# Large-Scale Non-Interactive Threshold Cryptosystems Through Anonymity

Andreas Erwig, Sebastian Faust, and Siavash Riahi

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

**Abstract.** A $(t, n)$-public key threshold cryptosystem allows distributing the execution of a cryptographic task among a set of $n$ parties by splitting the secret key required for the computation into $n$ shares. A subset of at least $t + 1$ honest parties is required to execute the task of the cryptosystem correctly, while security is guaranteed as long as at most $t < \frac{n}{2}$ parties are corrupted. Unfortunately, traditional threshold cryptosystems do not scale well, when executed at large-scale (e.g., in the Internet-environment). In such settings, a possible approach is to select a subset of $n$ players (called a committee) out of the entire universe of $N \gg n$ parties to run the protocol. If done naively, however, this means that the adversary's corruption power does not scale with $N$ as otherwise, the adversary would be able to corrupt the entire committee. A beautiful solution for this problem is given by Benhamouda et al. (TCC 2020) who present a novel form of secret sharing, where the efficiency of the protocol is *independent* of $N$, but the adversarial corruption power *scales* with $N$. They achieve this through a novel mechanism that guarantees that parties in a committee stay anonymous until they start to interact within the protocol.

In this work, we initiate the study of large-scale threshold cryptosystems. We present novel protocols for distributed key generation, threshold encryption, and signature schemes that guarantee security in large-scale environments with complexity independent of $N$. One of our key contributions is to show how to generically transform threshold encryption and signature schemes, which are secure against static adversaries (and satisfy certain additional properties), to secure threshold cryptosystems that offer strong security in the large-scale setting.

## 1 Introduction

In a threshold cryptosystem [18, 22, 17], a secret key $sk$ is distributed among a set of $n$ parties, where each party holds a share $sk_i$ of the secret key. A subset of $t+1$ parties is needed to re-construct the secret key (or carry out the cryptographic task such as signing), while $\leq t$ parties learn nothing about the sensitive information. A threshold cryptosystem consists of two components. First, a protocol for securely generating the secret key – so-called *distributed key generation (DKG)* [41] – that enables the parties to securely generate a shared $sk$ and the corresponding public key $pk$. At the end of this protocol each party holds its secret key share $sk_i$ and is aware of the public key $pk$. Second, a distributed version of the cryptosystem, where the parties can use their shares to perform the cryptographic tasks. Two important examples of threshold cryptosystems are threshold signatures for signing messages in a distributed fashion (e.g., [9, 37]), and threshold public key encryption for distributed decryption of ciphertexts (e.g., [43, 13, 36]).

Traditionally, threshold cryptographic schemes have been considered in a setting where the number of parties $n$ is relatively small and the adversary is restricted to corrupt at most $t < n/2$ parties. This upper bound is required to achieve guaranteed output delivery [16], i.e., the adversary cannot stall the system even when behaving in an arbitrary malicious way. When we move to a large-scale Internet setting with a large user base (e.g., as prominently considered in the blockchain setting), several new challenges arise. In particular, we aim for a solution that is *scalable* when the number of users in the universe – denoted by $N$ – drastically increases. On the one hand, we aim at a protocol where only a small subset of the entire population carry out the cryptographic computation (e.g., running DKG with thousands of users is practically infeasible). On the other hand, we want that the adversary can corrupt a fraction of all parties $N$, such that its corruption power is not bounded by a small value $t$.

To address these two challenges, the recent work of Benhamouda et al. [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 share holders in this protocol. Let us provide some more details on the solution of [7]. Dynamic proactive secret sharing is a secret sharing protocol that proceeds in epochs, where the adversary is allowed to corrupt at most $t$ parties per epoch. Such an adversary is often also referred to as a *mobile adversary* [40]. To ensure security in this setting dynamic proactive secret sharing schemes deploy a so-called *handover* protocol, where the shared secret is re-shared to a new committee at the end of each epoch. While a mobile adversary can corrupt over time $\gg N$ users, the naive application of dynamic proactive secret sharing only tolerates $t < n/2$ corruptions. To circumvent this, inspired by recent advances in blockchain consensus, Benhamouda et al. introduce a novel concept of anonymity. More precisely, after a party in a committee is activated and communicates (e.g., for reconstructing the shared secret), a new committee is selected in such a way that the members of the new committee stay anonymous. This feature guarantees that an adversary cannot target the members of the small-sized committee, even for a corruption power of $\gg n$ corruptions per epoch.

The original work of Benhamouda et al. [7] considers only the question of how to store a secret in a large scale environment. This work has recently been extended in the so-called YOSO model of computation ("You only speak once") [26], which presents a general framework for committee-style secure computation with anonymity. The result of [26] is however mainly a feasibility result (we will discuss it – and in particular its limitations – in more detail in Sec. 1.2), and relies on techniques from general purpose secure multiparty computation. In this work we initiate the study of scalable threshold cryptography for large scale environments, where the adversary's corruption power per epoch can grow with $N$. We discuss our main contributions in more detail below.

### 1.1 Our Contribution

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

In our work, we first formalize the concept of discrete-log-based large-scale distributed key generation schemes and show a concrete instantiation. We follow the idea of Benhamouda et al. [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 challenge arises from the fact that distributed key generation protocols are typically highly interactive. This poses a problem in our full security setting since parties can at most speak once in order to preserve their anonymity. In particular, distributed key generation schemes typically require parties to commit to a value and, in a later round of the protocol, send the correct opening to the commitment. In our construction, we address this problem by relying on the *future broadcast* primitive as introduced in [26]. Future broadcast allows parties to secret share the opening of a commitment to a future committee which will then publish the shares at a later time. We believe that our formalization of large-scale DKG protocols and the corresponding discrete-log-based protocol can pave the way for designing DKG protocols under different assumptions such as the RSA assumption which has been mentioned as an open problem in [26].

We next consider the setting of large-scale non-interactive threshold public key encryption and signature schemes. To this end, we give a generic transformation for discrete-log-based threshold encryption and signature schemes that are secure against static corruptions (and satisfy certain properties), to achieve security against fully mobile adversaries. In general, the challenge when proving security of threshold schemes with adaptive or mobile security is that one must exhibit a simulator that simulates the view of the adversary without knowing the secret key of the scheme. That is, the simulator does not know the secret key shares of some honest parties and thereby, upon corruption of these parties, annot provide an internal state that is consistent with previous information that the adversary has seen.

We circumvent this issue by designing our protocols carefully in such a way that allows us to construct a simulator whose answers to the adversary's corruption queries are consistent with the view of the adversary. At a high level, we achieve this by postponing the publication of secret key share dependent values to the end of an epoch while still guaranteeing correctness and security of our scheme. As such, the simulator can maintain a set of secret key shares which *look* consistent from the adversary's view of the protocol execution, as long as the adversary does not corrupt more than $t$ secret key shareholders per epoch. We ensure this corruption upper bound by leveraging on the idea of Benhamouda et al. [7] of keeping the identities of secret key shareholders anonymous until the end of an epoch.

We emphasize that previous works on threshold public key crpytosystems in the proactive (mobile) setting considered an adversary who can either be (1) static or (2) adaptive, where in the former the adversary must commit to its corruption choice at the beginning of an epoch and in the latter, the adversary is allowed to corrupt parties at any time. However, in these previous works, the adversary is restricted to corrupting at most a minority of secret key shareholders and is therefore strictly weaker than a fully mobile adversary.

Finally, we give concrete instantiations of our generic transformation based on the threshold encryption scheme from Shoup and Gennaro [43] and the threshold signature scheme from Boldyreva [9]. We emphasize that our large-scale schemes are non-interactive as long as the underlying scheme is non-interactive as well.

### 1.2 Related Work

*Anonymity and You Only Speak Once Paradigm.* The most relevant previous work for us is by Benhamouda et al. [7] who introduce the notion of evolving-committee proactive secret sharing (ECPSS) which extends previous secret sharing notions in the following ways: (1) an ECPSS scheme includes a procedure to select the committee of secret shareholders, and (2) an ECPSS scheme does not *assume* that an adversary corrupts at most a minority of parties in a committee but rather provides a mechanism for *provably* achieving this. Benhamouda et al. present an instantiation of an ECPSS scheme which they prove secure against a fully mobile adversary by keeping the identities of the secret shareholders *anonymous* among a large set of parties. That is, they prove that if the adversary corrupts at most 25% of all parties in the universe, their ECPSS scheme remains secure. A recent work by Gentry et al. [25] improves Benhamouda et al.'s solution by allowing for a more powerful adversary that can corrupt up to less than 50% of all parties. We recall the definition of ECPSS schemes in Sec. 2.6 and we provide a more detailed description of the ECPSS scheme of Benhamouda et al. in Sec. 3.

Another recent work of Gentry et al. [26] generalizes the concept of computing on secrets among anonymous parties by introducing the *you only speak once* (YOSO) model and showing how to realize information theoretical and computational secure multi-party computation in this model. However, this work is mainly a feasibility result and it relies on certain idealized functionalities that currently cannot be instantiated. In a similar spirit, a recent work by Choudhuri et al. [15] 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 [15] do not analyze their solution w.r.t. a fully mobile adversary who has sufficient corruption power to potentially corrupt a majority of the universe's participants.

*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., [41, 24, 33, 14]). Recently, Gurkan et al. [29] 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. Abe and Fehr [1] and Canetti et al. [13] proposed DKG protocols which are secure against adaptive adversaries. Most related to our work is the recent work by Groth [28] which introduces a non-interactive distributed key generation protocol, which is secure against a mobile adversary who corrupts at most a minority of parties at a time. However, this work does not consider the fully mobile setting that we consider in our work.

Threshold public key cryptosystems have been extensively studied with security against static adversaries (e.g., [12, 10, 44, 43, 9]) and adaptive adversaries (e.g., [23, 13, 32, 35, 36, 37, 19]). Herzberg et al. [31] 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. Finally, there have been works in the adaptive proactive adversarial setting (e.g., [21, 13, 2]) which is the setting that is most similar to the setting we consider in this work. However, all of the above mentioned works focus on an adversary (static, adaptive or mobile) that is restricted to only corrupt at most a minority of the participants in the universe, whereas we consider a fully mobile adversary that has sufficient corruption power to corrupt a large fraction of all parties.

We provide discussion on additional related work in Appendix A.

## 2 Preliminaries

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

## 2.1 Notation

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

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

## 2.2 Communication and Adversarial Model

Our communication and adversarial model follows the model of Benhamouda et al. [7]. We assume that parties have access to an authenticated broadcast channel and a public key infrastructure (PKI). Furthermore, we consider a synchronous communication model where messages broadcasted in some round $i$ are received by all other parties in round $i + \delta$ where $\delta$ is a fixed upper bound. A blockchain network satisfies this communication model.

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

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

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

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

### 2.3 Public Key Encryption

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

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

$\mathsf{KeyGen}(1^\lambda)$*: This probabilistic algorithm takes as input a security parameter* $\lambda$ *and outputs a public key* $pk$ *and a secret key* $sk$*.*
$\mathsf{Enc}(pk, m)$*: This probabilistic algorithm takes as input a public key* $pk$ *and a message* $m$ *and outputs a ciphertext* $ct$*.*
$\mathsf{Dec}(sk, ct)$*: This algorithm takes as input a secret key* $sk$ *and a ciphertext* $ct$ *and it outputs either* $\perp$ *or a message* $m$*.*

**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. [7] for public key encryption schemes, which in turn is based on [20] and [6]. The RIND-SO notion defines a security game between a challenger and an adversary in which the challenger first samples a set of key pairs and then sends the public keys to the adversary. The adversary then chooses a distribution $\mathcal{D}$ and receives a vector of ciphertexts, which encrypt messages that are sampled from $\mathcal{D}$. The adversary can then choose some of the ciphertexts and receives the corresponding secret keys which can be used to decrypt them. Finally, for the remaining ciphertexts, the adversary either receives the correct plaintext or randomly sampled messages from $\mathcal{D}$, conditioned on the opened plaintext[1].

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

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

1. $(\mathbf{sk}, \mathbf{pk}) \coloneqq (sk_i, pk_i)_{i \in [n]} \leftarrow (\mathsf{Gen}(1^\lambda))_{i \in [n]}$
2. $(\mathcal{D}, \mathsf{Resamp}_{\mathcal{D}}, state_1) \leftarrow \mathcal{A}(\mathbf{pk})$
3. $\mathbf{m} \coloneqq (m_i)_{i \in [n]} \leftarrow \mathcal{D}$

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

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

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

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

**Definition 4.** *A public key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is anonymous if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\nu$ in the security parameter $\lambda$ 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*

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

### Threshold Public Key Encryption

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

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

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

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

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

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

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

*Consistency* A $(t, n) - \mathsf{TPKE}$ 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. Decryption consistency: 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. Reconstruction consistency: For any message $m$, any label $L$, any ciphertext $ct \leftarrow_\$ \mathsf{TEnc}(pk, m, L)$ and any set $T = \{ct_1, \cdots, ct_t\}$ of valid decryption shares $ct_i \leftarrow \mathsf{TDec}(sk_i, ct, L)$ with $sk_i$ being $t$ distinct secret key shares, it must hold that $\mathsf{TCombine}(T, ct) = m$.

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

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

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

### 2.4 Non-Interactive Zero-Knowledge

We now recall the definition of a non-interactive zero-knowledge (NIZK) proof of knowledge which has first been introduced in [8].

**Definition 7.** *A NIZK proof of knowledge with respect to a polynomial-time recognizable binary relation* $\mathsf{R}$ *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* $\mathsf{crs}$*; (ii)* $\mathsf{Prove}(\mathsf{crs}, (Y, y))$ *outputs a proof* $\pi$ *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.* $\mathsf{R}$ *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* $\mathsf{Setup}'$ *and* $\mathsf{S}$*, where* $\mathsf{Setup}'(1^\lambda)$ *on input the security parameter, outputs a pair* $(\widetilde{\mathsf{crs}}, \tau)$ *with* $\tau$ *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.*

### 2.5 Secret Sharing

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

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

1. Secrecy: $\mathcal{A}$ learns no information about $s$, if the dealer is uncorrupted.
2. Reconstruction: Any set of honest secret shareholder of size $\geq t$ can reconstruct the original secret $s$.

Shamir's secret sharing [42] is the most prominent $(t, n)$-secret sharing scheme and we will recall it here briefly. Let $q$ be a prime and let $1 \leq t \leq n < q$. The dealer chooses a secret $s \in \mathbb{Z}_q$ and a random polynomial $F(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$ where $a_0 = s$. For $1 \leq i \leq n$, the dealer computes $s_i = F(i)$ and sends $s_i$ to secret shareholder $P_i$. A set $S$ of honest shareholders with $|S| \geq t$ can reconstruct $s$ via interpolation. More concretely, for any $i \in \mathbb{Z}_q$ and any $j \in S$ there exist lagrange coefficients $l_{i,j}$ such that $F(i) = \sum_{j \in S} l_{i,j} s_j$.

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

### 2.6 Evolving-Committee Proactive Secret Sharing

Recently, Benhamouda et al. [7] introduced the notion of evolving-committee proactive secret sharing (ECPSS), which is defined w.r.t. a universe of $N$ parties and parameters $t \leq n < N$. Similar to dynamic proactive secret sharing (see Appendix A), ECPSS allows to share a secret to a committee of parties and to periodically exchange the secret shareholders of the committee. ECPSS further extends previous secret sharing notions by providing a procedure that selects a size $n$ committee from all $N$ parties and by proving that a fully mobile adversary with corruption power $p$ s.t. $p \cdot N > t-1$ can at most corrupt $t-1$ shareholders at a time.

We now recall the definition of an ECPSS scheme as given in [7].

**Definition 8 (ECPSS).** *An evolving-committee proactive secret sharing scheme with parameters $t \leq n < N$ consists of the following procedures:*

**Setup:** *Optional procedure that provides the initial state for a universe of $N$ parties.*

**Sharing:** *Shares a secret $s$ among an initial committee of size $n$.*

**Committee-Selection:** *This procedure is executed among all $N$ parties and selects the next $n$-party committee.*

**Handover:** *This procedure is executed among $n$ parties, takes the output of committee-selection and the current shares and re-shares them among the next committee.*

**Reconstruction:** *Takes $t$ or more shares from the current committee and reconstructs the secret $s$ or outputs $\perp$ on failure.*

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

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

We call an ECPSS scheme $(\lambda, n, t-1, p)$-secure, if it satisfies the secrecy and reconstruction property w.r.t. a security parameter $\lambda$, 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. [7]

In this section, we recall the scalable ECPSS scheme with security against a fully mobile adversary as presented by Benhamouda et al. [7] as it constitutes an important building block of our work. We denote the scheme by $\Sigma'_{\mathsf{ECPSS}}$. The main idea behind the scheme is to achieve scalability by choosing small committees of secret shareholders whose size $n$ does not depend on the total set of parties $N$. However, due to the small committee size, a fully mobile adversary, whose corruption power is proportional to $N$ instead of $n$, is able to corrupt all members of the committee, thereby compromising security. Benhamouda et al.'s idea to tackle this issue is to keep the identity of the committee members hidden from the adversary, i.e., the committee members should be anonymous until they have to communicate for the first time.

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

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

Upon broadcasting these values the members of the nominating committee erase their internal state as their identity is now known to the adversary. All $N$ parties can now check if they were selected to the next holding committee by trying to decrypt the published ciphertexts. At this point, the previous-epoch holding committee (which holds the shares of the secret) encrypts (a sharing of) the secret shares under the ephemeral public keys and broadcasts the resulting ciphertexts. Again, as broadcasting compromises anonymity, the parties in the previous holding committee first erase their internal states. We refer the reader to [7] for the full description of the $\Sigma'_{\mathsf{ECPSS}}$ scheme.

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

The Setup procedure works as follows.

**$\mathsf{Setup}(1^\lambda)$:** On input a security parameter $\lambda$, this procedure chooses a $\lambda$-bit prime $q$ and executes $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ to generate the common reference string $\mathsf{crs}$ of a $\mathsf{NIZK}$ proof system (cf. Def. 7). The procedure outputs public parameters $pp := (\mathsf{crs}, q)$.[2]

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

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

**Select procedure:**

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

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

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

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

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

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

**Handover procedure:**

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

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

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

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

[a] For simplicity, we assume that Select is executed as part of the Handover procedure. This is different from the original protocol description in [7], but does not affect the functionality or security of the protocol.

Finally, we note that Benhamouda et al.'s solution can be instantiated with different parameters. For instance, it has been shown to be $(128, 889, 425, 0.05)$-secure or $(128, 38557, 19727, 0.25)$-secure. A recent work by Gentry et al. [25] showed how to improve these parameters by introducing a new Select mechanism that allows $\Sigma'_{\mathsf{ECPSS}}$ to remain secure for any $p < \frac{1}{2}$. More concretely, for $p = 0.25$ they only require a holding committee of size 680 and for $p = 0.40$ a committee of size 3500, which are significant improvements compared to the original parameters from [7].

*Our (slightly) modified $\Sigma'_{\mathsf{ECPSS}}$ Construction.* In our work, we consider a slightly modified version of the Handover and Select procedures. Instead of erasing the long-term secret key $sk_i$ in step 4 of the Handover procedure, $sk_i$ is already erased at the end of step 5 of the Select procedure. It is easy to see that this change does not affect the functionality of the $\Sigma'_{\mathsf{ECPSS}}$ scheme as $sk_i$ is never used after step 5 of the Select procedure. For the rest of this paper we use this modified version and denote it by $\Sigma_{\mathsf{ECPSS}}$. This subtle but important modification is essential for the security proofs of our large-scale distributed key generation protocol and our generic large-scale threshold public key encryption scheme.

## 4 Large-Scale Distributed Key Generation

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

### 4.1 Model

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

A large-scale $(t, n)$-distributed key generation protocol (LS–DKG) differs from the above notion of distributed key generation protocols in the sense that it is defined w.r.t. a universe of parties $U$ from which a committee of parties $C$ of size $n$ with $n < |U|$ is selected. This committee can then execute the key generation protocol. In terms of security, an 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 LS–DKG protocol.

**Definition 9.** *A large-scale $(t, n)$-distributed key generation protocol* (LS–DKG) *is run among a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple* LS–DKG = (Setup, TKeyGen) *of an efficient algorithm and a protocol which are defined as follows:*

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

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

In this work, we focus on discrete-log-based threshold cryptosystems, i.e., threshold schemes that operate over a cyclic group $\mathbb{G}$ of prime order $q$ and output secret/public key pairs of the form $(x, g^x)$, where $x \in \mathbb{Z}_q$ and $g$ is a generator of $\mathbb{G}$. We now present the correctness properties of an LS–DKG scheme, which are similar to the correctness properties for DKG schemes in the discrete-log setting as introduced by Gennaro et al. [24].

**Correctness:** An LS–DKG protocol must satisfy the following three correctness properties.

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

In addition to correctness, an $\mathsf{LS–DKG}$ scheme must satisfy the following secrecy property similar to the definition of Gennaro et al. [24].

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

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

### 4.2 Construction

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

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

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

*Overview of our construction.* Our LS–DKG protocol (which we denote throughout this paper by $\Pi_{\mathsf{LS-DKG}}$) follows the same three-phase framework of previous DKG protocols, while addressing the challenges of being executed in the YOSO model. At the core of our protocol lies the $\Sigma_{\mathsf{ECPSS}}$ scheme which allows selecting a committee of anonymous parties and passing on secret information from one committee to another. We further employ a non-interactive zero-knowledge (NIZK) proofs such that honest behavior can be proven without requiring any interaction, and we carefully design our protocol in such a way that parties can commit to a value and hand-over the corresponding opening to the next committee which broadcasts the opening at a later time. This latter technique has first been introduced by Gentry et al. [26] who called it *future broadcast* and who pointed out its importance in the YOSO model. In the following, we provide a more detailed description of our solution.

The starting point of our protocol is the *committee selection phase*, during which two anonymous committees $C$ and $C'$ of size $n$ are selected via executions of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure. The protocol then proceeds to the *share distribution phase*, during which each party $P_i \in C$ chooses two random values $s_i$ and $r_i$ which it shares to committee $C'$ via the techniques of the $\Sigma_{\mathsf{ECPSS}}$ scheme. Additionally, $P_i$ broadcasts the value $g^{s_i+r_i}$ and finally, in order to make the sharing publicly verifiable without the need for any interaction, $P_i$ broadcasts a NIZK proof that proves honest behavior of $P_i$ during the share distribution phase.

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

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

reconstructing the values $g^{r_i}$ of qualified parties. This requires parties to speak and therefore to handover the protocol execution to a new committee $C''$.

In the final phase of the protocol, the *public key reconstruction phase*, all parties in $U$ can use the unblinded values $g^{s_i}$ for all $P_i \in Qual$ to compute the public key as $pk = \prod_{i \in Qual} g^{s_i}$.

We now give a formal description of our LS–DKG protocol $\Pi_{\mathsf{LS\text{-}DKG}}$ w.r.t. $\Sigma_{\mathsf{ECPSS}}$, a NIZK proof system NIZK and a public key encryption scheme PKE.

$\mathsf{Setup}(1^\lambda)$**:** On input a security parameter $\lambda$, execute $pp^{\mathsf{ECPSS}} \leftarrow \Sigma_{\mathsf{ECPSS}}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{crs}_1 \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$. Parse $pp^{\mathsf{ECPSS}} := (\mathsf{crs}_2, q)$ and set $\mathsf{crs} := \mathsf{crs}_1 || \mathsf{crs}_2$[4]. Choose a group $\mathbb{G}$ of prime order $q$ with generator $g$ such that the dlog problem is hard in $\mathbb{G}$. Output public parameters $pp^{\mathsf{LS\text{-}DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$.

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

In the following, we denote by PKE the public key encryption scheme that generates the ephemeral keys of the $\Sigma_{\mathsf{ECPSS}}$ scheme.

**Input:** $pp^{\mathsf{LS\text{-}DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$ and integers $t, n \in \mathbb{N}$, s.t. $n \geq 2t + 1$.

**Committee Selection Phase:**

1. During the committee selection phase, all parties in $U$ execute $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ twice to select committees $C$ and $C'$ where $|C| = |C'| = n$. Note that after the execution of $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ each party $P_j{}' \in C'$ is associated to an ephemeral public key $\mathsf{epk}_j$ which is known to all parties in $C$.

**Share Distribution Phase:**

2. Each party $P_i \in C$ does the following:
   (a) Choose $s_i \leftarrow_\$ \mathbb{Z}_q$ and $r_i \leftarrow_\$ \mathbb{Z}_q$.
   (b) Choose random degree-$t$ polynomials $F_i^s(x) = a_{i,0}^s + a_{i,1}^s x + \cdots + a_{i,t}^s x^t \in \mathbb{Z}_q[x]$ with $a_{i,0}^s = s_i$ and $F_i^r(x) = a_{i,0}^r + a_{i,1}^r x + \cdots + a_{i,t}^r x^t \in \mathbb{Z}_q[x]$ with $a_{i,0}^r = r_i$.
   (c) Compute shares $s_{i,j} := F_i^s(j)$ and $r_{i,j} := F_i^r(j)$ for $j \in [n]$.
   (d) For all $j \in [n]$ compute $c_{i,j}^s \leftarrow \mathsf{PKE}.\mathsf{Enc}_{\mathsf{epk}_j}(s_{i,j})$ and $c_{i,j}^r \leftarrow \mathsf{PKE}.\mathsf{Enc}_{\mathsf{epk}_j}(r_{i,j})$.
   (e) Compute $A_i = g^{s_i + r_i}$.
   (f) Compute a NIZK proof $\pi_i$ for the statement that the decrypted values of all $\{c_{i,j}^s\}_{j \in [n]}$ and all $\{c_{i,j}^r\}_{j \in [n]}$ are shares of two degree-$t$ polynomials s.t. the sum of the polynomials' free terms equals the dlog of $A_i$.
   (g) Erase all secret values, i.e., shares $s_{i,j}$ and $r_{i,j}$, polynomials $F_i^s$ and $F_i^r$ and values $s_i$ and $r_i$.
   (h) Broadcast $(A_i, \pi_i, \{c_{i,j}^s\}_{j \in [n]}, \{c_{i,j}^r\}_{j \in [n]})$.

**Qualification Phase:**

3. Let $Qual = \emptyset$. Upon receiving the tuples $(A_i, \pi_i, \{c_{i,j}^s\}_{j \in [n]}, \{c_{i,j}^r\}_{j \in [n]})$ for $i \in [n]$, all parties $P_j{}' \in C'$ check if $\pi_i$ is valid and if so, store $i$ in $Qual$ until $|Qual| = t + 1$.[a]
4. For all $i \in Qual$ do the following:

[4] For simplicity we only use the concatenated CRS in our protocols and proofs, and assume that each NIZK proof system can extract the correct CRS and uses it for the proof generation.

(a) Compute $s_{i,j} \leftarrow \mathsf{PKE.Dec}_{\mathsf{esk}_j}(c^s_{i,j})$.
(b) Compute $r_{i,j} \leftarrow \mathsf{PKE.Dec}_{\mathsf{esk}_j}(c^r_{i,j})$.

5. Compute the secret key share $sk'_j \in \mathbb{Z}_q$ as $sk'_j = \sum_{i \in Qual} s_{i,j}$.
6. Compute $r'_j = \sum_{i \in Qual} r_{i,j}$ and $R'_j = g^{r'_j}$.
7. Compute a NIZK proof $\pi'_j$ that proves that the dlog of $R'_j$ is the sum of the decryptions of $c^r_{i,j}$ under $\mathsf{esk}_j$.
8. Erase values $\{s_{i,j}, r_{i,j}\}_{i \in Qual}$ and $r'_j$.
9. Execute $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}[C'_{\langle sk'_1, \cdots, sk'_n \rangle, U}](pp^{\mathsf{ECPSS}})$. This execution selects a committee $C''$ with $|C''| = n$ s.t. each $P_k'' \in C''$ learns a refreshed secret key share $sk''_k$.
10. Broadcast $(R'_j, \pi'_j)$.

**Public Key Reconstruction Phase:**

11. Let $Rand = \emptyset$. Upon receiving $\{(R'_j, \pi'_j)\}_{j \in [n]}$, all parties in $U$ compute the set $Qual$ as above and check for all $j \in [n]$ if $\pi'_j$ is valid and if so store $R'_j$ in $Rand$ until $|Rand| = t + 1$.
12. Compute the public key $pk \in \mathbb{G}$ as $pk = \left(\prod_{k \in Qual} A_k\right) \cdot \left(\prod_{j \in Rand} R'^{l_j}_j\right)^{-1}$ where $l_j$ are the corresponding lagrange coefficients.

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

**Theorem 1.** *Let the discrete-log assumption hold in $\mathbb{G}$, let* NIZK *be a non-interactive zero-knowledge proof system as per Def. 7, $\Sigma_{\mathsf{ECPSS}}$ be a $(\lambda, n, t, p)$-secure instantiation of the evolving-committee proactive secret sharing scheme as presented in Sec. 2.6 and* PKE *be a RIND-SO secure public key encryption scheme. Then the protocol $\Pi_{\mathsf{LS-DKG}}$ from Sec. 4.2 is a $(\lambda, n, t, p)$-secure large-scale $(t, n)$-distributed key generation scheme.*

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

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

We provide the proof of Lemma 1 in Appendix B.

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

*Proof Sketch:* We provide the full proof of Lemma 2 in Appendix B, and only give the main ideas of the proof here. For the proof we need to construct a simulator which on input a public key $pk$, can simulate an execution of the $\Pi_{\mathsf{LS-DKG}}$ protocol to a fully mobile adversary $\mathcal{A}$ in such a way that (1) the simulated execution is computationally indistinguishable from a real execution of $\Pi_{\mathsf{LS-DKG}}$ from $\mathcal{A}$'s point of view and (2) the public key that is output by the simulated execution is $pk$.

At a high level, the main challenge in the simulation is the following. The simulator, on behalf of honest parties $P_i$ in committee $C$, has to first broadcast the elements $g^{s_i+r_i}$ and send a sharing of $r_i$ and later adjust these values depending on the set of qualified parties and the values chosen by the adversary such that the following condition holds:

$$pk = \prod_{k \in Qual} g^{s_k+r_k} \cdot \left( \prod_{j \in Rand} g^{{r'_j}^{l_j}} \right)^{-1}. \tag{1}$$

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

The main idea behind our simulation is to use the fact that the adversary corrupts at most a minority of parties in each committee. More concretely, this fact allows to make the following two observations: (1) there must exist at least one honest party in $Qual$ since the set of qualified parties $Qual$ consists of $t+1$ parties, and (2) the simulator has sufficient information to reconstruct all values $r'_j$ for all parties ${P_j}' \in C'$ and thereby to learn all elements $R'_j$. This allows the simulator to first compute a value $R'$ s.t. $pk = \prod_{k \in Qual} g^{s_k+r_k} \cdot R'$. The simulator then adjusts the elements $R'_j$ for some honest parties ${P_j}'$ in $C'$ such that for any subset $S \subset \{R'_1, \cdots, R'_n\}$ with $|S| = t+1$ it is possible to reconstruct $R'$ via interpolation in the exponent. The simulator thereby ensures that Eq. (1) holds. The simulator can then broadcast the elements $R'_j$ along with a simulated NIZK proof. The fact that there is at least one honest party in the set $Qual$ ensures that the adversary cannot distinguish the simulated values $R'_j$ from the real ones.

## 5 Large-Scale Threshold Public Key Encryption from Discrete-Log-Based TPKE Schemes

We are now ready to present our generic transformation from a discrete-log-based statically secure threshold PKE scheme (TPKE) according to Definitions 5 and 6, to a large-scale non-interactive threshold PKE scheme (LS–TPKE) through anonymization techniques similar to what we have seen in the previous sections. We first start by introducing the notion of large-scale threshold public key encryption and then provide our generic transformation from a discrete-log-based TPKE to an LS–TPKE scheme using the evolving-committee proactive secret sharing solution $\Sigma_{\mathsf{ECPSS}}$ from Section 3, the large-scale distributed key generation scheme $\Pi_{\mathsf{LS\text{-}DKG}}$ from the previous section and a NIZK proof system. In

Appendix D, we show a concrete instantiation of our generic transformation based on the TPKE scheme by Shoup and Gennaro [43].

### 5.1 Large-Scale Non-Interactive Threshold Public Key Encryption

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

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

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

**Definition 10.** *A large-scale non-interactive $(t, n)$-threshold public key encryption scheme* (LS–TPKE) *is defined w.r.t. a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple* LS–TPKE = (Setup, TKeyGen, TEnc, TDec, TShareVrfy, TCombine, Refresh) *of efficient algorithms and protocols which are defined as follows:*

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

$\mathsf{Refresh}[C_{\langle sk_1,\cdots,sk_n\rangle}, U](pp)$*: This is a protocol involving a committee* $C$ *with* $|C| = n$ *and the universe of parties* $U$*. Each* $P_i \in C$ *takes as secret input a secret key share* $sk_i$ *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* $sk'_i$*.*

We will now define the properties that an LS–TPKE must satisfy, namely *Consistency* and *CCA-Security*. 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 $sk_i^j$ and decryption shares as $ct_i^j$). We require an LS–TPKE scheme to satisfy the following consistency properties.

*Consistency* A $(t, n)$ − LS–TPKE scheme must fulfill the following two consistency properties. 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

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

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

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

2. Reconstruction consistency: For any message $m$, any label $L$, any ciphertext $ct \leftarrow \mathsf{TEnc}(pk, m, L)$ and any set $T = \{ct_1^j, \cdots, ct_{t+1}^j\}$ of valid decryption shares where each decryption share is computed as $ct_i^j \leftarrow \mathsf{TDec}(sk_i^j, ct, L)$ with $t+1$ distinct secret key shares $sk_i^j$, it holds that $\mathsf{TCombine}(T, ct) = m$.

*CCA-Security* In the following, we give the definition of chosen-ciphertext security for a $(t, n)$−LS–TPKE scheme considering an efficient fully mobile adversary $\mathcal{A}$ with corruption power $p$ s.t. $p \cdot |U| > t$. The state-of-the-art communication model for designing and analyzing protocols that are secure against such a strong adversary, is the YOSO model, in which committee members can speak at most once before having to hand over their secret key share to the next committee. This has the following interesting implications on the definition of the security game as compared to the notion of CCA-security for a TPKE scheme (cf. Appendix A). First, upon a decryption oracle query, the game has to output decryption shares on behalf of honest secret key shareholders. However, this requires these shareholders to "speak". As each party should speak only once, outputting decryption shares must be executed simultaneously with a refreshing of the secret key shares to the next committee. 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 $\lambda$:

1. The game executes $\mathsf{Setup}(1^\lambda)$ and obtains public parameters $pp$, which it forwards to the adversary $\mathcal{A}$. For each epoch $j \geq 0$, the game maintains a set of corrupted parties $B^j$ which is initialized as $B^j := \emptyset$.
2. The adversary $\mathcal{A}$ is given access to the following corruption oracle:
   – **Corruption oracle**: On input an index $i \in [N]$, the game checks if $\left\lfloor \frac{|B^j|+1}{|U|} \right\rfloor \leq p$. If so, $\mathcal{A}$ receives the internal state of party ${P_i}^j$ and the game sets $B^j \leftarrow B^j \cup \{{P_i}^j\}$.
3. The game executes $\mathsf{TKeyGen}[U](pp, t, n)$. 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 oracles:
   – **Refresh oracle**: On input a set $NB^j \subseteq B^j$, the game executes the protocol $\mathsf{Refresh}[C^j_{\langle sk_1^j, \cdots, sk_n^j \rangle}, U](pp)$ and sets $B^{j+1} \leftarrow B^j \setminus NB^j$.
   – **Decryption oracle**: On input a set $NB^j \subseteq B^j$ and a set of ciphertexts $CT^j$ with a set of associated labels $AL^j$, the game computes $ct_{i,k}^j \leftarrow \mathsf{TDec}(sk_i^j, ct_k^j, L_k^j)$ for all $ct_k^j \in CT^j$ and $L_k^j \in AL^j$ for all parties ${P_i}^j \in C^j \setminus B^j$. Then, the oracle calls the Refresh oracle on input $NB^j$ and finally the game returns all tuples $(\{ct_{i,k}^j\}_{k \in |CT^j|}, vk_i^j)$ to $\mathcal{A}$.
5. Eventually, $\mathcal{A}$ chooses two messages $m_0, m_1$ with $|m_0| = |m_1|$ and a label $L$ and sends them to the game. The game chooses a random bit $b \leftarrow_\$ \{0,1\}$ and sends $ct' \leftarrow_\$ \mathsf{TEnc}(pk, m_b, L)$ to $\mathcal{A}$.
6. $\mathcal{A}$ is allowed to make queries as described in steps 2. and 4. with the exception that it cannot make a decryption query on ciphertext $ct'$.
7. Eventually, $\mathcal{A}$ outputs a bit $b'$. The game outputs 1 if $b' = b$ and 0 otherwise.

**Definition 11.** *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 $(\lambda, n, t, p)$ s.t. $p \cdot |U| > t$ if for every fully mobile PPT adversary $\mathcal{A}$ with corruption power $p$ there exists a negligible function $\nu$ in the security parameter $\lambda$, such that*

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

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

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

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

### 5.2 Generic Transformation from TPKE to LS–TPKE

We are now ready to show our generic transformation of discrete-log-based non-interactive threshold encryption schemes secure against static adversaries to a large-scale non-interactive threshold encryption scheme. We will show in Appx. E how our techniques can be similarly applied to non-interactive threshold signature schemes. At a high level, we utilize our LS–DKG protocol $\Pi_{\mathsf{LS-DKG}}$ as presented in Sec. 4, the evolving-committee proactive secret sharing scheme $\Sigma_{\mathsf{ECPSS}}$ as described in Sec. 3 and a NIZK proof system to transform a non-interactive $(t, n)$-threshold encryption scheme secure against static adversaries TPKE to a large-scale non-interactive $(t, n)$-threshold encryption scheme secure against fully mobile adversaries LS–TPKE.

**Properties of the TPKE scheme.** For our transformation, we require the underlying TPKE scheme to fulfill the three properties Has-DKG, Sk-To-Vk and Has-Sim. The first property Has-DKG states that the TPKE scheme must be compatible with the $\Pi_{\mathsf{LS-DKG}}$ protocol, i.e., the public key and secret key shares as output by $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ can be used in TPKE. The property Sk-To-Vk states that a secret key share and its corresponding verification key must form a discrete-log instance in the TPKE scheme. Finally, the property Has-Sim says that for TPKE there must exist a simulator that can simulate the TPKE–CCA game (cf. Appendix A) to a static adversary on input a public key, verification keys and $t$ secret key shares.

We now formally define the required properties for our transformation. Let $U$ be a universe of parties. Let $\lambda \in \mathbb{N}$ be the security parameter, $pp^{\mathsf{LS-DKG}} \leftarrow \Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}(1^\lambda)$, $pp^{\mathsf{TPKE}} \leftarrow \mathsf{TPKE.Setup}(1^\lambda)$ and $t, n \in \mathbb{N}$ s.t. $1 \leq t \leq n$. Note that $pp^{\mathsf{LS-DKG}}$ can be parsed as $pp^{\mathsf{LS-DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$.

1. Has-DKG: For all
$$(pk, \{sk_i^1\}_{i\in[n]}) \leftarrow \Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}[U](pp^{\mathsf{LS-DKG}}, t, n)$$
and all
$$(pk', \{vk'_i\}_{i\in[n]}, \{sk'_i\}_{i\in[n]}) \leftarrow \mathsf{TPKE.KeyGen}(pp^{\mathsf{TPKE}}, t, n),$$
the distributions of the tuples $(pk, \{sk_i^1\}_{i\in[n]})$ and $(pk', \{sk'_i\}_{i\in[n]})$ are indistinguishable[5].

2. Sk-To-Vk: For all $(pk', \{vk'_i\}_{i\in[n]}, \{sk'_i\}_{i\in[n]}) \leftarrow \mathsf{TPKE.KeyGen}(pp^{\mathsf{TPKE}}, t, n)$, it holds that $vk'_i = g^{sk'_i}$.

3. Has-Sim: There exists an efficient algorithm $\mathcal{S}$ that behaves as follows: Let $(pk, \{vk'_j\}_{j\in[n]}, \{sk'_i\}_{i\in[n]})$ be a tuple generated by the TPKE.KeyGen procedure, i.e., $(pk, \{vk'_j\}_{j\in[n]}, \{sk'_i\}_{i\in[n]}) \in \mathsf{TPKE.KeyGen}(pp^{\mathsf{TPKE}}, t, n)$. Then the

[5] This property implies that $\Pi_{\mathsf{LS-DKG}}$ and TPKE operate over the same group $\mathbb{G}$ with the same generator $g$.

algorithm $\mathcal{S}$ on input the public key $pk$, verification keys $\{vk'_j\}_{j\in[n]}$ and only $t$ of the secret key shares $\{sk'_i\}_{i\in[t]}$, can simulate the game $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ as presented in Def. 6 w.r.t. $(pp^{\mathsf{TPKE}}, pk, \{vk'_j\}_{j\in[n]})$ to an efficient static adversary $\mathcal{A}$ such that $\mathcal{A}$ cannot distinguish the simulation from the execution of the real game except with negligible probability in $\lambda$.

**Construction** Let us now provide our generic construction of a large-scale threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}} = (\mathsf{Setup}, \mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{TDec}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Refresh})$ with CCA-security using the large-scale distributed key generation scheme $\Pi_{\mathsf{LS-DKG}} = (\mathsf{Setup}, \mathsf{TKeyGen})$ as described in Sec. 4, the ECPSS scheme $\Sigma_{\mathsf{ECPSS}} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Select}, \mathsf{Handover}, \mathsf{Rec})$ as presented in Sec. 3, a threshold public key encryption scheme $\mathsf{TPKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{TEnc}, \mathsf{TShareVrfy}, \mathsf{TCombine})$ secure against static adversaries and a NIZK proof system $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as per Def. 7. We detail the generic construction below.

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

$$pp^{\mathsf{TPKE}} \leftarrow \mathsf{TPKE.Setup}(1^\lambda), pp^{\mathsf{LS-DKG}} \leftarrow \Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}(1^\lambda)$$
$$\mathsf{crs}_1 \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$$

Recall that $pp^{\mathsf{LS-DKG}}$ can be parsed as $pp^{\mathsf{LS-DKG}} := (\mathsf{crs}_2, \mathbb{G}, q, g)$. Output public parameters $pp := (pp^{\mathsf{TPKE}}, pp^{\mathsf{LS-DKG}}, \mathsf{crs}_1)$.

$\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}(pp, t, n)$: On input public parameters $pp$ and two integers $t, n \in \mathbb{N}$ s.t. $n \geq 2t+1$, this protocol parses $pp := (pp^{\mathsf{TPKE}}, pp^{\mathsf{LS-DKG}}, \mathsf{crs}_1)$ and calls the $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}(pp^{\mathsf{LS-DKG}}, t, n)$ procedure, which selects a committee $C^1$ with $|C^1| = n$ and outputs a public key $pk$ to all parties in $U$ and secret key shares $sk_i^1$ to each party ${P_i}^1 \in C^1$. Additionally, all ${P_i}^1$ compute $vk_i^{1'} := g^{sk_i^1}$ and a NIZK proof $\pi_i^1$ proving that $vk_i^{1'}$ was computed correctly[6]. ${P_i}^1$ then sets the verification key $vk_i^1 := \{vk_i^{1'}, \pi_i^1\}$.

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

$\Pi_{\mathsf{LS-TPKE}}.\mathsf{TDec}(sk_i^j, ct, L)$: This is the $\mathsf{TPKE.TDec}$ procedure.

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

[6] We note that $\pi_i^1$ can be efficiently computed and verified since its witness is $sk_i^1$ and its statement consists of the ciphertexts that are broadcast during the $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ procedure.

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

$\Pi_{\mathsf{LS-TPKE}}.\mathsf{Refresh}[C^{j-1}_{\langle sk_1^{j-1},\cdots,sk_n^{j-1}\rangle}, U](pp)$: This protocol is executed between a committee $C^{j-1}$ in epoch $j-1$ and the universe $U$, where each $P_i^{j-1} \in C^{j-1}$ receives as input a secret key share $sk_i^{j-1}$ and each party $P_k \in U$ receives as input $pp := (pp^{\mathsf{TPKE}}, pp^{\mathsf{LS-DKG}}, \mathsf{crs}_1)$. The protocol first runs $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ which selects a committee $C^j$ with $|C^j| = n$ and outputs refreshed secret key shares $sk_i^j$ to each $P_i^j \in C^j$. Additionally, all $P_i^j \in C^j$ compute $vk_i^{j'} := g^{sk_i^j}$, generate a NIZK proof $\pi_i^j$ that the verification key was computed correctly[7] and set $vk_i^j := \{vk_i^{j'}, \pi_i^j\}$.

**Theorem 2.** *Let $\Pi_{\mathsf{LS-DKG}}$ be a $(\lambda, n, t, p)$-secure instantiation of the large-scale $(t, n)$-distributed key generation protocol from Sec. 4,* $\mathsf{TPKE}$ *a non-interactive $(t, n)$-threshold public key encryption scheme which is secure against chosen-ciphertext attacks with static corruptions according to Def. 6, $\Sigma_{\mathsf{ECPSS}}$ a $(\lambda, n, t, p)$-secure instantiation of the evolving-committee proactive secret sharing scheme as presented in Sec. 3 and* $\mathsf{NIZK}$ *a non-interactive zero-knowledge proof system as per Def. 7. Assume that* $\mathsf{TPKE}$ *satisfies the properties* $\mathsf{Has\text{-}DKG}$*,* $\mathsf{Sk\text{-}To\text{-}Vk}$*, and* $\mathsf{Has\text{-}Sim}$*, then $\Pi_{\mathsf{LS-TPKE}}$ is a $(\lambda, n, t, p)$-secure large-scale non-interactive $(t, n)$-threshold public key encryption scheme.*

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

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

*Proof.* This lemma follows directly from the consistency properties of the $\mathsf{TPKE}$ scheme, the properties $\mathsf{Has\text{-}DKG}$ and $\mathsf{Sk\text{-}To\text{-}Vk}$, the completeness property of the $\mathsf{NIZK}$ proof system and from the (robust) reconstruction property of the $\Sigma_{\mathsf{ECPSS}}$ scheme. We provide a proof outline for Lemma 3 in Appendix C.

**Lemma 4.** *The large-scale non-interactive $(t, n)$-threshold public key encryption scheme $\Pi_{\mathsf{LS-TPKE}}$ as described in Sec. 5.2 is secure against chosen-ciphertext attacks w.r.t. parameters $(\lambda, n, t, p)$.*

*Proof Sketch.* We provide the full formal proof of Lemma 4 in Appendix C. We provide now a high level proof sketch that summarizes the main ideas of our proof. At a high level, we show that if there exists a fully mobile adversary $\mathcal{B}$ with corruption power $p$ who can win game $\mathsf{LSTPKE\text{–}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ with non-negligible

[7] We note that $\pi_i^j$ can be efficiently computed and verified since its witness is $sk_i^j$ and its statement consists of the ciphertexts that are broadcast during the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure.

probability, then there exists an efficient static adversary $\mathcal{A}$ who can use $\mathcal{B}$ to win its own game $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ (cf. Def. 6) with non-negligible probability. Therefore, we show how $\mathcal{A}$ can simulate game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ to $\mathcal{B}$ in such a way that the simulation is indistinguishable from a real execution to $\mathcal{B}$ and how $\mathcal{A}$ can use $\mathcal{B}$'s output bit $b'$ to win its own game.

The first step of our proof, similar to the proof of Lemma 2, is to show that $\mathcal{B}$ corrupts at most $t$ secret key shareholders (i.e., committee members) per epoch. We show this by providing reductions to the secrecy properties of the $\Pi_{\mathsf{LS-DKG}}$ and $\Sigma_{\mathsf{ECPSS}}$ schemes.

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 $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$. $\mathcal{A}$ embeds $pk$ in game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ by executing the simulator code of the $\Pi_{\mathsf{LS-DKG}}$ scheme (cf. Fig. 1) on input $pk$. After this execution, $\mathcal{A}$ knows the secret key shares of all honest and malicious parties. Note however that, according to the simulation strategy for the $\Pi_{\mathsf{LS-DKG}}$ scheme, these secret key shares are merely random values in $\mathbb{Z}_q$ that are independent of $pk$. The main idea of the proof is now to show that $\mathcal{A}$ can simulate game $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ to the adversary $\mathcal{B}$, without $\mathcal{B}$ noticing that the committee members hold a sharing of a random value.

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

*Remark 1.* We note that the scheme $\Pi_{\mathsf{LS-TPKE}}$ inherits the security guarantee of the underlying $\mathsf{TPKE}$ scheme, i.e., if the $\mathsf{TPKE}$ scheme is chosen-ciphertext secure, then so is $\Pi_{\mathsf{LS-TPKE}}$. On the other hand, if $\mathsf{TPKE}$ is chosen-plaintext secure, then so is $\Pi_{\mathsf{LS-TPKE}}$. We omitted chosen-plaintext security for $\mathsf{LS{-}TPKE}$ schemes in our model since we opt for the strongest possible security guarantees.

We note that our generic transformation can be instantiated with multiple different discrete-log-based TPKE schemes (e.g., [43, 4] and with slight adjustments [38]). We show in Appendix D how to instantiate our generic transformation with the TPKE scheme from [43].

In Appendix E, we present a model for large-scale non-interactive threshold signature schemes and argue that the transformation that we presented in this section can be applied for threshold signature schemes as well. At a high level, the model follows the ideas of the model of LS–TPKE schemes, i.e., a large-scale non-interactive threshold signature scheme is defined w.r.t. to a universe $U$ of parties and proceeds in epochs at the beginning of which a new committee of secret key shareholders is selected. The definition includes a refresh procedure which allows to transition from one epoch to the next by selecting a new committee and refreshing the secret key shares. Finally, an LS–TSIG scheme must be secure w.r.t. a fully mobile adversary whose corruption power suffices to corrupt an entire committee of secret key shareholders.

# 6 Applications

In this section, we show several interesting applications of our LS–DKG protocol combined with LS–TSIG and LS–TPKE schemes.

Our schemes are perfectly suited to be used in blockchain networks, which have increasingly gained attention in the cryptography community as they have proven to be surprisingly versatile for the realization of cryptographic primitives and protocols. We split applications of our solutions into two categories, (1) storage of blockchain-backed secrets and (2) adding signing functionality to a blockchain.

## 6.1 Storage of Blockchain-Backed Secrets

Any information stored on a blockchain is publicly available to all users which severely restricts the usefulness of a blockchain and the class of applications it supports. Recently, Benhamouda et al. [7] and Goyal et al. [27] presented solutions based on secret sharing to allow the storage of secret values on a blockchain[8]. At a high level, these solutions allow a client to secret share a value to a committee, which then stores the secret and periodically refreshes the shares to a new committee. However, for many applications it is not necessary to store the secret on the entire blockchain, but it rather suffices to have a functionality that allows to commit to a secret and have the blockchain open the commitment in case of malicious behavior during the execution of the application.

Consider the example of a fair exchange protocol. Assume two parties, say Alice and Bob, wish to exchange secrets $a$ and $b$, where Alice initially owns $a$ and Bob owns $b$. Alice and Bob could now use either solution of Benhamouda et al. or Goyal et al. to share $a$ and $b$ to the committee and once both parties have done so, the committee could send the shares of $b$ to Alice and vice versa. There are, however, several issues with this solution: (1) Alice and Bob have to interact with the committee, (2) the committee has to store shares of $a$ and $b$

[8] Likewise Kokoris-Kogias et al. [34] presented a solution for auditable data-management based on blockchain and threshold encryption which allows for storage of secrets on a blockchain. We refer to Appendix A for further discussion.

and has to possibly refresh the shares to a new committee and (3) the committee members learn that Alice and Bob exchange secrets, thereby compromising the two parties' privacy. Instead, assume that each committee member holds a secret key share of an LS–TPKE scheme and that the corresponding public key $pk$ is stored on the blockchain. In this case, Alice and Bob could just encrypt their secrets under $pk$ and exchange the ciphertexts[9]. Once each of them have received the respective ciphertext, they can reveal their secrets to each other. If one party misbehaves, say Alice, by not revealing her secret, Bob can let the committee decrypt Alice's ciphertext[10]. Note that, in the optimistic case, i.e., when no party misbehaves, then we have that (1) there is no interaction with the committee required, (2) the committee does not have to store and refresh the secrets $a$ and $b$ and (3) the committee does not learn which parties interact with each other.

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

### 6.2 Adding Signing Functionality to a Blockchain

Our LS–TSIG scheme can be used to generate signatures "on behalf" of the blockchain. This allows to sign individual blocks of the blockchain, thereby certifying that the block is indeed a valid part of the blockchain or sign certain messages indicating that a specific event has occurred on the blockchain. Benhamouda et al. [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.

9 Along with NIZK proofs that prove that the ciphertexts indeed encrypt $a$ and $b$.

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

*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.

## Acknowledgments

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

## 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] J. F. Almansa et al. "Simplified Threshold RSA with Adaptive and Proactive Security". In: *Advances in Cryptology - EUROCRYPT 2006*. Berlin, Heidelberg, 2006.
[3] A. Asayag et al. *Helix: A Scalable and Fair Consensus Algorithm Resistant to Ordering Manipulation*. Cryptology ePrint Archive, Report 2018/863. 2018.
[4] J. Baek and Y. Zheng. "Simple and efficient threshold cryptosystem from the Gap Diffie-Hellman group". In: *GLOBECOM '03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489)*. 2003.
[5] M. Bellare et al. "Key-Privacy in Public-Key Encryption". In: *ASIACRYPT 2001*. 2001.
[6] M. Bellare et al. "Standard Security Does Not Imply Security against Selective-Opening". In: *Advances in Cryptology – EUROCRYPT 2012*. Berlin, Heidelberg, 2012.
[7] F. Benhamouda et al. "Can a Public Blockchain Keep a Secret?" In: *Theory of Cryptography*. Cham, 2020.
[8] M. Blum et al. "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.

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

[11] D. Boneh et al. "Short Signatures from the Weil Pairing". In: *ASIACRYPT 2001*. 2001.

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

[13] R. Canetti et al. "Adaptive Security for Threshold Cryptosystems". In: *CRYPTO'99*. 1999.

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

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

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

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

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

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

[20] C. Dwork et al. "Magic Functions". In: *40th FOCS*. 1999.

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

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

[23] Y. Frankel et al. "Adaptively-Secure Distributed Public-Key Systems". In: *Algorithms - ESA' 99*. Berlin, Heidelberg, 1999.

[24] R. Gennaro et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *EUROCRYPT'99*. 1999.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Supplementary Material

# A Additional Related Work and Preliminaries

## A.1 Additional Related Work

*Threshold Cryptography in Blockchains* Numerous works have considered the use of threshold cryptographic primitives in the context of blockchain (e.g. [34, 3, 30]). The works of Maram et al. [39] and Goyal et al. [27] both present dynamic proactive secret-sharing (DPSS) constructions for blockchain networks under an honest majority assumption. Goyal et al. then proceed to define the notion of extractable witness encryption on blockchains and show an instantiation based on their DPSS scheme. Benhamouda et al. [7] extend the notion of DPSS to an evolving-committee proactive secret sharing scheme that does not require the honest majority assumption. Kokoris-Kogias et al. [34] present an auditable data-management solution that is based on blockchain and threshold encryption. However, their work lacks a formal security analysis of the presented solution and focuses mostly on a static committee of secret key shareholders. Finally, Groth [28] presents a non-interactive distributed key generation protocol together with a refresh procedure that allows refreshing the secret key shares to a new committee.

## A.2 Further Notions of Secret Sharing

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

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

# B Proof of Theorem 1

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

## B.1 Proof of Lemma 1

*Proof.* In order to prove Lemma 1, we have to show that the correctness properties 1.-3. hold. The correctness proof for properties 1. and 3. proceeds in a similar manner as for the DKG protocol in [24]. 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 thereby identify valid tuples and (2) there exists an order on the tuples. Therefore it holds that all honest parties in $U$ compute the same set $Qual$ consisting of $t+1$ valid tuples.

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

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

   The correctness for secret key shares $sk''_j$ of parties $P_j{}'' \in C''$ follows directly from the above and from the (robust) reconstruction property of the $\Sigma_{\mathsf{ECPSS}}$ scheme.

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 $pk = g^{sk} = g^{\sum_{k\in Qual} a^s_{k,0}}$ after an execution of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$. If $k \in Qual$, then $P_k \in C$ must have sent shares $r_{k,j}$ of the value $a^r_{k,0}$ to party $P_j{}' \in C'$ such that for any set $S$ with $|S| \geq t+1$ of valid shares it holds that $a^r_{k,0} = \sum_{j\in S} l_j \cdot r_{k,j}$. Additionally, $P_k \in C$ must have correctly shared $a^s_{k,0}$ and broadcast $A_k = g^{a^s_{k,0}+a^r_{k,0}}$.

   Upon receiving shares $r_{k,j}$, each $P_j{}' \in C'$ computes $R'_j = g^{\sum_{k\in Qual} r_{k,j}} = g^{r'_j}$ along with a NIZK proof $\pi'_j$ that proves correctness of $R'_j$. All parties in $U$ then construct a set $Rand$ consisting of $t+1$ correct elements $R'_j$ for $j \in [n]$. The public key is reconstructed as

$$
\begin{aligned}
pk &= \left(\prod_{k \in Qual} A_k\right) \cdot \left(\prod_{j \in Rand} R_j'^{l_j}\right)^{-1} \\
&= \left(\prod_{k \in Qual} A_k\right) \cdot \left(g^{\sum_{j \in Rand} l_j \cdot r_j'}\right)^{-1} \\
&= \left(\prod_{k \in Qual} A_k\right) \cdot \left(g^{\sum_{j \in Rand} l_j \cdot \sum_{k \in Qual} r_{k,j}}\right)^{-1} \\
&= \left(\prod_{k \in Qual} g^{a^s_{k,0} + a^r_{k,0}}\right) \cdot \left(g^{\sum_{k \in Qual} a^r_{k,0}}\right)^{-1} \\
&= \prod_{k \in Qual} \left(g^{a^s_{k,0}}\right) = g^{\sum_{k \in Qual} a^s_{k,0}}
\end{aligned}
$$

3. Since the secret key is computed as $sk = \sum_{k \in Qual} a^s_{k,0}$ and $a^s_{k,0}$ is chosen uniformly at random from $\mathbb{Z}_q$, it holds that $sk$ is uniformly distributed in $\mathbb{Z}_q$. Since $sk$ is uniformly distributed in $\mathbb{Z}_q$, so is $pk = g^{sk} \in \mathbb{G}$.

### B.2 Proof of Lemma 2

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

**Lemma 5.** *Let $\Pi_{\mathsf{LS-DKG}}$ be the large-scale distributed key generation protocol from Sec. 4 instantiated with a non-interactive zero-knowledge proof system* NIZK*, a RIND-SO secure public key encryption scheme* PKE *and a $(\lambda, n, t, p)$-secure instantiation of $\Sigma_{\mathsf{ECPSS}}$. 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 $\lambda$.*

*Proof.* We prove this lemma by reduction to the secrecy property of the $\Sigma_{\mathsf{ECPSS}}$ 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 $\Sigma_{\mathsf{ECPSS}}$. In a nutshell, the secrecy property of $\Sigma_{\mathsf{ECPSS}}$ states that upon choosing two secrets $x \in \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$ and subsequently interacting with the $\Sigma_{\mathsf{ECPSS}}$ scheme, the adversary $\mathcal{B}$ cannot distinguish whether $x$ or $y$ has been shared in $\Sigma_{\mathsf{ECPSS}}$.

In order to break this property using $\mathcal{A}$, the adversary $\mathcal{B}$ simulates an execution of the $\Pi_{\mathsf{LS-DKG}}$ protocol as follows: As mentioned above, $\mathcal{B}$ first chooses two secrets $x \in \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$, one of which is shared in $\Sigma_{\mathsf{ECPSS}}$. $\mathcal{B}$ then sets the universe of parties in $\Pi_{\mathsf{LS-DKG}}$ to the same universe as in $\Sigma_{\mathsf{ECPSS}}$ and simulates the behavior of all honest parties in this universe to $\mathcal{A}$.

At the beginning of each epoch, all parties in the universe broadcast a new long-term public key via the PKI. For all corrupted parties $P_j$ that broadcast a long-term public key $pk_j$ in $\Pi_{\mathsf{LS\text{-}DKG}}$, $\mathcal{B}$ chooses a fresh long-term key pair $(sk'_j, pk'_j)$ and broadcasts $pk'_j$ in $\Sigma_{\mathsf{ECPSS}}$.

$\mathcal{B}$ must now ensure that in both schemes the same committees are selected in each epoch. A naive approach would be to simply relay messages between schemes $\Sigma_{\mathsf{ECPSS}}$ and $\Pi_{\mathsf{LS\text{-}TPKE}}$ during executions of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure. However, this is not sufficient. Recall that the nominating committee selects parties to a holding committee by broadcasting ephemeral public key and ciphertext pairs $\{(\mathsf{epk}'_i, c'_i)\}_{i\in[n]}$ where the ciphertext $c'_i$ encrypts the ephemeral secret key $\mathsf{esk}'_i$ corresponding to $\mathsf{epk}'_i$. The issue with simply relaying messages between both schemes during the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure is that $\mathcal{B}$ cannot learn the ephemeral secret keys $\mathsf{esk}'_i$ of honest committee members $P_i$ and thereby $\mathcal{B}$ cannot learn the secret shares that are exchanged during executions of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure in $\Pi_{\mathsf{LS\text{-}DKG}}$. However, it is crucial for $\mathcal{B}$ to learn these secret shares in order to simulate the rest of the protocol.

Therefore, $\mathcal{B}$ simulates the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure as follows: upon the nominating committee in $\Sigma_{\mathsf{ECPSS}}$ sending the pairs $\{(\mathsf{epk}'_i, c'_i)\}_{i\in[n]}$, $\mathcal{B}$ first uses the long-term secret keys $sk'_j$ that it chose at the beginning of the epoch for corrupted parties $P_j$ to identify whether any of these parties has been selected to the committee. If so, $\mathcal{B}$ learns the ephemeral secret key $\mathsf{esk}'_j$ and encrypts it under the long-term public key $pk_j$ of party $P_j$ in $\Pi_{\mathsf{LS\text{-}DKG}}$ to obtain a ciphertext $c_j$. It forwards the pairs $(\mathsf{epk}'_j, c_j)$ to $\mathcal{A}$ for all corrupted parties $P_j$.

For all honest parties $P_i$ that have been selected to the committee, $\mathcal{B}$ chooses new ephemeral key pairs $(\mathsf{esk}_i, \mathsf{epk}_i) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and forwards the pairs $(\mathsf{epk}_i, c'_i)$ to $\mathcal{A}$. Note that now the decryption of the ciphertexts $c'_i$ and the corresponding ephemeral public keys $\mathsf{epk}_i$ do not form valid key pairs. However, this remains undetected by $\mathcal{A}$ (except with negligible probability) due to the RIND-SO security of the $\mathsf{PKE}$ scheme and due to the fact that parties have erased their long-term secret keys already at this point. Note that as a consequence of this simulation, $\mathcal{A}$ encrypts its shares during executions of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure under the ephemeral public keys $\mathsf{epk}_i$ for which $\mathcal{B}$ knows the corresponding ephemeral secret keys $\mathsf{esk}_i$.

Apart from the simulation of the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Select}$ procedure, $\mathcal{B}$ executes the $\Pi_{\mathsf{LS\text{-}DKG}}$ correctly for all honest parties, i.e., it follows the protocol instructions. Note, however, that $\mathcal{B}$ does not know the identities of uncorrupted parties in the current-epoch committee (i.e., $C$, $C'$ or $C''$). Therefore, $\mathcal{B}$ follows the protocol instructions of $\Pi_{\mathsf{LS\text{-}DKG}}$ for all honest parties in the current-epoch committee without knowing the identities of these parties. Said differently, $\mathcal{B}$ prepares the internal states of honest parties in the current-epoch committee without knowing the identities of these parties. Upon $\mathcal{A}$ corrupting a party $P_i$, $\mathcal{B}$ corrupts the corresponding party in $\Sigma_{\mathsf{ECPSS}}$ and hence learns whether or not $P_i$ is part of the current-epoch committee. If it is, $\mathcal{B}$ returns one of the internal states that it had previously prepared for honest parties in the current-epoch committee along with the ephemeral secret key $\mathsf{esk}_i$[11]. Otherwise, if $P_i$ is not in the current-epoch committee, $\mathcal{B}$ forwards the internal state of $P_i$ to $\mathcal{A}$. Once committee members in $\Pi_{\mathsf{LS-DKG}}$ have to broadcast values (i.e., reveal their identities), $\mathcal{B}$ executes the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure in $\Sigma_{\mathsf{ECPSS}}$ by which it learns the identities of all honest parties in the current-epoch committee. This allows $\mathcal{B}$ to broadcast the prepared internal states on behalf of the honest committee members in $\Pi_{\mathsf{LS-DKG}}$.

It is easy to see that the committees in $\Sigma_{\mathsf{ECPSS}}$ and $\Pi_{\mathsf{LS-DKG}}$ consist of the same parties. Therefore, if $\mathcal{A}$ corrupts more than $t$ parties in either of $C$, $C'$ or $C''$, then $\mathcal{B}$ corrupts more than $t$ parties in a committee in $\Sigma_{\mathsf{ECPSS}}$, thereby learning whether secret $x$ or $y$ was shared in $\Sigma_{\mathsf{ECPSS}}$. This concludes the proof.

With Lemma 5 in place, we can now prove Lemma 2.

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

During the $\Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}(1^\lambda)$ procedure, $\mathcal{S}$ runs $(\tilde{\mathsf{crs}}_1, \tau) \leftarrow \mathsf{NIZK}.\mathsf{Setup}'(1^\lambda)$ instead of $\mathsf{crs}_1 \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$. This allows $\mathcal{S}$ to obtain a trapdoor $\tau$ 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 4b. 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 ${P_j}' \in H'$[12], in particular the values $sk'_j$ and $r_{i,j}$ for $i \in Qual$. Further, note that due to Lemma 5 there is an honest majority in each of $C$, $C'$ and $C''$ through which $\mathcal{S}$ can learn the values $sk'_k$ and $r_{i,k}$ for all ${P_k}' \in B'$ and $i \in Qual$. Therefore, $\mathcal{S}$ can learn the elements $R'_j$ for all parties ${P_j}' \in C'$.

In step 4b, $\mathcal{S}$ computes

$$R' = pk^{-1} \cdot \prod_{k \in Qual} A_k$$

such that it holds that $\prod_{k \in Qual} A_k \cdot R' = pk$.

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

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

$\mathcal{S}$ then 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 4b while for parties ${P_j}' \in H'$, $\mathcal{S}$ sets

$$R'_j = R'^{l_{j,0}} \cdot \prod_{i \in B' \cup SH'} R_i'^{l_{j,i}} \tag{2}$$

where $l_{j,i}$ are the appropriate lagrange coefficients. Note that this allows any set $S \subset \{R'_1, \cdots, R'_n\}$ with $|S| = t + 1$ to reconstruct $R'$ via interpolation in the exponent.

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

During the simulation of steps 4b and 7, $\mathcal{S}$ handles corruptions as follows:

- Upon $\mathcal{A}$ corrupting a party ${P_j}' \in SH'$, $\mathcal{S}$ sends the internal state of ${P_j}'$ to $\mathcal{A}$ and sets $SH' = SH' \setminus \{{P_j}'\}$ and $B' = B' \cup \{{P_j}'\}$.
- Upon $\mathcal{A}$ corrupting a party ${P_i}' \in H'$, $\mathcal{S}$ sends the original (and not the simulated) internal state which includes $R'_i = g^{r'_i}$ (instead of the simulated value as computed in Eq. (2)) to $\mathcal{A}$ and chooses a party ${P_j}' \leftarrow_{\$} SH'$. It then sets $H' = H' \setminus \{{P_i}'\}$, $SH' = SH' \setminus \{{P_j}'\}$, $H' = H' \cup \{{P_j}'\}$ and $B' = B' \cup \{{P_i}'\}$.

If $\mathcal{A}$ corrupts a party in $H'$ during the simulation of steps 4b or 7, $\mathcal{S}$ executes the simulation of the respective step again for the updated sets $B'$, $SH'$ and $H'$.

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

We now show that the simulation is computationally indistinguishable to $\mathcal{A}$ from a real protocol execution. Note that $\mathcal{S}$ only deviates from the protocol instructions during the $\mathsf{NIZK.Setup}$ procedure and during steps 4b and 7 for parties ${P_j}' \in H'$. Due to the zero-knowledge property of the $\mathsf{NIZK}$ proof system, it holds that the distributions $\{\mathsf{crs}_1 : \mathsf{crs}_1 \leftarrow \mathsf{NIZK.Setup}(1^\lambda)\}$ and $\{\tilde{\mathsf{crs}}_1 : (\tilde{\mathsf{crs}}_1, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)\}$ are computationally indistinguishable to $\mathcal{A}$. In step 4b, $\mathcal{S}$ replaces the elements $R'_j$ for all ${P_j}' \in H'$ by elements computed as $R'^{l_{j,0}} \cdot \prod_{i \in B' \cup SH'} R_i'^{l_{j,i}}$. Note that, due to Lemma 5, there exists at least one honest party $P_i \in Qual$ and therefore all elements $R'_j$ contain at least one uniformly random value $r_{i,j}$ from an honest party in the exponent. Therefore $\mathcal{A}$ can distinguish the simulated and real elements $R'_j$ only by breaking the RIND-SO security of the PKE scheme which happens at most with negligible probability. Finally, by the zero-knowledge property of the $\mathsf{NIZK}$ proof system, $\mathcal{A}$ cannot distinguish the simulated NIZK proofs $\tilde{\pi}'_j$ from the real proofs $\pi_j$ except with negligible probability.

## C Proof of Theorem 2

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

**Simulator Code:** During the simulation of $\Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}$, $\mathcal{S}$ executes $\mathsf{NIZK.Setup}'(1^\lambda)$ instead of $\mathsf{NIZK.Setup}(1^\lambda)$ through which it learns a trapdoor $\tau$ 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 5.

On input a public key $pk$, $\mathcal{S}$ then simulates $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ as follows:

- $\mathcal{S}$ follows the protocol instructions for all honest parties until step 4b.
- In step 4b, $\mathcal{S}$ proceeds as follows:
  - $\mathcal{S}$ chooses $t - |B'|$ parties from $H'$ and assigns those parties to a new set $SH'$ s.t. $SH' \cap H' = \emptyset$.
  - $\mathcal{S}$ computes the element

$$R' = pk^{-1} \cdot \prod_{k \in Qual} A_k.$$

  - For all parties in $SH'$, $\mathcal{S}$ follows the protocol instructions.
  - For all parties $P_i{}' \in B'$, $\mathcal{S}$ computes the elements $R_i'$.
  - For all parties $P_j{}' \in H'$, $\mathcal{S}$ sets $R_j' = R'^{l_{j,0}} \cdot \prod_{i \in B' \cup SH'} R_i'^{l_{j,i}}$ where $l_{j,i}$ are the appropriate lagrange coefficients.
- In step 7, $\mathcal{S}$ then uses the trapdoor $\tau$ as generated during the $\mathsf{NIZK.Setup}'$ procedure, to generate simulated NIZK proofs $\tilde{\pi}_j'$ that prove correctness of the elements $R_j'$.
- During the simulation of steps 4b and 7, $\mathcal{S}$ handles corruptions as follows:
  - Upon $\mathcal{A}$ corrupting a party $P_j{}' \in SH'$, $\mathcal{S}$ sends the internal state of $P_j{}'$ to $\mathcal{A}$ and sets $SH' = SH' \setminus \{P_j{}'\}$ and $B' = B' \cup \{P_j{}'\}$.
  - Upon $\mathcal{A}$ corrupting a party $P_i{}' \in H'$, $\mathcal{S}$ sends the original (and not the simulated) internal state which includes $R_i' = g^{r_i'}$ (instead of the simulated value as computed in Eq. (2)) to $\mathcal{A}$ and chooses a party $P_j{}' \leftarrow_\$ SH'$. It then sets $H' = H' \setminus \{P_i{}'\}$, $SH' = SH' \setminus \{P_j{}'\}$, $H' = H' \cup \{P_j{}'\}$ and $B' = B' \cup \{P_i{}'\}$.

  If $\mathcal{A}$ corrupts a party in $H'$ during the simulation of steps 4b or 7, $\mathcal{S}$ executes the simulation of the respective step again for the updated sets $B'$, $SH'$ and $H'$.
- The simulator $\mathcal{S}$ executes the rest of the protocol correctly for all honest parties.

Fig. 1: Simulator code for our large-scale distributed key generation protocol $\Pi_{\mathsf{LS-DKG}}$.

### C.1 Proof outline of Lemma 3

We show that decryption and reconstruction consistency hold for $\Pi_{\mathsf{LS-TPKE}}$ for the first epoch. For all subsequent epochs, these properties then follow from the (robust) reconstruction property of the $\Sigma_{\mathsf{ECPSS}}$ scheme. 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 $pp := (pp^{\mathsf{TPKE}}, pp^{\mathsf{LS-DKG}}, \mathsf{crs}_1)$.

1. Due to the Has-DKG and Sk-To-Vk properties, the following holds:

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

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

$$(pk, \{vk_i^{1'}\}_{i\in[n]}, \{sk_i^1\}_{i\in[n]}) \in \mathsf{TPKE.TKeyGen}(pp^{\mathsf{TPKE}}).$$

2. Due to the completeness property of the NIZK scheme, it holds that $\pi_i^1$ is a correct NIZK proof for the correctness of $vk_i^{1'}$.
3. Decryption consistency in epoch 1 then follows from the above and from the decryption consistency property of the TPKE scheme.
4. Reconstruction consistency in epoch 1 follows from the above and from the reconstruction consistency property of the TPKE scheme.

### C.2 Proof of Lemma 4

*Proof.* We now present the proof of Lemma 4. 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 $\mathsf{TPKE{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ (cf. Section 2.3) with non-negligible advantage. More precisely, we show in a series of computationally indistinguishable games that $\mathcal{A}$ can use $\mathcal{B}$'s output bit $b'$ to win its own game.

**Game $G_0$**: This is the original $\mathsf{LSTPKE{-}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ 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 $\left\lfloor \frac{|B^j|+1}{|U|} \right\rfloor \le p$. Further, $\mathcal{B}$ obtains access to a decryption and refresh oracle.

**Game $G_1$**: This game proceeds as the previous game with the difference that it aborts in case during any epoch $j$ of the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ 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 $\Pi_{\mathsf{LS-DKG}}$ scheme. In more detail, 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 simulates an execution of the $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ procedure to $\mathcal{B}$ and subsequently breaks the secrecy property of $\Pi_{\mathsf{LS-DKG}}$ by corrupting more than $t$ parties in $\Pi_{\mathsf{LS-DKG}}$. Upon $\mathcal{B}$ issuing a corruption oracle query, $\mathcal{A}'$ corrupts the corresponding party in $\Pi_{\mathsf{LS-DKG}}$ and relays the obtained internal state to $\mathcal{B}$ (i.e., gives the control of this party to $\mathcal{B}$). Due to property Sk-To-Vk, $\mathcal{A}'$ can correctly compute verification keys for corrupted parties in the last epoch of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$. Therefore, we get that $\Pr[G_1 = 0] \le \Pr[G_1 = 1] + \nu_1(\lambda)$ where $\nu_1$ is a negligible function in $\lambda$.

**Game $\boldsymbol{G_2}$**: This game proceeds as the previous game with the difference that it aborts in case in any epoch $j$ after the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ 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 $\Sigma_{\mathsf{ECPSS}}$ 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 $\Sigma_{\mathsf{ECPSS}}$ by corrupting more than $t$ parties in $\Sigma_{\mathsf{ECPSS}}$. The reduction works in a similar fashion as the one in Lemma 5, i.e., $\mathcal{A}'$ trying to break the secrecy property of $\Sigma_{\mathsf{ECPSS}}$ can simulate game $\mathsf{LSTPKE\text{–}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ 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 5. As a consequence of this simulation, $\mathcal{A}'$ does not know the identities of parties in the current-epoch committee, therefore it is crucial that decryption shares in $\mathsf{LSTPKE\text{–}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}$ are output only after the current-epoch committee has been replaced by a new committee, such that $\mathcal{A}'$ learns the identities of the parties it needs to output decryption shares for. It is easy to see that upon $\mathcal{B}$ corrupting more than $t$ parties in a committee $C^j$ in $\Pi_{\mathsf{LS-TPKE}}$, $\mathcal{A}'$ can corrupt more than $t$ parties in the same-epoch committee in $\Sigma_{\mathsf{ECPSS}}$, thereby breaking the secrecy property of $\Sigma_{\mathsf{ECPSS}}$.

Hence, we get that $\Pr[\boldsymbol{G_1} = 1] \leq \Pr[\boldsymbol{G_2} = 1] + \nu_2(\lambda)$ where $\nu_2$ is a negligible function in $\lambda$.

**Game $\boldsymbol{G_3}$**: This game is the same as the previous game with only a syntactical change. For each epoch $j$ after the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ the game maintains another list $SH^{C^j}$ with $|SH^{C^j}| = t - |B^{C^j}|$, in addition to the sets $H^{C^j}$ and $B^{C^j}$. At the beginning of epoch $j$, the game then randomly assigns $t - |B^{C^j}|$ parties from $H^{C^j}$ to $SH^{C^j}$ and removes these parties from $H^{C^j}$ (i.e., $H^{C^j} \cap SH^{C^j} = \emptyset$).

This change is only syntactical and therefore we get that $\Pr[\boldsymbol{G_2} = 1] = \Pr[\boldsymbol{G_3} = 1]$.

**Game $\boldsymbol{G_4}$**: This game is similar to the previous game with a modification to the corruption oracle. In each epoch $j$ after the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$, the corruption oracle behaves as follows:

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

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

The change in this game is only syntactical and 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 $\mathsf{crs}_1$ of the NIZK proof system is generated, the game executes $(\tilde{\mathsf{crs}}_1, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)$ instead of $\mathsf{crs}_1 \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. This allows the game to learn a trapdoor $\tau$. Since the distributions $\{\mathsf{crs}_1 : \mathsf{crs}_1 \leftarrow \mathsf{NIZK.Setup}(1^\lambda)\}$ and $\{\tilde{\mathsf{crs}}_1 : (\tilde{\mathsf{crs}}_1, \tau) \leftarrow \mathsf{NIZK.Setup}'(1^\lambda)\}$ are indistinguishable to $\mathcal{B}$ except with negligible probability (due to the zero-knowledge property of $\mathsf{NIZK}$), it holds that $\Pr[\boldsymbol{G_4} = 1] \leq \Pr[\boldsymbol{G_5} = 1] + \nu_3(\lambda)$ where $\nu_3$ is a negligible function in $\lambda$.

**Game $\boldsymbol{G_6}$**: This game works as the previous game with the following difference. For each party $P_i{}^j \in H^{C^j}$, the game computes a simulated NIZK proof $\pi_i^j$ (i.e., without using the secret key share $sk_i^j$) using the trapdoor $\tau$ and algorithm $\mathsf{S}$ (cf. Def. 7) which proves that the verification key $vk_i^{j'}$ has been computed correctly w.r.t. $sk_i^j$.

Due to the zero-knowledge property of the NIZK proof system, the simulated proof $\pi_i^j$ is indistinguishable from a real proof except with negligible probability. It holds that $\Pr[\boldsymbol{G_5} = 1] \leq \Pr[\boldsymbol{G_6} = 1] + \nu_4(\lambda)$ where $\nu_4$ is a negligible function in $\lambda$.

**Game $\boldsymbol{G_7}$**: This game proceeds as the previous game with the following modification. After the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$, the game uses the secret key shares $sk_i^1$ of all $P_i{}^1 \in H^{C^1}$ to reconstruct the secret key $sk$ corresponding to $pk$. Note that this is possible because $|H^{C^1}| \geq t+1$. During a decryption oracle query in an epoch $j$, the game then proceeds as follows: It reconstructs a degree-$t$ polynomial $\tilde{F}^j$ from the secret key shares $\{sk_k^j\}_{k \in B^{C^j} \cup SH^{C^j}}$ and $sk$, s.t. $\tilde{F}^j(k) = sk_k^j$ and $\tilde{F}^j(0) = sk$. The game then computes secret key shares $\tilde{F}^j(i) = \tilde{sk}_i^j$ for all $P_i{}^j \in H^{C^j}$ and uses $\tilde{sk}_i^j$ to compute decryption shares for $P_i{}^j$.

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

We therefore get that $\Pr[\boldsymbol{G_6} = 1] = \Pr[\boldsymbol{G_7} = 1]$.

**Game $\boldsymbol{G_8}$**: This game proceeds similarly as the previous game with a modification in the refresh oracle. Instead of executing the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure on input the secret key shares $sk_i^j$ for all parties $P_i{}^j \in H^{C^j}$, the game first chooses a uniformly random element $x_i^j \in \mathbb{Z}_q$ for each $P_i{}^j$ and then executes the simulator code of $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ on input secret key shares $sk_k^j$ for all ${P_k}^j \in SH^{C^j}$ and $x_i^j$ for all ${P_i}^j \in H^{C^j}$.

In case $\mathcal{B}$ sends a corruption query for a party ${P_i}^j \in H^{C^j}$ *during* a refresh oracle execution, the game returns the secret key share $sk_i^j$ (instead of $x_i^j$) and repeats the steps of $\boldsymbol{G_8}$ w.r.t. the updated sets $H^{C^j}$, $SH^{C^j}$ and $B^{C^j}$.

The indistinguishability argument of this game to the previous one follows from the secrecy property of the $\Sigma_{\mathsf{ECPSS}}$ scheme. Note that for $\Sigma_{\mathsf{ECPSS}}$ to satisfy secrecy, there must exist a simulator that can simulate the scheme in such a way that an efficient fully mobile adversary cannot distinguish whether the simulation is executed w.r.t. the secret key $sk$ or a uniformly random secret except with negligible probability. Hence, if $\mathcal{B}$ could detect the modification in $\boldsymbol{G_8}$ with non-negligible probability, then we could construct an efficient adversary who could break the secrecy property of the $\Sigma_{\mathsf{ECPSS}}$ scheme with non-negligible probability. Therefore, it holds that $\Pr[\boldsymbol{G_7} = 1] \leq \Pr[\boldsymbol{G_8} = 1] + \nu_5(\lambda)$ where $\nu_5$ is a negligible function in $\lambda$.

**Game $\boldsymbol{G_9}$**: This game proceeds similarly to the previous game with the following modification in the decryption oracle. For all $P_i^j \in H^{C^j}$, the game first computes the verification key $vk_i^{j'}$ as $vk_i^{j'} = pk^{l_{i,0}} \prod_{k \in B^{C^j} \cup SH^{C^j}} vk_k^{l_{i,k}}$. The game then executes the algorithm $\mathcal{S}$ as per property $\mathsf{Has\text{-}Sim}$ on input $(pk, \{vk_i^{j'}\}_{i\in[n]}, \{sk_k^j\}_{k \in B^{C^j} \cup SH^{C^j}})$ to answer decryption queries.

In more detail, upon a decryption query from $\mathcal{B}$ on input a set of ciphertexts $CT^j$ with a set of associated labels $AL^j$, the game proceeds as follows:

1. it first computes verification keys for all parties $P_i^j \in H^{C^j}$ as described above
2. it executes $\mathcal{S}$ on input $(pk, \{vk_i^{j'}\}_{i\in[n]}, \{sk_k^j\}_{k \in B^{C^j} \cup SH^{C^j}})$
3. for each $ct_m^j \in CT^j$, $L_m^j \in AL^j$ with $m \in [|CT^j|]$ and $P_i^j \in H^{C^j}$, the game uses $\mathcal{S}$ to receive decryption shares $\{ct_{i,m}^j\}_{m \in |CT^j|}$
4. the game outputs $\{ct_{i,m}^j\}_{m \in |CT^j|}$ together with the verification key $vk_i^{j'}$ and a NIZK proof $\pi_i^j$ to answer decryption oracle queries by $\mathcal{B}$.

Note that for all parties ${P_k}^j \in SH^{C^j}$, the game computes decryption shares w.r.t. the secret key share $sk_k^j$.

Recall that upon a corruption query for a party ${P_i}^j \in H^{C^j}$, the game removes a party ${P_k}^j$ at random from the set $SH^{C^j}$ and adds it to $H^{C^j}$. Therefore, in case such a corruption query occurs *during* a decryption oracle execution, the game first computes the decryption shares of ${P_i}^j$ w.r.t. the secret key share $sk_i^j$ and then repeats the steps of $\boldsymbol{G_9}$ w.r.t. the updated sets $H^{C^j}$, $SH^{C^j}$ and $B^{C^j}$.

Due to property $\mathsf{Has\text{-}Sim}$, there exist an efficient algorithm $\mathcal{S}$ as used in $\boldsymbol{G_9}$. $\mathcal{S}$ simulates game $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}'}_{\mathsf{TPKE}}$ such that it is indistinguishable to an efficient static adversary $\mathcal{A}'$ from the real game except with negligible probability. Since the game uses $\mathcal{S}$ only to simulate decryption shares to $\mathcal{B}$, it holds that $\Pr[\boldsymbol{G_8} = 1] \leq \Pr[\boldsymbol{G_9} = 1] + \nu_6(\lambda)$ where $\nu_6$ is a negligible function in $\lambda$.

**Game $\boldsymbol{G_{10}}$**: This game proceeds similarly to the previous game $\boldsymbol{G_9}$ with the exception that before the execution of $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$, the game chooses at random a public key $pk$, s.t. $(pk, \cdot, \cdot) \in \mathsf{TPKE.KeyGen}$. Then during the $\Pi_{\mathsf{LS-TPKE}}.\mathsf{TKeyGen}$ procedure, instead of executing $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$, the game executes the simulator code of $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ (cf. 1) on input $pk$. This code simulates $\Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}$ in a way such that the output public key is equal to $pk$.

The indistinguishability argument follows from the secrecy property of the $\Pi_{\mathsf{LS-DKG}}$ scheme. More precisely, for any secure LS–DKG scheme, there must exist an efficient simulator which on input a public key $pk$ can simulate the execution of LS–DKG.TKeyGen in such a way that the execution is indistinguishable to an efficient fully mobile adversary except with negligible probability and the output public key equals $pk$. For $\Pi_{\mathsf{LS-DKG}}$ we presented this simulator code in Fig. 1. Therefore, it holds that $\Pr[\boldsymbol{G_9} = 1] \leq \Pr[\boldsymbol{G_{10}} = 1] + \nu_7(\lambda)$ where $\nu_7$ is a negligible function in $\lambda$.

By the transition from game $\boldsymbol{G_0}$ to $\boldsymbol{G_{10}}$ we get that

$$\begin{aligned}
\Pr[\mathsf{LSTPKE\text{–}CCA}^{\mathcal{B}}_{\Pi_{\mathsf{LS-TPKE}}}(\lambda) = 1] &= \Pr[\boldsymbol{G_0} = 1] \\
&\leq \Pr[\boldsymbol{G_{10}} = 1] + \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) \\
&\qquad + \nu_5(\lambda) + \nu_6(\lambda) + \nu_7(\lambda) \\
&\leq \Pr[\boldsymbol{G_{10}} = 1] + \nu(\lambda).
\end{aligned}$$

where $\nu(\lambda) \geq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) + \nu_5(\lambda) + \nu_6(\lambda) + \nu_7(\lambda)$ is a negligible function in $\lambda$.

Having shown that the transition from game $\boldsymbol{G_0}$ to game $\boldsymbol{G_{10}}$ is indistinguishable, it remains to show that there exists an efficient static adversary $\mathcal{A}$ who plays in game $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ and simulates game $\boldsymbol{G_{10}}$ to $\mathcal{B}$. We have to show that $\mathcal{A}$ can then use $\mathcal{B}$ to win its own game. The only difference between game $\boldsymbol{G_{10}}$ and $\mathcal{A}$'s simulation is as follows: In $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$, $\mathcal{A}$ receives a challenge public key $pk_C$, which it uses instead of the randomly chosen public key in game $\boldsymbol{G_{10}}$. Since the challenge public key $pk_C$ is chosen uniformly at random, this change is only syntactical.

Finally, we have to show that $\mathcal{A}$ can use $\mathcal{B}$ to win the $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ game. Note that the encryption procedure is the same in both the $\Pi_{\mathsf{LS-TPKE}}$ and TPKE 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 $ct'$, $\mathcal{A}$ forwards it to $\mathcal{B}$. Upon $\mathcal{B}$ outputting a bit $b'$, $\mathcal{A}$ forwards this bit to its own game. Hence, there exists a negligible function $\nu'$ in $\lambda$ such that it holds that

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

where $\nu'$ is a negligible function in $\lambda$.

*Remark 2.* We note that the adversary in game $\mathsf{LSTPKE\text{–}CCA}_{\Pi_{\mathsf{LS-TPKE}}}$ might obtain access to additional oracles, depending on the model in which property Has-Sim is satisfied. For instance, if the TPKE scheme satisfies property Has-Sim in the random oracle model, then the adversary in game $\mathsf{LSTPKE\text{–}CCA}_{\Pi_{\mathsf{LS-TPKE}}}$ obtains access to a random oracle as well. Similar to [31], we abstract from this in our proof, since such oracles are captured by algorithm $\mathcal{S}$ according to property Has-Sim.

## D Concrete Instantiation with the Shoup-Gennaro Threshold Encryption Scheme

In the following we show a concrete instantiation of the generic transformation as presented in the previous section. We first briefly recall the $(t, n)$-threshold encryption scheme from Shoup and Gennaro [43], which we denote by SG. This scheme has previously been proven secure against static adversaries. We show that SG satisfies the properties Has-DKG, Sk-To-Vk and Has-Sim as defined in Sec. 5 and thereby can be transformed into a large-scale non-interactive threshold encryption scheme.

We now briefly recall the SG scheme and its security proof as given in [43].

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

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

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

$\mathsf{KeyGen}(pp, t, n)$*:* On input public parameters $pp$ and integers $t, n \in \mathbb{N}$ s.t. $n \geq 2t + 1$, this procedure chooses a random degree-$t$ polynomial $F(x) = a_0 + a_1 x + \cdots + a_t x^t \in \mathbb{Z}_q[x]$ 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 $sk_i$.

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

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

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

$\mathsf{TDec}(sk_i, ct, L, vk_i)$: On input a secret key share $sk_i$, a ciphertext $ct = (c, L, u, \bar{u}, e, f)$, a label $L$ and a verification key $vk_i$ 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: $u_i = u^{sk_i}, \hat{u_i} = u^{s_i}, \hat{h}_i = g^{s_i}, e_i = H_4(u_i, \hat{u}_i, \hat{h}_i), f_i = s_i + sk_i e_i$.

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

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

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

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

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

The output is the message $m$.

We will now show that the $\mathsf{SG}$ scheme fulfills the required properties $\mathsf{Has\text{-}DKG}$, $\mathsf{Sk\text{-}To\text{-}Vk}$ and $\mathsf{Has\text{-}Sim}$ as presented in Sec. 5.2 such that it can be transformed into a large-scale non-interactive threshold PKE scheme.

$\mathsf{SG}$ *satisfies the properties from Sec. 5.2* In the following, for security parameter $\lambda \in \mathbb{N}$, let $pp^{\mathsf{LS\text{-}DKG}} \leftarrow \Pi_{\mathsf{LS\text{-}DKG}}.\mathsf{Setup}(1^\lambda)$ and $pp^{\mathsf{SG}} \leftarrow \mathsf{SG}.\mathsf{Setup}(1^\lambda)$ and let $U$ denote a universe of parties. Further, let $t, n \in \mathbb{N}$, s.t. $n \geq 2t + 1$.

– $\mathsf{Has\text{-}DKG}$: We have to show that the distribution of the tuples $(pk^1, \{sk_i^1\}_{i \in [n]})$ and $(pk', \{sk_i'\}_{i \in [n]})$ are indistinguishable, where

$$(pk^1, \{sk_i^1\}_{i \in [n]}) \leftarrow \Pi_{\mathsf{LS\text{-}DKG}}.\mathsf{TKeyGen}[U](pp^{\mathsf{LS\text{-}DKG}}, t, n)$$
$$\text{and } (pk', \cdot, \{sk_i'\}_{i \in [n]}) \leftarrow \mathsf{SG}.\mathsf{KeyGen}(pp^{\mathsf{SG}}, t, n).$$

In order to do so, we have to show that the values $\{sk_i^1\}_{i \in [n]}$ and $\{sk_i'\}_{i \in [n]}$ are uniformly distributed in $\mathbb{Z}_q$ and that the elements $pk^1$ and $pk'$ are uniformly distributed in $\mathbb{G}$. From the correctness property of $\Pi_{\mathsf{LS\text{-}DKG}}$ and the

description of $\mathsf{SG}$ we know that the values $\{sk^1_i\}_{i\in[n]}$ and $\{sk'_i\}_{i\in[n]}$ define degree-$t$ polynomials $F^1$ and $F'$ over $\mathbb{Z}_q[x]$ with uniformly random coefficients, s.t. $F^1(i) = sk^1_i$, $F'(i) = sk'_i$ and $F^1(0) = sk^1$ and $F'(0) = sk'$. Therefore, all values $\{sk^1_i\}_{i\in[n]}$ and $\{sk'_i\}_{i\in[n]}$ are distributed uniformly in $\mathbb{Z}_q$.
Further, we know that $sk^1$ and $sk'$ are uniformly distributed in $\mathbb{Z}_q$. Since the values $pk^1$ and $pk'$ are each computed as $pk^1 = g^{sk^1}$ and $pk' = g^{sk'}$ respectively, both $pk^1$ and $pk'$ are uniformly distributed in $\mathbb{G}$[13].

– $\mathsf{Sk\text{-}To\text{-}Vk}$: As evident from the description of the $\mathsf{TKeyGen}$ procedure of the $\mathsf{SG}$ scheme, it holds for all $(pk', \{vk'_i\}_{i\in[n]}, \{sk'_i\}_{i\in[n]}) \leftarrow \mathsf{SG.TKeyGen}(pp^{\mathsf{SG}})$ that $vk'_i = g^{sk'_i}$ for $i \in [n]$.
– $\mathsf{Has\text{-}Sim}$: In order to show that the $\mathsf{SG}$ scheme satisfies this property, we briefly recall the main idea behind the security reduction of the $\mathsf{SG}$ scheme to the computational Diffie-Hellman problem in presence of a static adversary $\mathcal{A}$ in game $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ (see Sec. 2.3) as presented in [43].

*Reduction of* $\mathsf{SG}$ *to computational Diffie-Hellman.* The reduction in [43] shows that if there exists a static adversary $\mathcal{A}$ that wins the $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ game (cf. Def. 6) with non-negligible probability, then there also exists a simulator $\mathcal{S} := (\mathcal{S}_1, \mathcal{S}_2)$[14] which solves the CDH problem with non-negligible probability, i.e., upon receiving a CDH instance $(g, g^a, g^b) \in \mathbb{G}^3$, $\mathcal{S}$ can compute $g^{ab}$. W.l.o.g. let $\mathcal{A}$ corrupt parties $(P_1, \cdots, P_t)$. Further, let $pk = g^a$, $sk_i \leftarrow_\$ \mathbb{Z}_q$ and $vk_i \leftarrow g^{sk_i}$ for $i \in [t]$. Then the simulator $\mathcal{S}$ first executes subprocedure $\mathcal{S}_1$ on input $(pk, \{vk_i, sk_i\}_{i\in[t]})$, which computes $vk_j = pk^{l_{j,0}} \prod_{i=1}^{t} vk_i^{l_{j,i}}$ for $t+1 \le j \le n$. 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 $\mathsf{TPKE\text{–}CCA}^{\mathcal{A}}_{\mathsf{SG}}$ as follows: In the beginning of the game, $\mathcal{S}_2$ sends the tuple $(pk, \{vk_j\}_{j\in[n]}, \{sk_i\}_{i\in[t]})$ to the adversary. Upon $\mathcal{A}$ initiating the challenge phase of the game by sending a label $L'$ and two messages $m_0, m_1$, $\mathcal{S}_2$ sets the challenge ciphertext $ct'$ as follows: it chooses $c' \in \{0,1\}^l$ and $k', e', f' \in \mathbb{Z}_q$ at random. Then it sets

$$u' = g^b, \bar{g}' = g^{k'}, \bar{u}' = (u')^{k'}, w' = g^{f'}/(u')^{e'}, \bar{w}' = (\bar{g}')^{f'}/(\bar{u}')^{e'}.$$

Finally, the simulator programs $H_2$ and $H_3$ such that $H_2(c', L', u', w') = \bar{g}'$ and $H_3(\bar{g}', \bar{u}', \bar{w}') = e'$ and $\mathcal{S}_2$ outputs $ct' = (c', L', u'\bar{u}', e', f')$ to $\mathcal{A}$. The idea is that in order to guess which message is encrypted in $ct'$, the adversary has to query $H_1$ on input $u'^{sk} = g^{ab}$ and thereby $\mathcal{S}_2$ can learn the solution to the CDH problem. It remains to show the simulation of the decrpytion oracle. For each query to $H_2$ on a different input than $(c', L', u', w')$, $\mathcal{S}_2$

[13] We assume that $\Pi_{\mathsf{LS\text{–}DKG}}$ and $\mathsf{SG}$ operate over the same group $\mathbb{G}$ with generator $g$.
[14] For ease of explanation, we present the simulator w.r.t. two subprocedures $\mathcal{S}_1$ and $\mathcal{S}_2$.

chooses a random $k \in \mathbb{Z}_q$ and outputs $\bar{g} = h^k$. Upon receiving a decryption query for ciphertext $ct = (c, L, u, \bar{u}, e, f)$, $\mathcal{S}_2$ has to compute $u_j = vk_j^r$ for $t+1 \leq j \leq n$ along with a simulated NIZK proof to prove the correctness of the decryption. $\mathcal{S}_2$ first verifies that $ct$ is a valid ciphertext and if this check passes, it computes $u_j = (\bar{u})^{l_{j,0}/k} \prod_{i=1}^{t} u^{sk_i l_{j,i}}$.
$\mathcal{S}_2$ satisfies the requirements of property Has-Sim.

## E Large-Scale Non-Interactive Threshold Signature Schemes

In this section, we first introduce the notion of large-scale non-interactive threshold signature schemes (LS–TSIG), before we argue that our generic transformation from a discrete-log-based TPKE scheme to a large-scale threshold PKE scheme as presented in Sec. 5 works similarly for discrete-log-based threshold signature schemes (TSIG). In Appendix F, we then show that the threshold signature scheme by Boldyreva [9], which we denote by TH–BLS, satisfies the (slightly adjusted) properties Has-DKG, Sk-To-Vk and Has-Sim as required for our generic transformation in Sec. 5.

### E.1 Large-Scale Non-Interactive Threshold Signature Scheme

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

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

**Definition 12.** *A large-scale non-interactive $(t, n)$-threshold signature scheme* (LS–TSIG) *is defined w.r.t. a universe of parties $U = \{P_1, \cdots, P_N\}$ with $N > n$ and consists of a tuple* LS–TSIG = (Setup, TKeyGen, TSign, TShareVrfy, TCombine, Verify, Refresh) *of efficient algorithms and protocols which are defined as follows:*

- Setup$(1^\lambda)$*: This probabilistic algorithm takes a security parameter $\lambda \in \mathbb{N}$ as input and outputs public parameters $pp$.*
- 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 $sk_i$.*
- TSign$(sk_i, m)$*: This algorithm takes as input a secret key share $sk_i$ and a message $m$ and outputs a signature share $\sigma_i$.*

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

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

$\mathsf{Refresh}[C_{\langle sk_1,\cdots,sk_n\rangle}, U](pp)$: *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 $sk_i$ 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 $sk'_i$.*

*Robustness* A $(t, n) - \mathsf{LS{-}TSIG}$ scheme must fulfill the following two robustness 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 $(pk, \{vk_i^j\}_{i\in[n]}, \{sk_i^j\}_{i\in[n]})$ recursively as

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

1. Share verification: For any message $m$ it must hold that:
$$\mathsf{TShareVrfy}(vk_i^j, m, \mathsf{TSign}(sk_i^j, m)) = 1$$
2. Reconstruction: For any message $m$ and any set $T = \{\sigma_1, \cdots, \sigma_{t+1}\}$ of valid signature shares where each signature share is computed as $\sigma_i \leftarrow \mathsf{TSign}(sk_i^j, m)$ with $t + 1$ distinct $sk_i^j$ secret key shares, it must hold that:
$$\mathsf{Verify}(pk, m, \mathsf{TCombine}(pk, m, T)) = 1$$

*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 $\lambda$ 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 game executes $\mathsf{TKeyGen}[U](pp, t, n)$. The protocol selects a committee $C^1$ with $|C^1| = n$ and outputs a public key $pk$, a set of verification keys $\{vk_1^1, \cdots, vk_n^1\}$ and a set of secret key shares $\{sk_1^1, \cdots, sk_n^1\}$, such that ${P_i}^1 \in C^1$ learns $vk_i^1$ and $sk_i^1$.
4. Additionally to the corruption oracle, the adversary $\mathcal{A}$ obtains access to the following oracles:
   – **Refresh oracle**: On input a set $NB^j \subseteq B^j$, the game executes the protocol $\mathsf{Refresh}[C^j_{\langle sk_1^j, \cdots, sk_n^j \rangle}, U](pp)$ and sets $B^{j+1} \leftarrow B^j \setminus NB^j$.
   – **Signing oracle**: On input a set $NB^j \subseteq B^j$ and a set of messages $M^j$, the game computes $\sigma_{i,k}^j \leftarrow \mathsf{TSign}(sk_i^j, m_k^j)$ for $m_k^j \in M^j$ for all parties ${P_i}^j \in C^j \setminus B^j$. Then, the oracle calls the Refresh oracle on input $NB^j$ and finally the game returns all tuples $(\{\sigma_{i,k}^j\}_{k \in |M^j|}, vk_i^j)$ to $\mathcal{A}$.
5. Eventually, $\mathcal{A}$ outputs a message $m'$ and a signature $\sigma'$. $\mathcal{A}$ wins the game if it has never previously queried the signing oracle on message $m'$ and if $\mathsf{Verify}(pk, m', \sigma') = 1$.

**Definition 13.** *A large-scale non-interactive $(t, n)$-threshold signature scheme* LS–TSIG *with a universe of parties $U$ is $(\lambda, n, t, p)$-unforgeable with $p \cdot |U| > t$ if for every efficient fully mobile adversary $\mathcal{A}$ with corruption power p there exists a negligible function $\nu$ in the security parameter $\lambda$, such that*

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

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

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

We call a large-scale non-interactive $(t, n)$-threshold signature scheme LS–TSIG scheme $(\lambda, n, t, p)$-secure, if it satisfies the robustness and $(\lambda, n, t, p)$-unforgeability property.

### E.2 Generic Transformation from TSIG to LS–TSIG

We argue that we can generically transform a discrete-log-based non-interactive $(t, n)$-threshold signature scheme TSIG as defined in Definition 16 into a fully-secure non-interactive $(t, n)$-threshold signature scheme LS–TSIG. Our construction works in the same spirit as the generic construction from TPKE to LS–TPKE as presented in Sec. 5.2, i.e., we make use of our $\Pi_{\mathsf{LS\text{–}DKG}}$ protocol as presented in Sec. 4, the $\Sigma_{\mathsf{ECPSS}}$ scheme as described in Sec. 3 and a non-interactice zero-knowledge proof system NIZK. In fact, there are only the following three minor changes required as compared to the generic transformation from Sec. 5.2: (1) the properties Has-DKG, Sk-To-Vk and Has-Sim must be defined w.r.t. TSIG instead of TPKE, (2) the algorithm $\mathcal{S}$ in property Has-Sim obtains access to a signing oracle and (3) all procedures that are defined w.r.t. TPKE in $\Pi_{\mathsf{LS\text{–}TPKE}}$ must be replaced by procedures from TSIG, where TEnc is replaced by TSign and TDec is replaced by Verify.

We now formally define the required properties for our transformation. As discussed above, we only require minimal changes as compared to the properties and transformation from Sec. 5.2.

Let $U$ be a universe of parties. Let $\lambda \in \mathbb{N}$ be the security parameter, $pp^{\mathsf{LS-DKG}} \leftarrow \Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}(1^\lambda)$, $pp^{\mathsf{TSIG}} \leftarrow \mathsf{TSIG}.\mathsf{Setup}(1^\lambda)$ and $t, n \in \mathbb{N}$ s.t. $1 \leq t \leq n$. Note that $pp^{\mathsf{LS-DKG}}$ can be parsed as $pp^{\mathsf{LS-DKG}} := (\mathsf{crs}, \mathbb{G}, q, g)$.

1. Has-DKG: For all
$$(pk, \{sk_i^1\}_{i\in[n]}) \leftarrow \Pi_{\mathsf{LS-DKG}}.\mathsf{TKeyGen}[U](pp^{\mathsf{LS-DKG}}, t, n)$$
and all
$$(pk', \{vk'_i\}_{i\in[n]}, \{sk'_i\}_{i\in[n]}) \leftarrow \mathsf{TSIG}.\mathsf{KeyGen}(pp^{\mathsf{TSIG}}, t, n),$$
the distributions of the tuples $(pk, \{sk_i^1\}_{i\in[n]})$ and $(pk', \{sk'_i\}_{i\in[n]})$ are indistinguishable[15].

2. Sk-To-Vk: Let $(pk', \{vk'_i\}_{i\in[n]}, \{sk'_i\}_{i\in[n]}) \leftarrow \mathsf{TSIG}.\mathsf{KeyGen}(pp^{\mathsf{TSIG}}, t, n)$. It holds that $vk'_i = g^{sk_i}$ for all $i \in [n]$.

3. Has-Sim: There exists an efficient algorithm $\mathcal{S}$ that behaves as follows: Let $(pk, \{vk'_j\}_{j\in[n]}, \{sk_i\}_{i\in[n]})$ be a tuple generated by the TSIG.KeyGen procedure i.e., $(pk, \{vk'_j\}_{j\in[n]}, \{sk_i\}_{i\in[n]}) \in \mathsf{TSIG}.\mathsf{KeyGen}(pp^{\mathsf{TSIG}}, t, n)$ and let the algorithm $\mathcal{S}$ obtain access to an oracle that generates valid signatures on arbitrary messages under $pk$. Then $\mathcal{S}$, on input the public key $pk$, verification keys $\{vk'_j\}_{j\in[n]}$ and only $t$ of the secret key shares $\{sk_i\}_{i\in[t]}$, can simulate the game $\mathsf{SIG-UFCMA}^{\mathcal{A}}_{\mathsf{TSIG}}$ as presented in Def. 16 w.r.t. $(pp^{\mathsf{TSIG}}, pk, \{vk'_j\}_{j\in[n]})$ to an efficient static adversary $\mathcal{A}$ such that $\mathcal{A}$ cannot distinguish the execution of the simulated game from the execution of the real game except with negligible probability in $\lambda$.

**Construction** We generically construct a large-scale threshold signature scheme $\Pi_{\mathsf{LS-TSIG}} = (\mathsf{Setup}, \mathsf{TKeyGen}, \mathsf{TSign}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Verify}, \mathsf{Refresh})$ which is secure against fully mobile adversaries using the large-scale distributed key generation scheme $\Pi_{\mathsf{LS-DKG}} = (\mathsf{Setup}, \mathsf{TKeyGen})$ as described in Sec. 4, the ECPSS scheme $\Sigma_{\mathsf{ECPSS}} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Select}, \mathsf{Handover}, \mathsf{Rec})$ as presented in Sec. 3, a threshold signature scheme $\mathsf{TSIG} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{TSign}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Verify})$ secure against a static adversary and a NIZK proof system $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as per Def. 7. We detail the generic construction below.

$\Pi_{\mathsf{LS-TSIG}}.\mathsf{Setup}(1^\lambda)$: On input a security parameter $\lambda$, execute
$$pp^{\mathsf{TSIG}} \leftarrow \mathsf{TSIG}.\mathsf{Setup}(1^\lambda), pp^{\mathsf{LS-DKG}} \leftarrow \Pi_{\mathsf{LS-DKG}}.\mathsf{Setup}(1^\lambda).$$
$$\mathsf{crs}_1 \leftarrow \mathsf{NIZK}.\mathsf{Setup}(1^\lambda)$$

[15] This property implies that $\Pi_{\mathsf{LS-DKG}}$ and TSIG operate over the same group $\mathbb{G}$ with the same generator $g$.

Recall that $pp^{\mathsf{LS\text{-}DKG}}$ can be parsed as $pp^{\mathsf{LS\text{-}DKG}} := (\mathsf{crs}_2, \mathbb{G}, q, g)$. Output public parameters $pp := (pp^{\mathsf{TSIG}}, pp^{\mathsf{LS\text{-}DKG}}, \mathsf{crs}_1)$.

$\Pi_{\mathsf{LS\text{-}TSIG}}.\mathsf{TKeyGen}(pp, t, n)$: On input public parameters $pp$ and two integers $t, n \in \mathbb{N}$ s.t. $n \geq 2t + 1$, this protocol parses $pp := (pp^{\mathsf{TSIG}}, pp^{\mathsf{LS\text{-}DKG}}, \mathsf{crs}_1)$ and calls $\Pi_{\mathsf{LS\text{-}DKG}}.\mathsf{TKeyGen}(pp^{\mathsf{LS\text{-}DKG}}, t, n)$. The protocol selects a committee $C^1$ and outputs a public key $pk$ to all parties in $U$ and secret key shares $sk_i^1$ to each party $P_i{}^1 \in C^1$. Additionally, all $P_i{}^1 \in C^1$ compute $vk_i^{1'} := g^{sk_i^1}$ and a NIZK proof $\pi_i^1$ that the verification key $vk_i^{1'}$ was computed correctly[16]. $P_i{}^1$ then sets $vk_i^1 := \{vk_i^{1'}, \pi_i^1\}$.

$\Pi_{\mathsf{LS\text{-}TSIG}}.\mathsf{TSign}(sk_i, m)$: This procedure executes $\mathsf{TSIG.TSign}$.

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

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

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

$\Pi_{\mathsf{LS\text{-}TSIG}}.\mathsf{Refresh}[C^{j-1}_{\langle sk_1^{j-1}, \cdots, sk_n^{j-1} \rangle}, U](pp)$: This protocol is executed between a committee $C^{j-1}$ in epoch $j-1$ and the universe $U$, where each $P_i{}^{j-1} \in C^{j-1}$ receives as input a secret key share $sk_i^{j-1}$ and each party $P_k \in U$ receives as input $pp := (pp^{\mathsf{TSIG}}, pp^{\mathsf{LS\text{-}DKG}}, \mathsf{crs}_1)$. The protocol first runs $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ which selects a committee $C^j$ and outputs refreshed secret key shares $sk_i^j$ to each $P_i{}^j \in C^j$. Additionally, all $P_i{}^j \in C^j$ compute $vk_i^{j'} := g^{sk_i^j}$, generate a NIZK proof $\pi_i^j$ that the verification key was computed correctly[17] and set $vk_i^j := \{vk_i^{j'}, \pi_i^j\}$.

**Theorem 3.** *Let $\Pi_{\mathsf{LS\text{-}DKG}}$ be the large-scale $(t, n)$-distributed key generation protocol from Sec. 4,* TSIG *a non-interactive $(t, n)$-threshold signature scheme which is secure against static adversaries according to Def. 15, $\Sigma_{\mathsf{ECPSS}}$ a $(\lambda, n, t, p)$-secure instantiation of the evolving-committee proactive secret sharing scheme as presented in Sec. 3 and* NIZK *a non-interactive zero-knowledge proof system as per Def. 7. Assume that* TSIG *satisfies the adjusted properties* Has-DKG, Sk-To-Vk, *and* Has-Sim *as defined in Appendix F. Then $\Pi_{\mathsf{LS\text{-}TSIG}}$ is a $(\lambda, n, t, p)$-secure large-scale non-interactive $(t, n)$-threshold signature scheme.*

---

[16] We note that $\pi_i^1$ can be efficiently computed and verified since its witness is $sk_i^1$ and its statement consists of the ciphertexts that are broadcast during the $\Pi_{\mathsf{LS\text{-}DKG}}.\mathsf{TKeyGen}$ procedure.

[17] We note that $\pi_i^j$ can be efficiently computed and verified since its witness is $sk_i^j$ and its statement consists of the ciphertexts that are broadcast during the $\Sigma_{\mathsf{ECPSS}}.\mathsf{Handover}$ procedure.

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

**Lemma 6.** *The large-scale non-interactive* $(t, n)$*-threshold signature scheme,* $\Pi_{\mathsf{LS-TSIG}}$*, satisfies robustness.*

*Proof.* This lemma follows directly from the robustness property of the TSIG scheme (cf. Definition 16), the properties Has-DKG and Sk-To-Vk as defined above, the completeness property of the NIZK proof system and from the reconstruction property of the $\Sigma_{\mathsf{ECPSS}}$ scheme.

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

The proof of Lemma 7 is similar to the proof of Lemma 4.

In Appendix F, we show that the threshold signature scheme by Boldyreva [9], which we denote by TH–BLS, satisfies the adjusted properties Has-DKG, Sk-To-Vk and Has-Sim as discussed above.

# F Concrete LS–TSIG Scheme from Boldyreva TSIG Scheme

## F.1 Background on Digital Signatures and Threshold Signature Schemes

Before we provide a concrete instantiation of our generic transformation with a threshold signature scheme, we first recall the basic definitions of digital signature schemes and threshold signature schemes.

**Definition 14 (Digital signatures).** *A digital signature scheme* SIG *consists of a triple of algorithms* SIG = (KeyGen, Sign, Verify) *defined as:*

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

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

Verify($pk, m, \sigma$)**:** *This deterministic algorithm takes as input a public key pk, message m and signature* $\sigma$ *and outputs a bit either* 1 *or* 0*. If the output is* 1*,* $\sigma$ *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 15 (Unforgeability).** *A signature scheme* $\mathsf{SIG}$ *is unforgeable if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\nu$ *in the security parameter* $\lambda$ *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* $\sigma$ *for* $m$ *under public key* $pk$*.*
3. *Eventually,* $\mathcal{A}$ *outputs a forgery* $(m^*, \sigma^*)$ *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.*

### F.2 Threshold Signature Scheme

**Definition 16.** *A non-interactive* $(t, n)$*-threshold signature scheme* $\mathsf{TSIG}$ *consists of a tuple* $\mathsf{TSIG} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{TSign}, \mathsf{TShareVrfy}, \mathsf{TCombine}, \mathsf{Verify})$ *of efficient algorithms 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* $\sigma_i$*.*
$\mathsf{TShareVrfy}(\sigma_i, vk_i, m)$*: This deterministic algorithm takes as input a signature share* $\sigma_i$*, a verification key* $vk_i$ *and a message* $m$ *and it outputs either* $1$ *or* $0$*. If the output is* $1$*,* $\sigma_i$ *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* $\sigma$*.*
$\mathsf{Verify}(pk, m, \sigma)$*: This deterministic algorithm takes as input a public key* $pk$*, message* $m$ *and signature* $\sigma$ *and outputs a bit either* $1$ *or* $0$*. If the output is* $1$*,* $\sigma$ *is called a valid signature.*

*Robustness* A $(t, n)-\mathsf{TSIG}$ scheme must fulfill the following two robustness 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. Signing robustness: For any message $m$ it must hold that
$$\mathsf{TShareVrfy}(\mathsf{TSign}(sk_i, m), vk_i, m) = 1.$$
2. Reconstruction robustness: 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 $\mathsf{SIG}\text{–}\mathsf{UFCMA}^{\mathcal{A}}_{\mathsf{TSIG}}$ which receives as input a security parameter $\lambda$:

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 $(m^*, \sigma^*)$ 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 17.** *A non-interactive* $(t, n)$*-threshold signature scheme* $\mathsf{TSIG}$ *is unforgeable if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\nu$ *in the security parameter* $\lambda$*, such that* $\Pr[\mathsf{SIG}\text{–}\mathsf{UFCMA}^{\mathcal{A}}_{\mathsf{TSIG}}(\lambda) = 1] \leq \nu(\lambda)$.

### F.3 Concrete Instantiation with the Boldyreva Threshold Signature Scheme

In the following we present the non-interactive threshold signature scheme from [9], which we denote by TH–BLS and which has been proven secure against static adversaries according to Definition 17. We then show that TH–BLS can be transformed into a large-scale non-interactive threshold signature scheme, which is secure against fully mobile adversaries using our generic transformation from Appendix E. In order to do so, we must show that the properties Has-DKG, Sk-To-Vk and Has-Sim hold for TH–BLS. To show that Has-Sim holds, we give a proof sketch for a reduction of TH–BLS to the single party signature scheme BLS as introduced in [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 18 (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* V-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 [11], before presenting its threshold variant TH–BLS as presented in [9].

### F.4 The BLS scheme

KeyGen$(1^\lambda)$*:* On input a security parameter $\lambda$, 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 $(sk', pk')$.

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 $\sigma$.

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

The authors of [11] show that the BLS scheme is unforgeable as per Definition 15.

### F.5 The TH–BLS scheme

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

Setup$(1^\lambda)$*:* On input a security parameter $\lambda$, 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 $pp := (\mathbb{G}, p, g, H)$.

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 $sk_i$.

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

TShareVrfy$(vk_i, m, \sigma_i)$*:* On input a verification key $vk_i$, a message $m$ and a signature share $\sigma_i$, this algorithm outputs V-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 $\sigma$.

**Theorem 4.** *If the* BLS *scheme as presented in Section F.4 is unforgeable as per Definition 15, then the* TH–BLS *scheme is unforgeable as per Definition 17 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 $\sigma_i$ 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 $(m^*, \sigma^*)$, $\mathcal{S}_2$ 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 $(m^*, \sigma^*)$ 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 $(m^*, \sigma^*)$ constitutes a valid forgery in game $\mathsf{SIG\text{–}UFCMA}^{\mathcal{S}}_{\mathsf{BLS}}$.

TH–BLS *satisfies the properties from Appendix E.2* We now show that the TH–BLS scheme satisfies the three properties Has-DKG, Sk-To-Vk and Has-Sim as required by our generic transformation from Appendix E.2.

- Has-DKG: Same as in Appenidx D. Note that we assume that $\Pi_{\mathsf{LS\text{–}DKG}}$ and TH–BLS operate over the same group $\mathbb{G}$. This is not an issue since $\Pi_{\mathsf{LS\text{–}DKG}}$ only requires that the dlog assumption holds in $\mathbb{G}$ which must be the case for any GDH group.
- Sk-To-Vk: Same as in Appendix D.
- Has-Sim: As shown in the proof sketch of Theorem 4, the algorithm $\mathcal{S}_2$ satisfies the requirements of the Has-Sim property.