Large-Scale Non-Interactive Threshold Cryptosystems in the YOSO Model

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 parties in a committee to stay anonymous – also referred to as the YOSO (You Only Speak Once) model – until they start to interact within the protocol.

In this work, we initiate the study of large-scale threshold cryptography in the YOSO model of communication. We formalize and present novel protocols for distributed key generation, threshold encryption, and signature schemes that guarantee security in large-scale environments. A key challenge in our analysis is that we cannot use the secret sharing protocol of Benhamouda et al. as a black-box to construct our schemes, and instead we require a more generalized version, which may be of independent interest. Finally, we show how our protocols can be concretely instantiated in the YOSO model, and discuss interesting applications of our schemes.

1 Introduction

In a threshold cryptosystem [19, 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 a key pair – so-called *distributed key generation (DKG)* [38, 39] – that enables the parties to securely generate a shared secret key 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 secret key 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., [9, 34]), and threshold public key encryption for distributed decryption of ciphertexts (e.g., [14, 41]). While these threshold cryptosystems can be constructed from general purpose multi-party computation (MPC), such solutions are typically not desirable due to their lack of efficiency. Instead, there has recently been a broad interest in constructing concretely efficient threshold cryptographic schemes as evidenced by the extensive number of works in recent years (e.g., [20, 13, 33]).

Traditionally, threshold cryptographic schemes have been considered in a setting where the adversary is restricted to corrupt at most $t < n/2$ parties. This upper bound is required to achieve guaranteed output delivery [17], i.e., the adversary cannot stall the system even when behaving in an arbitrary malicious way. When we move to a large-scale setting with many users (e.g., as prominently considered in the blockchain setting), several new challenges arise. In particular, we aim to achieve two conflicting goals. First, we require a solution that is *scalable* when the number of users in the universe – denoted by N – increases. Ideally, we want a protocol with efficiency independent of the size of the universe, as the size of the universe is typically very large. Our second goal is that the adversary can corrupt a linear fraction of all parties N, and in particular its corruption power shall increase when the number of parties in the universe increases. We call such a strong adversary a *fully mobile adversary*.

To address these two challenges, the recent work of Benhamouda et al. [7] 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 [7]. Dynamic proactive secret sharing is a 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* [36]. To ensure security in this setting, dynamic proactive secret sharing schemes deploy a so-called *handover* protocol, where the 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, 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, via a so-called *role assignment* mechanism, 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 if $\gg n$ parties can be corrupted per epoch. We recall the definition of ECPSS schemes in Sec. 2.5, and we provide a more detailed description of the ECPSS scheme of Benhamouda et al. in Sec. 3.

The original work of Benhamouda et al. [7] considers only the question of how to store a secret in a large-scale environment. This work has recently been extended by Gentry et al. [23] in the so-called YOSO model of computation ("You Only Speak Once"), which presents a general framework for committee-style secure computation with anonymity. This framework crucially relies on a role assignment UC ideal functionality, which is responsible for the selection of anonymous committees and for assigning concrete roles to each party in the selected committee. Unfortunately, to date this functionality has not been instantiated[1] and therefore, so far the solution of Gentry et al. is not suited for concretely instantiating cryptographic schemes in the YOSO model. Moreover, the most efficient solution of [23] relies on a trusted party to generate initial secret key shares and the corresponding public key of a threshold encryption scheme (for a detailed discussion on [23] see Sec. 1.2). Kolby et al. [32] later showed how to overcome this assumption and the necessity for any trusted setup, however, at the cost of efficiency.

1.1 Our Contribution

In this work, we address the above drawbacks of Gentry et al.'s general purpose YOSO MPC by formalizing and constructing concrete schemes for distributed key generation as well as threshold encryption and signatures that can be instantiated in the YOSO model. In particular, we show that the role assignment functionality of Benhamouda et al.'s ECPSS construction suffices to prove our schemes secure in the YOSO model. While our constructions build on earlier works on efficient threshold encryption and signature schemes, the resulting YOSO schemes turn out to be rather complex, and proving their security becomes quite involved. One main reason for this is that we cannot use Benhamouda et al.'s ECPSS scheme (which is one of our main building-blocks) in a black-box fashion. Let us now discuss our contribution in more detail.

Large-Scale Distributed Key Generation. Distributed key generation (DKG) [38, 39] is a fundamental primitive of threshold cryptography. It allows a set of parties to securely generate a shared secret key (and

[1]In particular, Gentry et al. do not claim that the role assignment of Benhamouda et al. can UC-realize their functionality.

corresponding public key) in a distributed way, thereby avoiding a trusted setup. Our first contribution is to introduce and formalize the concept of discrete-log-based large-scale distributed key generation in the YOSO model, and to show a concrete instantiation. We follow the idea of Benhamouda et al. [7] 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 challenges arise from the fact that we cannot use the ECPSS construction from Benhamouda et al. entirely as black-box (see below for further details) and that we must prove that an adversary indeed corrupts at most a minority of protocol participants. We do the latter by exhibiting a reduction to the security of the ECPSS construction. In more detail, we use the role assignment mechanism of the ECPSS construction in our protocol and show that if a majority of parties get corrupted in our DKG, then we can construct an adversary that corrupts a majority of parties in the ECPSS scheme, thereby breaking its security. Finally, we deal with the fact that distributed key generation protocols are typically highly interactive which poses a problem in our full security setting where parties can speak only once to preserve their anonymity. Gentry et al. [23] mentioned the construction of a DKG protocol in the YOSO model (albeit for the RSA setting) as an open problem and we make the first step towards addressing this problem with our construction.

Large-Scale Threshold PKE and Signatures. We next consider the setting of large-scale non-interactive threshold public key encryption and signature schemes in the YOSO model 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 [41] can be transformed to the large-scale setting with security against fully mobile adversaries. In a similar way we show in Appendix F how to transform the statically-secure threshold signature scheme from Boldyreva [9] to the setting of large-scale security. A key building-block for this transformation is again the ECPSS construction from Benhamouda et al. For the security proof of our schemes, we have to deal with the general issue of providing a consistent view to an adversary who can adaptively corrupt parties. More importantly, however, we face similar issues as in our DKG protocol, i.e., we must prove that at most a minority of parties are corrupted by the adversary and we must deal with the challenges that arise from the non-black-box use of the ECPSS construction (see below).

Generalization, Instantiation and Applications. 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. The conditions that such schemes have to satisfy are given in Sec. 5.2. Finally, we discuss how all of our schemes can be concretely instantiated in the YOSO model and we describe 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.

Non-Black-Box Use of Benhamouda et al.'s ECPSS Construction. As a first attempt, one may try to build large-scale threshold cryptographic schemes by leveraging ECPSS in a black-box fashion to select anonymous committees and to re-share the secret key of the threshold scheme to future committees. However, in the YOSO model, each party can speak at most once per epoch, restricting committee members to communicate only *during* the state handover to the next-epoch committee. Therefore, any communication that is specific to the threshold scheme (e.g., the publication of decryption or signature shares) must happen during the handover. Even worse, all communication required to initially generate the secret key in the DKG protocol must occur during a state handover as well. Such communication is not needed (and thus not supported) in the ECPSS construction itself. To tackle this issue, we provide a *generalized handover* procedure which allows parties to broadcast auxiliary information while handing over their internal state.

An additional challenge arises from the fact that Benhamouda et al. only consider an honest dealer that shares one secret to an initial committee. For our distributed key generation protocol, however, we require n parties to each share a secret random value, where some of the dealers may be malicious. This issue alone prevents us from using ECPSS as black-box, yet it leads to further issues when trying to make a direct reduction from the security of ECPSS. Concretely, we cannot simulate non-interactive zero knowledge (NIZK) proofs in the reduction, since our protocol requires NIZK proofs w.r.t. all n initially shared secrets, whereas the NIZK proofs in Benhamouda et al.'s solution are w.r.t. only one secret.

1.2 Comparison to YOSO MPC

Gentry et al. [23] introduced the YOSO model and proposed two protocols to generically achieve YOSO MPC in the information theoretical and computational setting, respectively. Both of these protocols rely on an ideal UC role assignment functionality that is responsible for selecting anonymous committees with honest majority. To the best of our knowledge, no protocol exists to date which can UC-realize this functionality and therefore, both proposed protocols cannot currently be instantiated.[2] We note that, as part of their computationally secure protocol, Gentry et al. use a threshold version of the Paillier encryption scheme [37] and refresh the secret key shares using the techniques of Benhamouda et al.'s ECPSS handover procedure, which is similar to the general idea of our large-scale threshold encryption construction. However, in contrast to the work of Gentry et al., we formalize threshold encryption in the YOSO model as a standalone primitive and formally prove the security of our construction w.r.t. the concrete role assignment mechanism of Benhamouda et al. We believe that this formalization/instantiation can be helpful in future works in order to instantiate generic YOSO MPC protocols.

In terms of communication efficiency, we note that the main efficiency bottleneck of protocols in the YOSO model appears during the handover of values from one committee to another and when selecting a new committee. In order to compile a circuit into a YOSO protocol, the solution of Gentry et al. requires at least one committee per layer of multiplication gates, regardless of the protocol choice (information theoretic or computational). In addition, the computationally secure protocol requires a trusted party to generate the initial encryption key of the committee members. This assumption can be removed by executing a DKG protocol using either the information theoretic MPC protocol or the solution of Kolby et al. [32], which are both setup-free YOSO MPC protocols. However, both of these protocols require linear communication rounds in the depth of the DKG circuit. In contrast, we provide a dlog-based DKG for our threshold schemes, which requires only 3 communication rounds.

1.3 Related Work

As mentioned in the Introduction, in the last years there has been extensive work on threshold cryptography [28, 2, 42, 27, 20, 13, 33]. We focus here on schemes that are important to the YOSO setting. We provide a discussion on additional related work in Appendix A.

Extensions to Benhamouda et al.'s ECPSS Construction. The original ECPSS solution of Benhamouda et al. [7] suffers from two drawbacks, namely (1) it provides security only against fully mobile adversaries with corruption power of roughly 25%, while requiring huge committee sizes, and (2) it relies on the use of rather complex non-interactive zero knowledge (NIZK) proofs. Two recent works of Gentry et al. [25, 24] improve on these drawbacks. In [25], Gentry et al. introduce a novel committee selection mechanism which allows for a more powerful adversary that can corrupt less than 50% of all parties. Furthermore, their solution allows to decrease the required committee size significantly.

The issue that Benhamouda et al.'s solution relies on complex NIZK proofs was addressed in [24], where the authors propose an efficient non-interactive publicly verifiable secret sharing scheme that might be used to efficiently instantiate the handover procedure of Benhamouda et al.'s solution.

Other YOSO MPC protocols. In a similar spirit to the YOSO MPC paper, a recent work by Choudhuri et al. [16] 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 [16] do not analyze their solution

[2]We note that Campanelli et al. [12] propose protocols for role assignments in the YOSO model, however, none of these protocols have been shown to UC-realize the role assignment functionality of [23].

w.r.t. a fully mobile adversary who has sufficient corruption power to potentially corrupt a majority of the universe's participants. As mentioned previously, Kolby et al. [32] propose a setup-free YOSO MPC protocol, which however requires communication rounds linear in the depth of the computed circuit. Campanelli et al. [12] and Cascudo et al. [15] analyze the notion of *encryption to the future* which generally allows parties to send messages to an anonymous and yet to be selected committee. Finally, Acharya et al. [3] propose an MPC model, which deviates from the YOSO model only by considering some parties (so-called MPC input providers) that speak more than once. While this is a compelling model, the focus of our work is to construct instantiable threshold cryptosystems in the YOSO model.

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. We defer the definitions of anonymous public key encryption, non-interactive threshold public key encryption (TPKE) and non-interactive zero-knowledge (NIZK) proofs of knowledge to Appendix A.

2.1 Notation

We write $s \leftarrow_\$ H$ to denote that a variable s is sampled uniformly at random from a set H. For integers i, j with $i < j$, we use $[i]$ to denote the set $\{1, \cdots, i\}$ and $[i; j]$ to denote the set $\{i, \cdots, j\}$. We write $s \leftarrow_\$ A(x)$ for a probabilistic algorithm A that on input x outputs s, and $s \leftarrow B(x, r)$ to denote running a deterministic algorithm B that produces output s on input x and randomness r. $s \in A(x)$ denotes that s is in the set of possible outputs of A on input x. For a polynomial F, we write $deg(F) = t$ to denote that F is of degree t.

For a set of parties C and a protocol Π, we write $\Pi[C_{\langle x_1, \cdots, x_{|C|} \rangle}]$ to denote that Π is jointly executed by all parties $P_i \in C$ on secret inputs x_i for $i \in [|C|]$. Furthermore, we write $\Pi[C_{\langle x_1, \cdots, x_{|C|} \rangle}](y)$ 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 the notation $\Pi[C_{\langle x_1, \cdots, x_{|C|} \rangle}, U](y)$ denotes 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

Following Benhamouda et al. [7], we assume that parties have access to an authenticated broadcast channel and a public key infrastructure (PKI). 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. Furthermore, we consider synchronous communication where messages broadcast in some round i are received by all other parties in round $i + \delta$ where δ is a fixed upper bound. 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. Communication via a blockchain realizes such a communication model.

We consider a fully mobile adversary which can monitor the broadcast channel and for messages sent in round i by honest parties, we allow the adversary to receive those messages in the same round i, i.e., without delay δ. The adversary can corrupt parties at any point in time. Corrupted parties are controlled by the adversary and can deviate arbitrarily from the protocol execution. We assume that the adversary corrupts a fraction p of *all* parties in the system. The fraction p is called the adversary's *corruption power*. Notice that we allow the adversary to "uncorrupt" parites, and consider a party as no longer controlled by the adversary when the uncorrupted party broadcasts a new public key to the PKI. 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 protocols with proactive security.

```
$\mathsf{Rind\text{-}SO}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda)$
00 $\mathcal{I} := \emptyset$, $(\mathbf{sk}, \mathbf{pk}) \leftarrow (\mathsf{Gen}(1^\lambda))_{i \in [n]}$
01 $(\mathcal{D}, \mathsf{Resamp}_{\mathcal{D}}, \mathrm{state}_1) \leftarrow \mathcal{A}(\mathbf{pk})$
02 $\mathbf{m_0} := (m_i)_{i \in [n]} \leftarrow \mathcal{D}$
03 $\mathbf{c} := (c_i)_{i \in [n]} \leftarrow (\mathsf{Enc}(pk_i, m_i))_{i \in [n]}$
04 $\mathrm{state}_2 \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{C}}}(\mathbf{c}, \mathrm{state}_1)$
05 $\mathbf{m_1} \leftarrow \mathsf{Resamp}_{\mathcal{D}}(\mathbf{m}_{\mathcal{I}})$
06 $b \leftarrow \{0,1\}$, $b' \leftarrow \mathcal{A}(\mathbf{m}_b, \mathrm{state}_2)$
07 Return $b = b'$

Oracle $\mathcal{O}_{\mathrm{C}}(i)$
08 If $i \notin [n]$: Return $\perp$
09 $\mathcal{I} := \mathcal{I} \cup \{i\}$
10 Return $sk_i$
```

Fig. 1: The RIND-SO security game from Hazay et al. [29] adjusted to allow adaptive corruptions of keys.

2.3 Public Key Encryption

Throughout this work, we use different notions of public key encryption (PKE). Recall that a public key encryption scheme PKE consists of three algorithms KeyGen, Enc and Dec, where (1) KeyGen on input a security parameter λ outputs a public key pk and a secret key sk; (2) Enc on input a public key pk and a message m outputs a ciphertext ct; and (3) Dec on input a secret key sk and a ciphertext ct 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 that given a secret key sk generated by KeyGen, it is possible to derive the corresponding public key pk via a 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. [29] for PKE with adaptive corruptions. The RIND-SO notion considers a security game between a challenger and an adversary in which the challenger first samples a set of key pairs $(pk_i, sk_i)_{i \in [n]}$ and then sends all pk_i to the adversary. The adversary then chooses a distribution $\mathcal{D}$ and receives a vector of n ciphertexts, each encrypting a message sampled from $\mathcal{D}$ under pk_i. The adversary can then adaptively choose to open some of the ciphertexts by receiving the corresponding secret keys sk_i. Finally, for the remaining unopened ciphertexts, the adversary either receives the correct plaintext, or a randomly sampled messages from $\mathcal{D}$[3] and the adversary has to decide whether it received the correct or random messages. We note that Hazay et al. [29] introduced this notion for a semi-adaptive adversary which opens all public keys in one shot, but mentioned that their results hold for adaptive corruptions as well.

Definition 1 (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 $\mathcal{I} \subseteq [n]$ 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 2 (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 in Fig. 1. The advantage of the adversary $\mathcal{A}$ is defined as $|\Pr[\mathsf{Rind\text{-}SO}^{\mathcal{A}}_{\mathsf{PKE}}(\lambda) = 1] - \frac{1}{2}|$. A* PKE *scheme is RIND-SO secure, if every PPT $\mathcal{A}$ only has negligible advantage (in λ) in winning the above game.*

[3]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.

2.4 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 size $\geq t$ to reconstruct s. Essentially, for a security parameter $\lambda \in \mathbb{N}$, a (t, n)-secret sharing scheme must fulfill two properties against an adversary $\mathcal{A}$ corrupting 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. $\mathcal{A}$ is only able to distinguish which secret was shared with negligible probability in λ.
2. Reconstruction: The secret can be reconstructed from any set of honest secret shares of size $\geq t$.

Shamir's secret sharing [40] 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 $F(i) = \sum_{j \in S} l_{i,j} s_j$.

2.5 Evolving-Committee Proactive Secret Sharing

Recently, Benhamouda et al. [7] introduced the notion of evolving-committee proactive secret sharing (ECPSS), which is defined w.r.t. a universe of N parties and parameters $t \leq n < N$. ECPSS allows to share a secret to a committee of parties and to periodically exchange the secret shareholders of the committee. It further extends previous secret sharing notions by providing a role assignment 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 [7].

Definition 3 (ECPSS). *An evolving-committee proactive secret sharing scheme with parameters $t \leq n < N$ consists of the following procedures:*

Setup (optional): *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 polynomial $\mathsf{poly}(n, \lambda)$, independent of N.

An ECPSS scheme must fulfill the same secrecy and reconstruction properties as a secret sharing scheme, but w.r.t. a fully mobile adversary with corruption power p s.t. $p \cdot N > t - 1$. 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.

In this section, we recall the scalable ECPSS scheme Σ_{ECPSS} due to Benhamouda et al. [7], which is an important building block of our schemes. The key observation of Benhamouda et al. 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. This prevents targeted attacks from a fully mobile adversary. The scheme proceeds in epochs, where at the beginning of each epoch all N parties in the universe generate a key pair for an anonymous public key encryption scheme. These key pairs are the *long-term keys*, and are broadcasted to the PKI.

In each epoch two committees are selected, a nominating and a holding committee. The latter maintains the shares of the secret while the former selects the members of the holding committee. The nominating committee self-selects, for instance by the use of verifiable random functions. 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.

Before broadcasting, the nominating committee members must erase their internal state as the broadcast reveals their identity to the adversary. All N parties 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 [7] for the full description of the Σ_{ECPSS} scheme.

We will now describe the Setup and Select procedures as well as our generalized handover procedure $\mathsf{G–Handover}$ (which differs only slightly from the original $\mathsf{Handover}$ procedure of Benhamouda et al.) in more detail as these are most relevant for our work.

$\mathsf{Setup}(1^\lambda)$: 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. 13). The procedure outputs public parameters $pp := (\mathsf{crs}, q)$.[4]

During the Select procedure, a nominating committee first self-selects and then chooses the next n-party holding committee. We omit the details on the self-selection and focus on the selection of the holding committee. Let APKE be the anonymous public key encryption scheme and PKE a public key encryption scheme that is used to generate the ephemeral keys

Select procedure:

Let C_{nom} be the nominating committee which selects the next n-party holding committee. Let t_0 be the initial round of the protocol. $P_i \in C_{\mathsf{nom}}$ proceeds as follows:

1. Choose a nominee for the next holding committee $p \in [N]$ with long-term public key pk_p, 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).

In round $t_0 + \delta$, all parties P_j where $j \in [N]$ do:

4. Verify that the broadcasters were indeed in C_{nom}. Otherwise, ignore this tuple.
5. For each tuple (epk_i, c_i) broadcast by a party in C_{nom}, 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.

Intuitively, due to the self-selection and using an anonymous scheme APKE, the adversary does not know the identities of honest members of the holding committee. Indeed [7] show that (using the correct parameters) the adversary cannot corrupt more than $t < n/2$ members of the holding committee except with negligible probability. Since malicious parties can simply refuse to nominate a party in the holding committee, the Select procedure might nominate less than n parties. Yet this is similar to selecting a malicious party to the new holding committee. Hence, for simplicity we assume that the holding committees are of size exactly n.

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

The Handover procedure presented by Benhamouda et al. is tailor-made for the ECPSS construction, i.e., parties in the holding committee re-share their secret share to the next holding committee by encrypting the corresponding shares with the ephemeral public keys received from the Select procedure. Unfortunately, the construction cannot be used off the shelf to distribute additional secret-dependent material. Since this is required in our applications, we define a *generalized* Handover procedure, called G–Handover. More precisely, we write $\mathsf{G\text{–}Handover}[C_{\langle(s_1,aux_1),\cdots,(s_n,aux_n)\rangle}, U](pp)$ to state that the G–Handover procedure is executed between the current holding committee members C and the parties in the universe U where each committee member $P_i \in C$ has secret input s_i and additionally broadcasts auxiliary input aux_i. The only difference to the original Handover procedure of Benhamouda et al. is the additional broadcast of aux_i, which does not affect the correctness of the Handover procedure.

We will later see how protocols can be proven secure when using G–Handover as a building block. In a nutshell, we must prove that the auxiliary values do not leak any information about the secret value that is being passed on to the next committee. We believe that this simple extension of ECPSS is of independent interest as it is required for many applications in which ECPSS can be used.

G–Handover procedure:

Let C be a holding committee such that each party $P_i \in C$ for $i \in [n]$ knows a secret share $s_i \in \mathbb{Z}_q$ and has auxiliary input aux_i. Let t_0 denote the round in which the protocol execution begins. We assume that the Select procedure has been previously executed to select the next-epoch holding committee C' such that each $P_j{}' \in C'$ is associated with an ephemeral public key epk_j.
In round t_0, 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 $s_{i,j} := F_i(j)$ and $c_{i,j} \leftarrow \mathsf{PKE.Enc}(\mathsf{epk}_j, s_{i,j})$ for $j \in [n]$.
2. Let com_i be a commitment to s_i from the Handover procedure of the previous epoch[a]. 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 $j \in [n]$.
3. Choose a new long-term key pair $(sk'_i, pk'_i) \leftarrow \mathsf{APKE.KeyGen}(1^\lambda)$ and erase all secrets from the previous epoch.
4. Broadcast $(pk'_i, \pi_{i,\mathsf{Handover}}, \{c_{i,j}\}_{j\in[n]}, aux_i)$.
5. In round $t_0 + \delta$, for all tuples of the form $(pk'_i, \pi_{i,\mathsf{Handover}}, \{c_{i,j}\}_{j\in[n]})$ where $i \in [n]$, all parties $P'_j \in C'$ do the following:
   – Verify the NIZKs $\pi_{i,\mathsf{Handover}}$ and for the first $t+1$ valid proofs π_i store i in a set $Qual$. Compute $s_{i,j} \leftarrow \mathsf{PKE.Dec}(\mathsf{esk}_j, c_{i,j})$ for all $i \in Qual$ and reconstruct the new secret share s'_j using Lagrange polynomial interpolation.

[a]The com_i in the first epoch is generated by the dealer

We point out that the encryption schemes PKE and APKE are used as a form of hybrid encryption, 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 give a formal specification of this scheme in Appendix B.

4 Large-Scale Distributed Key Generation

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 and each party learns a *share* of the secret key sk. A DKG protocol shall satisfy *correctness*, which means that any subset of $t + 1$ parties can reconstruct sk; and *security*, which states that by corrupting at most t parties the adversary learns no information about sk.

A large-scale (t, n)-distributed key generation protocol (LS–DKG) in the YOSO model differs from the above notion in the sense that it is defined w.r.t. a universe of parties U. From this universe we select a committee of parties C of size n with $n \ll |U|$, which executes the key generation protocol. The key difference to traditional DKG protocols is in the security. Concretely, an 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 $\mathsf{LS\text{–}DKG}$ protocol.

Definition 4. *A large-scale (t, n)-distributed key generation protocol* $\mathsf{LS\text{–}DKG} = (\mathsf{Setup}, \mathsf{TKeyGen})$ *is a protocol run among a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ 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 and secrecy properties of an $\mathsf{LS\text{–}DKG}$ scheme, which follow the standard definitions of correctness and secrecy of discrete-log-based DKG schemes as introduced by Gennaro et al. [22].

Correctness: A (t, n)-$\mathsf{LS\text{–}DKG}$ 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.

Secrecy: A (t, n)-$\mathsf{LS\text{–}DKG}$ scheme is (λ, n, t, p)-*secret* if for any security paremeter $\lambda \in \mathbb{N}$ and for every fully mobile adversary $\mathcal{A}$ with corruption power p s.t. $p \cdot |U| > t$, the following holds: 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 in an execution of the real protocol $\mathsf{LS\text{–}DKG}$ that outputs the public key 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.

4.1 Construction

We are now ready to present our construction of a large-scale distributed key generation ($\mathsf{LS\text{–}DKG}$) protocol. Typically, discrete-log based DKG protocols are executed among a fixed set of parties where the execution proceeds in three phases: (1) share distribution, (2) qualification, and (3) public key reconstruction phase. In the first phase, each party P_i chooses a random value s_i and shares it 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 in the second phase of the protocol, as it allows to identify misbehaving parties and consequently to exclude them from the protocol execution, while the honest parties "qualify" to further execute the protocol. At this point, all parties can reconstruct their respective secret key share by summing up the secret shares each party received from all qualified parties. In the final phase of the protocol, each qualified party P_i opens its commitment to g^{s_i}, which allows all parties to reconstruct the final public key.

In our setting, we need to design a DKG protocol in the YOSO model. To do so, we use the ideas of the ECPSS scheme Σ_{ECPSS} as described in Sec. 3 to keep the identity of the parties anonymous. Our DKG protocol must hence be non-interactive, i.e., any communication must occur only during the state handover from one committee to another. In order to construct such a non-interactive protocol, we follow the ideas of Damgård et al.'s DKG protocol [18], which deviates from the above DKG description in the following two points: (1) it uses plain Shamir secret sharing instead of verifiable secret sharing to share secrets s_i, and (2) it removes the commitment to elements g^{s_i}. Essentially, in their protocol all parties first share their secret s_i, then compute their secret key shares sk_i from all received shares and only then broadcast g^{sk_i} such that all parties can compute the public key via Lagrange interpolation in the exponent. The advantage of this approach is that parties do not first have to send a commitment and later open it, thereby improving communication complexity and allowing for non-interactivity. However, due to the missing verifiability of the secret sharing, a single malicious party can cause the protocol to abort. In our case, we want that not even a minority of t corrupted parties can abort the protocol execution. We therefore add NIZK proofs to achieve public verifiability of the secret sharing, which allows to identify malicious parties. Furthermore, we must extend the protocol such that for each round of communication, a fresh committee is being selected and we must prove that a fully mobile adversary can corrupt at most a minority of parties in each committee.

As mentioned in the Introduction, we cannot employ the Σ_{ECPSS} scheme as black-box. Instead, we use the generalized handover procedure $\mathsf{G\text{–}Handover}$ as described in Sec. 3 to allow parties to broadcast DKG-specific values during the state handover. Further, we extend the sharing procedure of Σ_{ECPSS} in a non-black-box way to let n parties each share a secret to the same committee while proving honest behavior. Note, however, that the role assignment $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ is not affected by any of the issues mentioned in the Introduction and hence, we can use it directly to select anonymous committees. In the following, we provide a more detailed description of our solution.

Suppose that an anonymous committee C has been previously selected via an execution of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure. Our protocol proceeds in four phases and requires a total of three anonymous committees. In the first phase, the *committee selection phase*, a fresh anonymous committee C' is selected. In the *share distribution phase*, 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, P_i broadcasts a NIZK proof that proves correctness of the sharing.

After δ rounds, the *qualification phase* begins, in which each party $P_j{}' \in C'$ verifies the NIZK proofs and stores in a set $Qual$ the identities of the first $t+1$ parties from committee C who sent a valid NIZK proof. The waiting period of δ rounds ensures that all honest parties in C' compute the same set $Qual$. 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 that 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, a new committee C'' is selected and all parties $P_j{}' \in C'$ re-share their secret key shares sk'_j to C'' while broadcasting elements $g^{sk'_j}$ along with a NIZK proof that proves that $g^{sk'_j}$ was computed correctly. This is done by executing the $\mathsf{G\text{–}Handover}$ procedure.

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 $\mathsf{LS\text{–}DKG}$ protocol $\Pi_{\mathsf{LS\text{–}DKG}}$ using the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ and $\mathsf{G\text{–}Handover}$ procedures and a NIZK proof system NIZK.[5]

$\mathsf{Setup}(1^\lambda)$**:** 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 discrete-log problem is hard in $\mathbb{G}$. Output public parameters $pp := (\mathsf{crs}, \mathbb{G}, q, g)$.

$\mathsf{TKeyGen}(pp, t, n)$ **procedure:**

In the following, we denote by PKE the public key encryption scheme for the ephemeral keys of the Σ_{ECPSS} scheme. Recall that PKE together with the anonymous public key encryption scheme APKE form the combined public key encryption scheme CPKE (cf. Sec. 3). We further denote by t_0 the round in which the protocol execution begins.

[5]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.

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

Committee Selection Phase:

1. During the committee selection phase, the procedure $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}\,[U]\,(pp)$ is executed by the universe U to select a new committee C' where $|C| = |C'| = n$. Note that after the execution of $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ for each party $P_j{}' \in C'$ there exists an ephemeral public key epk_j.

Share Distribution Phase:

2. In round $t_0 + \delta$, each party $P_i \in C$ does the following:
   (a) Choose $s_i \leftarrow_\$ \mathbb{Z}_q$ and 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 $a_{i,0} = s_i$. Moreover, compute shares $s_{i,j} := F_i(j)$ and $c_{i,j} \leftarrow \mathsf{PKE.Enc}(\mathsf{epk}_j, s_{i,j}; r_{i,j})$ for $j \in [n]$.
   (b) Compute a NIZK proof π_i for the following language:

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

   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 $s_{i,j}$ in $\mathbb{Z}_q$ such that any size t subset of $\{s_{i,1}, \cdots, s_{i,n}\}$ lies on the t-degree polynomial F_i.
   (c) Erase all secret values, i.e., shares $s_{i,j}$, polynomial F_i and the value s_i.
   (d) Broadcast $(\pi_i, \{c_{i,j}\}_{j\in[n]})$[a].

Qualification Phase:

3. Let $Qual = \emptyset$. In round $t_0 + 2\delta$, all parties $P_j{}' \in C'$ proceed as follows:
   (a) Check for each received tuple $(\pi_i, \{c_{i,k}\}_{k\in[n]})$ if π_i is valid and if so, store i in $Qual$ until $|Qual| = t+1$.[b]
   (b) For all $i \in Qual$ compute $s_{i,j} \leftarrow \mathsf{PKE.Dec}(\mathsf{esk}_j, c_{i,j})$.
   (c) Compute the secret key share $sk'_j \in \mathbb{Z}_q$ as $sk'_j = \sum_{i\in Qual} s_{i,j}$ and compute $S'_j = g^{sk'_j}$.
   (d) Compute a NIZK proof π'_j for the following language:

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

   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.
   (e) Execute $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}\,[U]\,(pp)$ to select a committee C'' with $|C''| = n$.
   (f) In round $t_0 + 3\delta$, execute the $\mathsf{G{-}Handover}[C'_{\langle (sk'_1,(S'_1,\pi'_1)),\cdots,(sk'_n,(S'_n,\pi'_n))\rangle}, U](pp)$ procedure s.t. each party $P_i{}'' \in C''$ learns a refreshed secret key share sk''_i.

Public Key Reconstruction Phase:

4. Let $PK = \emptyset$. In round $t_0 + 4\delta$, all parties in U check for all tuples (S'_j, π'_j) for $j \in [n]$ if π'_j is valid w.r.t. set $Qual$ and if so store S'_j in PK until $|PK| = t+1$.
5. The public key $pk \in \mathbb{G}$ can then be computed as $pk = \prod_{k\in PK} S_k'^{l_k}$ 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.

4.2 Security analysis

Theorem 1. *Let* NIZK *be a NIZK proof system,* Σ_{ECPSS} *be a* (λ, n, t, p)*-secure instantiation of the ECPSS scheme and* CPKE *be a RIND-SO secure PKE scheme. Suppose the discrete logarithm assumption holds in group* $\mathbb{G}$*, then the protocol* $\Pi_{\mathsf{LS-DKG}}$ *from Sec. 4.1 is a* (λ, n, t, p)*-secure large-scale distributed key generation protocol.*

In order to prove Theorem 1, we have to show that $\mathit{\Pi}_{\mathsf{LS-DKG}}$ satisfies the correctness and secrecy property w.r.t. a fully mobile adversary with corruption power p. The proof of correctness can be found in Appendix C.1. For the secrecy property, we give a proof sketch below and its full proof in Appendix C.2.

Lemma 1. *The large-scale distributed key generation scheme $\mathit{\Pi}_{\mathsf{LS-DKG}}$ as presented in Sec. 4.1 is (λ, n, t, p)-secret.*

Proof Sketch. 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 $\mathit{\Pi}_{\mathsf{LS-DKG}}$ protocol to a fully mobile adversary $\mathcal{A}$ in such a way that: (1) the simulated and real executions of $\mathit{\Pi}_{\mathsf{LS-DKG}}$ are computationally indistinguishable to $\mathcal{A}$, and (2) the public key that is output by the simulated execution is pk.

In our simulation, the simulator first "commits" to the secrets s_i for all honest $P_i \in C$ by sharing them to committee C' and later adjusts these secrets based on the set of qualified parties and the secrets chosen by the adversary such that:

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

We first show that a fully mobile adversary $\mathcal{A}$ can corrupt at most t parties in each committee via a reduction to the secrecy property of the $\mathit{\Sigma}_{\mathsf{ECPSS}}$ scheme. We can make this reduction because we use the role assignment functionality $\mathit{\Sigma}_{\mathsf{ECPSS}}.\mathsf{Select}$. That is, we show that if $\mathcal{A}$ is able to corrupt more than t parties in either of C, C' or C'', then we can construct an adversary that corrupts more than t parties of a holding committee in $\mathit{\Sigma}_{\mathsf{ECPSS}}$, thereby breaking the secrecy property of $\mathit{\Sigma}_{\mathsf{ECPSS}}$. The high level idea of this reduction is to set C, C' and C'' to the same set of parties as a holding committee in the execution of an $\mathit{\Sigma}_{\mathsf{ECPSS}}$ instance. Clearly, if then more than t parties get corrupted in any of C, C' or C'', there exists an adversary that can corrupt more than t parties in a holding committee of $\mathit{\Sigma}_{\mathsf{ECPSS}}$.

Hence, we must have an honest majority in each of C, C' and C'', which means: (1) there is at least one honest party in $Qual$ since $|Qual| = t + 1$, and (2) the simulator has sufficient information to reconstruct all secret key shares sk'_k for all parties $P_k{}' \in C'$ and to learn all elements S'_k. The simulator then "adjusts" the elements S'_j for some honest parties $P_j{}' \in C'$ such that Eq. (1) holds and broadcasts the adjusted elements S'_j along with a simulated NIZK proof. We must show that the adversary cannot distinguish the adjusted elements S'_j from the real ones. Since we cannot use $\mathit{\Sigma}_{\mathsf{ECPSS}}$ as black-box, we must exhibit two reductions to the RIND-SO security of the CPKE scheme to show this indistinguishability.

5 Large-Scale Threshold Public Key Encryption

A non-interactive threshold public key encryption scheme (TPKE) allows to distribute the secret key of a PKE scheme among a fixed set of parties, who can then non-interactively generate so-called decryption shares for a ciphertext. Given valid decryption shares from a certain threshold of parties, the ciphertext can then be decrypted. Security relies on the assumption that an adversary can corrupt less parties than this threshold. In contrast to traditional threshold public key encryption, a large-scale threshold public key encryption scheme, denoted by $\mathsf{LS–TPKE}$, operates in the YOSO model, i.e., it is defined w.r.t. a universe of parties U and relies on a role assignment functionality. An $\mathsf{LS–TPKE}$ scheme must include a refresh procedure which allows to handover the secret key shares of the scheme from one committee to another. This procedure is crucial to ensure security against a fully mobile adversary. Since committee members can speak only once per epoch, we require that each member can generate decryption shares locally. Note that a committee member can generate decryption shares for multiple ciphertexts in one epoch. All generated decryption shares are then broadcast during the refresh procedure. We now provide the formal definition of an $\mathsf{LS–TPKE}$ scheme.

Definition 5. *A large-scale non-interactive (t, n)-threshold public key encryption scheme* ($\mathsf{LS–TPKE}$) *is defined w.r.t. a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple* $\mathsf{LS–TPKE} =$ (Setup, $\mathsf{TKeyGen}$, TEnc, TDec, $\mathsf{TShareVrfy}$, $\mathsf{TCombine}$, $\mathsf{Refresh}$) *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 key share* sk_i*, verification key* vk_i *and a list of decryption shares* 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*, *Decryption Consistency* and *Efficiency*. 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) − 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 all $i \in [n]$, $j \geq 1$ and for any message m, label L and 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 $i \in [n]$ and $j > 1$, the k-th 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) − LS–TPKE scheme in the YOSO model considering an efficient fully mobile adversary $\mathcal{A}$ with corruption power p s.t. $p \cdot |U| > t$. 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 $\mathsf{LSTPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{LS\text{–}TPKE}}(\lambda)$ 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 $i \in [N]$, the game checks if $\left\lfloor \frac{|B^j|+1}{|U|} \right\rfloor \leq p$. If so, the game sets $B^j \leftarrow B^j \cup \{{P_i}^j\}$ and $\mathcal{A}$ receives the internal state of ${P_i}^j$.
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. At this point, $\mathcal{A}$ additionally obtains access to the following two oracles. Let $dl_i^1 := \emptyset$ for parties ${P_i}^1 \in C^1 \setminus B^1$:
   – **Decryption oracle**: On input a ciphertext ct with an associated label L, the game computes $ct_i^j \leftarrow \mathsf{TDec}(sk_i^j, ct, L)$ for all parties ${P_i}^j \in C^j \setminus B^j$. Then, the oracle adds ct_i^j to the list dl_i^j.
   – **Refresh oracle**: On input a set NB^j, the game executes $\mathsf{Refresh}[C^j_{\langle (sk_1^j, vk_1^j, dl_1^j), \cdots, (sk_n^j, vk_n^j, dl_n^j) \rangle}, U](pp)$ and sets $B^{j+1} \leftarrow B^j \setminus NB^j$. It further initializes lists $dl_i^{j+1} := \emptyset$ for parties ${P_i}^{j+1} \in C^{j+1} \setminus B^{j+1}$.
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.

We note that in the security game above the adversary is controlling the malicious parties who can deviate from the protocol description.

Definition 6. *A large-scale non-interactive (t,n)-threshold public key encryption scheme* $\mathsf{LS\text{–}TPKE}$ *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*

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

Further, we require a (t,n)-$\mathsf{LS\text{–}TPKE}$ scheme to satisfy decryption consistency, which intuitively says that a fully mobile adversary cannot generate a valid decryption share w.r.t. a ciphertext s.t. the share cannot be used for decryption. Our definition follows closely the decryption consistency definition of traditional threshold encryption schemes (e.g., [35]) with adaptions to the YOSO model.

Definition 7 (Decryption Consistency). *An* $\mathsf{LS\text{–}TPKE}$ *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* $\mathsf{LSTPKE\text{–}DC}$ *described below with non-negligible probability:*

$\mathsf{LSTPKE\text{–}DC}$*: The game proceeds as steps 1.-4. described in game* $\mathsf{LSTPKE\text{–}CCA}$ *with the difference that the adversary is allowed to learn all secret key shares in each epoch j*[6]*. The adversary eventually outputs 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 $\tilde{T} = \{\tilde{ct}_1^j, \cdots, \tilde{ct}_{t+1}^j\}$ and wins the game if the following conditions hold:*

[6]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

$$\mathsf{TShareVrfy}(ct^*, vk_i^j, ct_i^j) = 1 \text{ and } \mathsf{TShareVrfy}(ct^*, \tilde{vk}_i^j, \tilde{ct}_i^j) = 1.$$

2. $\mathsf{TCombine}(T, ct^*) \neq \mathsf{TCombine}(\tilde{T}, ct^*)$

Finally, we require an efficiency property. Intuitively, this property states that (1) similar to an ECPSS scheme, the total communication complexity during the execution of the refresh procedure for honest parties depends only on the committee size n and the number of decryption shares honest parties have to broadcast, and (2) an execution of the refresh procedure must terminate within a constant number of rounds.

Definition 8 (Efficiency). *Let dl_i^j be the decryption share list of an honest party $P_i{}^j \in C^j$ in epoch j. Then we call a (t,n)-LS–TPKE scheme efficient, if for all epochs $j \geq 1$ and a constant c:*

1. *The communication complexity of all honest parties during an execution of the* Refresh *procedure is upper bounded by some fixed polynomial* $\mathsf{poly}(n, \lambda, |dl_i^j|)$.
2. *An execution of the* Refresh *procedure takes at most $c \cdot \delta$ rounds.*

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

5.1 Construction

Shoup and Gennaro [41] introduced two TPKE schemes denoted as TDH1 and TDH2, which are both CCA-secure against static adversaries in the random oracle model [6]. Both of these schemes are well suited for our construction of an LS–TPKE scheme, since they are non-interactive and use discrete-log key pairs. In the following, we show how the scheme TDH1 = (Setup, KeyGen, TEnc, TShareVrfy, TCombine) can be transformed into an LS–TPKE scheme $\Pi_{\mathsf{LS-TPKE}}$ = (Setup, TKeyGen, TEnc, TDec, TShareVrfy, TCombine, Refresh). For this transformation, we make use of the following building-blocks: (1) our large-scale DKG protocol $\Pi_{\mathsf{LS-DKG}}$ = (Setup, TKeyGen) as described in Sec. 4, (2) the role assignment mechanism of Benhamouda et al., (3) the G–Handover procedure as presented in Sec. 3 and (4) a NIZK proof system NIZK = (Setup, Prove, Verify) as per Def. 13. Note that for similar reasons as for our $\Pi_{\mathsf{LS-DKG}}$ protocol, we cannot use the Σ_{ECPSS}.Handover procedure as black-box. Instead, we have to use the generalized handover procedure G–Handover to broadcast verification keys and decryption shares during a state handover.

We detail the construction of the $\Pi_{\mathsf{LS-TPKE}}$ scheme below, which is based on the TDH1 scheme given in Appendix E.

$\underline{\Pi_{\mathsf{LS-TPKE}}.\mathsf{Setup}(1^\lambda)}$

Execute:

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

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

$\underline{\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}[U](pp, t, n)}$

Let t_0 be the round in which the protocol execution begins. All parties in U do:

1. Parse $pp := (pp^{\mathsf{TDH1}}, pp^{\mathsf{LS-DKG}})$.
2. Run $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}[U](pp^{\mathsf{LS-DKG}}, t, n)$. This protocol selects a committee C^1 with $|C^1| = n$, outputs a public key pk to all parties in U and distributes secret key shares sk_i^1 to each party $P_i{}^1 \in C^1$.

In round $t_0 + 4\delta$ all parties $P_i{}^1 \in C^1$ do:

3. Compute $\widehat{vk}_i^1 := g^{sk_i^1}$ and a proof π_i^1 proving that $\widehat{vk}_i^1$ was computed correctly[a].
4. Set $vk_i^1 := \{\widehat{vk}_i^1, \pi_i^1\}$ and initialize a decryption share list $dl_i^1 := \emptyset$.

$\underline{\Pi_{\mathsf{LS-TPKE}}.\mathsf{TEnc}(pk, m, L)}$

Execute $\mathsf{TDH1}.\mathsf{TEnc}(pk, m, L)$ and output the resulting ciphertext ct.

$\underline{\Pi_{\mathsf{LS-TPKE}}.\mathsf{TDec}(sk_i^j, ct, L)}$

Execute $\mathsf{TDH1}.\mathsf{TDec}(sk_i^j, ct, L)$ and add the output (ct_i^j or $\perp$) to the list dl_i^j.

$\underline{\Pi_{\mathsf{LS-TPKE}}.\mathsf{TShareVrfy}(ct, vk_i^j, ct_i^j)}$

Parse $vk_i^j := \{\widehat{vk}_i^j, \pi_i^j\}$, verify π_i^j and output 0 if the verification fails. Otherwise, execute $\mathsf{TDH1}.\mathsf{TShareVrfy}(ct, \widehat{vk}_i^j, ct_i^j)$ and output the resulting bit.

$\underline{\Pi_{\mathsf{LS-TPKE}}.\mathsf{TCombine}(T, ct)}$

Execute $\mathsf{TDH1}.\mathsf{TCombine}(T, ct)$ and output the resulting message m.

$\underline{\Pi_{\mathsf{LS-TPKE}}.\mathsf{Refresh}[C^j_{\langle(sk_1^j, vk_1^j, dl_1^j), \cdots, (sk_n^j, vk_n^j, dl_n^j)\rangle}, U](pp)}$

Let t_j be the round in which this protocol execution begins. This protocol is executed between a committee C^j in epoch $j \geq 1$ and the universe U.

1. Run $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}[U](pp)$ to select a committee C^{j+1} with $|C^{j+1}| = n$.
2. In round $t_j + \delta$ run $\mathsf{G\text{-}Handover}[C^j_{\langle(sk_1^j, (vk_1^j, dl_1^j)), \cdots, (sk_n^j, (vk_n^j, dl_n^j))\rangle}, U](pp)$.

   Afterwards, each $P_i^{j+1} \in C^{j+1}$ receives a refreshed secret key shares sk_i^{j+1}

In round $t_j + 2\delta$ all parties $P_i^{j+1} \in C^{j+1}$ do:

3. 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 [a] and set $vk_i^{j+1} := \{\widehat{vk}_i^{j+1}, \pi_i^{j+1}\}$.
4. Initialize a decryption share list $dl_i^{j+1} := \emptyset$.

[a]The language for this proof is the same as the language L' in the $\Pi_{\mathsf{LS-DKG}}$ protocol (cf. Sec. 4).

Theorem 2. *Let $\Pi_{\mathsf{LS-DKG}}$ be a (λ, n, t, p)-secure instantiation of the* LS–DKG *protocol from Sec. 4,* TDH1 *be the non-interactive (t, n)-TPKE scheme as described in Appendix E, Σ_{ECPSS} a (λ, n, t, p)-secure instantiation of the ECPSS scheme,* NIZK *a NIZK proof system and* CPKE *a RIND-SO secure PKE scheme. Then $\Pi_{\mathsf{LS-TPKE}}$ is a (λ, n, t, p)-secure large-scale non-interactive 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, decryption consistency and efficiency as well as security against chosen-ciphertext attacks w.r.t. parameters (λ, n, t, p). We therefore state the following lemmas.

Lemma 2. *The large-scale non-interactive threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}}$ as described in Sec. 5.1 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 [7] of G–Handover. We provide a proof outline for Lemma 2 in Appendix D.

Lemma 3. *The large-scale non-interactive threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}}$ as described in Sec. 5.1 satisfies decryption consistency w.r.t. parameters (λ, n, t, p).*

Proof. We provide a proof sketch for Lemma 3 in Appendix D.

Lemma 4. *The large-scale non-interactive threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}}$ as described in Sec. 5.1 satisfies the efficiency property.*

Proof. This lemma follows directly from the fact that Σ_{ECPSS} is a scalable ECPSS scheme, i.e., the communication complexity of $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ and $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ is upper bounded by $\mathsf{poly}(n, \lambda)$. Note that we use the $\mathsf{G{-}Handover}$ procedure instead of $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ to additionally broadcast verification keys, NIZK proofs and decryption lists, whose communication complexity for honest parties, however, is upper bounded by $\mathsf{poly}(n, \lambda, |dl_i|)$. Furthermore, it is easy to see that the refresh procedure takes 2δ rounds.

Lemma 5. *The large-scale non-interactive threshold public key encryption scheme $\Pi_{\mathsf{LS{-}TPKE}}$ as described in Sec. 5.1 is secure against chosen-ciphertext attacks w.r.t. parameters (λ, n, t, p).*

Proof Sketch. We provide the full proof of Lemma 5 in Appendix D, and give a proof sketch here. At a high level, we show that if there exists a fully mobile adversary $\mathcal{B}$ with corruption power p who can win game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS{-}TPKE}}}$, then there exists an efficient static adversary $\mathcal{A}$ who can use $\mathcal{B}$ to win its own game $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ (cf. Def. 11). Therefore, we show how $\mathcal{A}$ can simulate the game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS{-}TPKE}}}$ such that it is indistinguishable from a real execution for $\mathcal{B}$, and how $\mathcal{A}$ can use $\mathcal{B}$'s output bit b' to win its own game.

The first step is similar to the proof of Lemma 1, i.e., we show that $\mathcal{B}$ corrupts at most t committee members per epoch via a reduction to the secrecy property of the Σ_{ECPSS} scheme. We then show how $\mathcal{A}$ simulates the game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS{-}TPKE}}}$ to $\mathcal{B}$ w.r.t. the public key pk that it receives from its own game $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$. $\mathcal{A}$ embeds pk in game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS{-}TPKE}}}$ by executing the simulator of the $\Pi_{\mathsf{LS{-}DKG}}$ scheme (cf. Fig. 2) on input pk. Afterwards, $\mathcal{A}$ knows the secret key shares of all honest and malicious parties. Note that these secret key shares are merely random values in $\mathbb{Z}_q$ and independent of pk. We have to show now that $\mathcal{A}$ can simulate game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS{-}TPKE}}}$ without $\mathcal{B}$ noticing that the committee members hold a sharing of a random value.

$\mathcal{A}$ has to simulate verification keys and decryption shares for honest committee members that are consistent with $\mathcal{B}$'s view. These simulated verification keys and decryption shares are, however, not consistent with the secret key shares of honest parties (since $\mathcal{A}$ does not know the correct secret key shares). Yet, as committee members erase their secret state before outputting their verification key and decryption shares, this inconsistency between public information and internal state of honest parties remains undetected by $\mathcal{B}$. More concretely, if $\mathcal{B}$ corrupts a committee member *before* it outputs 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 at most t committee members are corrupted). On the other hand, if $\mathcal{B}$ corrupts a committee member *after* it outputs its verification key and decryption shares, the secret key share has already been erased. Note, however, that at this point an encrypted sharing of the secret key share has been broadcast (as part of the $\mathsf{G{-}Handover}$ procedure) and that the encrypted shares are inconsistent with the verification key and decryption shares. Therefore, we must provide a reduction to the RIND-SO security of the CPKE scheme to show that this inconsistency remains undetected.

We note that parts of our proof use similar techniques as the proof of Gentry et al. [23] for their computational protocol (e.g., for the simulation of decryption shares). However, we additionally consider our $\Pi_{\mathsf{LS{-}DKG}}$ protocol and show reductions to, e.g., the secrecy of the Σ_{ECPSS} scheme and the RIND-SO security of the underlying CPKE scheme.

5.2 Transformation Framework from TPKE to LS–TPKE

The $\mathsf{TDH1}$ scheme satisfies certain properties that allow us to transform it into an $\mathsf{LS{-}TPKE}$ scheme. In the following, we will informally abstract these properties to argue that any CCA-secure non-interactive TPKE scheme satisfying these properties can be transformed into an $\mathsf{LS{-}TPKE}$ scheme. We will show in Appendix F how this idea can be applied to non-interactive threshold signature schemes. For our transformation, a TPKE must satisfy the following properties .

1. **Compatibility with $\Pi_{\mathsf{LS{-}DKG}}$.** The TPKE scheme must be 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 TPKE. More concretely, it must hold that $(pk, \cdot, (sk_1, \cdots, sk_n)) \in \mathsf{TPKE}.\mathsf{KeyGen}(pp, t, n)$.

2. **Dlog-Based Verification Keys.** Secret key shares and the corresponding verification keys must form a discrete-log instance in TPKE, i.e., for a secret key share sk_i the corresponding verification key is of the form $vk_i = g^{sk_i}$.
3. **Simulatability.** There must exist a simulator for the TPKE scheme that simulates the $\mathsf{TPKE\text{–}CCA}$ game (cf. Appx. A.2) 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 $\mathsf{TPKE\text{–}CCA}$ 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 $\mathsf{LSTPKE\text{–}CCA}_{\mathsf{LS\text{–}TPKE}}$. However, since $\mathsf{LS\text{–}TPKE}$ is run in epochs, we must guarantee that the simulation of the oracles remains consistent from the adversary's view across multiple consecutive executions of the protocol with differing secret key shares and verification keys while the public key stays the same.

The first property is naturally required for using our $\Pi_{\mathsf{LS\text{–}DKG}}$ protocol jointly with the TPKE scheme to construct the $\mathsf{LS\text{–}TPKE}$ scheme. For the second property recall that in our security proof of $\Pi_{\mathsf{LS\text{–}TPKE}}$, 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 public key pk and the secret key shares of all corrupted parties. Therefore, it can also compute the corresponding verification keys (say w.l.o.g. $(vk_1, \cdots, vk_t)$). 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 knowledge of the corresponding secret key shares. Finally, the third property gives us a simulator which we can use in our security proof to simulate the responses to oracle queries from a fully mobile adversary in game $\mathsf{LSTPKE\text{–}CCA}_{\mathsf{LS\text{–}TPKE}}$.

6 Instantiations and Applications

In this section, we first show how to instantiate our large-scale schemes and afterwards discuss various applications.

6.1 Instantiations

Benhamouda et al. show that their ECPSS construction is secure w.r.t. different choices of parameters. For instance, their scheme is $(128, 889, 425, 0.05)$-secure. Since our large-scale schemes use the concrete role assignment functionality of Benhamouda et al., our schemes are secure w.r.t. the same parameters. Further, our NIZK proofs for the languages L and L' can be instantiated using bulletproofs [11].

Since our $\mathsf{G\text{–}Handover}$ extends $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ only by allowing to broadcast auxiliary values, it is likely that any improved handover mechanism that is compatible with Σ_{ECPSS} can be used in our schemes as well. Indeed, a recent work by Gentry et al. [24] proposes an efficient non-interactive publicly verifiable secret sharing (PVSS) scheme with the goal of using the PVSS scheme to efficiently instantiate the $\mathsf{Handover}$ procedure of Σ_{ECPSS}. If the PVSS scheme can indeed efficiently instantiate the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$, then we believe that it can also instantiate our $\mathsf{G\text{–}Handover}$ procedure. Finally, another work by Gentry et al. [25] introduced a new role assignment mechanism that improves the one from Benhamouda et al. by allowing to significantly reduce the committee size and guaranteeing security against stronger adversaries. It is an interesting open question, if this role assignment mechanism can be used to improve our large-scale schemes as well.

6.2 Applications

Our protocols can directly be used for blockchain applications, in particular for: (1) storage of blockchain-backed secrets, and (2) adding signing functionality to a blockchain. We defer application (2) to Appendix H.

Storage of Secrets on a Blockchain Any information stored on a blockchain is publicly available to all users which severely restricts applications running on top of a blockchain. Recently, Benhamouda et al. [7] and Goyal et al. [26] presented solutions based on secret sharing to allow the storage of secret values on a blockchain. 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 blockchain, instead it suffices to (privately) exchange a commitment to a secret and have the blockchain open the commitment in case of malicious behavior.

Consider the example of a fair exchange protocol. Assume two parties, say Alice and Bob, wish to exchange secrets a and b. They can use the solutions of Benhamouda or Goyal et al. to share a and b to the committee, which then 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 possibly refresh the shares to new committees, 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 the corresponding public key pk is stored on the blockchain. In this case, Alice and Bob can encrypt their secrets under pk and exchange the ciphertexts[7]. Once each of them have received the respective ciphertext, they can reveal their secrets to each other. If one party, say Alice, does not reveal her secret, Bob can let the committee decrypt Alice's ciphertext[8]. Note that, in the optimistic case, i.e., when no party misbehaves, we have (1) no interaction with the committee, (2) the committee does not have to store and refresh a and b, and (3) the committee does not learn which parties interact with each other.

Naturally, this idea can be used to store secrets on a blockchain, 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 and Goyal et al. is that the committee only refreshes the secret key shares (instead of all secrets that are stored on the blockchain), and therefore the communication complexity of a state handover is *independent* of the number of stored secrets.

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, P. Jovanovic, M. Maller, 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] A. Acharya, C. Hazay, V. Kolesnikov, et al. *SCALES: MPC with Small Clients and Larger Ephemeral Servers*. Cryptology ePrint Archive, Paper 2022/751. 2022.

[4] J. F. Almansa, I. Damgård, and J. B. Nielsen. "Simplified Threshold RSA with Adaptive and Proactive Security". In: *Advances in Cryptology - EUROCRYPT 2006*. Berlin, Heidelberg, 2006.

[5] M. Bellare, A. Boldyreva, A. Desai, et al. "Key-Privacy in Public-Key Encryption". In: *ASIACRYPT 2001*. 2001.

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

[7] F. Benhamouda, C. Gentry, S. Gorbunov, et al. "Can a Public Blockchain Keep a Secret?" In: *Theory of Cryptography*. Cham, 2020.

[8] M. Blum, P. Feldman, and S. Micali. "Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)". In: *20th ACM STOC*. 1988.

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

[7] With NIZK proofs that prove the ciphertexts indeed encrypt a and b, respectively.

[8] The label of the ciphertext can be used as a decryption policy i.e.,"If Bob publishes his ciphertext on the blockchain, he can learn the content of Alice's ciphertext."

[10] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing". In: *ASIACRYPT 2001*. 2001.

[11] B. Bünz, J. Bootle, D. Boneh, et al. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. 2018.

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

[13] R. Canetti, R. Gennaro, S. Goldfeder, et al. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA, 2020.

[14] R. Canetti, R. Gennaro, S. Jarecki, et al. "Adaptive Security for Threshold Cryptosystems". In: *CRYPTO'99*. 1999.

[15] I. Cascudo, B. David, L. Garms, et al. *YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model*. Cryptology ePrint Archive, Report 2022/242. 2022.

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

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

[18] I. Damgård, T. P. Jakobsen, J. B. Nielsen, et al. "Fast Threshold ECDSA with Honest Majority". In: *Security and Cryptography for Networks*. Cham, 2020.

[19] Y. Desmedt and Y. Frankel. "Threshold Cryptosystems". In: *CRYPTO'89*. 1990.

[20] J. Doerner, Y. Kondi, E. Lee, et al. "Threshold ECDSA from ECDSA Assumptions: The Multiparty Case". In: *2019 IEEE Symposium on Security and Privacy*. 2019.

[21] Y. Frankel. "A Practical Protocol for Large Group Oriented Networks". In: *EUROCRYPT'89*. 1990.

[22] R. Gennaro, S. Jarecki, H. Krawczyk, et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *EUROCRYPT'99*. 1999.

[23] C. Gentry, S. Halevi, H. Krawczyk, et al. "YOSO: You Only Speak Once - Secure MPC with Stateless Ephemeral Roles". In: *CRYPTO 2021*. 2021.

[24] C. Gentry, S. Halevi, and V. Lyubashevsky. *Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties*. Cryptology ePrint Archive, Report 2021/1397. 2021.

[25] C. Gentry, S. Halevi, B. Magri, et al. "Random-Index PIR and Applications". In: *Theory of Cryptography*. Cham, 2021.

[26] V. Goyal, A. Kothapalli, E. Masserova, et al. *Storing and Retrieving Secrets on a Blockchain*. Cryptology ePrint Archive, Report 2020/504. 2020.

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

[28] K. Gurkan, P. Jovanovic, M. Maller, et al. *Aggregatable Distributed Key Generation*. Cryptology ePrint Archive, Report 2021/005. 2021.

[29] C. Hazay, A. Patra, and B. Warinschi. "Selective Opening Security for Receivers". In: *ASIACRYPT 2015, Part I*. 2015.

[30] A. Herzberg, M. Jakobsson, S. Jarecki, et al. "Proactive Public Key and Signature Systems". In: *ACM CCS 97*. 1997.

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

[32] S. Kolby, D. Ravi, and S. Yakoubov. *Towards Efficient YOSO MPC Without Setup*. Cryptology ePrint Archive, Report 2022/187. 2022.

[33] C. Komlo and I. Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: *Selected Areas in Cryptography*. Cham, 2021.

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

[35] B. Libert and M. Yung. "Adaptively Secure Non-interactive CCA-Secure Threshold Cryptosystems: Generic Framework and Constructions". In: *J. Cryptol.* 4 (2020).
[36] R. Ostrovsky and M. Yung. "How to Withstand Mobile Virus Attacks (Extended Abstract)". In: *10th ACM PODC*. 1991.
[37] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *EUROCRYPT'99*. 1999.
[38] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO'91*. 1992.
[39] T. P. Pedersen. "A Threshold Cryptosystem without a Trusted Party". In: *Advances in Cryptology — EUROCRYPT '91*. Berlin, Heidelberg, 1991.
[40] A. Shamir. "How to Share a Secret". In: *Communications of the Association for Computing Machinery* 11 (1979).
[41] V. Shoup and R. Gennaro. "Securing Threshold Cryptosystems against Chosen Ciphertext Attack". In: *EUROCRYPT'98*. 1998.
[42] N. Shrestha, A. Bhat, A. Kate, et al. *Synchronous Distributed Key Generation without Broadcasts.* Cryptology ePrint Archive, Report 2021/1635. 2021.
[43] A. Tomescu, R. Chen, Y. Zheng, et al. "Towards Scalable Threshold Cryptosystems". In: *2020 IEEE Symposium on Security and Privacy*. 2020.

Supplementary Material

A Additional Related Work and Preliminaries

A.1 Additional Related Work

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., [38, 22]). Recently, Gurkan et al. [28] 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. [42] 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. [14] proposed DKG protocols which are secure against adaptive adversaries. The recent work by Groth [27] introduces a non-interactive distributed key generation protocol, which is secure against a mobile adversary, but not in 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., [41, 9]) and adaptive adversaries (e.g., [14, 31]). Herzberg et al. [30] 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., [14, 4]) 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. Finally, Tomescu et al. [43] consider the notion of scalable threshold cryptosystems, however, in an entirely different setting than the one we consider in our work.

A.2 Additional Preliminaries

Anonymous PKE We now briefly recall the definition of an anonymous PKE scheme as introduced by Bellare et al. [5].

Definition 9. *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 In the following, we recall the notion of a non-interactive threshold public key encryption scheme.

Definition 10. *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 $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$:

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 11. *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* $\Pr[\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}(\lambda) = 1] \leq 1/2 + \nu(\lambda)$*. We define the advantage of* $\mathcal{A}$ *in game* $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ *as* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{TPKE\text{–}CCA}_{\mathsf{TPKE}}} = |\Pr[\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}} = 1] - 1/2|$.

Definition 12 (Decryption Consistency). *A* (t, n)*-*TPKE *scheme satisfies decryption consistency if for all* $\lambda \in \mathbb{N}$*, all* $pp \leftarrow \mathsf{Setup}(1^\lambda)$ *and all PPT adversaries* $\mathcal{A}$ *it holds that*

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

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

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 [8].

Definition 13. *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.*

B Combined Encryption Scheme from [7]

The security of the Σ_{ECPSS} scheme of Benhamouda et al. [7] 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 [7].

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 [7]. 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

C Proof of Theorem 1

In this section, we provide a proof of Theorem 1. We do so by first proving Lemma 6 and then Lemma 1.

C.1 Proof of Correctness

Lemma 6. *The large-scale distributed key generation scheme $\Pi_{\mathsf{LS-DKG}}$ as presented in Sec. 4.1 satisifies the correctness property.*

Proof. In order to prove Lemma 6, 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 [22]. 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 and all honest parties wait until round $t_0 + 2\delta$ to consider the same set of tuples broadcast after the share distribution phase. 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' by round $t_0 + 2\delta$. 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 [7] 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).[9] 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 $pk = g^{sk} \in \mathbb{G}$.

C.2 Proof of Lemma 1

In order to prove Lemma 1, we first state and prove the following lemma:

Lemma 7. *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 (Sketch). We can 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

[9]Note that all honest parties in U compute the same set PK since we assume an order on the broadcast tuples and all honest parties wait until round $t_0 + 4\delta$ to consider the same set of tuples.

property of Σ_{ECPSS}. In our reduction, we assume for simplicity that the self-selection process in Σ_{ECPSS} is implemented via verifiable random functions as suggested in [7]. We further assume that the verifiable random functions are evaluated on the same input values in Σ_{ECPSS} and $\Pi_{\mathsf{LS-DKG}}$, which is easy to achieve in, e.g., the blockchain setting. We finally assume that a malicious party in the nominating committee always selects itself.

- **Case 1:** $\mathcal{A}$ corrupts more than t parties in C
  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}, s.t., if more than t parties in C get corrupted by $\mathcal{A}$, then $\mathcal{B}$ corrupts more than t parties in the holding committee of Σ_{ECPSS}. Therefore, $\mathcal{B}$ first 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 $\mathcal{A}$. Whenever $\mathcal{A}$ corrupts a party in $\Pi_{\mathsf{LS-DKG}}$, $\mathcal{B}$ corrupts the same party in Σ_{ECPSS} and forwards its internal state to $\mathcal{A}$. 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 $\mathcal{A}$. $\mathcal{B}$ lets all corrupted parties in the nominating committee self-select[10]. This ensures that the holding committee in Σ_{ECPSS} and committee C contain the same parties.
  Whenever $\mathcal{A}$ corrupts a party P_i after the selection of C, $\mathcal{B}$ corrupts the corresponding party in Σ_{ECPSS} and hence learns whether P_i is part of C. If it is, $\mathcal{B}$ prepares an internal state for P_i corresponding to the protocol description of $\Pi_{\mathsf{LS-DKG}}$ and sends this state along with the ephemeral secret key esk_i to $\mathcal{A}$[11].
  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}. Recall that the long-term public keys of all parties in the universe are updated in each epoch. Therefore, $\mathcal{B}$ must update the long-term public keys of honest parties in $\Pi_{\mathsf{LS-DKG}}$ to the same public keys used by honest parties in Σ_{ECPSS}. Additionally, for all long-term public keys pk'_k that are chosen by $\mathcal{A}$ in $\Pi_{\mathsf{LS-DKG}}$, $\mathcal{B}$ has to choose its own long-term key pair (pk_k, sk_k) and publish pk_k in Σ_{ECPSS}. During the execution of $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$, $\mathcal{B}$ first receives a ciphertext c_i[12] from each honest party P_i in the nominating committee of Σ_{ECPSS}. $\mathcal{B}$ then tries to decrypt all received ciphertexts with its long-term secret keys sk_k. If decryption is successful, $\mathcal{B}$ learns an ephemeral secret key esk_i, which it then encrypts under $\mathcal{A}$'s corresponding long-term public key pk'_k, i.e., $c'_i \leftarrow \mathsf{APKE.Enc}(pk'_k, \mathsf{esk}_i)$. $\mathcal{B}$ then forwards all received ciphertexts to $\mathcal{A}$ but replaces c_i by c'_i.
  This simulation ensures again that the holding committee in Σ_{ECPSS} and committee C' consist of the same parties. Similar to the reduction above, whenever $\mathcal{A}$ corrupts an honest party, $\mathcal{B}$ corrupts the corresponding party in Σ_{ECPSS}. If the corrupted party is in the holding committee, then $\mathcal{B}$ learns its ephemeral secret key and thereby, $\mathcal{B}$ can simulate the party's internal state. It again holds that if $\mathcal{A}$ corrupts more than t parties in C', then $\mathcal{B}$ corrupts more than t parties in Σ_{ECPSS}. Therefore, $\mathcal{A}$ corrupts more than t parties in C with at most negligible probability in λ.
- **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}.

[10] $\mathcal{B}$ obtains the proof that a malicious party has been selected to the nominating committee from $\mathcal{A}$ and the proof is valid in Σ_{ECPSS} since the same input values are used in the self-selection processes of Σ_{ECPSS} and $\Pi_{\mathsf{LS-DKG}}$.

[11] To be exact, $\mathcal{B}$ returns the 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., the secret key for the verifiable random function.

[12] Recall that these ciphertexts nominate a public key pk_k to the holding committee by encrypting an ephemeral secret key under pk_k.

With Lemma 7 in place, we can now prove Lemma 1.

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 $(\tilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)$ instead of $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. 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 3c. 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 states of all parties $P_j{}' \in H'$[13], in particular the values sk'_j. Further, note that due to Lemma 7 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 3c, $\mathcal{S}$ 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 3c while for parties $P_j{}' \in H'$, $\mathcal{S}$ sets

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

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 3d, $\mathcal{S}$ 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.

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 3c and 3d 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 3c, $\mathcal{S}$ 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 7, there exists at least one honest party $P_i \in Qual$ and we further know that all honest parties compute the same set $Qual$. 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.

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

Game $\boldsymbol{G_1}$: 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.

Game $\boldsymbol{G_2}$: This game works as the previous game with the difference that $\mathcal{S}$ aborts if any party $P_k \in B$[14] generates a valid proof π_k for a statement $x = (\{c_{k,i}, \mathsf{epk}_i\}_{i\in[n]})$ such that $x \notin L$. The simulator can identify this situation, since it knows the ephemeral secret keys of all parties in H' and since it learns at least $t+1$ decryptions of the ciphertexts $(c_{k,1}, \cdots, c_{k,n})$.

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

Game $\boldsymbol{G_3}$: 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 $\tilde{\pi}'_{j,\mathsf{Handover}}$[15] (i.e., without using the secret key share sk'_j) using the trapdoor τ and algorithm S (cf. Def. 13).

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

Game $\boldsymbol{G_4}$: 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. 13).

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 $\boldsymbol{G_5}$: 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 $\boldsymbol{G_6}$: This game works as the previous game with the following difference. The simulator first chooses uniformly at random a secret key $\tilde{sk} \leftarrow_\$ \mathbb{Z}_q$ 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 $\tilde{sk}'_j$.

Claim. $\mathcal{A}$ can distinguish this game from the previous game with at most negligible probability.

Proof. Due to Lemma 7, there are at most t corrupted parties in C'. Further, note that all honest parties compute the same set $Qual$ and that there must exist at least one honest party in $Qual$. As such, each sk'_j for $j \in H'$ contains at least one uniformly random value. Therefore, $\mathcal{A}$ can distinguish this game from the previous one only by breaking the RIND-SO security of the combined PKE scheme CPKE.

We now 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 $F(x) = a_0 + a_1x + \cdots + a_tx^t \in \mathbb{Z}_q[x]$ 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 $|I| \le t$ samples a uniform random element $z \leftarrow_\$ \mathbb{Z}_q$ and outputs a (t,n)-sharing of z conditioned on $\mathbf{m}_I$.

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

[15]$\tilde{\pi}'_{j,\mathsf{Handover}}$ is used during the $\mathsf{G-Handover}$ procedure, see Sec. 3.

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 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 public key/ciphertext 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.[16] 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 $\boldsymbol{G_5}$ and $\boldsymbol{G_6}$. Therefore, $\mathcal{A}$ succeeds at most with negligible probability.

Game $\boldsymbol{G_7}$: 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.[17] $\mathcal{S}$ 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 $\boldsymbol{G_6}$, 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 $\tilde{sk}'_j$.

Game $\boldsymbol{G_8}$: 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 $\boldsymbol{G_9}$: This game works as the previous game with the difference that $\mathcal{S}$ aborts if any party $P_k{}' \in B'$ generates a valid 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'.

[16]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 $c_{i,\mathsf{PKE}}$.

[17]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 7.
On input a public key pk, trapdoor τ and public parameters pp, $\mathcal{S}_2$ then simulates $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ as follows:
- $\mathcal{S}_2$ follows the protocol instructions for all honest parties until step 3c.
- In step 3c, $\mathcal{S}_2$ 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 $P_j{}' \in H'$, $\mathcal{S}_2$ 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 3d, $\mathcal{S}_2$ then uses the trapdoor τ to generate simulated NIZK proofs $\tilde{\pi}'_j$ that prove correctness of the elements $\tilde{S}'_j$.
- The simulator $\mathcal{S}_2$ executes the rest of the protocol correctly for all honest parties.

Fig. 2: 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.

D Proof of Theorem 2

In this section, we first provide a proof outline of Lemma 2 before giving a formal proofs of Lemma 3 and Lemma 5.

D.1 Proof outline of Lemma 2

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 [7] of the $\mathsf{G{-}Handover}$ 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:
$$\text{for all } (pk, \{vk_i^1\}_{i\in[n]}, \{sk_i^1\}_{i\in[n]}) \leftarrow \Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}\,[U]\,(pp, t, n),$$
where $vk_i^1 := (\widehat{vk}_i^1, \pi_i^1)$ it holds that
$$(pk, \cdot, \{sk_i^1\}_{i\in[n]}) \in \mathsf{TDH1.KeyGen}(pp^{\mathsf{TDH1}}, t, n).$$
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.

D.2 Proof Sketch of Lemma 3

We can prove Lemma 3 via reduction to the decryption consistency property of the $\mathsf{TDH1}$ scheme. Assume an adversary $\mathcal{B}$ winning game $\mathsf{LSTPKE{-}DC}$ 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 $\mathsf{LSTPKE{-}DC}$ hold, then we can use $\mathcal{B}$ to construct a PPT adversary $\mathcal{A}$ that breaks the decryption consistency property of the $\mathsf{TDH1}$ scheme.

For this reduction, $\mathcal{A}$ simulates game $\mathsf{LSTPKE{-}DC}$ to $\mathcal{B}$ in a straightforward way, i.e., it executes the game code of $\mathsf{LSTPKE{-}DC}$. Note that by the soundness property of the NIZK proof system for language L', it holds except with negligible probability that $VK = \tilde{VK}$, i.e., the sets contain the same verification keys. Further, we know that for all epochs $j \geq 1$ and for the tuple $(pk, \{vk_i^j\}_{i\in[n]}, \{sk_i^j\}_{i\in[n]})$ with $vk_i^j := (\widehat{vk}_i^j, \pi_i^j)$, it holds that

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

Finally, since the algorithms $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TCombine}$ and $\mathsf{TDH1.TCombine}$ are identical and the algorithms $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TShareVrfy}$ and $\mathsf{TDH1.TShareVrfy}$ differ only in the fact that $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TShareVrfy}$ 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 $(\widehat{vk}_1^j, \cdots, \widehat{vk}_{t+1}^j)$ and secret key shares $(sk_1^j, \cdots, sk_{t+1}^j)$.

Therefore, upon $\mathcal{B}$ winning the $\mathsf{LSTPKE{-}DC}$ game with non-negligible probability by outputting $(ct^*, VK, \tilde{VK}, T, \tilde{T})$, $\mathcal{A}$ breaks the decryption consistency property of $\mathsf{TDH1}$ with non-negligible probability by outputting $((pk, \{\widehat{vk}_i^j\}_{i\in[n]}, \{sk_i^j\}_{i\in[n]}), ct^*, T, \tilde{T})$.

D.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 $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ game with non-negligible advantage, then there also exists a static adversary $\mathcal{A}$ who can win the game $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ (cf. Section A.2) 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 $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ game. In the beginning of this game, the $\Pi_{\mathsf{LS-TPKE}}.\mathsf{Setup}$ 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 $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ 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 7, i.e., $\mathcal{A}'$ trying to break the secrecy property of Σ_{ECPSS} can simulate game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ 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 7.

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

Game G_2: This game is the same as the previous game with only two syntactical changes. First, 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., $H^{C^j} \cap SH^{C^j} = \emptyset$).

Second, the game maintains a list CT^j for each epoch j. Upon the adversary querying the decryption oracle on input a ciphertext ct^j in epoch j, the game stores ct^j in CT^j.

The changes in this game are only syntactical. Therefore, we have that $\Pr[\boldsymbol{G_1} = 1] = \Pr[\boldsymbol{G_2} = 1]$.

Game $\boldsymbol{G_3}$: 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 via Lagrange interpolation. Note that this is possible because $|H^{C^1}| \geq t+1$. During a refresh 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 $\tilde{ct}_i^j$ for ${P_i}^j$ for all ciphertexts $ct^j \in CT^j$. Finally, the game replaces all decryption shares in decryption list dl_i^j by the decryption shares $\tilde{ct}_i^j$.

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 ${P_i}^j \in H^{C^j}$. Therefore, all decryption shares $\tilde{ct}_i^j$ are valid w.r.t. the verification key vk_i^j. We therefore get that $\Pr[\boldsymbol{G_2} = 1] = \Pr[\boldsymbol{G_3} = 1]$.

Game $\boldsymbol{G_4}$: This game works as the previous game with the following difference. Upon a refresh oracle query in epoch j, the game parses the verification key of parties ${P_i}^j \in H^{C^j}$ as $vk_i^j := (\widehat{vk}_i^j, \pi_i^j)$ and computes a new verification key $\tilde{vk}_i^j := (\widehat{vk'}_i^j, \pi_i^j)$ where $\widehat{vk'}_i^j = g^{\tilde{sk}_i^j}$. The game then uses $\tilde{vk}_i^j$ instead of vk_i^j during the refresh oracle execution.

The indistinguishability argument for this game follows in the same way as for the previous game, i.e., since we have that $\tilde{sk}_i^j = sk_i^j$, it holds that $\tilde{vk}_i^j = vk_i^j$. Therefore, we have that $\Pr[\boldsymbol{G_3} = 1] = \Pr[\boldsymbol{G_4} = 1]$.

Game $\boldsymbol{G_5}$: 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 the NIZK proof system), it holds that $\Pr[\boldsymbol{G_4} = 1] \leq \Pr[\boldsymbol{G_5} = 1] + \nu_2(\lambda)$ where ν_2 is a negligible function in λ.

Game $\boldsymbol{G_6}$: This game works as the previous game with the following difference. Upon a refresh oracle query j, the game computes for each party ${P_i}^j \in H^{C^j}$ a simulated NIZK proof $\tilde{\pi}_i^j$ using the trapdoor τ and algorithm S (cf. Def. 13) which proves that $\widehat{vk'}_i^j$ is a valid verification key for party ${P_i}^j$. It then sets $\tilde{vk}_i^j := (\widehat{vk'}_i^j, \tilde{\pi}_i^j)$.

Due to the zero-knowledge property of the NIZK proof system, the simulated proof $\tilde{\pi}_i^j$ is indistinguishable from a real proof except with negligible probability. It holds that $\Pr[\boldsymbol{G_5} = 1] \leq \Pr[\boldsymbol{G_6} = 1] + \nu_3(\lambda)$ where ν_3 is a negligible function in λ.

Game $\boldsymbol{G_7}$: 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 $\pi_{k,\mathsf{Handover}}^j$, but there exists at least one party ${P_i}^{j+1} \in H^{C^{j+1}}$ such that $\mathsf{PKE.Dec}(\mathsf{esk}_i^{j+1}, c_{k,i}^j) \notin \mathbb{Z}_q$ or the decryptions of the ciphertexts $(c_{k,1}^j, \cdots, c_{k,n}^j)$ do not form a (t, n)-sharing of sk_k^j. 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 $\Pr[\boldsymbol{G_6} = 1] \leq \Pr[\boldsymbol{G_7} = 1] + \nu_4(\lambda)$ where ν_4 is a negligible function in λ.

Game $\boldsymbol{G_8}$: This game proceeds similarly as the previous game with the following modification. In the beginning of the game, a uniformly random value $x \leftarrow_\$ \mathbb{Z}_q$ is chosen. Then during the refresh oracle execution, the game computes a degree-t polynomial $\bar{F}^j$, s.t., $\bar{F}^j(0) = x$ and $\bar{F}^j(k) = sk_k^j$ for all $P_k \in \{SH^{C^j} \cup B^{C^j}\}$. Then the game executes the G–Handover procedure on input $x_i^j \leftarrow \bar{F}^j(i)$ for all parties ${P_i}^j \in H^{C^j}$ and on input sk_k^j for all ${P_k}^j \in SH^{C^j}$.

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 $\boldsymbol{G_6}$ of Lemma 1, with the only difference that the resamplable distribution must output either a sharing of sk_i^j or of x_i^j. Therefore, 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.

The following games $\boldsymbol{G_9}$-$\boldsymbol{G_{12}}$ show how random oracle and decryption oracle queries are handled, which was previously described in [41].

Game $\boldsymbol{G_9}$: This game proceeds as the previous game with a modification to the random oracle H_2. The game programs H_2 by maintaining a list H_2 in the following way. Upon a query (c, L, u, w) to 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 $\Pr[\boldsymbol{G_8} = 1] = \Pr[\boldsymbol{G_9} = 1]$.

Game $\boldsymbol{G_{10}}$: 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 H_4. Second, upon $\mathcal{B}$ issuing a refresh oracle query, the game does the following for all ciphertexts $ct^j \in CT^j$ where $ct^j = (c, L, u, \bar{u}, e, f)$: it computes the decryption share $\tilde{ct}_i^j$ for each party ${P_i}^j \in H^{C^j}$ by computing $u_i = u^{\tilde{sk}_i^j}$ 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 $\tilde{ct}_i^j = (i, u_i, e_i, f_i)$ is valid w.r.t. ciphertext ct^j and verification key $\tilde{vk}_i^j$, i.e., it holds that $\mathsf{TShareVrfy}(ct^j, \tilde{vk}_i^j, \tilde{ct}_i^j) = 1$.

Adversary $\mathcal{B}$ can distinguish this game from the previous one only in the event that game $\boldsymbol{G_{10}}$ 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 $\mathbb{G}$. Since the adversary makes only a polynomially bounded number of decryption queries, the abort event happens at most with negligible probability. Therefore, we have that $\Pr[\boldsymbol{G_9} = 1] \leq \Pr[\boldsymbol{G_{10}} = 1] + \nu_6(\lambda)$ where ν_6 is a negligible function in λ.

Game $\boldsymbol{G_{11}}$: This game proceeds similarly to the previous game with the following modification in the refresh oracle. For all ${P_i}^j \in H^{C^j}$, the game computes the verification key $\widehat{vk'}_i^j$ as $\widehat{vk'}_i^j = pk^{l_{i,0}} \prod_{k \in B^{C^j} \cup SH^{C^j}} \widehat{vk}_k^{j^{l_{i,k}}}$ instead of $\widehat{vk'}_i^j = g^{\tilde{sk}_i^j}$.

Note that it holds that

$$\widehat{vk'}_i^j = pk^{l_{i,0}} \cdot \prod_{k \in B^{C^j} \cup SH^{C^j}} \widehat{vk}_k^{j^{l_{i,k}}} = g^{\tilde{F}^j(i)} = g^{\tilde{sk}_i^j}.$$

Therefore, we have that $\Pr[\boldsymbol{G_{10}} = 1] = \Pr[\boldsymbol{G_{11}} = 1]$.

Game $\boldsymbol{G_{12}}$: This game proceeds similarly to the previous game with the following modification in the refresh oracle. Upon a refresh oracle query, the game computes the decryption shares $\tilde{ct}_i^j$ for all ${P_i}^j$ and for all $ct^j \in CT^j$ as follows:

It 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 $\boldsymbol{G_9}$ and that the game knows o. It then computes $(\bar{u})^{1/o} = (\bar{g})^{r/o} = pk^r$ and $u_i = (\bar{u})^{l_{i,0}/o} \cdot \prod_{k \in B^{C^j} \cup SH^{C^j}} u^{sk_k l_{i,k}}$. The computation of e_i and f_i as well as the programming of H_4 is done in the same way as described in $\boldsymbol{G_{10}}$.

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 o is chosen uniformly at random from $\mathbb{Z}_q$ this event happens only with negligible probability and therefore it holds that $\Pr[\boldsymbol{G_{11}} = 1] \leq \Pr[\boldsymbol{G_{12}} = 1] + \nu_7(\lambda)$ where ν_7 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 $\mathit{\Pi}_{\mathsf{LS\text{-}TPKE}}.\mathsf{TKeyGen}$, the game chooses at random a public key pk, s.t. $(pk, \cdot, \cdot) \in \mathsf{TDH1.KeyGen}$. Then during the $\mathit{\Pi}_{\mathsf{LS\text{-}TPKE}}.\mathsf{TKeyGen}$ procedure, instead of executing $\mathit{\Pi}_{\mathsf{LS\text{-}DKG}}.\mathsf{TKeyGen}$, the game executes $\mathcal{S}_2$, i.e., the simulator code of $\mathit{\Pi}_{\mathsf{LS\text{-}DKG}}.\mathsf{TKeyGen}$ (cf. Fig. 2) on input pk, the NIZK trapdoor τ and the public parameters $pp^{\mathsf{LS\text{-}DKG}}$. This code simulates $\mathit{\Pi}_{\mathsf{LS\text{-}DKG}}.\mathsf{TKeyGen}$ in a way such that the output public key is equal to pk.

The indistinguishability argument follows from the secrecy property of the $\mathit{\Pi}_{\mathsf{LS\text{-}DKG}}$ scheme. More precisely, we showed that for the $\mathit{\Pi}_{\mathsf{LS\text{-}DKG}}$ protocol there exists a simulator which on input a public key pk, trapdoor τ and public parameters $pp^{\mathsf{LS\text{-}DKG}}$ can simulate the execution of $\mathit{\Pi}_{\mathsf{LS\text{-}DKG}}.\mathsf{TKeyGen}$ 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 $(pk, \tau, pp^{\mathsf{LS\text{-}DKG}})$ is identical to the output distribution of $\mathcal{S}_1$ in Fig. 2. Therefore, it holds that $\Pr[\boldsymbol{G_{12}} = 1] \leq \Pr[\boldsymbol{G_{13}} = 1] + \nu_8(\lambda)$ where ν_8 is a negligible function in λ.

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

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

where $\nu(\lambda) \geq \sum_{i=1}^{8} \nu_i(\lambda)$ 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 $\mathsf{TPKE\text{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ 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 $\mathsf{TPKE\text{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$, $\mathcal{A}$ 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, H_3 and H_4 to the corresponding oracles in the $\mathsf{TPKE\text{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ 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, these changes are only syntactical.

Finally, we have to show that $\mathcal{A}$ can use $\mathcal{B}$ to win the $\mathsf{TPKE\text{-}CCA}^{\mathcal{A}}_{\mathsf{TDH1}}$ game. Note that the encryption procedure is the same in both the $\mathit{\Pi}_{\mathsf{LS\text{-}TPKE}}$ 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 but programs the oracle on all other queries from $\mathcal{B}$, 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

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

E The TDH1 Threshold Public Key Encryption Scheme From Shoup and Gennaro [41]

In the following we briefly recall the (t, n)-threshold encryption scheme from Shoup and Gennaro [41], denoted by TDH1.

Setup(1^λ)*:* 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 $pp := (\mathbb{G}, q, g, l, H_1, H_2, H_3, H_4)$.

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.

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 $ct = (c, L, u, \bar{u}, e, f)$.

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 $ct_i = (i, u_i, e_i, f_i)$.

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.

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 [41], Shoup and Gennaro prove in the random oracle model that TDH1 is CCA-secure against static adversaries corresponding to Def. 11.

F Large-Scale Non-Interactive Threshold Signature Schemes

F.1 Model

The formal definition of a large-scale non-interactive threshold signature scheme ($\mathsf{LS{-}TSIG}$) in the YOSO model follows the ideas of the definition of $\mathsf{LS{-}TPKE}$ schemes. That is, an $\mathsf{LS{-}TSIG}$ scheme is defined w.r.t. 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 $\mathsf{LS{-}TPKE}$ schemes, the definition of $\mathsf{LS{-}TSIG}$ 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. An $\mathsf{LS{-}TSIG}$ scheme must be secure w.r.t. a fully mobile adversary. Finally, we require that each committee member can generate signature shares locally and that a committee member can generate multiple signature shares per epoch. All generated signatures are then broadcast during the refresh procedure. We now provide the formal definition of a non-interactive $\mathsf{LS{-}TSIG}$ scheme.

Definition 14. *A large-scale non-interactive (t, n)-threshold signature scheme ($\mathsf{LS{-}TSIG}$) is defined w.r.t. a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple $\mathsf{LS{-}TSIG} = (\mathsf{Setup}, \mathsf{TKeyGen}, \mathsf{TSign}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Verify}, \mathsf{Refresh})$ 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 $i \in [n]$ and $j \geq 1$ and 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 $i \in [n]$ and $j > 1$, the k-th 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 $\{vk^1_1, \cdots, vk^1_n\}$ 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. Additionally to the corruption oracle, the adversary $\mathcal{A}$ obtains access to the following two oracles. Let $sl^1_i := \emptyset$ for parties ${P_i}^1 \in C^1 \setminus B^1$.
   – **Signing oracle**: On input a message m, the game computes $\sigma^j_i \leftarrow \mathsf{TSign}(sk^j_i, m)$ for all parties ${P_i}^j \in C^j \setminus B^j$. Then, the oracle adds σ^j_i to the list sl^j_i.
   – **Refresh oracle**: On input a set NB^j, the game executes $\mathsf{Refresh}[C_{\langle (sk^j_1, vk^j_1, sl^j_1), \cdots, (sk^j_n, vk^j_n, sl^j_n) \rangle}, U](pp)$ is executed and the game sets $B^{j+1} \leftarrow B^j \setminus NB^j$. Additionally, the game initializes the lists $sl^{j+1}_i := \emptyset$ for parties ${P_i}^{j+1} \in C^{j+1} \setminus B^{j+1}$.
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 (Unforgeability). *A large-scale non-interactive (t, n)-threshold signature scheme* $\mathsf{LS{-}TSIG}$ *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*

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

Further, we require a (t, n)-$\mathsf{LS{-}TSIG}$ scheme to satisfy robustness, which intuitively says that a fully mobile adversary cannot generate signature shares, which are valid w.r.t. a message and the corresponding verification keys, but cannot be combined to a full signature on this message. That is, an honest majority can always generate a valid signature.

Definition 16 (Robustness). *An* $\mathsf{LS{-}TSIG}$ *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* $\mathsf{LSSIG{-}UFCMA}$ *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. $\mathsf{Verify}(pk, m, \mathsf{TCombine}(pk, m, T)) = 0$.

Finally, similar to $\mathsf{LS{-}TPKE}$ schemes, we require the following efficiency property.

Definition 17 (Efficiency). *Let sl^j_i be the signature share list of an honest party ${P_i}^j \in C^j$ in epoch j. Then we call a (t, n)-*$\mathsf{LS{-}TSIG}$ *scheme efficient, if for all epochs $j \geq 1$ and a constant c:*

1. *The communication complexity of all honest parties during an execution of the* $\mathsf{Refresh}$ *procedure is upper bounded by some fixed* $\mathsf{poly}(n, \lambda, |sl^j_i|)$.
2. *An execution of the* $\mathsf{Refresh}$ *procedure takes at most $c \cdot \delta$ rounds.*

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

F.2 Construction

Boldyreva [9] introduced a threshold signature scheme denoted as $\mathsf{TH\text{–}BLS}$, which is secure against a static adversary. This scheme is well suited for our construction of a large-scale threshold signature scheme, since it is non-interactive and uses discrete-log key pairs. In the following, we show how the scheme $\mathsf{TH\text{–}BLS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{TSign}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Verify})$ can be transformed into an $\mathsf{LS\text{–}TSIG}$ scheme $\Pi_{\mathsf{LS\text{–}TSIG}} = (\mathsf{Setup}, \mathsf{TKeyGen}, \mathsf{TSign}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Verify}, \mathsf{Refresh})$. Similarly as for our construction of $\Pi_{\mathsf{LS\text{–}TPKE}}$ (cf. Sec. 5), we make use of the following building-blocks: (1) our large-scale DKG protocol $\Pi_{\mathsf{LS\text{–}DKG}} = (\mathsf{Setup}, \mathsf{TKeyGen})$ as described in Sec. 4, (2) the role assignment mechanism of Benhamouda et al., (3) the $\mathsf{G\text{–}Handover}$ procedure as presented in Sec. 3 and (4) a NIZK proof system $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as per Def. 13. Note that for similar reasons as for our $\Pi_{\mathsf{LS\text{–}DKG}}$ and $\Pi_{\mathsf{LS\text{–}TPKE}}$ protocols, we cannot use the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure as black-box. Instead, we have to use the generalized handover procedure $\mathsf{G\text{–}Handover}$ to broadcast verification keys and signature shares during a state handover. We detail our construction below. We recall the definition of a threshold signature scheme and the $\mathsf{TH\text{–}BLS}$ scheme in Appx. G.

$\underline{\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{Setup}(1^\lambda)}$

Execute:

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

Parse $\tilde{pp}^{\mathsf{LS\text{–}DKG}} := (\mathsf{crs}', \mathbb{G}, q, g)$. Define $pp^{\mathsf{LS\text{–}DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$ and output public parameters $pp := (pp^{\mathsf{TH\text{–}BLS}}, pp^{\mathsf{LS\text{–}DKG}})$.

$\underline{\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{TKeyGen}[U](pp, t, n)}$

Let t_0 be the round in which the protocol execution begins. All parties in U do:

1. Parse $pp := (pp^{\mathsf{TH\text{–}BLS}}, pp^{\mathsf{LS\text{–}DKG}})$.
2. Run $\Pi_{\mathsf{LS\text{–}DKG}}.\mathsf{TKeyGen}(pp^{\mathsf{LS\text{–}DKG}}, t, n)$. This protocol selects a committee C^1 and outputs a public key pk to all parties in U and secret key shares sk_i^1 to each party $P_i{}^1 \in C^1$.

In round $t_0 + 4\delta$ all parties $P_i{}^1 \in C^1$ do:

3. All $P_i{}^1 \in C^1$ compute $\widehat{vk}_i^1 := g^{sk_i^1}$ and a NIZK proof π_i^1 that the verification key $\widehat{vk}_i^1$ was computed correctly[a].
4. All $P_i{}^1 \in C^1$ set $vk_i^1 := \{\widehat{vk}_i^1, \pi_i^1\}$ and initialize a signature share list $sl_i^1 := \emptyset$.

$\underline{\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{TSign}(sk_i^j, m)}$

Execute $\mathsf{TH\text{–}BLS}.\mathsf{TSign}(sk_i^j, m)$ and then add the resulting signature share σ_i^j to the list sl_i^j.

$\underline{\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{TShareVrfy}(\sigma_i^j, vk_i^j, m)}$

Parse $vk_i^j := \{\widehat{vk}_i^j, \pi_i^j\}$ and check if π_i^j is a valid proof w.r.t. $\widehat{vk}_i^j$ (i.e., check 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, output 0. Otherwise, output $\mathsf{TH\text{–}BLS}.\mathsf{TShareVrfy}(\sigma_i^j, \widehat{vk}_i^j, m)$.

$\underline{\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{TCombine}(pk, m, T)}$

Execute $\mathsf{TH\text{–}BLS}.\mathsf{TCombine}(pk, m, T)$ and output the resulting signature σ.

$\underline{\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{Verify}(pk, m, \sigma)}$

Execute $\mathsf{TH\text{–}BLS}.\mathsf{Verify}(pk, m, \sigma)$ and output the resulting bit.

$\underline{\Pi_{\mathsf{LS\text{–}TSIG}}.\mathsf{Refresh}[C_{\langle (sk_1^j, vk_1^j, sl_1^j), \cdots, (sk_n^j, vk_n^j, sl_n^j) \rangle}, U](pp)}$

Let t_j be the round in which the protocol execution begins. This protocol is executed between a committee C^j in epoch j and the universe U.

1. Run $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}\,[U]\,(pp)$ to select a committee C^{j+1} with $|C^{j+1}| = n$.

2. In round $t_j + \delta$ run $\mathsf{G\text{–}Handover}[C^j_{\langle(sk^j_1,(vk^j_1,sl^j_1)),\cdots,(sk^j_n,(vk^j_n,sl^j_n))\rangle}, U](pp)$.
   Afterwards, each $P_i^{j+1} \in C^{j+1}$ gets a refreshed secret key shares sk_i^{j+1}.

In round $t_j + 2\delta$ all parties $P_i^{j+1} \in C^{j+1}$ do:

3. 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[a] and set $vk_i^{j+1} := \{\widehat{vk}_i^{j+1}, \pi_i^{j+1}\}$.
4. Initialize a signature share list $sl_i^{j+1} := \emptyset$.

[a]The language for this proof is the same as the language L' in the $\Pi_{\mathsf{LS\text{–}DKG}}$ protocol (cf. Sec. 4).

Theorem 3. *Let $\Pi_{\mathsf{LS\text{–}DKG}}$ be a (λ, n, t, p)-secure instantiation of the* LS–DKG *protocol from Sec. 4,* TH–BLS *the non-interactive (t, n)-*TSIG *scheme as described in Appx. G, Σ_{ECPSS} a (λ, n, t, p)-secure instantiation of the ECPSS construction,* NIZK *a NIZK proof system and* CPKE *a RIND-SO secure PKE scheme. Then $\Pi_{\mathsf{LS\text{–}TSIG}}$ is a (λ, n, t, p)-secure large-scale non-interactive threshold signature scheme.*

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

Lemma 8. *The large-scale non-interactive threshold signature scheme $\Pi_{\mathsf{LS\text{–}TSIG}}$ as described in Appendix F.2 satisfies consistency.*

Proof. This lemma follows directly from the consistency property of the TH–BLS scheme (cf. Definition 20), the completeness property of the NIZK proof system and from the handover correctness of the G–Handover scheme. A proof outline of this lemma looks similar to the proof outline of Lemma 2 in Appx. D.

Lemma 9. *The large-scale non-interactive threshold signature scheme $\Pi_{\mathsf{LS\text{–}TSIG}}$ as described in Appendix F.2 satisfies the efficiency property.*

Proof. The proof for this lemma is the same as the proof for Lemma 4.

Lemma 10. *The large-scale non-interactive threshold signature scheme $\Pi_{\mathsf{LS\text{–}TSIG}}$ as described in Appendix F.2 satisfies (λ, n, t, p)-robustness.*

Proof. The proof of this lemma is similar to the proof of Lemma 3.

Lemma 11. *The large-scale non-interactive threshold threshold signature scheme $\Pi_{\mathsf{LS\text{–}TSIG}}$ as described in Appendix F.2 is (λ, n, t, p)-unforgeable.*

The proof of Lemma 11 is similar to the proof of Lemma 5 with the difference that we have to provide a reduction to the unforgeability of TH–BLS. As part of this reduction we have to show that signing oracle queries from the adversary in game $\mathsf{LSSIG\text{–}UFCMA}_{\Pi_{\mathsf{LS\text{–}TSIG}}}$ can be answered without knowing the correct secret key shares. We show briefly in Appx. G how such signing oracle answers can be simulated for the TH–BLS scheme.

F.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 any discrete-log-based non-interactive TSIG scheme can be transformed to an LS–TSIG scheme.

G The $\mathsf{TH\text{–}BLS}$ Scheme from Boldyreva [9]

G.1 Background on Digital Signature and Threshold Signature Schemes

Before we recall the $\mathsf{TH\text{–}BLS}$ scheme [9], we first recall the basic definitions of digital signature schemes and threshold signature schemes as we need these definitions to argue later about the simulatability of signing oracle queries from an adversary in game $\mathsf{LSSIG\text{–}UFCMA}_{\Pi_{\mathsf{LS\text{–}TSIG}}}$.

Definition 18 (Digital signatures). *A digital signature scheme* SIG *consists of a triple of algorithms* $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *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 19 (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 20. *A non-interactive (t, n)-threshold signature scheme* TSIG *consists of a tuple of efficient algorithms* $\mathsf{TSIG} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{TSign}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Verify})$ *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 21. *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* $\Pr[\mathsf{SIG–UFCMA}^{\mathcal{A}}_{\mathsf{TSIG}}(\lambda) = 1] \leq \nu(\lambda)$.

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

Definition 22 (Robustness). *A (t, n)-TSIG scheme satisfies robustness if for all $\lambda \in \mathbb{N}$, all $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and all PPT adversaries $\mathcal{A}$ the following holds:*

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

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

G.2 The BLS and TH–BLS Schemes

In the following we present the non-interactive threshold signature scheme from [9], 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 [10]. 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 F, this is crucial for the proof of Lemma 11. 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 [9].

Definition 23 (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 [10], before presenting its threshold variant TH–BLS as presented in [9].

G.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 [10] show that the BLS scheme is unforgeable as per Definition 19.

G.4 The TH–BLS scheme

We now recall the threshold variant of the BLS scheme, which we denote by TH–BLS. Boldyreva [9] proved the TH–BLS scheme unforgeable as per Definition 21.

$\mathsf{Setup}(1^\lambda)$*:* 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 G.3 is unforgeable as per Definition 19, then the* TH–BLS *scheme is unforgeable as per Definition 21 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 $\mathsf{SIG\text{–}UFCMA}^{\mathcal{S}}_{\mathsf{BLS}}$.

H Adding Signing Functionality to a Blockchain

An 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 it allows to sign certain messages indicating that a specific event has occurred on the blockchain. Benhamouda et al. [7] 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 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 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.