures "on behalf" of the blockchain. This allows to sign individual blocks of the blockchain, thereby certifying that the block is indeed a valid part of the blockchain or sign certain messages indicating that a specific event has occurred on the blockchain. Benhamouda et al. [9] previously mentioned that extending their solution to a threshold signature scheme (as we did in this work) opens the door to various interesting applications. We briefly recall two applications here. We note that Benhamouda et al. have never formally shown how to construct such a threshold signature scheme from their solution.

Blockchain Interoperability. Blockchain interoperability deals with the issue of running applications across multiple different blockchain networks. This often requires proving to a blockchain $\mathcal{B}$ that a certain event has occurred on another blockchain $\mathcal{A}$. In order to do so, trusted parties can be used that are part of both networks and therefore can mediate between two blockchains. With our $\mathsf{LS{-}TSIG}$ scheme, however, blockchain $\mathcal{A}$ can simply create a signature on a message indicating that the event in question has occurred and this message/signature pair can be sent to blockchain $\mathcal{B}$. Parties in blockchain $\mathcal{B}$ merely require the signing public key of blockchain $\mathcal{A}$ to verify the signature.

Blockchain Checkpointing. Checkpoints on a blockchain allow to certify that a certain blockchain state is valid. This proves to be particularly useful for new parties joining a blockchain network, as these parties are not anymore required to download and validate the entire blockchain starting at the first block. Instead, new parties can download the blocks since the latest checkpoint and validate the blocks that succeed this checkpoint. This significantly improves computation time of parties joining a blockchain system. A threshold signature scheme, like our $\mathsf{LS{-}TSIG}$ scheme, can be used to build such checkpoints by simply signing valid blocks. The signature serves as a proof for the block's validity.

Acknowledgments

This work is supported by the German Research Foundation (DFG) Emmy Noether Program FA 1320/1-1, by the German Research Foundation DFG - SFB 1119 - 236615297 (CROSSING Projects P1 and S7), by the German Federal Ministry of Education and Research (BMBF) *iBlockchain Project* (grant nr. 16KIS0902), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the *National Research Center for Applied Cybersecurity ATHENE*.

[8] Along with NIZK proofs that prove that the ciphertexts indeed encrypt a and b.

[9] We note that the label of the ciphertext can be used as a decryption policy that might say in this case: "If Bob publishes his ciphertext on the blockchain, he is allowed to learn the content of Alice's ciphertext."

References

[1] M. Abe and S. Fehr. “Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography”. In: *Advances in Cryptology – CRYPTO 2004*. Berlin, Heidelberg, 2004.

[2] I. Abraham et al. “Reaching Consensus for Asynchronous Distributed Key Generation”. In: *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. Virtual Event, Italy, 2021.

[3] J. F. Almansa et al. “Simplified Threshold RSA with Adaptive and Proactive Security”. In: *Advances in Cryptology - EUROCRYPT 2006*. Berlin, Heidelberg, 2006.

[4] A. Asayag et al. *Helix: A Scalable and Fair Consensus Algorithm Resistant to Ordering Manipulation*. Cryptology ePrint Archive, Report 2018/863. 2018.

[5] J. Baek and Y. Zheng. “Simple and efficient threshold cryptosystem from the Gap Diffie-Hellman group”. In: *GLOBECOM ’03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489)*. 2003.

[6] M. Bellare and P. Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *ACM CCS 93*. 1993.

[7] M. Bellare et al. “Key-Privacy in Public-Key Encryption”. In: *ASIACRYPT 2001*. 2001.

[8] M. Bellare et al. “Standard Security Does Not Imply Security against Selective-Opening”. In: *Advances in Cryptology – EUROCRYPT 2012*. Berlin, Heidelberg, 2012.

[9] F. Benhamouda et al. “Can a Public Blockchain Keep a Secret?” In: *Theory of Cryptography*. Cham, 2020.

[10] M. Blum et al. “Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)”. In: *20th ACM STOC*. 1988.

[11] F. Böhl et al. “On Definitions of Selective Opening Security”. In: *PKC 2012*. 2012.

[12] A. Boldyreva. “Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme”. In: *PKC 2003*. 2003.

[13] D. Boneh et al. “Chosen Ciphertext Secure Public Key Threshold Encryption Without Random Oracles”. In: *CT-RSA 2006*. 2006.

[14] D. Boneh et al. “Short Signatures from the Weil Pairing”. In: *ASIACRYPT 2001*. 2001.

[15] D. Boneh et al. “Threshold Cryptosystems from Threshold Fully Homomorphic Encryption”. In: *CRYPTO 2018, Part I*. 2018.

[16] M. Campanelli et al. *Encryption to the Future: A Paradigm for Sending Secret Messages to Future (Anonymous) Committees*. Cryptology ePrint Archive, Report 2021/1423. 2021.

[17] R. Canetti et al. “Adaptive Security for Threshold Cryptosystems”. In: *CRYPTO’99*. 1999.

[18] J. F. Canny and S. Sorkin. “Practical Large-Scale Distributed Key Generation”. In: *EUROCRYPT 2004*. 2004.

[19] A. R. Choudhuri et al. *Fluid MPC: Secure Multiparty Computation with Dynamic Participants*. Cryptology ePrint Archive, Report 2020/754. 2020.

[20] R. Cohen and Y. Lindell. “Fairness Versus Guaranteed Output Delivery in Secure Multiparty Computation”. In: *Journal of Cryptology* 4 (2017).

[21] A. De Santis et al. “How to Share a Function Securely”. In: *26th ACM STOC*. 1994.

[22] Y. Desmedt and Y. Frankel. “Threshold Cryptosystems”. In: *CRYPTO’89*. 1990.

[23] J. Devevey et al. “Non-Interactive CCA2-Secure Threshold Cryptosystems: Achieving Adaptive Security in the Standard Model Without Pairings”. In: (2021).

[24] C. Dwork et al. “Magic Functions”. In: *40th FOCS*. 1999.

[25] Y. Frankel et al. “Optimal-resilience proactive public-key cryptosystems”. In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. 1997.

[26] Y. Frankel. “A Practical Protocol for Large Group Oriented Networks”. In: *EUROCRYPT’89*. 1990.

[27] Y. Frankel et al. “Adaptively-Secure Distributed Public-Key Systems”. In: *Algorithms - ESA’ 99*. Berlin, Heidelberg, 1999.

[28] R. Gennaro et al. “Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”. In: *EUROCRYPT’99*. 1999.

[29] C. Gentry et al. *Random-index PIR and Applications*. Cryptology ePrint Archive, Report 2020/1248. 2020.

[30] C. Gentry et al. *YOSO: You Only Speak Once / Secure MPC with Stateless Ephemeral Roles*. Cryptology ePrint Archive, Report 2021/210. 2021.

[31] V. Goyal et al. *Storing and Retrieving Secrets on a Blockchain*. Cryptology ePrint Archive, Report 2020/504. 2020.

[32] J. Groth. *Non-interactive distributed key generation and key resharing*. Cryptology ePrint Archive, Report 2021/339. 2021.

[33] K. Gurkan et al. *Aggregatable Distributed Key Generation*. Cryptology ePrint Archive, Report 2021/005. 2021.

[34] T. Hanke et al. "DFINITY Technology Overview Series, Consensus System". In: *CoRR* (2018). arXiv: 1805.04548.

[35] C. Hazay et al. "Selective Opening Security for Receivers". In: *ASIACRYPT 2015, Part I*. 2015.

[36] A. Herzberg et al. "Proactive Public Key and Signature Systems". In: *ACM CCS 97*. 1997.

[37] S. Jarecki and A. Lysyanskaya. "Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures". In: *EUROCRYPT 2000*. 2000.

[38] A. Kate et al. *Distributed Key Generation in the Wild*. Cryptology ePrint Archive, Report 2012/377. 2012.

[39] E. Kokoris-Kogias et al. "CALYPSO: Private Data Management for Decentralized Ledgers". In: *Proc. VLDB Endow.* 4 (2020). ISSN: 2150-8097.

[40] B. Libert and M. Yung. "Adaptively Secure Non-interactive Threshold Cryptosystems". In: *ICALP 2011, Part II*. 2011.

[41] B. Libert and M. Yung. "Non-interactive CCA-Secure Threshold Cryptosystems with Adaptive Security: New Framework and Constructions". In: *TCC 2012*. 2012.

[42] B. Libert et al. "Fully Distributed Non-Interactive Adaptively-Secure Threshold Signature Scheme with Short Shares : Efficiency Considerations and Implementation ?" In: 2019.

[43] X.-J. Lin and L. Sun. *Efficient CCA-secure Threshold Public-Key Encryption Scheme*. Cryptology ePrint Archive, Report 2013/749. 2013.

[44] S. K. D. Maram et al. "CHURP: Dynamic-Committee Proactive Secret Sharing". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. London, United Kingdom, 2019.

[45] R. Ostrovsky and M. Yung. "How to Withstand Mobile Virus Attacks (Extended Abstract)". In: *10th ACM PODC*. 1991.

[46] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO'91*. 1992.

[47] A. Shamir. "How to Share a Secret". In: *Communications of the Association for Computing Machinery* 11 (1979).

[48] V. Shoup and R. Gennaro. "Securing Threshold Cryptosystems against Chosen Ciphertext Attack". In: *EUROCRYPT'98*. 1998.

[49] N. Shrestha et al. *Synchronous Distributed Key Generation without Broadcasts*. Cryptology ePrint Archive, Report 2021/1635. 2021.

[50] H. Wee. "Threshold and Revocation Cryptosystems via Extractable Hash Proofs". In: *EUROCRYPT 2011*. 2011.

Supplementary Material

A Additional Related Work and Preliminaries

A.1 Additional Related Work

Threshold Cryptography in Blockchains Numerous works have considered the use of threshold cryptographic primitives in the context of blockchain (e.g., [39, 4, 34]). The works of Maram et al. [44] and Goyal et al. [31] both present dynamic proactive secret-sharing (DPSS) constructions for blockchain networks under an honest majority assumption. Goyal et al. then proceed to define the notion of extractable witness encryption on blockchains and show an instantiation based on their DPSS scheme. Benhamouda et al. [9] extend the notion of DPSS to an evolving-committee proactive secret sharing scheme that does not require the honest majority assumption. Kokoris-Kogias et al. [39] present an auditable data-management solution that is based on blockchain and threshold encryption. However, their work lacks a formal security analysis of the presented solution and focuses mostly on a static committee of secret key shareholders. Finally, Groth [32] presents a non-interactive distributed key generation protocol together with a refresh procedure that allows refreshing the secret key shares to a new committee.

A.2 Further Notions of Secret Sharing

Proactive Secret Sharing Proactive secret sharing schemes (PSS) further extend robust secret sharing by providing an additional procedure, which allows to refresh the secret shares. More concretely, a PSS scheme proceeds in epochs, i.e., time intervals which are delimited by periodical executions of share refresh procedures. The refresh procedure guarantees that shares from different epochs cannot be combined in order to retrieve the original secret. The adversary model for PSS schemes considers mobile adversaries that can corrupt and uncorrupt parties, but never more than $t-1$ per epoch. PSS schemes must fulfill the secrecy and (robust) reconstruction properties as mentioned above.

Dynamic Proactive Secret Sharing Similar to PSS schemes, dynamic proactive secret sharing schemes (DPSS) proceed in epochs with the difference that the committee of shareholders changes in each epoch, i.e., the refresh procedure is executed between two different (but not necessarily disjoint) committees. DPSS schemes must fulfill the same properties as PSS schemes.

A.3 Combined Encryption Scheme from [9]

The security of the Σ_{ECPSS} scheme of Benhamouda et al. [9] relies on the fact that the combined public key encryption scheme CPKE, consisting of the anonymous and ephemeral schemes APKE and PKE, is RIND-SO secure. We recall here the construction of this combined scheme as used in [9].

Let $\mathsf{APKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be the anonymous public key encryption scheme and $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be the ephemeral public key encryption scheme as used in [9]. The combined encryption scheme CPKE consists of a tuple $\mathsf{CPKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ which are defined as follows:

$\mathsf{CPKE.KeyGen}(1^\lambda)$: This is the key generation of the APKE scheme, i.e., $(sk, pk) \leftarrow \mathsf{APKE.KeyGen}(1^\lambda)$.
$\mathsf{CPKE.Enc}(pk, m)$:
- Execute $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
- Encrypt esk under pk, i.e., $c_{\mathsf{APKE}} := \mathsf{APKE.Enc}(pk, \mathsf{esk})$
- Encrypt m under epk, i.e., $c_{\mathsf{PKE}} := \mathsf{PKE.Enc}(\mathsf{epk}, m)$
- Output $c := (\mathsf{epk}, c_{\mathsf{APKE}}, c_{\mathsf{PKE}})$

$\mathsf{CPKE.Dec}(sk, c)$: Parse c as $(\mathsf{epk}, c_{\mathsf{APKE}}, c_{\mathsf{PKE}})$ and:
- Decrypt c_{APKE} as $\mathsf{esk} := \mathsf{APKE.Dec}(sk, c_{\mathsf{APKE}})$
- Check if esk is a valid secret key corresponding to epk. If it is not, abort.
- Decrypt c_{PKE} as $m := \mathsf{PKE.Dec}(\mathsf{esk}, c_{\mathsf{PKE}})$
- Output m

B Proof of Theorem 1

In this section, we provide a proof of Theorem 1. We do so by first proving Lemma 1 and then Lemma 2.

B.1 Proof of Lemma 1

Proof. In order to prove Lemma 1, we have to show that the correctness properties 1.-3. hold. The correctness proof for properties 1. and 3. proceeds in a similar manner as for the DKG protocol in [28]. We briefly recall the proof here.

First, we note that all parties in U compute the same set $Qual$ during an execution of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$. This is because (1) each party in U can verify the NIZK proofs $\{\pi_i\}_{i\in[n]}$ that are output by parties $P_i \in C$ and, due to the completeness property of the NIZK proof system, identify valid tuples and (2) the fact that there exists an order on the tuples. Therefore it holds that all honest parties in U compute the same set $Qual$ consisting of $t+1$ valid tuples.

1. Note that if it holds that $k \in Qual$, then party $P_k \in C$ must have shared $a_{k,0}$ correctly to committee C'. Therefore, each party $P_j{}' \in C'$ receives secret shares $s_{k,j} \in \mathbb{Z}_q$ for all $k \in Qual$ and subsequently computes its secret key share as $sk'_j = \sum_{k\in Qual} s_{k,j}$. Further, from Shamir's secret sharing we know that it must hold for any set S with $|S| \geq t+1$ of correct secret shares that $a_{k,0} = \sum_{j\in S} l_j \cdot s_{k,j}$. From this, it follows that

$$sk = \sum_{k\in Qual} a_{k,0} = \sum_{k\in Qual}\left(\sum_{j\in S} l_j \cdot s_{k,j}\right) = \sum_{j\in S} l_j \cdot \left(\sum_{k\in Qual} s_{k,j}\right) = \sum_{j\in S} l_j \cdot sk'_j.$$

   The correctness for secret key shares sk''_j of parties $P_j{}'' \in C''$ follows directly from the above and from the handover correctness [9] of $\mathsf{G{-}Handover}$.
2. In order to show that correctness property 2. is satisfied, we have to show that all parties $P_j \in U$ know the same public key $pk = g^{sk} = g^{\sum_{k\in Qual} a_{k,0}}$ after an execution of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$. If $k \in PK$, then $P_k \in C'$ must have broadcast the group element $g^{sk'_k}$ alongside a valid NIZKs proof π'_k (which is possible to produce and verify due to the completeness property of the NIZK proof). All parties $P_j \in U$ then compute the public key as $pk = \prod_{k\in PK} g^{sk'^{l_k}_k} = g^{sk}$.
3. Since the secret key is computed as $sk = \sum_{k\in Qual} a_{k,0}$ and $a_{k,0}$ is chosen uniformly at random from $\mathbb{Z}_q$, it holds that sk is uniformly distributed in $\mathbb{Z}_q$. Since sk is uniformly distributed in $\mathbb{Z}_q$, so is $pk = g^{sk} \in \mathbb{G}$.

B.2 Proof of Lemma 2

In order to prove Lemma 2, we first state and prove the following lemma:

Lemma 6. *Let $\Pi_{\mathsf{LS-DKG}}$ be the large-scale distributed key generation protocol from Sec. 4 instantiated with a (λ, n, t, p)-secure instantiation of Σ_{ECPSS}. Then there exists no fully mobile adversary $\mathcal{A}$ with corruption power p who can corrupt more than t parties in either of C, C' or C'' with more than negligible probability in λ.*

Proof. We prove this lemma by reduction to the secrecy property of the Σ_{ECPSS} scheme. More precisely, we show that if there exists an adversary $\mathcal{A}$ who can corrupt more than t parties in either of C, C' or C'' with non-negligible probability, then we can construct a fully mobile adversary $\mathcal{B}$ with corruption power p who uses $\mathcal{A}$ to break the secrecy property of Σ_{ECPSS}. In fact, we distinguish the following three cases: (1) $\mathcal{A}$ corrupts more than t parties in C, (2) $\mathcal{A}$ corrupts more than t parties in C' and (3) $\mathcal{A}$ corrupts more than t parties in C''. We then show that in each of these cases we can write a reduction to the secrecy property of Σ_{ECPSS}.

Recall that the secrecy property of Σ_{ECPSS} states that upon choosing two secrets $x \in \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$ and subsequently interacting with the Σ_{ECPSS} scheme, the adversary $\mathcal{B}$ cannot distinguish whether x or y has been shared in Σ_{ECPSS}.

– **Case 1:** $\mathcal{A}$ corrupts more than t parties in C
The reduction in this case works as follows: Before the execution of $\Pi_{\mathsf{LS-DKG}}$ begins, $\mathcal{B}$ chooses two secrets $x \in \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$, one of which is then shared in Σ_{ECPSS}. $\mathcal{B}$ then sets the universe of parties in $\Pi_{\mathsf{LS-DKG}}$ to the same universe as in Σ_{ECPSS} and simulates the behavior of all honest parties in this universe to $\mathcal{A}$. The main idea of the reduction is that $\mathcal{B}$ sets committee C in $\Pi_{\mathsf{LS-DKG}}$ to the same committee of parties that is selected in Σ_{ECPSS}. Therefore, when the procedure $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ is executed in $\Pi_{\mathsf{LS-DKG}}$ to select committee C, $\mathcal{B}$ executes the same procedure in Σ_{ECPSS} and relays the outputs of honest parties in Σ_{ECPSS} to adversary $\mathcal{A}$ and the outputs of $\mathcal{A}$ to Σ_{ECPSS}. This ensures that the same committee is selected in Σ_{ECPSS} and $\Pi_{\mathsf{LS-DKG}}$.
Apart from the simulation of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure, $\mathcal{B}$ executes the $\Pi_{\mathsf{LS-DKG}}$ correctly for all honest parties in C, i.e., it follows the protocol instructions. Note, however, that $\mathcal{B}$ does not know the identities of uncorrupted parties in C. Therefore, $\mathcal{B}$ follows the protocol instructions of $\Pi_{\mathsf{LS-DKG}}$ for all honest parties in C without knowing the identities of these parties. Said differently, $\mathcal{B}$ prepares the internal states of honest parties in C without knowing the identities of these parties. Upon $\mathcal{A}$ corrupting a party P_i, $\mathcal{B}$ corrupts the corresponding party in Σ_{ECPSS} and hence learns whether P_i is part of C. If it is, $\mathcal{B}$ returns one of the internal states that it had previously prepared for honest parties in C along with the ephemeral secret key esk_i[10]. Otherwise, if P_i is not in C, $\mathcal{B}$ forwards the internal state of P_i to $\mathcal{A}$. Clearly, if $\mathcal{A}$ is able to corrupt more than t parties in committee C during the $\Pi_{\mathsf{LS-DKG}}$ execution, then $\mathcal{B}$ is able to corrupt more than t parties in the Σ_{ECPSS} scheme and can consequently break the secrecy of Σ_{ECPSS}. Hence, the probability of $\mathcal{B}$ corrupting more than t parties in Σ_{ECPSS} is equal to the probability of $\mathcal{A}$ corrupting more then t parties in C. Therefore, $\mathcal{A}$ corrupts more than t parties in C with at most negligible probability in λ.

– **Case 2:** $\mathcal{A}$ corrupts more than t parties in C'
The reduction in this case works in a similar way as the reduction in the previous case. The only difference is that committee C is now selected independently of Σ_{ECPSS} and instead committee C' is set to the same committee as in Σ_{ECPSS}.

– **Case 3:** $\mathcal{A}$ corrupts more than t parties in C''
The reduction in this case works as the reduction for Case 2 with the only difference that committee C'' is set to the same committee as in Σ_{ECPSS}.

We note that during our proof we do not use the NIZK crs of the underlying ECPSS scheme. Instead, as prescribed by the protocol description of $\Pi_{\mathsf{LS-DKG}}$, $\mathcal{B}$ generates a crs during the $\Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}$ procedure. We therefore do not use the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Setup}$ procedure in black-box. However, since the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure does not make use of the crs, we can still employ it in black-box in our protocol.

With Lemma 6 in place, we can now prove Lemma 2.

Proof. We describe a simulator $\mathcal{S}$ which on input a public key $pk = g^x \in \mathbb{G}$ where $x \in \mathbb{Z}_q$ simulates an execution of $\Pi_{\mathsf{LS-DKG}}$ to an efficient fully mobile adversary $\mathcal{A}$ such that the output distribution of $\mathcal{S}$ is computationally indistinguishable from $\mathcal{A}$'s view of an execution of the real protocol which ends with pk as its output public key. In the following, we first describe the behavior of simulator $\mathcal{S}$ on input pk and subsequently we show that the output distribution produced by $\mathcal{S}$ is computationally indistinguishable to $\mathcal{A}$ from the output distribution of a real protocol execution of $\Pi_{\mathsf{LS-DKG}}$.

During the $\Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}(1^\lambda)$ procedure, $\mathcal{S}$ runs $(\tilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK}.\mathsf{Setup}'(1^\lambda)$ instead of $\mathsf{crs} \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$. This allows $\mathcal{S}$ to obtain a trapdoor τ for the NIZK proof system. Afterwards the execution of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ begins during which the adversary $\mathcal{A}$ can corrupt honest parties at any time.

For all honest parties, $\mathcal{S}$ follows the protocol instructions of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ until step 6. As such, it correctly executes the protocol for all honest parties in committee C. Let $H' \subseteq C'$ and $B' \subset C'$ denote the sets of honest and corrupted parties in committee C' respectively. Note that $\mathcal{S}$ knows the correct internal

[10] To be exact, $\mathcal{B}$ returns the prepared internal state which is specific to the $\Pi_{\mathsf{LS-DKG}}$ protocol together with any other internal secrets that party P_i might hold, e.g., secret information for the self-selection functionality.

states of all parties ${P_j}' \in H'$[11], in particular the values sk'_j. Further, note that due to Lemma 6 there is an honest majority in each of C, C' and C'' through which $\mathcal{S}$ can learn the values sk'_k for all ${P_k}' \in B'$. Therefore, $\mathcal{S}$ can learn the elements S'_j for all parties ${P_j}' \in C'$.

In step 6, $\mathcal{S}$ chooses $t - |B'|$ parties from H' and assigns them to a new set SH' (i.e., $SH' \cap H' = \emptyset$). For all parties in SH', $\mathcal{S}$ follows the protocol instructions of step 6 while for parties ${P_j}' \in H'$, $\mathcal{S}$ sets

$$\tilde{S}'_j = pk^{l_{j,0}} \cdot \prod_{i \in B' \cup SH'} S_i'^{l_{j,i}} \tag{2}$$

where $l_{j,i}$ are the appropriate lagrange coefficients. Note that this allows any set $T \subset \{\{\tilde{S}'_j\}_{j \in H'} \cup \{S'_i\}_{i \in B' \cup SH'}\}$ with $|T| = t+1$ to reconstruct pk via interpolation in the exponent.

In step 7, $\mathcal{S}$ then uses the trapdoor τ as generated during the $\mathsf{NIZK.Setup}'$ procedure, to generate simulated NIZK proofs $\tilde{\pi}'_j$ that prove correctness of the elements S'_j.

During the simulation of steps 6 and 7, $\mathcal{S}$ handles corruptions as follows:

- Upon $\mathcal{A}$ corrupting a party ${P_j}' \in SH'$, $\mathcal{S}$ sends the internal state of ${P_j}'$ to $\mathcal{A}$ and sets $SH' = SH' \setminus \{{P_j}'\}$ and $B' = B' \cup \{{P_j}'\}$.
- Upon $\mathcal{A}$ corrupting a party ${P_i}' \in H'$, $\mathcal{S}$ sends the original internal state which includes $S'_i = g^{sk'_i}$ and π'_i (instead of the simulated values $\tilde{S}'_i$ and $\tilde{\pi}'_i$ as computed in Eq. (2)) to $\mathcal{A}$ and chooses a party ${P_j}' \leftarrow_\$ SH'$. It then sets $H' = H' \setminus \{{P_i}'\}$, $SH' = SH' \setminus \{{P_j}'\}$, $H' = H' \cup \{{P_j}'\}$ and $B' = B' \cup \{{P_i}'\}$.

If $\mathcal{A}$ corrupts a party in H' during the simulation of steps 6 or 7, $\mathcal{S}$ executes the simulation of the respective step again for the updated sets B', SH' and H'.

The simulator $\mathcal{S}$ executes the rest of the protocol correctly for all honest parties.

We now show that the simulation is computationally indistinguishable to $\mathcal{A}$ from a real protocol execution. Before providing the full formal proof of indistinguishability, we first give a high level overview of why indistinguishability holds. Note that $\mathcal{S}$ only deviates from the protocol instructions during the $\mathsf{NIZK.Setup}$ procedure and during steps 6 and 7 for parties ${P_j}' \in H'$. Due to the zero-knowledge property of the NIZK proof system, it holds that the distributions $\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)\}$ and $\{\tilde{\mathsf{crs}} : (\tilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)\}$ are computationally indistinguishable to $\mathcal{A}$. In step 6, $\mathcal{S}$ replaces the elements $S'_j = g^{sk'_j}$ for all ${P_j}' \in H'$ by elements $\tilde{S}'_j$ computed as in Eq. (2). Note that, due to Lemma 6, there exists at least one honest party $P_i \in Qual$ and therefore all elements S'_j contain at least one uniformly random value $s_{i,j}$ from an honest party in the exponent. Further, due to the soundness property of the NIZK proof system, all parties in $Qual$ behaved honestly during the share distribution phase except with negligible probability. Therefore $\mathcal{A}$ can distinguish the simulated elements $\tilde{S}'_j$ from the real elements S'_j only by breaking the RIND-SO security of the CPKE scheme. Finally, by the soundness and zero-knowledge properties of the NIZK proof system, $\mathcal{A}$ cannot generate a valid NIZK proof for a maliciously computed element S'_k for a party ${P_k}' \in B'$ and $\mathcal{A}$ cannot distinguish the simulated NIZK proofs $\tilde{\pi}'_j$ from the real proofs π_j except with negligible probability.

We now show formally that the simulated execution of $\Pi_{\mathsf{LS-DKG}}$ as described above is computationally indistinguishable from a real execution of $\Pi_{\mathsf{LS-DKG}}$ to an efficient fully mobile adversary $\mathcal{A}$.

Proof. We show indistinguishability in a series of games.

Game G_0: This is the real execution of the $\Pi_{\mathsf{LS-DKG}}$ protocol.

Game G_1: In this game we only modify the $\Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}$ procedure. When the common reference string crs of the NIZK proof system is generated, the simulator executes $(\tilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)$ instead of $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. This allows $\mathcal{S}$ to learn a trapdoor τ. Since the distributions $\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)\}$ and $\{\tilde{\mathsf{crs}} : (\tilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)\}$ are indistinguishable to $\mathcal{A}$ except with negligible probability (due to the zero-knowledge property of NIZK), it holds that this game is indistinguishable from the previous game except with negligible probability.

[11] For simplicity, we use the notations ${P_j}' \in H'$ and $j \in H'$ interchangeably throughout this paper.

Game G_2: This game works as the previous game with the difference that $\mathcal{S}$ aborts if any party $P_k \in B$[12] generates a valid proof π_k but there exists at least one party ${P_i}' \in H'$ such that $\mathsf{PKE.Dec}(\mathsf{esk}_i, c_{k,i}) \notin \mathbb{Z}_q$. Note that the simulator can identify this since it knows the ephemeral secret keys of all parties in H'.

Due to the soundness property of the NIZK proof system, the simulator aborts only with negligible probability.

Game G_3: This game is the same as the previous game with only a syntactical change. For committee C', the simulator maintains another list SH' with $|SH'| = t - |B'|$, in addition to the sets H' and B'. At the beginning of the epoch, the simulator randomly assigns $t - |B'|$ parties from H' to SH' and removes these parties from H' (i.e., $H' \cap SH' = \emptyset$).

Game G_4: This game is similar to the previous game, with the difference that $\mathcal{S}$ handles corruptions as follows:

- Upon $\mathcal{A}$ corrupting a party ${P_j}' \in SH'$, $\mathcal{S}$ sends the internal state of ${P_j}'$ to $\mathcal{A}$ and sets $SH' = SH' \setminus \{{P_j}'\}$ and $B' = B' \cup \{{P_j}'\}$.
- Upon $\mathcal{A}$ corrupting a party ${P_i}' \in H'$, $\mathcal{S}$ sends the internal state to $\mathcal{A}$ and chooses a party ${P_j}' \leftarrow_\$ SH'$. It then sets $H' = H' \setminus \{{P_i}'\}$, $SH' = SH' \setminus \{{P_j}'\}$, $H' = H' \cup \{{P_j}'\}$ and $B' = B' \cup \{{P_i}'\}$.

The changes in this game are only syntactical.

Game G_5: This game works as the previous game with the following difference. For each party ${P_j}' \in H'$, the game computes a simulated NIZK proof $\tilde{\pi}'_{j,\mathsf{Handover}}$ (i.e., without using the secret key share sk'_j) using the trapdoor τ and algorithm S (cf. Def. 8).

Due to the zero-knowledge property of the NIZK proof system, the simulated proof $\pi_{j,\tilde{\mathsf{Handover}}}{}'$ is indistinguishable from the real proof except with negligible probability.

Game G_6: This game works as the previous game with the following difference. For each party ${P_j}' \in H'$, the game computes a simulated NIZK proof $\tilde{\pi}'_j$ (i.e., without using the secret key share sk'_j) using the trapdoor τ and algorithm S (cf. Def. 8).

Due to the zero-knowledge property of the NIZK proof system, the simulated proof $\tilde{\pi_j}'$ is indistinguishable from the real proof except with negligible probability.

Game G_7: This game works as the previous game with the following difference. The simulator first chooses uniformly at random a secret key $\tilde{sk}$ with the corresponding public key $\tilde{pk} = g^{\tilde{sk}}$. Then for each party ${P_j}' \in H'$, the simulator chooses secret key shares $\tilde{sk}'_j$ conditioned on the secret key shares sk'_k for $k \in B' \cup SH'$, s.t. $(\{\tilde{sk}'_j\}_{j \in H'}, \{sk'_k\}_{k \in B' \cup SH'})$ form a (t, n)-sharing of $\tilde{sk}$. The simulator then replaces the secret key shares sk'_j by $\tilde{sk}'_j$.

Proof. We show that the probability of $\mathcal{A}$ being able to distinguish this game from the previous one is negligible by exhibiting a reduction to the RIND-SO security of the combined public key encryption scheme CPKE. More concretely, we show that if $\mathcal{A}$ is able to distinguish the two games, then we can construct an adversary $\mathcal{B}$ which can break the RIND-SO security of the CPKE scheme. In the beginning of the reduction, $\mathcal{B}$ chooses the following resamplable distribution $\mathcal{D}$: The distribution samples uniformly at random an element $y \leftarrow \mathbb{Z}_q$ and outputs a (t, n)-sharing of y, i.e., it chooses a random degree-t polynomial $F(x) = a_0 + a_1 x + \cdots + a_t x^t \in \mathbb{Z}_q[x]$ with $a_0 = y$ and outputs $(F(1), \cdots, F(n))$. The algorithm $\mathsf{Resamp}_{\mathcal{D}}$ on input a vector of messages $\mathbf{m}_I$ for $|I| \leq t$ samples a uniform random element $z \leftarrow \mathbb{Z}_q$ and outputs a (t, n)-sharing of z conditioned on $\mathbf{m}_I$.

Having chosen $\mathcal{D}$, $\mathcal{B}$ receives n public keys $(pk_1, \cdots, pk_n)$ from its RIND-SO game which $\mathcal{B}$ embeds in the execution of the $\Pi_{\mathsf{LS\text{-}DKG}}$ protocol on behalf of n honest parties in U. Additionally, $\mathcal{B}$ receives n

[12] By B we denote the set of corrupted parties in C.

ciphertexts $(c_1, \cdots, c_n)$ from its game where each ciphertext c_i consists of (1) an ephemeral public key epk_i, (2) the encryption of the corresponding ephemeral secret key under one of the public keys from the RIND-SO game, (i.e., $c_{i,\mathsf{APKE}} = \mathsf{APKE.Enc}(pk_i, \mathsf{esk}_i)$), and (3) the encryption of a message under the ephemeral public key (i.e., $c_{i,\mathsf{PKE}} = \mathsf{PKE.Enc}(\mathsf{epk}_i, m_i)$). Whenever $\mathcal{A}$ sends a corruption query for any of the public keys $(pk_1, \cdots, pk_n)$, $\mathcal{B}$ forwards the query to its own game. During the nomination of committee C', $\mathcal{B}$ nominates parties by embedding pairs $(\mathsf{epk}_i, \mathsf{APKE.Enc}(pk_i, \mathsf{esk}_i))$ for which $\mathcal{B}$ does not know the secret key sk_i. $\mathcal{B}$ then sends the ciphertexts $c_{j,\mathsf{PKE}}$ for $j \notin I$ to honest parties in C' and sends messages m_i for $i \in I$ to the corrupted parties in C' encrypted under their respective ephemeral public keys.[13] Finally, $\mathcal{B}$ starts the challenge phase in the RIND-SO game through which it receives messages $\tilde{m}_j$ for all $j \notin I$ which are either the correct messages encrypted in $c_{j,\mathsf{PKE}}$ or resampled messages conditioned on the messages of corrupted parties. Note that each honest party ${P_i}' \in H'$ receives n messages from committee C of which one is the message $\tilde{m}_i$. Adversary $\mathcal{B}$, on behalf of party ${P_i}'$, then sums up $t+1$ of those messages to obtain an element sk_i' and broadcasts $g^{sk_i'}$. If $\mathcal{A}$ realizes that sk_i' does not correspond to the sum of the decrypted ciphertexts for party ${P_i}'$, then $\mathcal{B}$ outputs 0, otherwise $\mathcal{B}$ outputs 1.

Note that $\mathcal{B}$ wins the RIND-SO game, whenever $\mathcal{A}$ successfully distinguishes games $\boldsymbol{G_6}$ and $\boldsymbol{G_7}$. Therefore, $\mathcal{A}$ succeeds at most with negligible probability.

Game $\boldsymbol{G_8}$: This game works as the previous game with the following difference. For each party ${P_j}' \in H'$, the simulator computes $\tilde{S}_j' = \tilde{pk}^{l_{j,0}} \cdot \prod_{i \in B' \cup SH'} {S_i'}^{l_{j,i}}$ where $l_{j,i}$ are the appropriate lagrange coefficients.[14] $\mathcal{S}$ then broadcasts $\tilde{S}_j'$, however uses sk_j' for the remaining protocol execution. That is, the ciphertexts $\{c_{j,k}'\}_{k \in [n]}$ and the elements $\tilde{S}_j'$, that are broadcast in the same epoch, are inconsistent.

The indistinguishability argument follows in a similar manner as for game $\boldsymbol{G_7}$, i.e., we can show a reduction to the RIND-SO security of the CPKE scheme. Note that the only difference is the fact that the elements $\tilde{S}_j'$ are broadcast in the same epoch as the ciphertexts from the RIND-SO game and that the resamplable distribution must output either a sharing of sk_j' or of the discrete logarithm of $\tilde{sk}_j'$.

Game $\boldsymbol{G_9}$: This game works as the previous game with the difference that $\mathcal{S}$ aborts if any party ${P_k}' \in B'$ generates a valid NIZK proof $\pi_{j,\mathsf{Handover}}'$, but there exists at least one party ${P_i}'' \in H''$ such that $\mathsf{PKE.Dec}(\mathsf{esk}_i'', c_{k,i}'') \notin \mathbb{Z}_q$ or the decryptions of the ciphertexts $(c_{k,1}', \cdots, c_{k,n}')$ do not form a (t,n)-sharing of sk_k'. The simulator can identify this situation, since it simulates all parties ${P_i}'' \in H''$.

Due to the soundness property of the NIZK proof system, the simulator aborts at most with negligible probability.

Game $\boldsymbol{G_{10}}$: This game works as the previous game with the difference that $\mathcal{S}$ aborts if any party ${P_k}' \in B'$ generates a valid proof π_k' for a statement $x = (\{c_{i,k}\}_{i \in Qual}, \mathsf{epk}_k, S_k')$ such that $x \notin L'$. Note that the simulator can identify this since it knows the secret key share of at least $t+1$ parties and therefore has sufficient information to reconstruct the correct elements S_k' for all parties in B'.

Due to the soundness property of the NIZK proof system, the simulator aborts only with negligible probability.

In the final simulation, the simulator replaces the public key $\tilde{pk}$ by the input public key pk. Note that if a corruption of a party ${P_i}' \in H'$ occurs, then the simulator has to output the correct (not simulated) internal state of ${P_i}'$ and recompute the internal states for the updated sets H'.

[13] Note that $\mathcal{B}$ learns m_i for $i \in I$ because it receives the secret key sk_i from the RIND-SO game, which allows $\mathcal{B}$ to open the ciphertext $c_{i,\mathsf{PKE}}$.

[14] Note that the elements $\tilde{S}_j'$ are the same in this game as in the previous game, i.e., the dlog of $\tilde{S}_j'$ is $\tilde{sk}_j'$.

Simulator Code: Let $\mathcal{S} := (\mathcal{S}_1, \mathcal{S}_2)$ where $\mathcal{S}_1$ simulates $\Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}$ and $\mathcal{S}_2$ simulates $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$. During the simulation of $\Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}$, $\mathcal{S}_1$ executes $\mathsf{NIZK.Setup}'(1^\lambda)$ instead of $\mathsf{NIZK.Setup}(1^\lambda)$ through which it learns a trapdoor τ for the NIZK proof system. Let $H' \subseteq C'$ and $B' \subset C'$ be the sets of honest and corrupted parties in committee C'. Note that there is an honest majority in committees C, C' and C'' due to Lemma 6.
On input a public key pk, trapdoor τ and public parameters $pp^{\mathsf{LS-DKG}}$, $\mathcal{S}_2$ then simulates $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ as follows:

- $\mathcal{S}_2$ follows the protocol instructions for all honest parties until step 6.
- In step 6, $\mathcal{S}$ proceeds as follows:
  - $\mathcal{S}_2$ chooses $t - |B'|$ parties from H' and assigns those parties to a new set SH' s.t. $SH' \cap H' = \emptyset$.
  - For all parties $P_j{}' \in H'$, $\mathcal{S}$ computes $\tilde{S}'_j = pk^{l_{j,0}} \cdot \prod_{i \in B' \cup SH'} S_i'^{l_{j,i}}$ where $l_{j,i}$ are the appropriate lagrange coefficients.
- In step 7, $\mathcal{S}_2$ then uses the trapdoor τ as generated during the $\mathsf{NIZK.Setup}'$ procedure, to generate simulated NIZK proofs $\tilde{\pi}'_j$ that prove correctness of the elements $\tilde{S}'_j$.
- During the simulation of steps 6 and 7, $\mathcal{S}_2$ handles corruptions as follows:
  - Upon $\mathcal{A}$ corrupting a party $P_j{}' \in SH'$, $\mathcal{S}_2$ sends the internal state of $P_j{}'$ to $\mathcal{A}$ and sets $SH' = SH' \setminus \{P_j{}'\}$ and $B' = B' \cup \{P_j{}'\}$.
  - Upon $\mathcal{A}$ corrupting a party $P_i{}' \in H'$, $\mathcal{S}_2$ sends the original internal state which includes $S'_i = g^{sk'_i}$ and π'_i (instead of the simulated values $\tilde{S}'_i$ and $\tilde{\pi}'_i$ as computed in Eq. (2) and in step 7) to $\mathcal{A}$ and chooses a party $P_j{}' \leftarrow_\$ SH'$. It then sets $H' = H' \setminus \{P_i{}'\}$, $SH' = SH' \setminus \{P_j{}'\}$, $H' = H' \cup \{P_j{}'\}$ and $B' = B' \cup \{P_i{}'\}$.

  If $\mathcal{A}$ corrupts a party in H' during the simulation of steps 6 or 7, $\mathcal{S}_2$ executes the simulation of the respective step again for the updated sets B', SH' and H'.
- The simulator $\mathcal{S}_2$ executes the rest of the protocol correctly for all honest parties.

Fig. 1: Simulator code for our large-scale distributed key generation protocol $\Pi_{\mathsf{LS-DKG}}$. The simulator code is divided into two parts $\mathcal{S}_1$ generates the simulated crs and the corresponding trapdoor τ and $\mathcal{S}_2$ simulates the DKG execution on input a public key pk and the trapdoor τ.

C Proof of Theorem 2

In this section, we first provide a proof outline of Lemma 3 before giving a formal proofs of Lemma 4 and Lemma 5.

C.1 Proof outline of Lemma 3

We show that the correctness property holds for $\Pi_{\mathsf{LS-TPKE}}$ for the first epoch. For all subsequent epochs, these properties then follow from the handover correctness property [9] of the $\mathsf{G\text{--}Handover}$ procedure. Let $\lambda \in \mathbb{N}$ be the security parameter and let $pp \leftarrow \Pi_{\mathsf{LS-TPKE}}.\mathsf{Setup}(1^\lambda)$ be the public parameters, where $pp := (pp^{\mathsf{TDH1}}, pp^{\mathsf{LS-DKG}})$.

1. Note that the following holds:

$$\text{for all } (pk, \{vk_i^1\}_{i\in[n]}, \{sk_i^1\}_{i\in[n]}) \leftarrow \Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}(pp),$$

where $vk_i^1 := (\widehat{vk}_i^1, \pi_i^1)$ it holds that

$$(pk, \{\widehat{vk}_i^1\}_{i\in[n]}, \{sk_i^1\}_{i\in[n]}) \in \mathsf{TDH1.TKeyGen}(pp^{\mathsf{TDH1}}).$$

2. Due to the completeness property of the NIZK scheme, it holds that π_i^1 is a valid NIZK proof for the correctness of $\widehat{vk}_i^1$.
3. Correctness in epoch 1 then follows from the above and from the correctness property of the $\mathsf{TDH1}$ scheme.

C.2 Proof of Lemma 4

We prove Lemma 4 via contradiction to the decryption consistency property of the $\mathsf{TDH1}$ scheme. Assume an adversary $\mathcal{B}$ winning game $\mathsf{LSTPKE\text{–}DC}$ with non-negligible probability, i.e., $\mathcal{B}$ outputs in some epoch j with non-negligible probability a ciphertext ct^*, two sets of verification keys $VK = \{vk_1^j, \cdots, vk_{t+1}^j\}$ and $\tilde{VK} = \{\tilde{vk}_1^j, \cdots, \tilde{vk}_{t+1}^j\}$ and two sets of decryption shares $T = \{ct_1^j, \cdots, ct_{t+1}^j\}$ and $\tilde{T} = \{\tilde{ct}_1^j, \cdots, \tilde{ct}_{t+1}^j\}$ such that the conditions of game $\mathsf{LSTPKE\text{–}DC}$ hold.

By the soundness property of the NIZK proof system it must hold with all but negligible probability that $VK = \tilde{VK}$, i.e., the sets contain the same verification keys. In fact, these verification keys must be the correct keys for epoch j. Further, it holds that the distributions $(pk, \{\widehat{vk}_i^1\}_{i\in[n]}, \{sk_i^1\}_{i\in[n]})$ and $(pk', \{vk_i'\}_{i\in[n]}, \{sk_i'\}_{i\in[n]})$ are computationally indistinguishable where

$$(pk, \{vk_i^1\}_{i\in[n]}, \{sk_i^1\}_{i\in[n]}) \leftarrow \Pi_{\mathsf{LS\text{–}TPKE}}.\mathsf{TKeyGen}[U](pp, t, n) \text{ and } vk_i^1 := (\widehat{vk}_i^1, \pi_i^1)$$
$$\text{and}$$
$$(pk', \{vk_i'\}_{i\in[n]}, \{sk_i'\}_{i\in[n]}) \leftarrow \mathsf{TDH1.KeyGen}(pp^{\mathsf{TDH1}}, t, n)$$

Due to the properties of the Σ_{ECPSS} scheme, this holds likewise for all epochs $j > 1$. Finally, since the algorithms $\Pi_{\mathsf{LS\text{–}TPKE}}.\mathsf{TCombine}$ and $\mathsf{TDH1.TCombine}$ are identical and the algorithms $\Pi_{\mathsf{LS\text{–}TPKE}}.\mathsf{TShareVrfy}$ and $\mathsf{TDH1.TShareVrfy}$ differ only in the fact that $\Pi_{\mathsf{LS\text{–}TPKE}}.\mathsf{TShareVrfy}$ includes the additional NIZK proof verification, it must hold that ct^*, T and $\tilde{T}$ satisfy the conditions of the decryption consistency property of $\mathsf{TDH1}$ w.r.t. public key pk, verification keys $(vk_1^j, \cdots, vk_{t+1}^j)$ and secret key shares $(sk_1^j, \cdots, sk_{t+1}^j)$.

Therefore, $\mathcal{B}$ winning the $\mathsf{LSTPKE\text{–}DC}$ game with non-negligible probability directly contradicts the decryption consistency property of the $\mathsf{TDH1}$ scheme.

C.3 Proof of Lemma 5

Proof. We now present the proof of Lemma 5. To this end, we show that if there exists a fully mobile adversary $\mathcal{B}$ that can win the $\mathsf{LSTPKE\text{–}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS\text{–}TPKE}}}$ game with non-negligible advantage, then there also exists a static adversary $\mathcal{A}$ who can win the game $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ (cf. Section 2.3) with non-negligible advantage. More precisely, we show in a series of computationally indistinguishable games that $\mathcal{A}$ can use $\mathcal{B}$'s output bit b' to win its own game.

Game G_0: This is the original $\mathsf{LSTPKE\text{–}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS\text{–}TPKE}}}$ game. In the beginning of this game, the $\Pi_{\mathsf{LS\text{–}TPKE}}.\mathsf{Setup}$ procedure is executed to generate public parameters pp. In each epoch j, the game maintains a list B^j which indicates the set of corrupted parties in the universe U. Additionally, the game maintains two lists H^{C^j} and B^{C^j} which indicate the sets of honest and corrupted parties in committee C^j. The execution of $\Pi_{\mathsf{LS\text{–}TPKE}}.\mathsf{TKeyGen}$ generates a public key pk, selects a committee of secret key shareholders C^1 and outputs a secret key share sk_i^1 to each party $P_i{}^1 \in C^1$.

Note that $\mathcal{B}$ gets access to a corruption oracle, which allows $\mathcal{B}$ to corrupt parties in epoch j at any point in time as long as it holds that $\left\lfloor \frac{|B^j|+1}{|U|} \right\rfloor \leq p$. Further, $\mathcal{B}$ obtains access to a decryption oracle, a refresh oracle and random oracles H_1, H_2, H_3 and H_4.

Game G_1: This game proceeds as the previous game with the difference that it aborts in case in any epoch j it holds that $|B^{C^j}| > t$, i.e., in case in epoch j there are more than t corrupted parties in C^j.

The indistinguishability argument for this game follows from the secrecy property of the Σ_{ECPSS} scheme. That is, if an adversary $\mathcal{B}$ was able to corrupt more than t parties in C^j, then we can construct an adversary $\mathcal{A}'$ who can break the secrecy property of Σ_{ECPSS} by corrupting more than t parties in Σ_{ECPSS}. The reduction works in a similar fashion as the one in Lemma 6, i.e., $\mathcal{A}'$ trying to break the secrecy property of Σ_{ECPSS} can simulate game $\mathsf{LSTPKE\text{–}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS\text{–}TPKE}}}$ to $\mathcal{B}$ by correctly executing all instructions for all honest parties except for the broadcasting of corrupted long-term public keys at the beginning of an epoch and executions of the

$\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure in $\Pi_{\mathsf{LS-TPKE}}$, which are simulated as described in the proof of Lemma 6. The only difference in this game compared to Lemma 6 is that the protocol execution can take polynomially many epochs. Hence, we rely on an induction proof here. We can show that in the first committee C^1, no more than t parties are corrupted (as explained in Lemma 6). Now assuming that in C^{j-1} at most t parties are corrupted, (similar to Lemma 6) it is easy to see that in C^j no more than t parties can be corrupted. Note that as in Lemma 6, $\mathcal{A}'$ executes $\Pi_{\mathsf{LS-TPKE}}$ honestly to $\mathcal{B}$ until the selection of committee C^j.

Hence, we get that $\Pr[\boldsymbol{G_0} = 1] \leq \Pr[\boldsymbol{G_1} = 1] + \nu_1(\lambda)$ where ν_1 is a negligible function in λ.

Game $\boldsymbol{G_2}$: This game is the same as the previous game with only a syntactical change. For each epoch j after the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ the game maintains another list SH^{C^j} with $|SH^{C^j}| = t - |B^{C^j}|$, in addition to the sets H^{C^j} and B^{C^j}. At the beginning of epoch j, the game then randomly assigns $t - |B^{C^j}|$ parties from H^{C^j} to SH^{C^j} and removes these parties from H^{C^j} (i.e., $H^{C^j} \cap SH^{C^j} = \emptyset$).

This change is only syntactical and therefore we get that $\Pr[\boldsymbol{G_1} = 1] = \Pr[\boldsymbol{G_2} = 1]$.

Game $\boldsymbol{G_3}$: This game is similar to the previous game with a modification to the corruption oracle. In each epoch j after the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$, the corruption oracle behaves as follows:

– If $\mathcal{B}$ sends a corruption query for a party ${P_k}^j \in SH^{C^j}$, the game returns the internal state of ${P_k}^j$ and sets $SH^{C^j} = SH^{C^j} \setminus \{{P_k}^j\}$ and $B^{C^j} = B^{C^j} \cup \{{P_k}^j\}$.
– If $\mathcal{B}$ sends a corruption query for a party ${P_i}^j \in H^{C^j}$, the game returns the internal state of ${P_i}^j$ to $\mathcal{B}$ and chooses a party ${P_k}^j \leftarrow_\$ SH^{C^j}$. The game then sets $SH^{C^j} = SH^{C^j} \setminus \{{P_k}^j\}$, $H^{C^j} = H^{C^j} \cup \{{P_k}^j\}$, $H^{C^j} = H^{C^j} \setminus \{{P_i}^j\}$ and $B^{C^j} = B^{C^j} \cup \{{P_i}^j\}$.

For all corruption queries for party ${P_i}^j \in U$, the game sets $B^j = B^j \cup \{{P_i}^j\}$ and $H^j = H^j \setminus \{{P_i}^j\}$.

The change in this game is only syntactical and therefore we have that $\Pr[\boldsymbol{G_2} = 1] = \Pr[\boldsymbol{G_3} = 1]$.

Game $\boldsymbol{G_4}$: This game is similar to the previous game with a modification in the $\Pi_{\mathsf{LS-TPKE}}.\mathsf{Setup}$ procedure. When the common reference string crs of the NIZK proof system is generated, the game executes $(\tilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)$ instead of $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. This allows the game to learn a trapdoor τ. Since the distributions $\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)\}$ and $\{\tilde{\mathsf{crs}} : (\tilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)\}$ are indistinguishable to $\mathcal{B}$ except with negligible probability (due to the zero-knowledge property of NIZK), it holds that $\Pr[\boldsymbol{G_3} = 1] \leq \Pr[\boldsymbol{G_4} = 1] + \nu_2(\lambda)$ where ν_2 is a negligible function in λ.

Game $\boldsymbol{G_5}$: This game works as the previous game with the following difference. For each party ${P_i}^j \in H^{C^j}$, the game computes a simulated NIZK proof π_i^j (i.e., without using the secret key share sk_i^j) using the trapdoor τ and algorithm S (cf. Def. 8) which proves that the verification key $\widehat{vk}_i^j$ has been computed correctly w.r.t. sk_i^j.

Due to the zero-knowledge property of the NIZK proof system, the simulated proof π_i^j is indistinguishable from a real proof except with negligible probability. It holds that $\Pr[\boldsymbol{G_4} = 1] \leq \Pr[\boldsymbol{G_5} = 1] + \nu_3(\lambda)$ where ν_3 is a negligible function in λ.

Game $\boldsymbol{G_6}$: This game proceeds as the previous game with the following modification. After the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$, the game uses the secret key shares sk_i^1 of all ${P_i}^1 \in H^{C^1}$ to reconstruct the secret key sk corresponding to pk. Note that this is possible because $|H^{C^1}| \geq t + 1$. During a decryption oracle query in an epoch j, the game then proceeds as follows: It reconstructs a degree-t polynomial $\tilde{F}^j$ from the secret key shares $\{sk_k^j\}_{k \in B^{C^j} \cup SH^{C^j}}$ and sk, s.t. $\tilde{F}^j(k) = sk_k^j$ and $\tilde{F}^j(0) = sk$. The game then computes secret key shares $\tilde{F}^j(i) = \tilde{sk}_i^j$ for all ${P_i}^j \in H^{C^j}$ and uses $\tilde{sk}_i^j$ to compute decryption shares for ${P_i}^j$.

First, note that in each epoch it holds that $|H^{C^j}| \geq t+1$ and therefore the game has sufficient information to compute the secret key shares $\{sk_k^j\}_{k \in B^{C^j}}$. Second, note that for each epoch j there exists a degree-t polynomial F^j, s.t. $F^j(i) = sk_i^j$ and $F^j(0) = sk$ for all ${P_i}^j \in C^j$. This polynomial is uniquely identified by any $t+1$-size subset of $\{sk, sk_1^j, \cdots, sk_n^j\}$. Therefore, we have that $\tilde{F}^j = F^j$ and $\tilde{sk}_i^j = sk_i^j$ for all ${P_i}^j \in H^{C^j}$.

We therefore get that $\Pr[\boldsymbol{G_5} = 1] = \Pr[\boldsymbol{G_6} = 1]$.

Game $\boldsymbol{G_7}$: This game works as the previous game with the following difference. After the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$, the game computes a simulated NIZK proof $\pi^j_{i,\mathsf{Handover}}$ for each party ${P_i}^j \in H^{C^j}$ (i.e., without using the secret key share sk^j_i) using the trapdoor τ and algorithm S.

Due to the zero-knowledge property of the NIZK proof system, the simulated proof $\pi^j_{i,\mathsf{Handover}}$ is indistinguishable from a real proof except with negligible probability. We therefore get that $\Pr[\boldsymbol{G_6} = 1] \leq \Pr[\boldsymbol{G_7} = 1] + \nu_4(\lambda)$ where ν_4 is a negligible function in λ.

Game $\boldsymbol{G_8}$: This game works as the previous game with the difference that the game aborts if any party ${P_k}^j \in B^{C^j}$ generates a valid NIZK proof $\pi^j_{k,\mathsf{Handover}}$, but there exists at least one party ${P_i}^{j+1} \in H^{C^{j+1}}$ such that $\mathsf{PKE.Dec}(\mathsf{esk}^{j+1}_i, c^j_{k,i}) \notin \mathbb{Z}_q$ or the decryptions of the ciphertexts $(c^j_{k,1}, \cdots, c^j_{k,n})$ do not form a (t,n)-sharing of sk^j_k. The game can identify this situation, since it knows the internal states of all parties ${P_i}^j \in H^{C^j}$ and ${P_i}^{j+1} \in H^{C^{j+1}}$.

Due to the soundness property of the NIZK proof system, it holds that $\Pr[\boldsymbol{G_7} = 1] \leq \Pr[\boldsymbol{G_8} = 1] + \nu_5(\lambda)$ where ν_5 is a negligible function in λ.

Game $\boldsymbol{G_9}$: This game proceeds similarly as the previous game with a modification in the refresh oracle. Instead of executing the $\mathsf{G{-}Handover}$ procedure on input the secret key shares sk^j_i for all parties ${P_i}^j \in H^{C^j}$, the game chooses a uniformly random element $x^j_i \leftarrow \mathbb{Z}_q$ for each ${P_i}^j$ and executes the $\mathsf{G{-}Handover}$ procedure on input secret key shares sk^j_k for all ${P_k}^j \in SH^{C^j}$ and x^j_i for all ${P_i}^j \in H^{C^j}$.

In case $\mathcal{B}$ sends a corruption query for a party ${P_i}^j \in H^{C^j}$ *during* a refresh oracle execution, the game returns the secret key share sk^j_i (instead of x^j_i) and repeats the steps of $\boldsymbol{G_9}$ w.r.t. the updated sets H^{C^j}, SH^{C^j} and B^{C^j}.

The indistinguishability argument of this game to the previous one follows by the RIND-SO security of the CPKE scheme. The reduction works in a similar manner as the reduction in Game $\boldsymbol{G_7}$ of Lemma 2. Therefore, it holds that $\Pr[\boldsymbol{G_8} = 1] \leq \Pr[\boldsymbol{G_9} = 1] + \nu_6(\lambda)$ where ν_6 is a negligible function in λ.

Game $\boldsymbol{G_{10}}$: This game proceeds as the previous game with a modification to the random oracle $\mathsf{H_2}$. The game programs $\mathsf{H_2}$ by maintaining a list H_2 in the following way. Upon a query (c, L, u, w) to $\mathsf{H_2}$ from $\mathcal{B}$, the game first checks if $H_2[c, L, u, w]$ has already been defined. Otherwise, the game chooses uniformly at random $o \leftarrow \mathbb{Z}_q$ and sets $H_2[c, L, u, w] = pk^o$. The game then returns $H_2[c, L, u, w]$.

Note that o is chosen uniformly at random from $\mathbb{Z}_q$ and therefore pk^o is a uniformly random element in $\mathbb{G}$. Hence, we have that $\Pr[\boldsymbol{G_9} = 1] = \Pr[\boldsymbol{G_{10}} = 1]$.

Game $\boldsymbol{G_{11}}$: This game differs from the previous game in the following ways: First, the game maintains a list H_4 which stores query/response pairs for the random oracle $\mathsf{H_4}$. Second, upon $\mathcal{B}$ issuing a decryption query for a ciphertext $ct = (c, L, u, \bar{u}, e, f)$, the game computes a decryption share for party ${P_i}^j \in H^{C^j}$ as follows: it computes $u_i = u^{sk_i}$ as prescribed by the protocol description of TDec and then chooses uniformly at random $e_i \leftarrow \mathbb{Z}_q$ and $f_i \leftarrow \mathbb{Z}_q$. Then the game computes $\hat{u}_i = u^{f_i}/u_i^{e_i}$ and $\hat{h}_i = g^{f_i}/\widehat{vk_i}^{j^{e_i}}$, checks if $H_4[u_i, \hat{u}_i, \hat{h}_i]$ has been set previously and if so, the game aborts. Otherwise the game sets $H_4[u_i, \hat{u}_i, \hat{h}_i] = e_i$. Note that the resulting decryption share $ct_i = (i, u_i, e_i, f_i)$ is valid w.r.t. ciphertext ct and verification key vk^j_i, i.e., it holds that $\mathsf{TShareVrfy}(ct, vk^j_i, ct_i) = 1$.

Recall that upon a corruption query for a party ${P_i}^j \in H^{C^j}$, the game removes a party ${P_k}^j$ at random from the set SH^{C^j} and adds it to H^{C^j}. Therefore, in case such a corruption query occurs after or during a decryption oracle execution, the game first computes the correct decryption shares of ${P_i}^j$, i.e., without the programming of $\mathsf{H_4}$ and then repeats the steps of $\boldsymbol{G_{11}}$ for ${P_k}^j$ (note that this can happen at most t times).

Adversary $\mathcal{B}$ can distinguish this game from the previous one only in the event that game $\boldsymbol{G_{11}}$ aborts. However, since f_i is chosen uniformly at random from $\mathbb{Z}_q$, the elements $\hat{u}_i$ and $\hat{h}_i$ are uniform random elements from $\mathbb{G}$ and hence the abort event happens at most with negligible probability. Therefore, we have that $\Pr[\boldsymbol{G_{10}} = 1] \leq \Pr[\boldsymbol{G_{11}} = 1] + \nu_7(\lambda)$ where ν_7 is a negligible function in λ.

Game $\boldsymbol{G_{12}}$: This game proceeds similarly to the previous game with the following modification in the decryption oracle. For all ${P_i}^j \in H^{C^j}$, the game first computes the verification key $\widehat{vk}_i^j$ as $\widehat{vk}_i^j = pk^{l_{i,0}} \prod_{k \in B^{C^j} \cup SH^{C^j}} \widehat{vk}_k^{l_{i,k}}$. The game then computes decryption shares for all ${P_i}^j$ as follows:

Upon a decryption query from $\mathcal{B}$ on input a ciphertext $ct = (c, L, u, \bar{u}, e, f)$ the game first looks up $\bar{g} = H_2[c, L, u, w]$. Recall that $H_2[c, L, u, w]$ was programmed to be pk^o in game $\boldsymbol{G_1 0}$ and that the game knows o. It then computes $(\bar{u})^{1/o} = (\bar{g})^{r/o} = pk^r$ and $u_i = (\bar{u})^{l_{i,0}/o} \cdot \prod_{k \in B^{C^j} \cup SH^{C^j}} u^{sk_k l_{i,k}}$.

Note that for all parties ${P_k}^j \in SH^{C^j}$, the game computes decryption shares w.r.t. the secret key share sk_k^j according to the protocol description. Upon a corruption query for a party ${P_i}^j \in H^{C^j}$ after or during a decryption oracle execution, the game first computes the decryption shares of ${P_i}^j$ w.r.t. the secret key share sk_i^j and then repeats the steps of $\boldsymbol{G_{12}}$ w.r.t. the updated sets H^{C^j}, SH^{C^j} and B^{C^j} (note that this gain can happen at most t times).

This game is indistinguishable from the previous game except if $o = 0$. In this case, the game cannot correctly compute the element u_i and consequently has to abort. Since t is chosen uniformly at random from $\mathbb{Z}_q$ this event happens only with negligible probability and therefore it holds that $\Pr[\boldsymbol{G_{11}} = 1] \leq \Pr[\boldsymbol{G_{12}} = 1] + \nu_8(\lambda)$ where ν_8 is a negligible function in λ.

Game $\boldsymbol{G_{13}}$: This game proceeds similarly to the previous game $\boldsymbol{G_{12}}$ with the exception that before the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$, the game chooses at random a public key pk, s.t. $(pk, \cdot, \cdot) \in \mathsf{TDH1.KeyGen}$. Then during the $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ procedure, instead of executing $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$, the game executes $\mathcal{S}_2$, i.e., the simulator code of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ (cf. Fig. 1) on input pk, the NIZK trapdoor τ and the public parameters $pp^{\mathsf{LS-DKG}}$. This code simulates $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ in a way such that the output public key is equal to pk.

The indistinguishability argument follows from the secrecy property of the $\Pi_{\mathsf{LS-DKG}}$ scheme. More precisely, we showed that for the $\Pi_{\mathsf{LS-DKG}}$ protocol there exists a simulator which on input a public key pk, trapdoor τ and public parameters $pp^{\mathsf{LS-DKG}}$ can simulate the execution of $\mathsf{LS-DKG.TKeyGen}$ in such a way that the execution is indistinguishable to an efficient fully mobile adversary except with negligible probability and the output public key equals pk. Note that the distribution of $(pk, \tau, pp^{\mathsf{LS-DKG}})$ is identical to the output distribution of $\mathcal{S}_1$ in Fig. 1. Therefore, it holds that $\Pr[\boldsymbol{G_{12}} = 1] \leq \Pr[\boldsymbol{G_{13}} = 1] + \nu_9(\lambda)$ where ν_9 is a negligible function in λ.

By the transition from game $\boldsymbol{G_0}$ to $\boldsymbol{G_{13}}$ we get that

$$\begin{aligned}
\Pr[\mathsf{LSTPKE-CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}(\lambda) = 1] &= \Pr[\boldsymbol{G_0} = 1] \\
&\leq \Pr[\boldsymbol{G_{13}} = 1] + \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) \\
&\qquad + \nu_5(\lambda) + \nu_6(\lambda) + \nu_7(\lambda) + \nu_8(\lambda) \\
&\qquad + \nu_9(\lambda) \\
&\leq \Pr[\boldsymbol{G_{13}} = 1] + \nu(\lambda).
\end{aligned}$$

where $\nu(\lambda) \geq \sum_{i=1}^{9} \nu_i(\lambda)$ is a negligible function in λ.

Having shown that the transition from game $\boldsymbol{G_0}$ to game $\boldsymbol{G_{13}}$ is indistinguishable, it remains to show that there exists an efficient static adversary $\mathcal{A}$ who plays in game $\mathsf{TPKE-CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ and simulates game $\boldsymbol{G_{13}}$ to $\mathcal{B}$. We have to show that $\mathcal{A}$ can then use $\mathcal{B}$ to win its own game. The only differences between game $\boldsymbol{G_{13}}$ and $\mathcal{A}$'s simulation are as follows: (1) In $\mathsf{TPKE-CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$, $\mathcal{A}$ receives a challenge public key pk_C, which it uses instead of the randomly chosen public key in game $\boldsymbol{G_{13}}$, (2) $\mathcal{A}$ forwards all queries to random oracles H_1 and H_3 to the corresponding oracles in the $\mathsf{TPKE-CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ game and (3) $\mathcal{A}$ forwards all queries to the random oracle H_2 that are related to the challenge ciphertext to the corresponding oracle of its own game. Since the challenge public key pk_C is chosen uniformly at random, this change is only syntactical.

Finally, we have to show that $\mathcal{A}$ can use $\mathcal{B}$ to win the $\mathsf{TPKE–CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ game. Note that the encryption procedure is the same in both the $\Pi_{\mathsf{LS–TPKE}}$ and $\mathsf{TDH1}$ scheme. Therefore, upon $\mathcal{A}$ receiving challenge messages m_0 and m_1 and a label L' from $\mathcal{B}$, $\mathcal{A}$ forwards these messages as challenge messages to its own game. Upon receiving the challenge ciphertext $ct' = (c', L', u', \bar{u}', e', f')$, $\mathcal{A}$ forwards it to $\mathcal{B}$. Upon $\mathcal{B}$ outputting a bit b', $\mathcal{A}$ forwards this bit to its own game. Since $\mathcal{A}$ forwards queries to H_2 that are related to ct' to its own oracle, there is a negligible probability that $\mathcal{B}$ has previously (before receiving ct') queried H_2 on input (c', L', u', w'). Hence, there exists a negligible function ν' in λ such that it holds that

$$\begin{aligned}\Pr[\mathsf{LSTPKE–CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS–TPKE}}}(\lambda) = 1] &\leq \Pr[\mathsf{TPKE–CCA}^{\mathcal{A}}_{\mathsf{TDH1}}(\lambda) = 1] + \nu(\lambda)\\ &\leq 1/2 + \nu'(\lambda) + \nu(\lambda).\end{aligned}$$

D The TDH1 Threshold Public Key Encryption Scheme From Shoup and Gennaro [48]

In the following we briefly recall the (t, n)-threshold encryption scheme from Shoup and Gennaro [48], which we denote by $\mathsf{TDH1}$. This scheme has previously been proven secure against chosen ciphertext attacks and static adversaries according to Def. 6.

$\mathsf{Setup}(1^\lambda)$*:* On input a security parameter λ, the setup procedure generates a group $\mathbb{G}$ of prime order q with generator g For simplicity, we assume that both, the messages and labels, are l bits long. In addition, the setup procedure defines the following hash functions:

$$H_1 : \mathbb{G} \to \{0,1\}^l, H_2 : \{0,1\}^l \times \{0,1\}^l \times \mathbb{G} \times \mathbb{G} \to \mathbb{G}, H_3, H_4 : \mathbb{G}^3 \to \mathbb{Z}_q$$

The setup procedure outputs public parameters $pp := (\mathbb{G}, q, g, l, H_1, H_2, H_3, H_4)$.

$\mathsf{KeyGen}(pp, t, n)$*:* On input public parameters pp and integers $t, n \in \mathbb{N}$ s.t. $n \geq 2t+1$, this procedure chooses a random degree-t polynomial $F(x) = a_0 + a_1x + \cdots + a_tx^t \in \mathbb{Z}_q[x]$ and sets $sk_i = F(i)$ and $vk_i = g^{sk_i}$. The procedure outputs $pk = g^{sk}$, where $sk = F(0)$, and all $\{vk_i\}_{i\in[n]}$ to all parties P_i. Additionally, it outputs to each party P_i the secret key share sk_i.

$\mathsf{TEnc}(pk, m, L)$*:* On input a public key pk, a message $m \in \{0,1\}^l$ and label $L \in \{0,1\}^l$ the encryption algorithm works as follows:

1. Choose $r, s \leftarrow_\$ \mathbb{Z}_q$ at random
2. Compute:
   $c = H_1(pk^r) \oplus m, u = g^r, w = g^s, \bar{g} = H_2(c, L, u, w)$
   $\bar{u} = \bar{g}^r, \bar{w} = \bar{g}^s, e = H_3(\bar{g}, \bar{u}, \bar{w}), f = s + re.$

The output is the ciphertext $ct = (c, L, u, \bar{u}, e, f)$.

$\mathsf{TDec}(sk_i, ct, L)$*:* On input a secret key share sk_i, a ciphertext $ct = (c, L, u, \bar{u}, e, f)$ and a label L the decryption algorithm for party P_i does the following:

1. Compute: $w = g^f/u^e, \bar{g} = H_2(c, L, u, w), \bar{w} = \bar{g}^f/\bar{u}^e$.
2. If $e \neq H_3(\bar{g}, \bar{u}, \bar{w})$, output $(i, ?)$
3. If $e = H_3(\bar{g}, \bar{u}, \bar{w})$, choose $s_i \leftarrow_\$ \mathbb{Z}_q$ at random and compute:
   $u_i = u^{sk_i}, \hat{u}_i = u^{s_i}, \hat{h}_i = g^{s_i}, e_i = H_4(u_i, \hat{u}_i, \hat{h}_i), f_i = s_i + sk_ie_i.$

The output is a decryption share $ct_i = (i, u_i, e_i, f_i)$.

$\mathsf{TShareVrfy}(ct, vk_i, ct_i)$*:* On input a ciphertext $ct = (c, L, u, \bar{u}, e, f)$, a verification key vk_i and a decryption share $ct_i = (i, u_i, e_i, f_i)$, the decryption share verification algorithm does the following:

1. Check if $e \neq H_3(\bar{g}, \bar{u}, \bar{w})$ as in the decryption procedure and if so output 1 only if the decryption share is $(i, ?)$ and 0 otherwise.
2. Compute: $\hat{u_i} = u^{f_i}/u_i^{e_i}, \hat{h_i} = g^{f_i}/vk_i^{e_i}$.
3. If $e_i \neq H_4(u_i, \hat{u_i}, \hat{h_i})$, output 1 and 0 otherwise.

$\mathsf{TCombine}(T, ct)$*:* On input a set of valid decryption shares $T := \{ct_i\}_{i \in [t+1]}$ and a ciphertext $ct = (c, L, u, \bar{u}, e, f)$, the share combination algorithm does the following:

1. Check if $e \neq H_3(\bar{g}, \bar{u}, \bar{w})$ as in the decryption procedure and if so, output ?. Otherwise, assume that it holds that all $ct_i \in T$ are of the form $ct_i = (i, u_i, e_i, f_i)$.
2. Compute $m = H_1(\prod_{i=1}^{t+1} u_i^{l_{i,0}}) \oplus c$.

The output is the message m.

In [48], Shoup and Gennaro prove in the random oracle model that TDH1 is CCA-secure against static adversaries corresponding to Def. 6.

E Large-Scale Non-Interactive Threshold Signature Schemes

In this section, we first introduce the notion of large-scale non-interactive threshold signature schemes (LS–TSIG), before we show an instantiation of an LS–TSIG scheme from the threshold signature scheme by Boldyreva [12], which we denote by TH–BLS. We then argue that security of the resulting scheme can be proven in a similar manner as for the $\Pi_{\mathsf{LS-TPKE}}$ scheme from Sec. 5.

E.1 Model

The formal definition of a large-scale non-interactive threshold signature scheme (LS–TSIG) follows the ideas of the definition of LS–TPKE schemes. That is, an LS–TSIG scheme is defined w.r.t. to a universe U of parties and proceeds in epochs at the beginning of which a new committee of secret key shareholders is selected. Similarly to LS–TPKE schemes, the definition of LS–TSIG schemes includes a refresh procedure which allows to transition from one epoch to the next by selecting a new committee and refreshing the secret key shares. Finally, an LS–TSIG scheme must be secure w.r.t. a fully mobile adversary whose corruption power suffices to corrupt an entire committee of secret key shareholders.

We now provide the formal definition of a non-interactive LS–TSIG scheme.

Definition 14. *A large-scale non-interactive (t, n)-threshold signature scheme* (LS–TSIG) *is defined w.r.t. a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple* LS–TSIG = (Setup, TKeyGen, TSign, TShareVrfy, TCombine, Verify, Refresh) *of efficient algorithms and protocols which are defined as follows:*

- $\mathsf{Setup}(1^\lambda)$*: This probabilistic algorithm takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs public parameters pp.*
- $\mathsf{TKeyGen}[U](pp, t, n)$*: This is a protocol involving all parties $P_j \in U$, where each P_j receives as input public parameters pp and two integers $t, n \in \mathbb{N}$ such that $1 \leq t \leq n$. The protocol selects a committee of parties C with $|C| = n$ and outputs to each party $P_j \in U$ a public key pk and to each party $P_i \in C$ a verification key vk_i and a secret key share sk_i.*
- $\mathsf{TSign}(sk_i, m)$*: This algorithm takes as input a secret key share sk_i and a message m and outputs a signature share σ_i.*
- $\mathsf{TShareVrfy}(vk_i, m, \sigma_i)$*: This deterministic algorithm takes as input a verification key vk_i, a message m and a signature share σ_i and it either outputs 1 or 0. If the output is 1, σ_i is called a valid signature share.*

$\mathsf{TCombine}(pk, m, T)$: *This deterministic algorithm takes as input a set of valid signature shares T such that $|T| = t+1$, a public key pk and a message m and it outputs a full signature σ valid under pk.*

$\mathsf{Verify}(pk, m, \sigma)$: *This deterministic algorithm takes as input a public key pk, a message m and a signature σ. It outputs either 1 or 0. If the output is 1, σ is called a valid signature.*

$\mathsf{Refresh}[C_{\langle(sk_1,vk_1,sl_1),\cdots,(sk_n,vk_n,sl_n)\rangle}, U](pp)$: *This is a protocol involving a committee C with $|C| = n$ and the universe of parties U, where each $P_i \in C$ takes as secret input a secret key share sk_i verification key vk_i and signature share list sl_i, and all parties $P_j \in U$ take as input public parameters pp. The protocol selects a committee of parties C' with $|C'| = n$ and outputs to each party $P_i{}' \in C'$ a verification key vk_i' and a secret key share sk_i'. Furthermore, all parties in the universe receive vk_i and sl_i for $i \in [n]$.*

Consistency A $(t,n) - \mathsf{LS{-}TSIG}$ scheme must fulfill the following two consistency properties. For any $\lambda \in \mathbb{N}$, $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(pk, \{vk_i^1\}_{i\in[n]}, \{sk_i^1\}_{i\in[n]}) \leftarrow \mathsf{TKeyGen}[U](pp, t, n)$ with selected committee C^1, for $j > 1$ we define the tuple $(\{vk_i^j\}_{i\in[n]}, \{sk_i^j\}_{i\in[n]})$ recursively as

$$(\{vk_i^j\}_{i\in[n]}, \{sk_i^j\}_{i\in[n]}) \leftarrow \mathsf{Refresh}[C^{j-1}_{\langle(sk_1^{j-1},vk_1^{j-1},\cdot),\cdots,(sk_n^{j-1},vk_n^{j-1},\cdot)\rangle}, U](pp)$$

Recall that during these executions verification keys vk_i^{j-1} and signature share lists sl_i^{j-1} for $i \in [n]$ are broadcasted.

1. For any message m it must hold that:
$$\mathsf{TShareVrfy}(vk_i^j, m, \mathsf{TSign}(sk_i^j, m)) = 1$$
2. For all signature share lists sl_i^{j-1} where $i \in [n]$, each element in the list is computed as $\sigma_{i,k} \leftarrow \mathsf{TSign}(sk_i^{j-1}, m_k)$ for a message m_k. Further, for any set $T_k = \{\sigma_{1,k}^{j-1}, \cdots, \sigma_{t+1,k}^{j-1}\}$, it holds that:
$$\mathsf{Verify}(pk, m_k, \mathsf{TCombine}(pk, m_k, T_k)) = 1$$

Unforgeability In the following, we give the definition of unforgeability under chosen-message attacks for a $(t,n) - \mathsf{LS{-}TSIG}$ scheme considering an efficient fully mobile adversary $\mathcal{A}$ with corruption power $p \cdot |U| > t$. We define the following game $\mathsf{LSSIG{-}UFCMA}^{\mathcal{A}}_{\mathsf{LS-TSIG}}(\lambda)$ which is affected by the same implications of the YOSO model as the *CCA-Security* game in Sec. 5. The game $\mathsf{LSSIG{-}UFCMA}^{\mathcal{A}}_{\mathsf{LS-TSIG}}(\lambda)$ is initialized with a security parameter λ and proceeds as follows:

1. The game executes $\mathsf{Setup}(1^\lambda)$ and obtains public parameters pp, which it forwards to the adversary $\mathcal{A}$. For each epoch $j \geq 0$, the game maintains a set of corrupted parties B^j which is initialized as $B^j := \emptyset$.
2. The adversary $\mathcal{A}$ is given access to the following oracle:
   - **Corruption oracle**: On input an index $i \in [N]$, the game checks if $\left\lfloor \frac{|B^j|+1}{|U|} \right\rfloor \leq p$. If so, $\mathcal{A}$ receives the internal state of party $P_i{}^j$ and the game sets $B^j \leftarrow B^j \cup \{P_i{}^j\}$.
3. The protocol $\mathsf{TKeyGen}[U](pp, t, n)$ is executed. The protocol selects a committee C^1 with $|C^1| = n$ and outputs a public key pk, a set of verification keys $\{vk_1^1, \cdots, vk_n^1\}$ and a set of secret key shares $\{sk_1^1, \cdots, sk_n^1\}$, such that $P_i{}^1 \in C^1$ learns vk_i^1 and sk_i^1.
4. Additionally to the corruption oracle, the adversary $\mathcal{A}$ obtains access to the following two oracles. Let $sl_i^1 := \emptyset$ for $i \in [n]$.
   - **Refresh oracle**: On input a set $NB^j \subseteq B^j$, the protocol $\mathsf{Refresh}[C_{\langle(sk_1^j,vk_1^j,sl_1^j),\cdots,(sk_n^j,vk_n^j,sl_n^j)\rangle}, U](pp)$ is executed and the game sets $B^{j+1} \leftarrow B^j \backslash NB^j$. Additionally, the game initializes the lists $sl_i^{j+1} := \emptyset$ for $i \in [n]$.
   - **Signing oracle**: On input a set of messages M^j, the game computes $\sigma_{i,k}^j \leftarrow \mathsf{TSign}(sk_i^j, m_k^j)$ for $m_k^j \in M^j$ for all parties $P_i{}^j \in C^j \setminus B^j$. Then, the oracle adds all $\sigma_{i,k}^j$ to the list sl_i^j.
5. Eventually, $\mathcal{A}$ outputs a message m' and a signature σ'. $\mathcal{A}$ wins the game if it has never previously queried the signing oracle on message m' and if $\mathsf{Verify}(pk, m', \sigma') = 1$.

Definition 15. *A large-scale non-interactive (t,n)-threshold signature scheme* $\mathsf{LS\text{–}TSIG}$ *with a universe of parties U is (λ,n,t,p)-unforgeable with $p\cdot|U|>t$ if for every efficient fully mobile adversary $\mathcal{A}$ with corruption power p there exists a negligible function ν in the security parameter λ, such that*

$$\Pr[\mathsf{LSSIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{LS\text{–}TSIG}}(\lambda)=1]\leq\nu(\lambda).$$

We define the advantage of $\mathcal{A}$ in game $\mathsf{LSSIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{LS\text{–}TSIG}}$ *as*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{LSSIG\text{–}UFCMA},\mathsf{LS\text{–}TSIG}}(\lambda)=\Pr[\mathsf{LSSIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{LS\text{–}TSIG}}(\lambda)=1].$$

Definition 16 (Robustness). *An* $\mathsf{LS\text{–}TSIG}$ *scheme satisfies (λ,n,t,p)-robustness if there exists no fully mobile PPT adversary $\mathcal{A}$ that wins the following game with non-negligible probability. The game begins with steps 1.-4. as in game* $\mathsf{LSSIG\text{–}UFCMA}$ *with the difference that the adversary is allowed to learn all secret key shares in each epoch j. The adversary then outputs a message m, a set of verification keys $VK=\{vk^j_1,\cdots,vk^j_{t+1}\}$ and a set of signature shares $T=\{\sigma^j_1,\cdots,\sigma^j_{t+1}\}$ and wins the game if the following conditions hold:*

1. *For all $i\in[t+1]$ it holds that* $\mathsf{TShareVrfy}(vk^j_i,m,\sigma^j_i)=1$.
2. $\mathsf{Verify}(pk,m,\mathsf{TCombine}(pk,m,T))=0$.

We call a large-scale non-interactive (t,n)-threshold signature scheme $\mathsf{LS\text{–}TSIG}$ scheme (λ,n,t,p)-secure, if it satisfies the consistency, (λ,n,t,p)-robustness and (λ,n,t,p)-unforgeability properties.

E.2 Construction

We construct a large-scale threshold signature scheme $\Pi_{\mathsf{LS\text{–}TSIG}}=(\mathsf{Setup},\mathsf{TKeyGen},\mathsf{TSign},\mathsf{TShareVrfy},\mathsf{TCombine},\mathsf{Verify},\mathsf{Refresh})$ which is secure against fully mobile adversaries using the large-scale distributed key generation scheme $\Pi_{\mathsf{LS\text{–}DKG}}=(\mathsf{Setup},\mathsf{TKeyGen})$ as described in Sec. 4, the two committee framework of Benhamouda et al. as well as the $\mathsf{G\text{–}Handover}$ procedure as presented in Sec. 3, the threshold signature scheme $\mathsf{TH\text{–}BLS}=(\mathsf{Setup},\mathsf{KeyGen},\mathsf{TSign},\mathsf{TShareVrfy},\mathsf{TCombine},\mathsf{Verify})$ secure against a static adversary as introduced by Boldyreva [12] and a NIZK proof system $\mathsf{NIZK}=(\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ as per Def. 8.

Note that for similar reasons as for our $\Pi_{\mathsf{LS\text{–}DKG}}$ protocol, we cannot use the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure in black-box. Instead we have to use the generalized handover procedure $\mathsf{G\text{–}Handover}$, which internally uses the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure and the combined public key encryption scheme CPKE (cf. Sec. 3). We detail our construction below. We recall the $\mathsf{TH\text{–}BLS}$ scheme in Appx. F.

$\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, execute

$$pp^{\mathsf{TH\text{–}BLS}}\leftarrow\mathsf{TH\text{–}BLS}.\mathsf{Setup}(1^\lambda),\ pp^{\mathsf{LS\text{–}DKG}}\leftarrow\Pi_{\mathsf{LS\text{–}DKG}}.\mathsf{Setup}(1^\lambda).$$
$$\mathsf{crs}\leftarrow\mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$$

Recall that $pp^{\mathsf{LS\text{–}DKG}}$ can be parsed as $pp^{\mathsf{LS\text{–}DKG}}:=(\mathsf{crs}',\mathbb{G},q,g)$. Define $pp^{\mathsf{LS\text{–}DKG}}:=(\mathsf{crs},\mathbb{G},q,g)$ and output public parameters $pp:=(pp^{\mathsf{TH\text{–}BLS}},pp^{\mathsf{LS\text{–}DKG}})$.

$\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{TKeyGen}(pp,t,n)$: On input public parameters pp and two integers $t,n\in\mathbb{N}$ s.t. $n\geq 2t+1$, this protocol parses $pp:=(pp^{\mathsf{TH\text{–}BLS}},pp^{\mathsf{LS\text{–}DKG}})$ and calls $\Pi_{\mathsf{LS\text{–}DKG}}.\mathsf{TKeyGen}(pp^{\mathsf{LS\text{–}DKG}},t,n)$. The protocol selects a committee C^1 and outputs a public key pk to all parties in U and secret key shares sk^1_i to each party ${P_i}^1\in C^1$. Additionally, all ${P_i}^1\in C^1$ compute $\widehat{vk}^1_i:=g^{sk^1_i}$ and a NIZK proof π^1_i that the verification key $\widehat{vk}^1_i$ was computed correctly [15]. ${P_i}^1$ then sets $vk^1_i:=\{\widehat{vk}^1_i,\pi^1_i\}$ and initializes a signature share list $sl^1_i:=\emptyset$.

[15] The language for this proof is the same as the language L' in the $\Pi_{\mathsf{LS\text{–}DKG}}$ protocol.

$\Pi_{\mathsf{LS-TSIG}}.\mathsf{TSign}(sk_i, m)$: This procedure executes $\mathsf{TH{-}BLS.TSign}$ and adds the resulting signature share to the list sl_i.

$\Pi_{\mathsf{LS-TSIG}}.\mathsf{TShareVrfy}(\sigma_i, vk_i^j, m)$: On input a signature share σ_i, a verification key $vk_i^j := \{\widehat{vk}_i^j, \pi_i^j\}$ and a message m, this procedure checks if π_i^j is a valid proof w.r.t. $\widehat{vk}_i^j$ (i.e., it checks if $\widehat{vk}_i^j$ is indeed the correct verification key of party $P_i^j \in C^j$). If this check does not hold, the procedure outputs 0. Otherwise, it outputs $\mathsf{TH{-}BLS.TShareVrfy}(\sigma_i, \widehat{vk}_i^j, m)$.

$\Pi_{\mathsf{LS-TSIG}}.\mathsf{TCombine}(T, ct)$: This procedure executes $\mathsf{TH{-}BLS.TCombine}$.

$\Pi_{\mathsf{LS-TSIG}}.\mathsf{Verify}(pk, m, \sigma)$: This procedure executes $\mathsf{TH{-}BLS.Verify}$.

$\Pi_{\mathsf{LS-TSIG}}.\mathsf{Refresh}[C_{\langle(sk_1^j, vk_1^j, sl_1^j), \cdots, (sk_n^j, vk_n^j, sl_n^j)\rangle}, U](pp)$: This protocol is executed between a committee C^j in epoch j and the universe U, where each $P_i^j \in C^j$ receives as input a secret key share sk_i^j, verification key vk_i^j and signature share list sl_i^j, and each party $P_k \in U$ receives as input $pp := (pp^{\mathsf{TSIG}}, pp^{\mathsf{LS-DKG}})$. The protocol first runs $\mathsf{G{-}Handover}[C^j_{\langle(sk_1^j,(vk_1^j, sl_1^j)), \cdots, (sk_n^j,(vk_n^j, sl_n^j))\rangle}, U](pp)$ which selects a committee C^{j+1} and outputs refreshed secret key shares sk_i^{j+1} to each $P_i^{j+1} \in C^{j+1}$. Furthermore, all parties in the universe receive vk_i and sl_i for $i \in [n]$. Additionally, all $P_i^{j+1} \in C^{j+1}$ compute $\widehat{vk}_i^{j+1} := g^{sk_i^{j+1}}$, generate a NIZK proof π_i^{j+1} that the verification key was computed correctly[16] and set $vk_i^{j+1} := \{\widehat{vk}_i^{j+1}, \pi_i^{j+1}\}$. Finally, all P_i^{j+1} initialize a signature share list $sl_i^{j+1} := \emptyset$.

Theorem 3. *Let $\Pi_{\mathsf{LS-DKG}}$ be the large-scale (t, n)-distributed key generation protocol from Sec. 4, $\mathsf{TH{-}BLS}$ the non-interactive (t, n)-threshold signature scheme as described in Appx. F, Σ_{ECPSS} a (λ, n, t, p)-secure instantiation of the evolving-committee proactive secret sharing scheme as presented in Sec. 3, NIZK a non-interactive zero-knowledge proof system as per Def. 8 and CPKE a RIND-SO secure public key encryption scheme. Then $\Pi_{\mathsf{LS-TSIG}}$ is a (λ, n, t, p)-secure large-scale non-interactive (t, n)-threshold signature scheme.*

In order to prove Theorem 3, we have to show that $\Pi_{\mathsf{LS-TSIG}}$ satisfies consistency, robustness and (λ, n, t, p)-unforgeability. We therefore state the following lemmas.

Lemma 7. *The large-scale non-interactive (t, n)-threshold signature scheme, $\Pi_{\mathsf{LS-TSIG}}$, satisfies consistency.*

Proof. This lemma follows directly from the consistency property of the $\mathsf{TH{-}BLS}$ scheme (cf. Definition 19), the completeness property of the NIZK proof system and from the handover correctness of the $\mathsf{G{-}Handover}$ scheme. A proof outline of this lemma looks similar to the proof outline of Lemma 3 in Appx. C.

Lemma 8. *The large-scale non-interactive (t, n)-threshold signature scheme, $\Pi_{\mathsf{LS-TSIG}}$, satisfies (λ, n, t, p)-robustness.*

Proof. The proof of this lemma is similar to the proof of Lemma 4.

Lemma 9. *The large-scale non-interactive threshold (t, n)-threshold signature scheme $\Pi_{\mathsf{LS-TSIG}}$ is (λ, n, t, p)-unforgeable.*

The proof of Lemma 9 is similar to the proof of Lemma 5 with the difference that we have to provide a reduction to the unforgeability of $\mathsf{TH{-}BLS}$. As part of this reduction we have to show that signing oracle queries from the adversary in game $\mathsf{LSSIG{-}UFCMA}_{\Pi_{\mathsf{LS-TSIG}}}$ can be answered without knowing the correct secret key shares. We show briefly in Appx. F how such signing oracle answers can be simulated for the $\mathsf{TH{-}BLS}$ scheme.

[16] The language for this proof is the same as the language L' in the $\Pi_{\mathsf{LS-DKG}}$ protocol.

E.3 Transformation Framework from TSIG to LS–TSIG

The same reasoning that we presented in Sec. 5 for the transformation framework from TPKE schemes to LS–TPKE schemes applies to TSIG and LS–TSIG schemes as well. In a nutshell, a discrete-log-based non-interactive threshold signature scheme TSIG can be transformed to a large-scale non-interactive threshold signature scheme LS–TSIG if TSIG satisfies the same properties as described in Sec. 5 for TPKE schemes. The only minor difference in the properties is that the simulator in the simulatability property receives as input a public key, n verification keys and t secret key shares and additionally obtains access to a signing oracle which on input a message outputs a valid signature under the input public key. Apart from this, the same argumentation from Sec. 5 can be used here to argue that a scheme TSIG can be transformed to an LS–TSIG scheme.

F The TH–BLS Scheme from Boldyreva [12]

F.1 Background on Digital Signature and Threshold Signature Schemes

Before we recall the TH–BLS scheme, we first recall the basic definitions of digital signature schemes and threshold signature schemes.

Definition 17 (Digital signatures). *A digital signature scheme* SIG *consists of a triple of algorithms* SIG = (KeyGen, Sign, Verify) *defined as:*

$\mathsf{KeyGen}(1^\lambda)$**:** *This probabilistic algorithm takes as input a security parameter λ and outputs a key pair (sk, pk);*

$\mathsf{Sign}(sk, m)$**:** *This probabilistic algorithm takes as input a secret key sk and message m and outputs a signature σ;*

$\mathsf{Verify}(pk, m, \sigma)$**:** *This deterministic algorithm takes as input a public key pk, message m and signature σ and outputs a bit either 1 or 0. If the output is 1, σ is called a valid signature.*

A signature scheme must satisfy that for all messages m it holds that:

$$\Pr\left[\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1 \mid (sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)\right] = 1,$$

where the probability is taken over the randomness of KeyGen *and* Sign*.*

Definition 18 (Unforgeability). *A signature scheme* SIG *is unforgeable if for every PPT adversary $\mathcal{A}$ there exists a negligible function ν in the security parameter λ such that $\Pr[\mathsf{SIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{SIG}}(\lambda) = 1] \leq \nu(\lambda)$, where the experiment $\mathsf{SIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{SIG}}$ is defined as follows:*

1. *The game executes $\mathsf{KeyGen}(1^\lambda)$ and obtains a key pair (sk, pk). It forwards the public key pk to the adversary $\mathcal{A}$.*
2. *$\mathcal{A}$ obtains access to a signing oracle, which on input a message m outputs a signature σ for m under public key pk.*
3. *Eventually, $\mathcal{A}$ outputs a forgery (m^*, σ^*) and wins the game if (1) it holds that $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$ and (2) m^* has never been queried to the signing oracle before.*

Definition 19. *A non-interactive (t, n)-threshold signature scheme* TSIG *consists of a tuple of efficient algorithms* TSIG = (Setup, KeyGen, TSign, TShareVrfy, TCombine, Verify) *which are defined as follows:*

$\mathsf{Setup}(1^\lambda)$*: This probabilistic algorithm takes a security parameter $\lambda \in \mathbb{N}$ as input and output public parameters pp.*

$\mathsf{KeyGen}(pp, t, n)$*: This probabilistic algorithm takes as input public parameters pp and two integers $t, n \in \mathbb{N}$. It outputs a public key pk, a set of verification keys $\{vk_i\}_{i\in[n]}$ and a set of secret key shares $\{sk_i\}_{i\in[n]}$.*

$\mathsf{TSign}(sk_i, m)$*: This probabilistic algorithm takes a secret key share sk_i and a message m as input and outputs a signature share σ_i.*

$\mathsf{TShareVrfy}(\sigma_i, vk_i, m)$: *This deterministic algorithm takes as input a signature share σ_i, a verification key vk_i and a message m and it outputs either 1 or 0. If the output is 1, σ_i is called a valid signature share.*

$\mathsf{TCombine}(pk, m, T)$: *This deterministic algorithm takes as input a public key pk; a message m and a set of valid signature shares T for m under pk such that $|T| = t + 1$ and it outputs a signature σ.*

$\mathsf{Verify}(pk, m, \sigma)$: *This deterministic algorithm takes as input a public key pk, message m and signature σ and outputs a bit either 1 or 0. If the output is 1, σ is called a valid signature.*

Consistency A $(t, n) - \mathsf{TSIG}$ scheme must fulfill the following two consistency properties. Let $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(pk, \{vk_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow_\$ \mathsf{KeyGen}(pp, t, n)$.

1. For any message m it must hold that
$$\mathsf{TShareVrfy}(\mathsf{TSign}(sk_i, m), vk_i, m) = 1.$$
2. For any message m and any set $T = \{\sigma_1, \cdots, \sigma_{t+1}\}$ of valid signature shares $\sigma_i \leftarrow \mathsf{TSign}(sk_i, m)$ with sk_i being t distinct secret key shares, it must hold that
$$\mathsf{Verify}(pk, m, \mathsf{TCombine}(pk, m, T)) = 1.$$

Unforgeability We recall the definition of unforgeability for a $(t, n) - \mathsf{TSIG}$ scheme with static corruptions. Consider a PPT adversary $\mathcal{A}$ playing in the following game $\mathsf{SIG{-}UFCMA}^{\mathcal{A}}_{\mathsf{TSIG}}$ which receives as input a security parameter λ:

1. The adversary outputs a set $B \subset \{1, \cdots, n\}$ with $|B| = t$ to indicate its corruption choice. Let $H := \{1, \cdots, n\} \setminus B$.
2. The game computes $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and sets $(pk, \{vk_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(pp, t, n)$. It sends pp, pk and $\{vk_i\}_{i\in[n]}$ as well as $\{sk_j\}_{j\in B}$ to the adversary.
3. The adversary $\mathcal{A}$ is allowed to adaptively query a signing oracle, i.e., on input (m, i) with $i \in H$, the signing oracle outputs $\mathsf{TSign}(sk_i, m)$.
4. Eventually, $\mathcal{A}$ outputs a forgery (m^*, σ^*) and wins the game if (1) it holds that $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$ and (2) $\mathcal{A}$ has not previously made a signing query on message m^*.

Definition 20. *A non-interactive (t, n)-threshold signature scheme* TSIG *is unforgeable if for every PPT adversary $\mathcal{A}$ there exists a negligible function ν in the security parameter λ, such that* $\Pr[\mathsf{SIG{-}UFCMA}^{\mathcal{A}}_{\mathsf{TSIG}}(\lambda) = 1] \le \nu(\lambda)$.

In this work, we define robustness of a TSIG scheme as follows.

Definition 21 (Robustness). *A* TSIG *scheme satisfies robustness if for all PPT adversary $\mathcal{A}$ the following holds:*

$$\Pr\left[\begin{array}{l|l} \forall i \in [t+1] : \mathsf{TShareVrfy}(vk_i, m, \sigma_i) = 1 & \mathsf{K} := (pk, \{vk_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(pp, t, n) \\ \wedge \mathsf{Verify}(pk, m, \mathsf{TCombine}(pk, m, T)) = 0 & (m, T) \leftarrow \mathcal{A}(\mathsf{K}) \ \textit{s.t.}, \\ & T := \{\sigma_1, \cdots, \sigma_{t+1}\} \end{array}\right] \le \nu(\lambda).$$

where ν is a negligible function in the security parameter λ.

F.2 The BLS and TH–BLS Schemes

In the following we present the non-interactive threshold signature scheme from [12], which we denote by TH–BLS. We then give a proof sketch for a reduction of TH–BLS to the single party signature scheme BLS as introduced in [14]. This proof sketch demonstrates how signing oracle responses for TH–BLS can be simulated without knowing the corresponding secret key shares. As mentioned in Appendix E, this is crucial for the proof of Lemma 9. Both BLS and TH–BLS operate over so-called Gap Diffie-Hellman (GDH) groups in which the computational Diffie-Hellman (CDH) problem is hard, whereas the decisional Diffie-Hellman (DDH) problem is easy. We briefly recall the notions of CDH, DDH and GDH in the following.

Computational/Decisional Diffie-Hellman Problem and GDH Groups Let $\mathbb{G}$ be a cyclic group of prime order q and with generator g. Let a, b, c be elements chosen uniformly at random from $\mathbb{Z}_q$.

Computational Diffie-Hellman (CDH) Given (g, g^a, g^b), the CDH problem is to compute g^{ab}.
Decisional Diffie-Hellman (DDH) Given (g, g^a, g^b, g^c), the DDH problem is to decide whether $c = ab$.

We now recall the definition of GDH groups as given in [12].

Definition 22 (Gap Diffie-Hellman Group). *A group $\mathbb{G}$ of prime order q is called a Gap Diffie-Hellman (GDH) group if there exists an efficient algorithm* $\mathsf{V\text{-}DDH}()$ *which solves the DDH problem in $\mathbb{G}$ and there is no polynomial-time (in $|q|$) algorithm which solves the CDH problem in $\mathbb{G}$.*

We now first recall the BLS scheme as presented in [14], before presenting its threshold variant TH–BLS as presented in [12].

F.3 The BLS scheme

$\mathsf{KeyGen}(1^\lambda)$*:* On input a security parameter λ, this procedure generates a GDH group $\mathbb{G}$ of prime order q with generator g. In addition, the procedure defines the hash function $H : \{0,1\}^* \to \mathbb{G}^*$ and sets $pp = (\mathbb{G}, q, g, H)$. Further, it picks a secret key $sk \leftarrow_\$ \mathbb{Z}_q$ and computes the corresponding public key $pk \leftarrow g^x$ and sets $sk' = (sk, pp)$, $pk' = (pk, pp)$ It outputs (sk', pk').

$\mathsf{Sign}(sk', m)$*:* On input a secret key sk' and a message m, this procedure parses $sk' := (sk, pp)$ and computes a signature $\sigma = H(m)^{sk}$ and outputs σ.

$\mathsf{Verify}(pk', m, \sigma)$*:* On input a public key pk', a message m and a signature σ, this procedure parses $pk' := (pk, pp)$ and checks if $\mathsf{V\text{-}DDH}(g, pk, H(m), \sigma) = 1$. If so, this procedure outputs 1 and 0 otherwise.

The authors of [14] show that the BLS scheme is unforgeable as per Definition 18.

F.4 The TH–BLS scheme

We now recall the threshold variant of the BLS scheme, which we denote by TH–BLS.

$\mathsf{Setup}(1^\lambda)$*:* On input a security parameter λ, the setup procedure generates a GDH group $\mathbb{G}$ of prime order q with generator g. In addition, the setup procedure defines the hash function $H : \{0,1\}^* \to \mathbb{G}^*$.

The setup procedure outputs public parameters $pp := (\mathbb{G}, p, g, H)$.

$\mathsf{KeyGen}(pp, t, n)$*:* On input public parameters pp and integers $t, n \in \mathbb{N}$ s.t. $n \geq 2t+1$, this procedure chooses a random degree-t polynomial $F(x) = a_0 + a_1 x + \cdots + a_t x^t \in \mathbb{Z}_q[x]$ and sets $sk_i = F(i)$ and $vk_i = g^{sk_i}$. The procedure outputs $pk = g^{sk}$, where $sk = F(0)$, and all $\{vk_i\}_{i \in [n]}$ to all parties P_i. Additionally, it outputs to each party P_i the secret key share sk_i.

$\mathsf{TSign}(sk_i, m)$*:* On input a secret key share sk_i and a message m, this algorithm outputs $\sigma_i = H(m)^{sk_i}$.

$\mathsf{TShareVrfy}(vk_i, m, \sigma_i)$*:* On input a verification key vk_i, a message m and a signature share σ_i, this algorithm outputs $\mathsf{V\text{-}DDH}(g, vk_i, H(m), \sigma_i)$.

$\mathsf{TCombine}(pk, m, T)$: On input a public key pk, a message m and a set of valid signature shares $T \subset \{\sigma_1, \cdots, \sigma_n\}$ with $|T| = t+1$, this algorithm computes $\sigma = \prod_{\sigma_i \in T}(\sigma_i^{l_i})$ and outputs σ.

Theorem 4. *If the* BLS *scheme as presented in Section F.3 is unforgeable as per Definition 18, then the* TH–BLS *scheme is unforgeable as per Definition 20 in the random oracle model.*

Proof sketch. We provide a proof sketch for Theorem 4 by exhibiting a simulator $\mathcal{S} := (\mathcal{S}_1, \mathcal{S}_2)$ who uses an adversary $\mathcal{A}$ playing in game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{TH\text{–}BLS}}$ to win its own game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{S}}_{\mathsf{BLS}}$. $\mathcal{S}$ receives a public key pk from its game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{S}}_{\mathsf{BLS}}$ as well as access to a signing and a random oracle and it has to simulate game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{TH\text{–}BLS}}$ to $\mathcal{A}$. On a high level, the simulation works as follows:

W.l.o.g. let $\mathcal{A}$ corrupt parties $(P_1, \cdots, P_t)$. Upon $\mathcal{S}$ receiving pk, $\mathcal{S}$ calls its subprocedure $\mathcal{S}_1$ on input $(pk, \{vk_i, sk_i\}_{i\in[t]})$ for $sk_i \leftarrow_{\$} \mathbb{Z}_q$ and $vk_i = g^{sk_i}$ with $i \in [t]$. $\mathcal{S}_1$ computes $vk_j = pk^{l_{j,0}} \prod_{i=1}^{t} vk_i^{l_{j,i}}$ for $t+1 \leq j \leq n$. Note that for any subset $T \subset \{vk_1, \cdots, vk_n\}$ with $|T| = t+1$ it holds that $pk = \prod_{vk_i \in T} vk_i^{l_i}$ and therefore any T uniquely identifies pk. $\mathcal{S}$ then executes $\mathcal{S}_2$ on input $(pk, \{vk_j\}_{j\in[n]}, \{sk_i\}_{i\in[t]})$, which simulates game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{TH\text{–}BLS}}$ as follows:

In the beginning of the game, $\mathcal{S}_2$ sends $(pk, \{vk_j\}_{j\in[n]}, \{sk_i\}_{i\in[t]})$ to the adversary. Upon $\mathcal{A}$ issuing a random oracle query on input m, $\mathcal{S}_2$ forwards the query to its own random oracle and receives a group element $H(m) \in \mathbb{G}$. Upon $\mathcal{A}$ issuing a signing query on input (m, i), $\mathcal{S}_2$ issues a signing query on message m to its signing oracle and receives a signature $\sigma = H(m)^{sk}$. $\mathcal{S}_2$ then computes the signature share σ_i as $\sigma_i = \sigma^{l_{i,0}} \prod_{j=1}^{t} H(m)^{sk_j l_{i,j}}$. Finally, upon $\mathcal{A}$ outputting a forgery (m^*, σ^*), $\mathcal{S}_2$ can simply forward the forgery to game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{S}}_{\mathsf{BLS}}$. $\mathcal{S}$ wins its game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{S}}_{\mathsf{BLS}}$ whenever $\mathcal{A}$ wins game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{TH\text{–}BLS}}$ due to the following reason. If (m^*, σ^*) is a valid forgery in game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{A}}_{\mathsf{TH\text{–}BLS}}$ (i.e., $\mathcal{A}$ has never previously queried the signing oracle on input m^*), then $\mathcal{S}_2$ has never previously queried its own signing oracle on message m^* and hence (m^*, σ^*) constitutes a valid forgery in game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{S}}_{\mathsf{BLS}}$.