Non-interactive distributed key generation and key resharing

Jens Groth[1]

jens@dfinity.org
DFINITY Foundation

Draft
March 16, 2021

Abstract. We present a non-interactive publicly verifiable secret sharing scheme where a dealer can construct a Shamir secret sharing of a field element and confidentially yet verifiably distribute shares to multiple receivers. We also develop a non-interactive publicly verifiable resharing scheme where existing share holders of a Shamir secret sharing can create a new Shamir secret sharing of the same secret and distribute it to a set of receivers in a confidential, yet verifiable manner.
A public key may be associated with the secret being shared in the form of a group element raised to the secret field element. We use our verifiable secret sharing scheme to construct a non-interactive distributed key generation protocol that creates such a public key together with a secret sharing of the discrete logarithm. We also construct a non-interactive distributed resharing protocol that preserves the public key but creates a fresh secret sharing of the secret key and hands it to a set of receivers, which may or may not overlap with the original set of share holders.
Our protocols build on a new pairing-based CCA-secure public-key encryption scheme with forward secrecy. As a consequence our protocols can use static public keys for participants but still provide compromise protection. The scheme uses chunked encryption, which comes at a cost, but the cost is offset by a saving gained by our ciphertexts being comprised only of source group elements and no target group elements. A further efficiency saving is obtained in our protocols by extending our single-receiver encryption scheme to a multi-receiver encryption scheme, where the ciphertext is up to a factor 5 smaller than just having single-receiver ciphertexts.
The non-interactive key management protocols are deployed on the Internet Computer to facilitate the use of threshold BLS signatures. The protocols provide a simple interface to remotely create secret-shared keys to a set of receivers, to refresh the secret sharing whenever there is a change of key holders, and provide proactive security against mobile adversaries.

1 Introduction

The Internet Computer hosts clusters of nodes running subnets (shards) that host finite state machines known as canisters (advanced smart contracts). The subnets authenticate all communication to end users and between themselves using BLS signatures, which happens automatically so canister developers do not need to deal directly with authentication mechanisms. The nodes running a subnet use threshold cryptography to create the BLS signatures such that the subnet is resilient to compromise of a few nodes. The Internet Computer makes key management easy by assigning each subnet a permanent public key. The nodes running the subnet may change over time as some of them are taken out for maintenance or upgrade or there are other reasons to change the topology of the Internet Computer, but even under these changes the subnet is known by the same public key. The motivation behind this work is to develop key management protocols that make it easy to spin up a new subnet and provide the participating nodes a secret sharing of the subnet's signing key, and to make it easy for existing participants to reshare the secret key to a new set of nodes whenever the composition of the subnet changes. It is desirable for these protocols to be non-interactive since it eliminates the need for incoming nodes to participate in the protocol before they have joined the subnet and to support deployment on an asynchronous network such as the Internet.

1.1 Our contribution

We construct non-interactive distributed key generation and key resharing protocols that support Shamir secret sharings of the secret keys. The distributed key generation protocol enables a set of dealers to participate in the creation of a public verification key for the threshold BLS signature scheme together with Shamir secret shares of the secret key that are encrypted under the public keys of a set of receivers. A set of nodes that already hold a Shamir secret sharing of a secret key can use the distributed key resharing protocol to act as dealers and create a fresh random secret sharing of the same secret key and encrypt the shares under the public keys of the receiving nodes.

The protocols are designed to be non-interactive, which simplifies their usage and the communication pattern. Each dealer creates a dealing without interacting with other participants and publishes it. Each receiver can verify whether a dealing is correct without interaction with other participants. And given an agreed set of verified dealings the receivers can derive a public key for the BLS signature scheme and retrieve their shares of the secret signing key.

The basic idea behind our protocols is to use existing information theoretic schemes for Shamir secret sharing and resharing and encrypt the messages. We then use non-interactive zero-knowledge proofs to ensure each dealing, essentially a batch of ciphertexts, is correct. Since anybody can verify the NIZK proof, this means the dealings are publicly verifiable and everybody will agree on whether a dealing is valid or not. This removes the need for interaction that appears in existing verifiable secret sharing schemes where only the receiver can verify whether her share is valid or not, and hence needs to complain if a dealer sent her a bad share, which in turn requires all participants to wait and see who complains and whether there is a sufficiently large quorum to proceed.

To make key management easy, each node has a static public encryption key. However, to provide some protection against compromise we use encryption with forward secrecy, where time is divided into epochs and the decryption key can evolve such that ciphertexts for future epochs can be decrypted but ciphertexts from past epochs cannot be decrypted. A general paradigm for constructing pairing-based encryption schemes with forward secrecy is to use hierarchical identity based encryption schemes and derive keys for different epoch according to a tree-shaped hierarchy. These forward secure encryption schemes use the target group as message space. However, in our case the plaintexts are field elements constituting Shamir secret shares. If we encode the field elements directly in the target group, the NIZK proofs become unwieldy. Instead we divide the plaintexts into small chunks, which can be encrypted in the exponent and later be extracted using the Baby-step Giant-step algorithm. While chunking incurs an overhead, it also makes it possible to encode the plaintext a source group pre-image of the target group element we want the receiver to get and decode. Exploring this idea we develop a novel pairing-based encryption scheme with forward secrecy where the ciphertext consists of source group elements. Since source group elements are smaller than target group elements this gives a significant saving offsetting the overhead that comes from chunking the plaintexts.

Our new forward-secure encryption scheme has an additional benefit, it is structured such that ciphertexts addressed to different public keys may share randomness. Sharing randomness gives a significant performance improvement, while a single-receiver ciphertext for a chunk consists of 3 small source group elements and one double-size source group element, amortizing over many receivers we can reduce that cost almost down to a single small source group element per receiver.

Finally, it has also been a design goal to facilitate efficient NIZK proofs, since they place a computational burden on the verifiers. Since the encryption algorithm for the forward secure encryption scheme mainly uses exponentiations, we are in a beneficial setting for developing Schnorr-style proofs and applying the Fiat-Shamir heuristic to make them non-interactive. Our resulting NIZK proofs are compact and smaller than the ciphertexts. It is natural to use range proofs to show the encrypted chunks are appropriately sized using range proofs, but range proofs are complicated to construct. Instead we opted for approximate range proofs. Here the verifier does not get the guarantee that a chunk is in a given small range, the guarantee is just that the chunk is in a set that is small enough to be decrypted by brute force, i.e., we just give decryptability proofs. We construct a novel and simple decryptability proof. While the proof system itself is simple, the analysis is not quite so easy, since the proof system does not have perfect completeness and we therefore use rejection sampling to make the completeness error negligible.

1.2 Related work

In the full paper, we will provide further recent improvements and compare our work to the extensive literature on secret sharing and distributed key generation.

2 Preliminaries

2.1 Notation

In the following we write $y := x$ for assigning y the value x. Sampling from a probability distribution, we write $y \leftarrow \mathcal{D}$. When $\mathcal{D}$ samples uniformly at random from a set S, we write $y \xleftarrow{\$} S$. For repeated independent and uniformly random sampling we write $y_1, \ldots, y_n \xleftarrow{\$} S$.

Let A be an algorithm, which may be randomized. We write $y \leftarrow A(x)$ for assigning y the output of A running on input x. If the algorithm is deterministic there is only one possible output, and we will sometimes write $y := A(x)$. We may also think of a randomized algorithm as a deterministic computation based on the input x and some randomness r, in which case we can write $y := A(x; r)$. When discussing algorithms, we will often require them to be efficient. It is context dependent what that means though, so we will not provide an explicit definition of efficiency.

We write $A^{\mathcal{O}}$ for an algorithm that has access to an oracle $\mathcal{O}$, which may be thought of as a subroutine. The oracle provides some interface to the algorithm to which it can send an input and get back an output from the oracle. If for instance H is a hash function, we can write A^H for an algorithm that can call the hash function on a message m and get back a message digest $H(m)$.

Functions may be parametrized, a hash function may for instance have output length $\lambda = 256$. In general we use λ to denote a parameter that indicates the size of an object, e.g., λ_e could be the bit-length of an integer e. It depends on the context whether it is more natural to consider a value a parameter that is implicitly known to all and part of the setup of a protocol, or whether it is more natural to give it as explicit input to algorithms.

When considering security, we will usually write $\mathcal{A}$ for an adversary. Security is defined through experiments, where we precisely define the power and access to system manipulation the attacker has. We say the advantage of an adversary is the probability that it wins in the experiment. As an example, let us take an attacker that tries to guess the value of a fair coin toss. Here the advantage would be

$$\mathbf{Adv}(\mathcal{A}) = |2\Pr[b^* \leftarrow \mathcal{A}; b \xleftarrow{\$} \{0,1\} : b^* = b] - 1|,$$

with the implicit experiment in the notation being

```
$\mathbf{Exp}(\mathcal{A})$
$b^* \leftarrow \mathcal{A}$               // the attacker outputs a guess
$b \xleftarrow{\$} \{0,1\}$               // we toss a coin
if $b^* = b$ return $\top$                // $\top$ indicates success
else return $\bot$                        // $\bot$ indicates failure
```

and the adversary's advantage being $\mathbf{Adv}(\mathcal{A}) = |2\Pr[\mathbf{Exp}(\mathcal{A}) = \top] - 1|$.

We sometimes distinguish between perfect security, where no attacker gets any advantage; statistical security where no attacker can get significant advantage; and computational security where we believe, based on specific assumptions, no realistic computationally bounded attacker gets a significant advantage.

2.2 Fields, groups and pairings

We write $\mathbb{Z}_p$ for integers modulo p. In this article, p will always be a known prime and therefore $\mathbb{Z}_p$ is a finite field (also sometimes written $\mathbb{F}_p$). We assume $\mathbb{Z}_p$ comes with a canonical representation of field elements as integers $0, 1, \ldots, p-1$ and accordingly there are efficient algorithms to compute the field operations. We write $\mathbb{Z}_p^*$ for the multiplicative subgroup of the field, i.e., $\{1, \ldots, p-1\}$. We write $y := x \bmod p$ for assigning to y the canonical representation of x modulo p.

We write $\mathbb{G}$ for a group of known prime order p. All prime order groups are cyclic, so using multiplicative notation $\mathbb{G} = \{1, g^1, \ldots, g^{p-1}\}$ for some generator g, where the unit is $1 = g^0$. When we refer to groups we always assume they have known prime order, a canonical and compact representation of group elements, and efficient algorithms to compute group operations and decide membership.

We use pairing-based cryptography, where we have two source groups $\mathbb{G}_1, \mathbb{G}_2$ and a target group $\mathbb{G}_T$ of known prime order p. A pairing (in cryptographic parlance, deviating from standard mathematical terminology) is a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. This means if g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$, then $e(g_1, g_2)$ generates $\mathbb{G}_T$ and for all $a, b \in \mathbb{Z}_p$ we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$. We require that the pairing be efficiently computable. For security purposes, we will assume there are no non-trivial efficiently computable homomorphisms between the source groups $\mathbb{G}_1$ and $\mathbb{G}_2$, i.e., we are working with Type III pairings in the categorization of Galbraith, Paterson and Smart [GPS08]. We follow the convention that $\mathbb{G}_1$ is the source group that has the most compact representation of group elements and the most efficient computation.

In the implementation, we instantiate the pairing groups using BLS12-381 with group size $p \approx 2^{255}$. The source groups $\mathbb{G}_1, \mathbb{G}_2$ are elliptic curves with a base field of size $q \approx 2^{381}$ and the target group $\mathbb{G}_T$ is an order p multiplicative subgroup of $\mathbb{F}_{q^{12}}^*$.[1]

2.3 Hash functions

We rely on cryptographic hash functions to compress data. A standard cryptographic hash function is an efficiently computable function $H : \{0,1\}^* \to \{0,1\}^\lambda$ that takes arbitrary length input and maps it to a λ-bit string, where λ is a fixed parameter. An example is SHA-256, which hashes arbitrary[2] length strings into 256-bit strings.

We will not just hash to fixed length outputs though. We let $H_\lambda : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function that maps arbitrary length input to a string of specified length λ. We also sometimes map in and let $H_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ be a hash function that maps into the group $\mathbb{G}$. And we may map into a field, and let $H_{\mathbb{Z}_p} : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function mapping into $\mathbb{Z}_p$. We implicitly assume

[1] For general information about this choice of groups, see https://hackmd.io/@benjaminion/bls12-381.

[2] Strictly speaking the domain is string of length up to $2^{2^{64}}$ bits, but that is equivalent to arbitrary length strings in practice.

the use of domain separators in the hash functions, to ensure they are specific to the schemes presented here and not used elsewhere.

2.4 Shamir secret sharing and Lagrange interpolation

Threshold secret sharing enables a dealer with a secret s to create shares $s_1, \ldots, s_n$ such that any t shares suffice to compute the secret s, while $t-1$ shares reveal no information about the secret.

Shamir secret sharing [Sha79] is a popular secret sharing scheme for secrets in a field $\mathbb{Z}_p$. The idea is to choose a random degree $t-1$ polynomial $a(x)$ such that $a(0) = s$ and let the shares be $s_1 = a(1), \ldots, s_n = a(n)$.[3] The idea behind this secret sharing is that any t points in the plane with distinct x-coordinates uniquely determine a degree $t-1$ polynomial $a(x)$ through them and allows reconstruction of the secret $s = a(0)$. On the other hand, given $t-1$ points with distinct non-zero x-coordinates there are p possible polynomials through them, each yielding a different secret, so the points do not leak any information about the secret.

To describe Shamir secret sharing formally, let us first define the Lagrange interpolation polynomials over $\mathbb{Z}_p$. Given a set $I = \{i_1, \ldots, i_t\}$ of distinct indices and an index $i_j \in I$, we define

$$L^I_{i_j}(x) = \prod_{i \in I \setminus \{i_j\}} \frac{x-i}{i_j - i} \bmod p.$$

We observe, that all the Lagrange polynomials have degree $t-1$ and satisfy $L^I_{i_j}(i_k) = 1$ for $k = j$ and $L_{i_j}(i_k) = 0$ for $k \neq j$. Consequently, given points and shares $(i_1, a(i_1)), \ldots, (i_t, a(i_t))$ on a degree $t-1$ polynomial $a(x)$, we see

$$a(x) = \sum_{i_j \in I} L^I_{i_j}(x) a(i_j) \bmod p.$$

With a secret sharing where $s_{i_1} = a(i_1), \ldots, s_{i_t} = a(i_t)$ this means the shared secret $s = a(0)$ can be reconstructed as

$$s = \sum_{i_j \in I} L^I_{i_j}(0) s_{i_j} \bmod p.$$

Which gives us the following algorithms for an (n,t)-Shamir secret sharing:

$\mathsf{Share}(n,t,s) \to (s_1, \ldots, s_n)$**:** Given $s \in \mathbb{Z}_p$ set $a_0 := s$. Pick $a_1, \ldots, a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$ and define $a(x) = \sum_{k=0}^{t-1} a_k x^k \bmod p$. Return $(s_1, \ldots, s_n) := (a(1), \ldots, a(n))$.

[3] It is straightforward to generalize Shamir secret sharing to use any $n+1$ distinct indices in $\mathbb{Z}_p$ for the secret and the n shares but for simplicity we just use $0, 1, \ldots, n$ in the article.

$\mathsf{Reconstruct}(I, s_{i_1}, \ldots, s_{i_t})$**:** Given a set I of distinct indices $1 \leq i_1 < \ldots < i_t \leq n$ and shares $s_{i_1}, \ldots, s_{i_t} \in \mathbb{Z}_p$ return

$$s := \sum_{i_j \in I} L^I_{i_j}(0) s_{i_j} \bmod p.$$

Secret sharing poses a problem for the receiver: did she get a correct share? The dealer may give her a bad share that does not correspond to the dealing, or give so many fake shares to different receiver that they do not correspond to a real dealing. Feldman [Fel87] proposed verifiable secret sharing (VSS) to deal with this problem. His scheme uses a group $\mathbb{G}$ of order p. The dealers distributes shares together with public group elements $A_0 = g^{a_0}, \ldots, A_{t-1} = g^{a_{t-1}}$. Receiver i is supposed to get share $s_i = a(i)$, which can now be checked since a correct share satisfies

$$g^{s_i} = g^{a(i)} = g^{\sum_{k=0}^{t-1} a_k i^k} = \prod_{k=0}^{t-1} A_k^{i^k}.$$

Many verifiable secret sharing schemes use related ideas. They then let the receiver issue a complaint in case his share is wrong. As it turns out, having a separate complaint phase makes the key distribution scheme we will build on top of the secret sharing scheme interactive. So instead we will be looking in this article to construct a publicly verifiable secret sharing (PVSS) scheme where it is immediately verifiable to everybody, not just the receiver, whether a share is correct or not.

2.5 Resharing

Suppose we already have an (n, t)-Shamir secret sharing of a secret s but want to convert it into an (n', t')-Shamir secret sharing of the same secret s. For this we can use a resharing scheme, which is intended to be run by t dealers holding the original shares and giving the new shares to n' receivers. The idea is that dealer i creates an (n', t') secret sub-sharing of her secret share s_i. Then she gives the n' sub-shares to the respective receivers. Once a receiver has her t sub-shares, she can then use Lagrange interpolation to recombine them to a new secret share. Formally, we define the following resharing scheme intended to be run on a secret sharing $(s_1, \ldots, s_n)$:

$\mathsf{Reshare}(n', t', s_i)$: Return $(s_{i,1}, \ldots, s_{i,n'}) \leftarrow \mathsf{Share}(n', t', s_i)$.
$\mathsf{Combine}(I, s_{i_1,j}, \ldots, s_{i_t,j})$**:** Return $s'_j \leftarrow \mathsf{Reconstruct}(I, s_{i_1,j}, \ldots, s_{i_t,j})$.

The important property is for all correct (n, t)-secret sharings $(s_1, \ldots, s_n)$ of a secret $s \in \mathbb{Z}_p$, positive integers $t' \leq n'$, index set $I \subset [1..n]$ of size t and index set $J \subset [1..n']$ of size t' we have

$$\Pr\left[\begin{array}{ll} \text{for } i \in I : (s_{i,j_1}, \ldots, s_{i,j_{t'}}) \leftarrow \mathsf{Reshare}(n', t', s_i) & \\ \text{for } j \in J : s'_j \leftarrow \mathsf{Combine}(I, s_{i_1,j}, \ldots, s_{i_t,j}) & : s' = s \\ s' \leftarrow \mathsf{Reconstruct}(J, s'_{j_1}, \ldots, s'_{j_{t'}}) & \end{array}\right] = 1.$$

A key observation to see this is true is that Lagrange interpolation is linear. The original secret shares can be recombined using Lagrange interpolation to reconstruct the secret. But applying the same reconstruction procedure to the sub-shares also yields a secret sharing of the same secret. The receivers, provided they know the indices of the dealers, can therefore locally reconstruct (n', t')-shares of their new shares using Lagrange interpolation. To see this, we calculate:

$$\begin{aligned} s' &= \sum_{j_\ell \in J} L^J_{j_\ell}(0) s'_{j_\ell} = \sum_{j_\ell \in J} L^J_{j_\ell}(0) \left(\sum_{i_k \in I} L^I_{i_k}(0) s_{i_k, j_\ell} \right) \\ &= \sum_{i_k \in I} L^I_{i_k}(0) \left(\sum_{j_\ell \in J} L^J_{j_\ell}(0) s_{i_k, j_\ell} \right) = \sum_{i_k \in I} L^I_{i_k}(0) s_{i_k} = s \end{aligned}$$

There are several special purpose resharing schemes, for instance if the set of dealers is the same as the set of receivers, they can instead jointly create a secret sharing of 0 and add it to their existing shares, to get a fresh (n, t)-secret sharing. Or they can increase the threshold by adding an $(n, t + 1)$-secret sharing of 0 to their shares. But for simplicity we will only work with the resharing scheme given above.

2.6 The Schwartz-Zippel lemma

We will later develop zero-knowledge proofs where a common theme is to test polynomial identities. The Schwartz-Zippel lemma is useful in randomized testing of polynomial identities and states that for a multi-variate polynomial $f(x_1, \ldots, x_n) \in \mathbb{Z}_p[x_1, \ldots, x_n]$ of total degree d that for any set $S \subset \mathbb{Z}_p$

$$\Pr[x_1, \ldots, x_n \xleftarrow{\$} S : f(x_1, \ldots, x_n) = 0 \bmod p] \leq \frac{d}{|S|}.$$

3 Signatures

For completeness, we recap the security definition for a signature scheme $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{SigVfy})$. It is perfectly correct if for all $m \in \{0, 1\}^*$ we have

$$\Pr[(vk, sk) \leftarrow \mathsf{KGen}; \sigma \leftarrow \mathsf{Sign}(sk, m) : \mathsf{SigVfy}(vk, m, \sigma) = \top] = 1.$$

The advantage of an adversary against strong existential unforgeability under adaptive chosen message attack is

$$\mathbf{Adv}(\mathcal{A}) = \Pr[(vk, sk) \leftarrow \mathsf{KGen}; (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}}(vk) : \mathsf{SigVfy}(m^*, \sigma^*) = \top \text{ and } (m^*, \sigma^*) \notin Q],$$

where the oracle Sign on input m return $\sigma \leftarrow \mathsf{Sign}(sk, m)$ and stores $Q := Q \cup \{(m, \sigma)\}$. The standard notion of existential unforgeability is achieved by relaxing the success condition to $m^* \notin Q_{|m}$, where $Q_{|m}$ is the set of messages in Q.

3.1 BLS signatures

BLS signatures [BLS04] work as follows:

Setup**:** We assume all users of the signature scheme work with agreed public parameters including descriptions of groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of known prime order p, generator g_2 of $\mathbb{G}_2$, and a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The parameters also include a hash function $H_{\mathbb{G}_1} : \{0,1\}^* \to \mathbb{G}_1$.

KGen**:** Sample $sk \leftarrow \mathbb{Z}_q$, set $vk := g_2^{sk}$, and return (vk, sk)

$\mathsf{Sign}(sk, m)$**:** Return $\sigma := H_{\mathbb{G}_1}(m)^{sk}$

$\mathsf{SigVfy}(vk, m, \sigma)$**:** If $vk \in \mathbb{G}_2$, $\sigma \in \mathbb{G}_1$ and

$$e(H_{\mathbb{G}_1}(m), vk) = e(\sigma, g_2)$$

return $\top$, else return $\perp$.

It is easy to see the BLS signature scheme is perfectly correct. We argue existential unforgeability later when we consider the threshold version of BLS signatures. However, we note the important point that BLS signatures are unique.

Uniqueness. A signature scheme is said to have unique signatures if there is at most one valid signature on a given message satisfying the verification algorithm. For all $vk, m, \sigma_1, \sigma_2$ with $\mathsf{SigVfy}(vk, m, \sigma_1) = \top$ and $\mathsf{SigVfy}(vk, m, \sigma_2) = \top$ it must be the case that $\sigma_1 = \sigma_2$.

Theorem 1. *The BLS signature scheme has unique signatures.*

Proof. If $\mathsf{SigVfy}(vk, m, \sigma) = \top$, we have $vk \in \mathbb{G}_2$, $\sigma \in \mathbb{G}_1$ and $e(H_{\mathbb{G}_1}(m), vk) = e(\sigma, g_2)$. Since g_2 is a generator for $\mathbb{G}_2$ the signature verification equation has a unique solution σ, so there cannot be two distinct signatures $\sigma_1 \neq \sigma_2$ for the same message and verification key. $\square$

It follows from the uniqueness of signatures that for BLS signatures existential unforgeability and strong existential unforgeability are equivalent.

Existential unforgeability. BLS signatures are existentially unforgeable in the random oracle model under the assumption that given g_1^a, g_2^a and g_1^b it is hard to compute g_1^{ab}. We do not prove that here, instead we will in Sect. 8 prove security of threshold BLS signatures under different assumptions.

4 Threshold signatures

A threshold signature scheme enables a threshold of t signers out of n potential contributors to collaboratively sign a message. In this article, we will work with non-interactive threshold signature schemes, where each signer by herself can produce a signature share on the message. Given t signature shares, they can then be combined to a digital signature on the message. We define the syntax of threshold signatures by describing the constituent efficient algorithms below. The threshold signature scheme takes parameters and keys as inputs. We will later describe distributed key generation protocols for generating parameters and keys, but here they are just taken for granted.

$\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) \to b$**:** Deterministic algorithm that on a key bundle with threshold t, a verification key vk, and share verification keys $shvk_1, \ldots, shvk_n$ returns $\top$ if the key bundle is considered valid, and else returns $\bot$. It can only return $\top$ if t, n are positive integers with $t \leq n$

$\mathsf{SKVfy}(sk, shvk) \to \bot$**:** Deterministic algorithm that on a share-signing key sk returns $\top$ if sk is considered a valid share-signing key with respect to share-verification key $shvk$, and else returns $\bot$.

$\mathsf{SigShare}(sk, m) \to sh$**:** Deterministic or randomized algorithm that given a share-signing key sk and a message $m \in \{0,1\}^*$ produces a signature share sh.

$\mathsf{SigShVfy}(shvk, m, sh) \to b$**:** Deterministic algorithm that given a share-verification key $shvk$, a message m and a signature share sh returns $\top$ if the signature share is to be considered valid, and else returns $\bot$.

$\mathsf{SigShCombine}(I, sh_1, \ldots, sh_t) \to \sigma$**:** Deterministic algorithm that takes a set I of distinct indices $i_1 < \ldots < i_t$ and t signature shares $sh_1, \ldots, sh_t$ and combines them to a signature σ.

$\mathsf{SigVfy}(vk, m, \sigma) \to b$**:** Deterministic algorithm that given a verification key vk, a message $m \in \{0,1\}^*$ and a signature σ returns $\top$ if the signature is to be considered valid, and else returns $\bot$.

Correctness. The threshold signature scheme, with unspecified key generation, is perfectly correct if:

– A valid key bundle has a threshold in the right range. If $\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) = \top$ then $t \in [1..n]$.
– Valid share-signing keys produce valid signature shares. For all $sk, shvk, m$ where $\mathsf{SKVfy}(sk, shvk) = \top$

$$\Pr[sh \leftarrow \mathsf{SigShare}(sk, m) : \mathsf{SigShVfy}(shvk, m, sh) = \top] = 1.$$

– Combining valid shares for a threshold of distinct indices yields a valid signature. For all $vk, shvk_1, \ldots, shvk_n, I, sh_{i_1}, \ldots, sh_{i_t}, m$ where $\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) = \top$, I is a set of t distinct indices $1 \leq i_1 < \ldots < i_t \leq n$, for all $i \in I\ :\ \mathsf{SigShVfy}(shvk_i, m, sh_i) = \top$

$$\Pr[\sigma \leftarrow \mathsf{SigShCombine}(I, sh_{i_1}, \ldots, sh_{i_t}) : \mathsf{SigVfy}(vk, m, \sigma) = \top] = 1.$$

Uniqueness. The unique signature property can be defined as for standard signatures schemes as it only depends on SigVfy.

Unforgeability. We define unforgeability in Sect. 8 for threshold signature schemes with associated distributed key generation algorithms.

4.1 BLS threshold signatures

We now describe BLS threshold signatures with unspecified key generation. We reuse the signature verification algorithm from standard BLS signatures but enable a situation where the secret key is secret shared into multiple secret share-signing keys. These share-signing keys can be used to produce signature shares, and given enough signature shares they can be combined to a signature. The following algorithms can therefore be seen as an alternative to the single signer BLS signing algorithm.

Setup**:** Public parameters include groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of known prime order p with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and generators g_1, g_2 for $\mathbb{G}_1, \mathbb{G}_2$. The parameters also include a hash function $H_{\mathbb{G}_1} : \{0,1\}^* \to \mathbb{G}_1$.

$\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n)$**:** Check $t \in [1..n]$ and $vk, shvk_1, \ldots, shvk_n \in \mathbb{G}_2$. Set $shvk_0 := vk$ and $I = \{0, \ldots, t-1\}$. For $j = t, \ldots, n$ check whether

$$shvk_j = \prod_{i \in I} shvk_i^{L_i^I(j)}.$$

Return $\top$ if all checks pass, else return $\bot$.

$\mathsf{SKVfy}(sk, shvk)$ **:** If $sk \in \mathbb{Z}_p$ and $shvk = g_2^{sk}$ return $\top$, else return $\bot$.

$\mathsf{SigShare}(sk, m)$ **:** Return

$$sh := H_{\mathbb{G}_1}(m)^{sk}.$$

$\mathsf{SigShVfy}(shvk, m, sh)$ **:** If $shvk \in \mathbb{G}_2, sh \in \mathbb{G}_1$ and

$$e(H_{\mathbb{G}_1}(m), shvk) = e(sh, g_2)$$

return $\top$, else return $\bot$.

$\mathsf{SigShCombine}(I, sh_{i_1}, \ldots, sh_{i_t})$**:** Parse I as a set of distinct indices $i_1 < \ldots < i_t$ and $sh_{i_1}, \ldots, sh_{i_t}$ as elements in $\mathbb{G}_1$. Return

$$\sigma := \prod_{i \in I} sh_i^{L_i^I(0)}.$$

$\mathsf{SigVfy}(vk, m, \sigma)$**:** Check whether $vk \in \mathbb{G}_2$, $\sigma \in \mathbb{G}_1$ and

$$e(H_{\mathbb{G}_1}(m), vk) = e(\sigma, g_2).$$

If all checks pass return $\top$, else return $\bot$.

Performance optimization. Instead of using Lagrange interpolation, it is possible to do a fast randomized check on $vk, shvk_1, \ldots, shvk_n$ by checking $\prod_{i=0}^{n} shvk_i^{e_i}$, where $(e_0, \ldots, e_n)$ is a random vector that is orthogonal to the space of discrete logarithms corresponding to valid (n,t)-secret sharings. This makes the verification non-deterministic though and requires slight modifications in the definitions since we no longer get *perfect* correctness.

Theorem 2. *The BLS threshold signature scheme is perfectly correct.*

Proof. If $t \notin [1..n]$ the check of the threshold in $\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n)$ fails and the verification returns $\perp$.

Let $sk, shvk, m$ be such that $\mathsf{SKVfy}(sk, shvk) = \top$. This means $sk \in \mathbb{Z}_p$ and $shvk = g_2^{sk}$. Since $\mathsf{SigShare}(sk, m)$ returns $sh = H_{\mathbb{G}_1}(m)^{sk}$ we now have that the share verifies, i.e.,

$$e(H_{\mathbb{G}_1}(m), shvk) = e(H_{\mathbb{G}_1}(m), g_2^{sk}) = e(H_{\mathbb{G}_1}(m)^{sk}, g_2) = e(sh, g_2).$$

Let $vk, shvk_1, \ldots, shvk_n$ be given together with a set I of distinct indices $1 \leq i_1 < \ldots < i_t \leq n$, signature shares $sh_{i_1}, \ldots, sh_{i_t}$, and a message m. If $\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) = \top$ it means that there is a degree $t-1$ polynomial $a(X) \in \mathbb{Z}_p[X]$ such that $shvk_i = g_2^{a(i)}$ for $i = 0, \ldots, n$ using $shvk_0 := vk$. The condition $\mathsf{SigShVfy}(sh_i, shvk_i) = \top$ for all $i \in I$ gives us $sh_i \in \mathbb{G}_1, shvk_i \in \mathbb{G}_2$, and $e(sh_i, g_2) = e(H_{\mathbb{G}_1}(m), g_2^{a(i)})$, which implies $sh_i = H_{\mathbb{G}_1}(m)^{a(i)}$. With t valid signature shares, the signature share combination algorithm produces $\sigma = \prod_{i \in I} sh_i^{L_i^I(0)}$. Since

$$\prod_{i \in I} sh_i^{L_i^I(0)} = \prod_{i \in I} \left(H_{\mathbb{G}_1}(m)^{a(i)}\right)^{L_i^I(0)} = H_{\mathbb{G}_1}(m)^{\sum_{i \in I} a(i) L_i^I(0)} = H_{\mathbb{G}_1}(m)^{a(0)},$$

we see $e(H_{\mathbb{G}_1}(m), vk) = e(H_{\mathbb{G}_1}(m), g_2^{a(0)}) = e(H_{\mathbb{G}_1}(m)^{a(0)}, g_2) = e(\sigma, g_2)$. This means $\mathsf{SigVfy}(vk, m, \sigma) = \top$. □

5 Public-key encryption with forward secrecy

In Sect. 7 we present our distributed key generation and distributed key resharing protocols. In these protocols, dealers will encrypt secret shares to receivers. All parties have long-term public encryption keys, which makes it easy to manage the public keys since they are static but carries with it a risk of compromise during their lifetime. We will partly mitigate that risk by using encryption with forward secrecy, which means if the adversary learns the decryption key it is possible to decrypt future ciphertext to this participant but not past ciphertexts. It is entirely plausible that an adversary may send maliciously crafted ciphertexts, so we also want the encryption scheme to be secure against chosen ciphertext attack. In this section, we will construct step-by-step a new CCA-secure multi-recipient public-key encryption scheme with forward secrecy using pairing-based cryptography.

5.1 Decisional assumption

Our encryption schemes rely on a decisional problem relative to groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p with generators $g_1, g_2, e(g_1, g_2)$, where e is a Type III pairing. We assume these groups are given in the parameters together with $f_0, f_1, \dots, f_\lambda, h \in \mathbb{G}_2$. We define the decisional problem in the experiment below and an adversary $\mathcal{A}$'s advantage against the decisional problem as $\mathbf{Adv}(\mathcal{A}) := |\Pr[\mathbf{Exp}_0(\mathcal{A}) = \top - \mathbf{Exp}_1(\mathcal{A}) = \top|$.[4]

```
$\mathbf{Exp}_b(\mathcal{A})$
$\tau_1, \dots, \tau_\lambda \xleftarrow{\$} \{0,1\}$
$x, r, s \xleftarrow{\$} \mathbb{Z}_p$
For each $j \in [1..\lambda]$ pick $\rho_j \xleftarrow{\$} \mathbb{Z}_p$
$c_0 := g_1^{xr}$ ; $c_1 \xleftarrow{\$} \mathbb{G}_1$
Return $\mathcal{A}\left( c_b, \begin{matrix} \tau_1, \dots, \tau_\lambda \ , \ g_1^x, g_1^r, g_1^s \ , \ f_0, \dots, f_\lambda, h, \left(f_0 \cdot \prod_{i=1}^{\lambda} f_i^{\tau_i}\right)^r h^s \\ \left\{ g_1^{\rho_j}, g_2^x \left(f_0 \cdot f_j^{1-\tau_j}\right)^{\rho_j}, \{f_i^{\rho_j}\}_{i \in [1..\lambda] \setminus \{j\}}, h^{\rho_j} \right\}_{j \in [1..\lambda]} \end{matrix} \right)$
```

The following theorem shows that there are no trivial attacks on the assumption assuming there is sufficient entropy in $f_0, \dots, f_\lambda, h$ from the attacker's perspective.

Theorem 3. *The decisional assumption holds in the generic group model when $f_0, \dots, f_\lambda, h$ are chosen uniformly at random, i.e., any attacker using only generic group operations and a bounded number of them has negligible advantage.*

Proof. In the generic group model, the attacker can multiply existing group elements to construct new group elements in the same group, use the pairing to map two existing source group elements into the target group, and test equality in the groups. Since one can use the pairing to lift an equality in the source groups to the target group, without loss of generality, we can assume the adversary on source group elements $A_1, \dots, A_m \in \mathbb{G}_1, B_1, \dots, B_n \in \mathbb{G}_2$ and no elements in the target group, constructs pairing-product equalities of the form

$$1 = e\left(\prod A_i^{u_{1,i}}, \prod B_j^{v_{j,1}}\right) \cdots e\left(\prod A_i^{u_{i,\ell}}, \prod B_j^{v_{j,\ell}}\right).$$

Let $a_1, \dots, a_m, b_1, \dots, b_n$ be the discrete logarithms. The equality holds if and only if

$$0 = \sum u_{i,1} a_i \cdot \sum v_{j,1} b_j + \cdots + \sum u_{i,\ell} a_i \cdot \sum v_{i,j} b_j \bmod p.$$

We can therefore reason about the adversary's ability to learn useful information based on such quadratic equations in the discrete logarithms of the available group elements.

[4] The intention is that λ specifies the height of a tree with 2^λ leaves. We will later define tree-based encryption where the sender encrypts a message to a leaf in the tree; and the receiver who holds a decryption key for a node in the tree derives a decryption key for the leaf to get the plaintext out. Our encryption schemes could be optimized by choosing a larger branching factor for the tree, e.g., choosing a 4-ary tree would reduce public parameter size to 1/2 and decryption key size to 3/4. However, the decisional assumption and the resulting encryption schemes become a bit harder to describe.

Let $\phi_0, \ldots, \phi_\lambda, \eta$ be the discrete logarithms of $f_0, \ldots, f_\lambda, h$. We will now show that when the discrete logarithms of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ are treated as formal polynomials in $\phi_0, \ldots, \phi_\lambda, \eta, x, r, s, \rho_1, \ldots, \rho_\lambda, \log(c_1)$ the adversary cannot use the generic group operations to construct an pairing-product equality that is non-trivial in c_b. This is easy to see when $c_b = c_1$, since any such bilinear equality could be rewritten in the form

$$e(c_1, *) = e(*, *) \cdots e(*, *),$$

where the unspecified entries do not depend on c_1. Since c_1 only appears on the left hand side, when we take discrete logarithms we get a formal equality of the form

$$\log c_1 \cdot * = \sum * \cdot * + \cdots + * \cdot *,$$

where the formal variable $\log c_1$ only appears on the left hand side. For the equality to be true, we must therefore have the left hand side is of the form $\log c_1 \cdot 0$ to cancel out the formal variable $\log c_1$. So the pairing-product equation the adversary uses is $e(c_1, 1) = e(*, *) \cdots e(*, *)$, with no other entries depending on c_1. That means c_1 is not used at all and therefore the equality tells the adversary nothing about it. We now set out to prove that the adversary cannot use generic group operations to construct a non-trivial bilinear equality of the form $e(c_0, *) = e(*, *) \cdot e(*, *) \cdots$ either.

When $c_b = c_0 = g_1^{xr}$ the adversary can use generic group operations in $\mathbb{G}_1$ to construct group elements with discrete logarithms of the form

$$u = u_c xr + u_1 + u_x x + u_r r + u_s s + \sum_{j \in [1..\lambda]} u_{\rho_j} \rho_j$$

for known field elements $u_c, u_1, u_x, \ldots$. Using generic group operations in $\mathbb{G}_2$ the adversary can construct group elements with discrete logarithms of the form

$$v = \begin{pmatrix} v_1 + \sum_{i=0}^{\lambda} v_{\phi_i} \phi_i + v_\eta \eta + v_{\phi,\eta,r,s} \left((\phi_0 + \sum_{i=1}^{\lambda} \tau_i \phi_i) r + \eta s \right) \\ + \sum_{j \in [1..\lambda]} \left(v_j \left(x + (\phi_0 + (1 - \tau_j) \phi_j) \rho_j \right) + \sum_{i \in [1..\lambda] \setminus \{j\}} v_{i,j} \phi_i \rho_j + v_{\eta \rho_j} \eta \rho_j \right) \end{pmatrix}.$$

To have a non-trivial equality in c_0, we can without loss of generality assume it is an equality in the target group of the form $e(c_0, *) = e(*, *) \cdots e(*, *)$, where the right hand side does not involve c_0. So after taking discrete logarithms the question is whether the adversary can find multi-variate polynomials $v \neq 0$ and $u^{(1)}, v^{(1)}, \ldots$ such that

$$xr \cdot v = \sum_{\ell} u^{(\ell)} v^{(\ell)}, \tag{1}$$

where each $u^{(\ell)}$ has $u_c = 0$. We will now analyze coefficients of various terms to show no such polynomials exist, i.e., if the equality holds formally in the indeterminates $x, r, s, \rho_1, \ldots, \rho_\lambda$ then $v = 0$.

The term $xr\eta\rho_j$ can arise on the left hand side of (1) as $xr \cdot v_{\eta\rho_j}\eta\rho_j$. Inspection of the possible products of terms in $u^{(\ell)}v^{(\ell)}$ shows that $xr\eta\rho_j$ cannot arise on the right hand side of (1) when we keep in mind that by design $u_c^{(\ell)} = 0$. This means the term cannot arise anywhere in the right hand side $\sum_{\ell=1}^n u^{(\ell)}v^{(\ell)}$ of (1). The only possible coefficient for the term $xr\eta\rho_j$ is therefore 0. We conclude that $v_{\eta\rho_j} = 0$ for all possible choices of j.

The term $xr\phi_i\rho_j$ for $i \in [1..\lambda] \setminus \{j\}$ can arise on the left hand side of (1) as $xr \cdot v_{i,j}\phi_i\rho_j$. Again inspection of products of terms in $\sum_{\ell=1}^n u^{(\ell)}v^{(\ell)}$ show that it cannot arise on the right hand side of (1). The only possible coefficient for the term $xr\phi_i\rho_j$ is therefore 0. We conclude that all $v_{i,j} = 0$.

The terms $xr\phi_0\rho_j$ tell us in a similar way that all $v_j = 0$.

The term $xr\phi_0\eta s$ tells us $v_{\phi,\eta,r,s} = 0$.

The term $xr\eta$ tells us $v_\eta = 0$.

The terms $xr\phi_i$ can arise on the left hand side of (1) and also arise on the right hand side of (1) as $u_x^{(\ell)} x \cdot v_{\phi,\eta,r,s}^{(\ell)} \left((\phi_0 + \sum_{i=1}^\lambda \tau_i\phi_i)r + \eta s\right)$. If $\sum_\ell u_x^{(\ell)} v_{\phi,\eta,r,s}^{(\ell)}\tau_i \neq 0$ though, we also have $\sum_\ell u_x^{(\ell)} v_{\phi,\eta,r,s}^{(\ell)} \neq 0$ giving a non-trivial coefficient for $x\eta s$ on the right hand side of (1). Inspection of products of terms in $u^{(\ell)}v^{(\ell)}$ show that there is no other coefficients for this term, so we end up with the right hand side having as non-trivial coefficient for $x\eta s$. And since the left hand side of (1) is $xr \cdot v$ we cannot have a term $x\eta s$. We therefore conclude that $v_{\phi_i} = 0$ for all $i = 0, \ldots, \lambda$.

With the above analysis showing many coefficients on the left hand side of (1) are zero what remains on the left hand side is $v_1 xr$. We can only hit xr on the right hand side $\sum_\ell u^{(\ell)}v^{(\ell)}$ of (1) with products of the form $r \cdot (x + (\phi_0 + k\phi_j)\rho_j)$. So there must be at least one j where we have a non-trivial sum $w = \sum_\ell u_r^{(\ell)} v_j^{(\ell)}$. However, this also gives us $wr\phi_0\rho_j$ and $w(1-\tau_j)r\phi_j\rho_j$ on the right hand side of (1). These terms do not appear on the left hand side of (1), so we must cancel them out on the right hand side. The only other way we can hit such terms is through products of the form $\rho_j \cdot \left((\phi_0 + \sum_{i=1}^\lambda \tau_i\phi_i)r + \eta s\right)$. To cancel out $wr\phi_0\rho_j$ the coefficients of the latter products must sum to $-w$. To cancel out the coefficients for the term $r\phi_j\rho_j$ we then need $w(1-\tau_j)r\phi_j\rho_j - w\rho_j\tau_j\phi_j r = 0$. This means $w = 0$ so it cannot be non-trivial after all. We conclude $v_1 = 0$ and therefore all equalities over the formal polynomials defining the discrete logarithms of the group elements in the assumption exclude the use of c_b.

The final question now is whether the adversary could be lucky and construct a quadratic equality, which does not holds for formal polynomials in the indeterminates $\phi_0, \ldots, \phi_\lambda, \eta, x, r, s, \rho_1, \ldots, \rho_\lambda$, yet happens to hold for the concrete values in the challenge. However, as noted in various Uber assumptions using the generic group model, this means the adversary has constructed a non-trivial low-degree multi-variate polynomial in the indeterminates that evaluates to zero. By the Schwarz-Zippel lemma this has negligible probability of happening. □

5.2 Binary tree encryption

We start by defining binary tree encryption for a single receiver. In BTE the parameters specify a binary tree of height λ.[5] The encryption algorithm takes a plaintext and encrypts it to a leaf in the tree. To each leaf may be associated a decryption key, and a holder of that decryption key can recover the plaintext. There are also decryption keys associated with internal nodes. A decryption key for an internal node lets you derive a decryption key for any children of that node. This means, if you have the decryption key for the root you can derive decryption keys for all leaves. But if you do not have the decryption key for the root, you can only decrypt ciphertexts pertaining to leaves in the subtrees of the decryption keys you hold.[6]

A BTE scheme has the following efficient algorithms:

Setup: The parameters include a message space $\mathcal{M}$ and height λ for a binary tree. We will write a path to a node in the tree as $\tau_1 \dots \tau_\ell$ with $\ell \leq \lambda$. The root node thus has an empty path with $\ell = 0$, while for a leaf $\ell = \lambda$.

$\mathsf{KGen} \to (pk, dk)$: Randomized key generation algorithm that produces a public key and a decryption key for the root of the tree

$\mathsf{KVfy}(pk) \to b$: Deterministic key verification algorithm that returns $\top$ if the public key is considered valid, and otherwise returns $\bot$

$\mathsf{Derive}(dk_{\tau_1 \dots \tau_{\ell-1}}, \tau_\ell) \to dk_{\tau_1,\dots,\tau_\ell}$: Randomized update algorithm that given a decryption key for the node $\tau_1 \dots \tau_{\ell-1}$ and a bit τ_ℓ returns a decryption key for the node $\tau_1 \dots \tau_\ell$. If $\ell > \lambda$ or $\tau_\ell \notin \{0, 1\}$ or something else goes wrong the derivation algorithm returns $\bot$.
We will throughout the paper use the convention that dk is a decryption key and $dk_{\tau_1,\dots,\tau_\ell}$ is a pair $(\tau_1 \dots \tau_\ell, dk)$ that explicitly indicates the node $\tau_1 \dots \tau_\ell$ the decryption key belongs to.

$\mathsf{Enc}(pk, m, \tau_1, \dots, \tau_\lambda) \to c$: Randomized encryption algorithm that given a public key, message and a leaf returns a ciphertext or $\bot$ in case of failure, e.g., if one of the inputs is malformed.

$\mathsf{Dec}(dk_{\tau_1,\dots,\tau_\lambda}, c) \to m$: Deterministic decryption algorithm that given a ciphertext and decryption key for a leaf returns a plaintext $m \in \mathcal{M}$ or $\bot$ in case of error.

Correctness. The tree-encryption scheme is perfectly correct if:

– Honestly generated keys verify as being valid.

$$\Pr[(pk, dk) \leftarrow \mathsf{KGen} : \mathsf{KVfy}(pk) = \top] = 1$$

[5] Larger branching factors are possible and give a modest performance improvement at the cost of greater complexity in defining and describing BTE.

[6] Binary tree encryption [CHK07] is closely related to hierarchical identity-based encryption. The main difference is that in HIBE identities can be arbitrary strings, while here we have a very small "identity" space. Another small difference is that in our work we always encrypt to a leaf, not to some internal node in the tree (and for this reason permit 0's to label the leaves, where e.g. [BBG05] require identities to be non-zero).

– Correctly generated ciphertexts decrypt to the original plaintext. I.e., for all $m \in \mathcal{M}, \tau_1, \dots, \tau_\lambda \in \{0,1\}$

$$\Pr\left[\begin{matrix}(pk, dk) \leftarrow \mathsf{KGen}; c \leftarrow \mathsf{Enc}(pk, m, \tau_1 \dots \tau_\lambda) \\ dk_{\tau_1} \leftarrow \mathsf{Derive}(dk); \dots, dk_{\tau_1 \dots \tau_\lambda} \leftarrow \mathsf{Derive}(dk_{\tau_1 \dots \tau_{\lambda-1}}); m' \leftarrow \mathsf{Dec}(dk_{\tau_1 \dots \tau_\lambda}, c)\end{matrix} : m = m'\right] = 1$$

Indistinguishability for random leaf under chosen plaintext attack. For a *stateful* adversary $\mathcal{A}$ we define its advantage against indistinguishability under chosen plaintext attack against a random leaf as $\mathbf{Adv}(\mathcal{A}) = |\Pr[\mathbf{Exp}_0(\mathcal{A}) = \top] - \Pr[\mathbf{Exp}_1(\mathcal{A}) = \top|$, where the experiment is

```
$\mathbf{Exp}_b(\mathcal{A})$
$(pk, dk) \leftarrow \mathsf{KGen}$
$\tau_1, \dots, \tau_\lambda \xleftarrow{\$} \{0,1\}$
For $j = 1, \dots, \lambda$
  $dk_{\tau_1 \dots \tau_{j-1} 0} \leftarrow \mathsf{Derive}(dk_{\tau_1 \dots \tau_{j-1}}, 0)$
  $dk_{\tau_1 \dots \tau_{j-1} 1} \leftarrow \mathsf{Derive}(dk_{\tau_1 \dots \tau_{j-1}}, 1)$
$(m_0, m_1) \leftarrow \mathcal{A}(pk, \tau_1 \dots \tau_\lambda, \{dk_{\tau_1 \dots \tau_{j-1}(1-\tau_j)}\}_{j \in [1..\lambda]})$
$c \leftarrow \mathsf{Enc}(pk, m_b, \tau_1 \dots \tau_\lambda)$
$b^* \leftarrow \mathcal{A}(c)$
If $b = b^*$ return $\top$, else return $\bot$
```

Construction.

Setup**:** The parameters specify pairing groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p, a message space $\mathcal{M} = [-R..S] \subset \mathbb{Z}_p$ that is small enough to be searched by brute force, group elements $f_0, f_1, \dots, f_\lambda, h \in \mathbb{G}_2$ and a tree height λ.

$\mathsf{KGen} \rightarrow (y, dk)$**:** Pick $x \xleftarrow{\$} \mathbb{Z}_p$ and compute $y := g_1^x$. Pick $\rho \xleftarrow{\$} \mathbb{Z}_p$ and let $dk := (g_1^\rho, g_2^x f_0^\rho, f_1^\rho, \dots, f_\lambda^\rho, h^\rho)$. Return (y, dk).

$\mathsf{KVfy}(pk) \rightarrow b$**:** If $pk = y \in \mathbb{G}_1$ return $\top$ else return $\bot$

$\mathsf{Derive}(dk_{\tau_1 \dots \tau_{\ell-1}}, \tau_\ell) \rightarrow dk_{\tau_1 \dots \tau_\ell}$**:** Given

$$dk_{\tau_1 \dots \tau_{\ell-1}} = (\tau_1 \dots \tau_{\ell-1}, a, b, d_\ell, \dots, d_\lambda, e) \in \{0,1\}^{\ell-1} \times \mathbb{G}_1 \times \mathbb{G}_2^{\lambda-\ell+2}$$

and $\tau_\ell \in \{0,1\}$ pick $\delta \xleftarrow{\$} \mathbb{Z}_p$ and return

$$dk_{\tau_1 \dots \tau_\ell} := (\tau_1 \dots \tau_\ell, a \cdot g_1^\delta, b \cdot d_\ell^{\tau_\ell} \cdot (f_0 \prod_{i=1}^{\ell} f_i^{\tau_i})^\delta, d_{\ell+1} \cdot f_{\ell+1}^\delta, \dots, d_\lambda \cdot f_\lambda^\delta, e \cdot h^\delta).$$

$\mathsf{Enc}(y, m, \tau_1 \dots \tau_\lambda) \rightarrow c$**:** Given $y \in \mathbb{G}_1, m \in \mathcal{M}, \tau_1 \dots \tau_\lambda \in \{0,1\}^\lambda$ pick $r, s \leftarrow \mathbb{Z}_p$ and return

$$c := \left(y^r g_1^m, g_1^r, g_1^s, \left(f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}\right)^r h^s\right).$$

$\mathsf{Dec}(dk_{\tau_1\ldots\tau_\lambda}, c) \to m$**:** Parse

$$dk_{\tau_1\ldots\tau_\lambda} = (\tau_1 \ldots \tau_\lambda, a, b, e) \in \{0,1\}^\lambda \times \mathbb{G}_1 \times \mathbb{G}_2^2$$

and $c = (C, R, S, Z) \in \mathbb{G}_1^3 \times \mathbb{G}_2$. Assert $e\left(R, f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}\right) \cdot e(S, h) = e(g_1, Z)$. Compute

$$M := e(C, g_2) \cdot e(R, b)^{-1} \cdot e(a, Z) \cdot e(S, e)^{-1}.$$

Search for $m \in \mathcal{M}$ such that $M = e(g_1, g_2)^m$. If everything succeeds return m, else return $\perp$.

Theorem 4. *The tree-encryption scheme is perfectly correct.*

Proof. The key generation algorithm produces $y := g_1^x$. So the check in $\mathsf{KVfy}(pk)$ that $pk = y \in \mathbb{G}_1$ always holds.

To see we have correct decryption, first let us observe that we derive decryption keys of a particular form. Let for a public key $y = g_1^x$ a matching decryption key

$$\begin{aligned} dk_{\tau_1\ldots\tau_{\ell-1}} =& (\tau_1 \ldots \tau_{\ell-1}, a, b, d_\ell, \ldots, d_\lambda, e) \\ =& \left(\tau_1, \ldots \tau_{\ell-1}, g_1^\rho, g_2^x (f_0 \prod_{i=1}^{\ell-1} f_i^{\tau_i})^\rho, f_\ell^\rho, \ldots, f_\lambda^\rho, h^\rho\right) \end{aligned}$$

be given. For $\ell \le \lambda$ and a bit τ_ℓ, when we derive a new decryption key $dk_{\tau_1\ldots\tau_\ell} \leftarrow \mathsf{Derive}(dk_{\tau_1\ldots\tau_{\ell-1}}, \tau_\ell)$ we do it by picking $\delta \leftarrow \mathbb{Z}_p$ and setting

$$\begin{aligned} dk_{\tau_1\ldots\tau_\ell} :=& \left(\tau_1 \ldots \tau_\ell, a \cdot g_1^\delta, b \cdot d_\ell^{\tau_\ell} \cdot (f_0 \prod_{i=1}^{\ell} f_i^{\tau_i})^\delta, d_{\ell+1} \cdot f_{\ell+1}^\delta, \ldots, d_\lambda \cdot f_\lambda^\delta, e \cdot h^\delta\right) \\ =& \left(\tau_1 \ldots \tau_\ell, g_1^{\rho+\delta}, g_2^x (f_0 \prod_{i=1}^{\ell} f_i^{\tau_i})^{\rho+\delta}, f_{\ell+1}^{\rho+\delta}, \ldots, f_\lambda^{\rho+\delta}, h^{\rho+\delta}\right) \end{aligned}$$

Please observe, the derived decryption key has the same type of format just with a longer path to the node and with randomness $\rho + \delta$. Since we after key generation start with a decryption key for the root node of the form $dk = (g_1^\rho, g_2^x f_0^\rho, f_1^\rho, \ldots, f_\lambda^\rho, h^\lambda)$, and all derivations preserve the form, we get by induction that the decryption key for a leaf derived in this manner will be of the form

$$dk_{\tau_1\ldots\tau_\lambda} = (\tau_1 \ldots \tau_\lambda, a, b, e) = \left(\tau_1 \ldots \tau_\lambda, g_1^\rho, g_2^x \left(f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}\right)^\rho, h^\rho\right) \in \{0,1\}^\lambda \times \mathbb{G}_1 \times \mathbb{G}_2^2.$$

An encryption of $m \in \mathcal{M}$ under public key y for a leaf $\tau_1 \ldots \tau_\lambda \in \{0,1\}^\lambda$ using randomness $r, s \in \mathbb{Z}_p$ gives us a ciphertext of the form

$$c = (C, R, S, Z) = \left(y^r g_1^m, g_1^r, g_1^s, \left(f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}\right)^r h^s\right) \in \mathbb{G}_1^3 \times \mathbb{G}_2.$$

The decryption algorithm first asserts $e(R, f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}) \cdot e(S, h) = e(g_1, Z)$. Using a ciphertext produced by the encryption algorithm this assertion is

$$e(g_1^r, f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}) \cdot e(g_1^s, h) = e(g_1, (f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i})^r h^s),$$

which holds by the bilinearity of the pairing.

Next, the decryption algorithm computes $M = e(C, g_2) \cdot e(R, b)^{-1} \cdot e(a, Z) \cdot e(S, e)^{-1}$. Plugging in the values in the ciphertext and decryption key we get

$$\begin{aligned} M =& e(y^r g_1^m, g_2) \cdot e(g_1^r, g_2^x (f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i})^{\rho})^{-1} \cdot e(g_1^{\rho}, (f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i})^r h^s) \cdot e(g_1^s, h^{\rho})^{-1} \\ =& e(g_1, g_2)^m \cdot e(g_1^{xr}, g_2) \cdot e(g_1^r, g_2^x)^{-1} \cdot e(g_1^r, (f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i})^{\rho})^{-1} \cdot e(g_1^{\rho}, (f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i})^r) \\ & \cdot e(g_1^{\rho}, h^s) \cdot e(g_1^s, h^{\rho})^{-1} \; = \; e(g_1, g_2)^m \end{aligned}$$

Since $m \in \mathcal{M}$ and the message space is of modest size, the decryption algorithm finds and returns m. □

Theorem 5. *The tree-encryption scheme is rleaf-IND-CPA secure under the decisional assumption.*

Proof. Suppose we have an rleaf-IND-CPA adversary $\mathcal{A}$ with advantage ϵ, then we can construct an adversary $\mathcal{A}'$ that uses $\mathcal{A}$ in a black box manner and only consumes a moderate amount of extra resources with advantage $\epsilon/2$ against the decisional assumption. We now describe $\mathcal{A}'$, which gets either $c_0 = g_1^{xr}$ or $c_1 \xleftarrow{\$} \mathbb{G}_1$ as input together with other group elements and tries to decide which is the case.

$$\mathcal{A}'\left(c_b, \begin{array}{c} \tau_1, \ldots, \tau_\lambda \;,\; g_1^x, g_1^r, g_1^s \;,\; f_0, \ldots, f_\lambda, h, \left(f_0 \cdot \prod_{i=1}^{\lambda} f_i^{\tau_i}\right)^r h^s \\ \left\{ g_1^{\rho_j}, g_2^x \left(f_0 \cdot f_j^{1-\tau_j}\right)^{\rho_j}, \{f_i^{\rho_j}\}_{i \neq 0, j}, h^{\rho_j} \right\}_{j \in [1..\lambda]} \end{array} \right)$$

```
Set $y := g_1^x$
For $j = 1, \ldots, \lambda$
    Set $dk_{\tau_1 \ldots \tau_{j-1}(1-\tau_j)} := \left(g_1^{\rho_j}, g_2^x (f_0 f_j^{1-\tau_j})^{\rho_j} \cdot \prod_{i=1}^{j-1} (f_i^{\rho_j})^{\tau_i}, f_{j+1}^{\rho_j}, \ldots, f_\lambda^{\rho_j}, h^{\rho_j}\right)$
$(m_0, m_1) \leftarrow \mathcal{A}(y, \tau_1 \ldots \tau_\lambda, \{dk_{\tau_1 \ldots \tau_{j-1}(1-\tau_j)}\}_{j \in [1..\lambda]})$
$b'' \leftarrow \{0, 1\}$
$b' \leftarrow \mathcal{A}(c_b \cdot g_1^{m_{b''}}, g_1^r, g_1^s, (f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i})^r h^s)$
If $b' = b''$ return $b''$, else return a uniformly random bit
```

It follows by inspection that given its inputs, $\mathcal{A}$ can compute the relevant group elements it uses. To analyze the success probability of $\mathcal{A}'$ let us first observe that $y = g_1^x$ for a randomly chosen x just as in the rleaf-IND-CPA experiment.

Moreover, both here and in the rleaf-IND-CPA experiment each $dk_{\tau_1\ldots\tau_{j-1}(1-\tau_j)}$ is distributed as a uniformly random decryption key for $\tau_1\ldots\tau_{j-1}(1-\tau_j)$.

If $b=0$ we have $c_b=g_1^{xr}$ and the input to the second invocation of $\mathcal{A}$ is a random encryption of $m_{b''}$. Since $\mathcal{A}$ has advantage ϵ and there is 50% chance of having $b=0$ we get advantage $\epsilon/2$.

Else if $b=1$, c_b is just a random group element. This gives $\Pr[b'=b'']=1/2$ so $\mathcal{A}$ returns a uniformly random bit. This does not add or subtract from the advantage, so taken together $\mathcal{A}$ gets advantage $\epsilon/2$. □

5.3 Multi-receiver binary tree-encryption

We now define multi-receiver BTE where the sender has multiple plaintexts to address to different receivers. We could do parallel repetition of a single-receiver tree-encryption scheme to encrypt several plaintexts to different receivers, but reuse of randomness can make the encryption scheme much more efficient, our construction later on gains almost a factor 5 compared to the trivial repetition scheme.

Multi-receiver BTE has the following efficient algorithms:

Setup: The parameters specify a message space $\mathcal{M}$ and a height λ for a tree with 2^λ leaves. We write a path to a node in the tree as $\tau_1\ldots\tau_\ell$ with $\ell\le\lambda$ and for a leaf $\ell=\lambda$

$\mathsf{KGen}\to(pk,dk)$: Randomized key generation algorithm that produces a public key and a decryption key for the root

$\mathsf{KVfy}(pk)\to b$: Deterministic key verification algorithm that returns $\top$ if the public key is considered valid, and otherwise returns $\bot$

$\mathsf{Derive}(dk_{\tau_1\ldots\tau_{\ell-1}},\tau_\ell)\to dk_{\tau_1\ldots\tau_\ell}$: Randomized update algorithm that given a decryption key for the node $\tau_1\ldots\tau_{\ell-1}$ returns a decryption key for the node $\tau_1\ldots\tau_{\ell-1}\tau_\ell$.

$\mathsf{Enc}(pk_1,m_1,\ldots,pk_n,m_n,\tau_1\ldots\tau_\lambda)\to c$: Randomized encryption algorithm that given public keys and messages addressed to their owners and a leaf returns a ciphertext (or $\bot$ in case of failure, e.g., if one of the inputs is malformed)

$\mathsf{Dec}(i,dk_{\tau_1\ldots\tau_\lambda},c)\to m$: Deterministic decryption algorithm that given a ciphertext, an index to decrypt and a decryption key for a leaf returns a plaintext $m\in\mathcal{M}$ or $\bot$ in case of error.

Correctness. The tree-encryption scheme is perfectly correct if:

– Honestly generated keys verify as being valid.

$$\Pr[(pk,dk)\leftarrow\mathsf{KGen}:\mathsf{KVfy}(pk)=\top]=1$$

– Correctly generated ciphertexts decrypt to the original plaintext. I.e., for all $m_1,\ldots,m_n\in\mathcal{M},\tau_1,\ldots,\tau_\lambda\in\{0,1\}$ and $pk_1,\ldots,pk_{i-1},pk_{i+1},\ldots,pk_n$ such that $\mathsf{KVfy}(pk_j)=\top$

$$\Pr\left[\begin{array}{c}(pk,dk)\leftarrow\mathsf{KGen};pk_i:=pk;c\leftarrow\mathsf{Enc}(pk_1,m_1,\ldots,pk_n,m_n,\tau_1\ldots\tau_\lambda)\\ dk_{\tau_1}\leftarrow\mathsf{Derive}(dk);\ldots,dk_{\tau_1\ldots\tau_\lambda}\leftarrow\mathsf{Derive}(dk_{\tau_1\ldots\tau_{\lambda-1}});m'\leftarrow\mathsf{Dec}(i,dk_{\tau_1\ldots\tau_\lambda},c)\end{array}:m_i=m'\right]=1$$

Indistinguishability for random leaf under chosen plaintext attack For a stateful adversary $\mathcal{A}$ we define its distinguishing advantage under chosen plaintext attack against a random leaf as $\mathbf{Adv}(\mathcal{A}) = |\Pr[\mathbf{Exp}_0(\mathcal{A}) = \top] - \Pr[\mathbf{Exp}_1(\mathcal{A}) = \top]|$, where the experiment is

```
$\mathbf{Exp}_b(\mathcal{A})$
$(pk, dk) \leftarrow \mathsf{KGen}$
$\tau_1, \dots, \tau_\lambda \xleftarrow{\$} \{0,1\}$
For $j = 1, \dots, \lambda$
  $dk_{\tau_1 \dots \tau_{j-1} 0} \leftarrow \mathsf{Derive}(dk_{\tau_1 \dots \tau_{j-1}}, 0)$
  $dk_{\tau_1 \dots \tau_{j-1} 1} \leftarrow \mathsf{Derive}(dk_{\tau_1 \dots \tau_{j-1}}, 1)$
$(pk_1, m_1, \dots, pk_{i-1}, m_{i-1}, m_i^{(0)}, m_i^{(1)}, pk_{i+1}, m_{i+1}, \dots, pk_n, m_m)$
    $\leftarrow \mathcal{A}(pk, \tau_1 \dots \tau_\lambda, \{dk_{\tau_1 \dots \tau_{j-1}(1-\tau_j)}\}_{j \in [1..\lambda]})$
If $(m_1, \dots, m_i^{(0)}, m_i^{(1)}, \dots, m_n) \notin \mathcal{M}^{n+1}$ or there is a malformed key $\mathsf{KVfy}(pk_k) = \bot$
or a key collision $pk_k = pk$ or $pk_k = pk_\ell$ for $k \neq \ell$ return $\bot$
$c \leftarrow \mathsf{Enc}(pk_1, m_1, \dots, pk, m_i^{(b)}, \dots, pk_n, m_n, \tau_1 \dots \tau_\lambda)$
$b' \leftarrow \mathcal{A}(c)$
If $b = b'$ return $\top$, else return $\bot$
```

Please observe that while the encryption scheme may be correct if a public key repeats, it is important for indistinguishability that the same public key is not repeated since an adversary might use correlations in a ciphertext to distinguish.

Construction. We now give a construction of a multi-receiver BTE scheme. Our construction is almost the same as for a single receiver, except we use the same randomness r, s to encrypt to multiple public keys. To prove the multi-receiver BTE is secure, we want to reduce it to the security of the single receiver BTE scheme. For the security proof to work, this means that from a single receiver ciphertext we need to simulate an extension to a multi-receiver ciphertext, and for this reduction to work it is useful to know the discrete logarithm of the public key. We therefore add to each public key y_i a proof of knowledge of the discrete logarithm.

Setup**:** The parameters specify a message space $\mathcal{M} = [-R..S] \subset \mathbb{Z}_p$ that is small enough to be searched by brute force, group elements $f_0, f_1, \dots, f_\lambda, h \in \mathbb{G}_2$ and a tree height λ.
The parameters also provide a setup for a simulation-extractable NIZK proof of knowledge of a discrete logarithm, see Sect. 6.3.

$\mathsf{KGen} \rightarrow (pk, dk)$**:** Pick $x \xleftarrow{\$} \mathbb{Z}_p$ and compute $y := g_1^x$. Generate a proof of knowledge of the discrete logarithm x as $\pi \leftarrow \mathsf{Prove}_{\mathsf{dlog}}(y; x)$ and set $pk := (y, \pi)$. Pick $\rho \xleftarrow{\$} \mathbb{Z}_p$ and let $dk := (g_1^\rho, g_2^x f_0^\rho, f_1^\rho, \dots, f_\lambda^\rho, h^\rho)$. Return (pk, dk).

$\mathsf{KVfy}(pk) \rightarrow b$**:** Parse $pk = (y, \pi)$. If $y \in \mathbb{G}_1$ return $\mathsf{PVfy}_{\mathsf{dlog}}(y, \pi)$, else return $\bot$

$\mathsf{Derive}(dk_{\tau_1 \dots \tau_{\ell-1}}, \tau_\ell) \rightarrow dk_{\tau_1 \dots \tau_\ell}$**:** Given

$$dk_{\tau_1 \dots \tau_{\ell-1}} = (\tau_1 \dots \tau_{\ell-1}, a, b, d_\ell, \dots, d_\lambda, e) \in \{0,1\}^{\ell-1} \times \mathbb{G}_1 \times \mathbb{G}_2^{\lambda-\ell+2}$$

and a bit τ_ℓ pick $\delta \xleftarrow{\$} \mathbb{Z}_p$ and return

$$dk_{\tau_1 \ldots \tau_\ell} := (\tau_1 \ldots \tau_\ell, a \cdot g_1^\delta, b \cdot d_\ell^{\tau_\ell} \cdot (f_0 \prod_{i=1}^{\ell} f_i^{\tau_i})^\delta, d_{\ell+1} \cdot f_{\ell+1}^\delta, \ldots, d_\lambda \cdot f_\lambda^\delta, e \cdot h^\delta).$$

$\mathsf{Enc}(pk_1, m_1, \ldots, pk_n, m_n, \tau_1 \ldots \tau_\lambda) \to c$**:** Given inputs $pk_i = (y_i, \pi_i)$ with $y_i \in \mathbb{G}_1$ and $m_i \in \mathcal{M}$ and $\tau_1 \ldots \tau_\lambda \in \{0,1\}^\lambda$ pick $r, s \leftarrow \mathbb{Z}_p$ and return

$$c := \left(y_1^r g_1^{m_1}, \ldots, y_n^r g_1^{m_n}, g_1^r, g_1^s, \left(f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i} \right)^r h^s \right)$$

$\mathsf{Dec}(i, dk_{\tau_1 \ldots \tau_\lambda}, c) \to m$**:** Parse

$$dk_{\tau_1 \ldots \tau_\lambda} = (\tau_1 \ldots \tau_\lambda, a, b, e) \in \{0,1\}^\lambda \times \mathbb{G}_1 \times \mathbb{G}_2^2$$

and $c = (C_1, \ldots, C_n, R, S, Z) \in \mathbb{G}_1^{n+2} \times \mathbb{G}_2$. Assert $e\left(R, f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}\right) \cdot e\left(S, h\right) = e\left(g_1, Z\right)$. Assuming $i \in [1..n]$ compute

$$M := e(C_i, g_2) \cdot e(R, b)^{-1} \cdot e(a, Z) \cdot e(S, e)^{-1}.$$

Search for $m \in \mathcal{M}$ such that $M = e(g_1, g_2)^m$. If everything succeeds return m, else return $\perp$.

Theorem 6. *The multi-receiver tree-encryption scheme is perfectly correct assuming the NIZK proof system for dlog has perfect completeness.*

Proof. The key generation algorithm produces $y := g_1^x$ and $\pi \leftarrow \mathsf{Prove}_{\mathsf{dlog}}(y; x)$. The perfect completeness of the proof system implies the check in $\mathsf{KVfy}(pk)$ that $\mathsf{PVfy}(y, \pi) = \top$ holds and we also have $y \in \mathbb{G}_1$.

To see we have correct decryption, observe that for any $i \in [1..n]$ restricting the output of the encryption algorithm to (C_i, R, S, Z) gives us a ciphertext generated as in the earlier single-receiver tree-encryption scheme. Moreover, the decryption algorithm works just as in the single-receiver tree encryption scheme. It follows from the perfect correctness of the single-receiver tree-encryption scheme that we also have perfect correctness for the multi-receiver tree-encryption scheme. □

Theorem 7. *The multi-receiver tree-encryption scheme is rleaf-IND-CPA secure assuming the single-receiver tree-encryption is rleaf-IND-CPA secure and the NIZK proof for dlog is simulation extractable.*

Sketch of proof. Let $\mathcal{A}$ be an rleaf-IND-CPA adversary against the multi-receiver TBE encryption scheme that outputs $n-1$ public keys in addition to the honest key in the rleaf-IND-CPA experiment. Then we can construct $\mathcal{A}_{\text{zk}}$ and $\mathcal{A}_{\text{se}}$ and $\mathcal{A}'$ adversaries against respectively the zero-knowledge or simulation-extractability property of the proof system or the single-receiver TBE scheme such that

$$\mathbf{Adv}(\mathcal{A}) \leq \mathbf{Adv}_{\text{zk}}(\mathcal{A}_{\text{zk}}) + \mathbf{Adv}_{\text{se}}(\mathcal{A}_{\text{se}}) + \mathbf{Adv}_{\text{single-TBE}}(\mathcal{A}').$$

The three types of adversaries use $\mathcal{A}$ in a black-box manner and only consume moderate additional resources.

Writing out the rleaf-IND-CPA experiment for a group with our concrete encryption scheme we get the experiment below. At the same time, we indicate experiments $\mathbf{Exp}_b^{\mathrm{zk}}, \mathbf{Exp}_b^{\mathrm{se}}$ implicitly defining a zero-knowledge adversary $\mathcal{A}_{\mathrm{zk}}$ and a simulation-extractability adversary $\mathcal{A}_{\mathrm{se}}$ (for simultaneous extraction from multiple proofs).

```
$\mathbf{Exp}_b(\mathcal{A})$
$x \xleftarrow{\$} \mathbb{Z}_p$
$y := g_1^x$
$\pi \leftarrow \mathsf{Prove}_{\mathsf{dlog}}(y; x)$       // In $\mathbf{Exp}_b^{\mathrm{zk}}(\mathcal{A}), \mathbf{Exp}_b^{\mathrm{se}}(\mathcal{A})$ run instead $\pi \leftarrow \mathsf{Simulate}_{\mathsf{dlog}}(y)$
$pk := (y, \pi)$
$\rho \xleftarrow{\$} \mathbb{Z}_p$
$dk := (g_1^\rho, g_2^x f_0^\rho, f_1^\rho, \ldots, f_\lambda^\rho, h^\rho)$
$\tau_1, \ldots, \tau_\lambda \xleftarrow{\$} \{0,1\}$
For $j = 1, \ldots, \lambda$
  $dk_{\tau_1 \ldots \tau_{j-1} 0} \leftarrow \mathsf{Derive}(dk_{\tau_1 \ldots \tau_{j-1}}, 0)$
  $dk_{\tau_1 \ldots \tau_{j-1} 1} \leftarrow \mathsf{Derive}(dk_{\tau_1 \ldots \tau_{j-1}}, 1)$
$(pk_1, m_1, \ldots, pk_{i-1}, m_{i-1}, m_i^{(0)}, m_i^{(1)}, pk_{i+1}, m_{i+1}, \ldots, pk_n, m_m)$
    $\leftarrow \mathcal{A}(pk, \tau_1 \ldots \tau_\lambda, \{dk_{\tau_1 \ldots \tau_{j-1}(1-\tau_j)}\}_{j \in [1..\lambda]})$
If there is $j$ where $m_j \notin \mathcal{M}$ or $\mathsf{KVfy}(pk_j) = \perp$ or $pk_j = pk$ return $\perp$
// In $\mathbf{Exp}_b^{\mathrm{se}}(\mathcal{A})$ for $j = 1, \ldots, n, j \neq i : x_j \leftarrow \mathsf{Extract}(y_j, \pi_j)$ and if $y_j \neq g_1^{x_j}$ return $\perp$
$r, s \xleftarrow{\$} \mathbb{Z}_p$
$c := \left(y_1^r g_1^{m_1}, \ldots, y_n^r g_1^{m_n}, g_1^r, g_1^s, \left(f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}\right)^r h^s\right)$
$b' \leftarrow \mathcal{A}(c)$
If $b = b'$ return $\top$, else return $\perp$
```

The difference between experiments $\mathbf{Exp}_b(\mathcal{A})$ and $\mathbf{Exp}_b^{\mathrm{zk}}(\mathcal{A})$ is whether the honest public key comes with a real proof or a simulated proof. If there is a difference, an adversary that picks the best b and runs the experiment $\mathbf{Exp}_b(\mathcal{A})/\mathbf{Exp}_b^{\mathrm{zk}}(\mathcal{A})$ can distinguish between the two cases and use it to break the zero-knowledge property of the proof system. Specifically, the experiment together with the adversary $\mathcal{A}$ become a joint zero-knowledge adversary $\mathcal{A}_{\mathrm{zk}}$ that produces the instance y together with the witness x, and then on the resulting proof π tries to distinguish whether it was simulated or not by means of the experiment and $\mathcal{A}$. Changing to the case where we extract discrete logarithms of the adversary's keys has similar negligible impact on the success probability, because otherwise we could use the joint experiment and adversary to break the simulation extractability of the proof system. Here one has to be careful though, since this fact relies on a careful analysis of many rewindings, and we will in the full version expand on this proof.

Finally, we construct an adversary for the single-receiver TBE that leverages any advantage in the $\mathbf{Exp}_b^{\mathrm{se}}(\mathcal{A})$ experiment to break the rleaf-IND-CPA for the single-receiver TBE.

```
$\mathcal{A}'\left(y, \tau_1, \ldots, \tau_\lambda, \{dk_{\tau_1 \ldots \tau_{j-1}(1-\tau_j)}\}_{j \in [1..\lambda]}\right) \to (m_0', m_1')$
$\pi \leftarrow \mathsf{Simulate}_{\mathsf{dlog}}(y)$
$pk := (y, \pi)$
$(pk_1, m_1, \ldots, m_i^{(0)}, m_i^{(1)}, \ldots, pk_n, m_n) \leftarrow \mathcal{A}(pk, \tau_1 \ldots \tau_\lambda, \{dk_{\tau_1 \ldots \tau_{j-1}(1-\tau_j)}\}_{j \in [1..\lambda]})$
Return $m_0' := m_i^{(0)}$ and $m_1' := m_i^{(1)}$
```

```
$\mathcal{A}'(C, R, S, Z) \to b$
If for any $j$ we have $\mathsf{KVfy}(pk_j) = \perp$ or $m_j = \perp$ return $\perp$
For $j \in [1..n] \setminus \{i\}$
   Parse $pk_j = (y_j, \pi_j)$
   If $j \neq i$ extract from the proof $x_j$ so that $y_j = g_1^{x_j}$ and if failing return $\perp$
   If extracted set $C_j := R^{x_j} g_1^{m_j}$
$C_i := C$
Return $\mathcal{A}(C_1, \ldots, C_n, R, S, Z)$
```

At this stage, the experiment is identical to the single-receiver experiment. So we have $\mathbf{Adv}^{\mathrm{se}}(\mathcal{A}) = \mathbf{Adv}_{\mathrm{single}}(\mathcal{A}')$. To see this again some care is needed in the analysis to see that the rewindings used in the witness extraction are meaningful across the two phases the adversary operates in during the experiment. □

5.4 Multi-receiver tree-encryption with message space $\mathbb{Z}_p$

For $m \in \mathbb{Z}_p$ it is in general infeasible to compute the discrete logarithm of $M = e(g_1, g_2)^m$ and decryption would take too long time. The natural solution is to use chunked encryption. Take m and write it as $m = \sum_{j=1}^{m} m_j B^{j-1}$ with chunks $m_j \in [0..B-1]$ where the bound B on the chunk size is small enough to make it possible to brute-force search through $[0..B-1]$.[7]

Let us formally write down the details of the multi-receiver BTE scheme for message space $\mathbb{Z}_p$ building on a multi-receiver BTE scheme for message space $\mathcal{M} = [-R..S]$.

Setup: The parameters include the setup for the basic multi-receiver tree-encryption scheme and positive integers B, m such that $[0..B-1] \subset \mathcal{M}$ and $p < B^m$.

$\mathsf{KGen}' \to (pk, dk)$: Return $(pk, dk) \leftarrow \mathsf{KGen}$

$\mathsf{KVfy}'(pk) \to b$: Return $\mathsf{KVfy}(pk)$

$\mathsf{Derive}'(dk_{\tau_1,\ldots,\tau_{\ell-1}}, \tau_\ell) \to dk'$: Return $\mathsf{Derive}(dk_{\tau_1,\ldots,\tau_{\ell-1}}, \tau_\ell)$

$\mathsf{Enc}'(pk_1, m_1, \ldots, pk_n, m_n, \tau_1, \ldots, \tau_\lambda) \to c'$: Given $m_1, \ldots, m_n \in \mathbb{Z}_p$ chunk them into pieces $m_{i,j} \in [0..B-1]$ so that $m_i = \sum_{j=1}^{m} m_{i,j} B^{j-1}$. For $j = 1, ..., m$ set $c_j \leftarrow \mathsf{Enc}(pk_1, m_{1,j}, \ldots, pk_n, m_{n,j}, \tau_1, \ldots, \tau_\lambda)$. Return $c' := (c_1, \ldots, c_m)$

$\mathsf{Dec}'(i, dk_{\tau_1,\ldots,\tau_\lambda}, c') \to m'$: Parse $c' = (c_1, \ldots, c_m)$ and for $j = 1, \ldots, m$ compute $m_{i,j} \leftarrow \mathsf{Dec}(i, dk_{\tau_1 \ldots \tau_\lambda}, c_j)$. If any $m_j = \perp$ return $\perp$, else return $m' := \sum_{j=1}^{m} m_{i,j} B^{j-1} \bmod p$

[7] Using the Baby-step Giant-step algorithm makes the search faster.

Theorem 8. *The chunked multi-receiver tree-encryption scheme has perfect correctness.*

Proof. It follows from perfect correctness of the multi-receiver tree-encryption scheme for message space $\mathcal{M}$ that the chunks $m_{i,j} \in [0..B-1] \subset \mathcal{M}$ that are retrieved when running $\mathsf{Dec}(i, dk_{\tau_1 \dots \tau_\lambda}, (c_1, \dots, c_m))$ return the chunks $m_{i,j}$ that were encrypted in c_j. Since the encryption algorithm chose the chunks so $m_i = \sum_{j=1}^{m} m_{i,j} B^{j-1}$ we see that $m' = m_i$. $\square$

Theorem 9. *The chunked multi-receiver tree-encryption scheme for $\mathbb{Z}_p$ is rleaf-IND-CPA secure.*

Proof. It follows from a hybrid argument that an adversary $\mathcal{A}$ against the chunked multi-receiver tree-encryption scheme can be used in a black-box manner to construct an adversary $\mathcal{A}'$ against the multi-receiver tree-encryption scheme with messages space $\mathcal{M} = [-R..S]$, giving us $\mathbf{Adv}_{\mathcal{M}'=\mathbb{Z}_p}(\mathcal{A}) \leq m \cdot \mathbf{Adv}_{\mathcal{M}=[-R..S]}(\mathcal{A}')$. $\square$

Let us write out the full encryption algorithm with some rearrangement of the group elements in order to observe that the encryption process and resulting ciphertext can be split in two parts. The first part depends on the public keys and the plaintexts, the second part depends on the leaf, and both parts depend on the randomness. We also write out the decryption algorithm in order to observe that the second part of the ciphertext is unique given the first part and the leaf.

$\mathsf{Enc}(pk_1, m_1, \dots, pk_n, m_n, \tau_1 \dots \tau_\lambda) \to c = (c_1, c_2)$**:** Pick $r_1, s_1, \dots, r_m, s_m \leftarrow \mathbb{Z}_p$ and compute $c_1 := \mathsf{Enc}_1(pk_1, m_1, \dots, pk_n, m_n; r_1, s_1, \dots, r_m, s_m)$ and $c_2 := \mathsf{Enc}_2(\tau_1, \dots, \tau_\lambda; r_1, s_1, \dots, r_m, s_m)$ where

- $\mathsf{Enc}_1(pk_1, m_1, \dots, pk_n, m_n; r_1, s_1, \dots, r_m, s_m)$ first parses $pk_1, \dots, pk_n$ as $pk_i = (y_i, \pi_i)$ with $y_i \in \mathbb{G}_1$ and chunks $m_1, \dots, m_n \in \mathbb{Z}_p$ as $m_i = \sum_{j=1}^{m} m_{i,j} B^{j-1}$ using chunks $m_{i,j} \in [0..B-1]$.

  It returns $c_1 := (C_{1,1}, \dots, C_{n,m}, R_1, S_1, \dots, R_m, S_m) \in \mathbb{G}_1^{n(m+2)}$, where

$$C_{i,j} := y_i^{r_j} g_1^{m_{i,j}} \qquad R_j := g_1^{r_j} \qquad S_j := g_1^{s_j}.$$

- $\mathsf{Enc}_2(\tau_1, \dots, \tau_\lambda; r_1, s_1, \dots, r_m, s_m)$ on $\tau_1, \dots, \tau_\lambda \in \{0,1\}$ returns $c_2 := (Z_1, \dots, Z_m) \in \mathbb{G}_2^m$, where

$$Z_j := \left(f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i} \right)^{r_j} h^{s_j}.$$

Return $c := (c_1, c_2)$ if everything worked. If the inputs were malformed giving $c_1 = \bot$ or $c_2 = \bot$, the encryption algorithm returns $c := \bot$.

$\mathsf{Dec}(i, dk_{\tau_1 \dots \tau_\lambda}, c) \to m$**:** Parse $c = (c_1, c_2)$ and $c_1 = (C_{1,1}, \dots, C_{n,m}, R_1, S_1, \dots, R_m, S_m) \in \mathbb{G}_1^{n(m+2)}$ and $c_2 = (Z_1, \dots, Z_m)$. Check that for all $j = 1, \dots, m$ we have

$$e(g_1, Z_j) = e(R_j, f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}) \cdot e(S_j, h).$$

Assuming $1 \leq i \leq n$ parse $dk_{\tau_1,\ldots,\tau_\lambda} = (\tau_1 \ldots \tau_\lambda, a, b, e) \in \{0,1\}^\lambda \times \mathbb{G}_1 \times \mathbb{G}_2^2$ as in the multi-receiver tree-encryption scheme for $\mathcal{M}$. For $j = 1, \ldots, m$ compute

$$M_j := e(C_{i,j}, g_2) \cdot e(R_j, b)^{-1} \cdot e(a, Z_j) \cdot e(S_j, e)^{-1}$$

and search for $m_j \in \mathcal{M}$ such that $M_j = e(g_1, g_2)^{m_j}$. If everything succeeds return $m := \sum_{j=1}^{m} m_j B^{j-1} \bmod p$, else return $\perp$.

The following theorem states that the second part of a well-formed ciphertext is unique.

Theorem 10. *For any ciphertext $c = (c_1, c_2)$ and leaf $\tau_1 \ldots \tau_\lambda$ that does not decrypt to $\perp$, there can be no other $c_2' \neq c_2$ such that $c' = (c_1, c_2')$ also decrypts non-trivially with respect to $\tau_1, \ldots, \tau_\lambda$.*

Proof. For the decryption algorithm to return a regular plaintext instead of the error $\perp$, the ciphertext pair $c = (c_1, c_2)$ must be parseable as described in the decryption algorithm to $c_1 = (C_{1,1}, \ldots, C_{n,m}, R_1, S_1, \ldots, R_m, S_m)$ and $c_2 = (Z_1, \ldots, Z_m)$. The decryption algorithm checks that for all $j = 1, \ldots, m$ we have $e(g_1, Z_j) = e(R_j, f_0 \prod_{i=1}^{\lambda} f_i^{\tau_i}) \cdot e(S_j, h)$. Since g_1 is a generator of $\mathbb{G}_1$ there is no different $Z_j' \neq Z_j$ that also satisfies this equality. Therefore the decryption algorithm will return $\perp$ on any $c_2' \neq c_2$. □

5.5 CCA-secure multi-receiver public-key encryption with forward secrecy

A multi-receiver encryption scheme with forward secrecy consists of the following efficient algorithms:

Setup**:** The parameters specify the message space $\mathcal{M}$ and a maximum number of epochs $T = 2^{\lambda_T}$

$\mathsf{KGen} \rightarrow (pk, dk_0)$**:** Randomized key generation algorithm that produces a public key and a decryption key for epoch $\tau = 0$

$\mathsf{KVfy}(pk) \rightarrow b$**:** Deterministic key verification algorithm that returns $\top$ if the public key is considered valid, and otherwise returns $\perp$

$\mathsf{KUpd}(dk_\tau) \rightarrow dk_{\tau+1}$**:** Randomized update algorithm that given a decryption key for epoch τ returns a decryption key for $\tau + 1$. If $\tau + 1 = T$ it returns $\perp$
We will throughout the paper assume dk_τ indicates the epoch τ it is intended for.

$\mathsf{Enc}(pk_1, m_1, \ldots, pk_n, m_n, \tau) \rightarrow c$**:** Randomized encryption algorithm that given n public keys and messages together with an epoch returns a ciphertext (or $\perp$ in case of failure, e.g., if one of the inputs is malformed).

$\mathsf{Dec}(i, dk_{\tau'}, c, \tau) \rightarrow m$**:** Deterministic decryption algorithm that given a decryption key dk_τ for index i and a ciphertext for epoch τ returns a plaintext $m \in \mathcal{M}$ or $\perp$ in case of error. It always returns $\perp$ in case $\tau \notin [\tau'..T-1]$.

Correctness. The encryption scheme with forward secrecy is perfectly correct if:

– Honestly generated public keys verify as being valid.

$$\Pr[(pk, dk_0) \leftarrow \mathsf{KGen} : \mathsf{KVfy}(pk) = \top] = 1$$

– Correctly generated ciphertexts decrypt to their original plaintexts. I.e., for all $m_1, \ldots, m_n \in \mathcal{M}, 0 \leq \tau' \leq \tau < T$ and $pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n$ where $\mathsf{KVfy}(pk_j) = \top$

$$\Pr\left[\begin{array}{c} (pk_i, dk_0) \leftarrow \mathsf{KGen}; c \leftarrow \mathsf{Enc}(pk_1, m_1, \ldots, pk_n, m_n, \tau) \\ dk_1 \leftarrow \mathsf{KUpd}(dk_0); \ldots, dk_{\tau'} \leftarrow \mathsf{KUpd}(dk_{\tau'-1}); m' \leftarrow \mathsf{Dec}(i, dk_{\tau'}, c, \tau) \end{array} : m_i = m'\right] = 1$$

Indistinguishability under fs-CCA attack. We define the distinguishing advantage of a stateful chosen ciphertext adversary $\mathcal{A}$ as $\mathbf{Adv}(\mathcal{A}) = |\Pr[\mathbf{Exp}_0(\mathcal{A}) = \top] - \Pr[\mathbf{Exp}_1(\mathcal{A}) = \top]|$, where

```
$\mathbf{Exp}_b(\mathcal{A})$
$(pk, dk_0) \leftarrow \mathsf{KGen}$
Store $dk_0$
$(pk_1, m_1, \ldots, pk_{i-1}, m_{i-1}, m_i^{(0)}, m_i^{(1)}, pk_{i+1}, m_{i+1}, \ldots, pk_n, m_n, \tau) \leftarrow \mathcal{A}^{\mathsf{KUpd},\mathsf{Corrupt},\mathsf{Dec}}(pk)$
$pk_i := pk$
If $(m_1, \ldots, m_i^{(0)}, m_i^{(1)}, \ldots, m_n) \notin \mathcal{M}^{n+1}$ or there is an invalid public key $\mathsf{KVfy}(pk_j) = \bot$
or there is a collision in the public keys with $pk_k = pk_\ell$ for $k \neq \ell$ or $\tau \notin [0..T-1]$ return $\bot$
$m_i := m_i^{(b)}$
$c \leftarrow \mathsf{Enc}(pk_1, m_1, \ldots, pk_n, m_n, \tau)$
$b' \leftarrow \mathcal{A}^{\mathsf{KUpd},\mathsf{Corrupt},\mathsf{Dec}}(c)$
If $\tau_{\mathsf{Corrupt}} \leq \tau$ or $(c, \tau) \in Q$ return $\bot$
If $b = b'$ return $\top$, else return $\bot$
```
,

KUpd	$\mathsf{Corrupt}$	$\mathsf{Dec}(i, c, \tau)$
Fetch the last dk_τ	Fetch the last dk_τ	Fetch the last $dk_{\tau'}$
$dk_{\tau+1} \leftarrow \mathsf{KUpd}(dk_\tau)$	$\tau_{\mathsf{Corrupt}} := \tau$	$Q := Q \cup \{(c, \tau)\}$
Store $dk_{\tau+1}$	Return dk_τ	Return $\mathsf{Dec}(i, dk_{\tau'}, c, \tau)$
Ignore further calls if $\tau + 1 = T$	Ignore further calls	

where the decryption queries may have arbitrary (i, c, τ). In particular, the decryption function may be called on an arbitrary ciphertext c that may not match the dimensions of the challenge ciphertext for n receivers.

Construction. We now present a multi-receiver fs-CCA-secure encryption scheme. It builds on the multi-receiver tree-encryption scheme for $\mathbb{Z}_p$ and we use the notation introduced in our observation that the multi-receiver BTE encryption algorithm can be split into two parts, one that is independent of the leaf and one that is independent of the plaintexts. In the security proof for the fs-CCA secure encryption scheme we rely on the random oracle model.

Setup**:** The parameters specify the message space $\mathbb{Z}_p$ and a maximum number of epoch $T = 2^{\lambda_T}$ and a hash function $H : \{0,1\}^* \to \{0,1\}^{\lambda_H}$.

The setup includes group elements $f_0, \dots, f_\lambda, h \in \mathbb{G}_2$ for the multi-receiver tree-encryption scheme defined in the previous section, where $\lambda = \lambda_T + \lambda_H$.

$\mathsf{KGen} \to (pk, dk_0)$**:** Pick $x \leftarrow \mathbb{Z}_p$ and compute $y := g_1^x$. Generate $\pi \leftarrow \mathsf{Prove}_{\mathsf{dlog}}(y, x)$ and let $pk := (y, \pi)$. Pick $\rho \leftarrow \mathbb{Z}_p$ and set $dk := (g_1^\rho, g_2^x f_0^\rho, f_1^\rho, \dots, f_\lambda^\rho, h^\rho)$. Let $dk_0 := (0, \{dk\})$ and return (pk, dk_0).

$\mathsf{KVfy}(pk) \to b$**:** Parse $pk = (y, \pi)$ and if $y \in \mathbb{G}_1$ return $\mathsf{PVfy}_{\mathsf{dlog}}(y, \pi)$, else return $\perp$

$\mathsf{KUpd}(dk_\tau, \tau') \to dk_{\tau'}$**:** Return $\perp$ if $\tau' \notin [\tau + 1 \dots T-1]$. Parse $\tau = \tau_1 \dots, \tau_{\lambda_T}$ in binary and $dk_\tau = (\tau, \{dk_{\tau_1 \dots \tau_\ell}\}_{\tau_1 \dots \tau_\ell \in \mathcal{T}_\tau})$, where $\mathcal{T}_\tau$ is a minimal set of nodes such that their subtrees cover exactly the leaves in $[\tau..T-1]$ ($\mathcal{T}_\tau$ has at most λ_T nodes). Let $\mathcal{T}_{\tau'}$ be the minimal set of nodes whose subtrees cover the leaves $[\tau'..T-1]$ and for all new $\tau_1 \dots \tau_\ell \in \mathcal{T}_{\tau'} \setminus \mathcal{T}_\tau$ derive a subkey $dk_{\tau_1 \dots \tau_\ell}$. Return

$$dk_{\tau'} = \left(\tau', \{dk_{\tau_1 \dots \tau_\ell}\}_{\tau_1 \dots \tau_\ell \in \mathcal{T}_{\tau'}}\right)$$

$\mathsf{Enc}(pk_1, m_1, \dots, pk_n, m_n, \tau) \to c$**:** Parse $\tau = \tau_1 \dots \tau_{\lambda_T}$ in binary. Pick $r_1, s_1, \dots, r_m, s_m \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$c_1 := \mathsf{Enc}_1(pk_1, m_1, \dots, pk_n, m_n; r_1, s_1, \dots, r_m, s_m).$$

Compute $\tau_{\lambda_T+1} \dots \tau_\lambda := H(pk_1, \dots, pk_n, c_1, \tau)$ and

$$c_2 := \mathsf{Enc}_2(\tau_1, \dots, \tau_\lambda; r_1, s_1, \dots, r_m, s_m).$$

If everything succeeds return $c := (c_1, c_2)$, else return $\perp$.

$\mathsf{Dec}(i, dk_{\tau'}, c, \tau) \to m$**:** Parse $c = (c_1, c_2)$ and $\tau = \tau_1 \dots \tau_{\lambda_T}$. Compute $\tau_{\lambda_T+1} \dots \tau_\lambda := H(pk_1, \dots, pk_n, c_1, \tau)$. Assuming $\tau \in [\tau'..T-1]$ derive $dk_{\tau_1 \dots \tau_\lambda}$ and return $\mathsf{Dec}(i, dk_{\tau_1 \dots \tau_\lambda}, c)$. If anything fails return $\perp$.

Theorem 11. *The multi-receiver encryption scheme has perfect correctness.*

Proof. The underlying multi-receiver BTE scheme for $\mathbb{Z}_p$ has perfect correctness, which implies perfect correctness also in the case where $\tau_{\lambda_T+1} \dots \tau_\lambda$ happens to equal $H(pk_1, \dots, pk_n, c_1, \tau)$. It follows that our purported fs-CCA secure encryption scheme has perfect correctness. □

Theorem 12. *The multi-receiver encryption scheme is fs-CCA secure.*

Proof. Suppose $\mathcal{A}$ is an fs-CCA adversary with advantage ϵ. We assume all parties treat H as a random oracle. In the random oracle model the hash function H returns a random string $\tau_{\lambda_T+1} \dots \tau_\lambda$ when queried on a fresh input. The random oracle may be programmable (benignly programmable as defined later: on a programming query it first returns a random value and then later somebody can program the input of that value) and we do not exclude $\mathcal{A}$ from being able to program the hash function as long as it cannot find collisions.

Writing out the fs-CCA experiment with our concrete encryption algorithm the adversary's advantage is $\epsilon = |\Pr[\mathbf{Exp}_0(\mathcal{A}) = \top] - \Pr[\mathbf{Exp}_1(\mathcal{A}) = \top]|$, where

```
$\mathbf{Exp}_b(\mathcal{A})$
$(pk, dk_0) \leftarrow \mathsf{KGen}$
Store $dk_0$
$(pk_1, m_1, \ldots, pk_{i-1}, m_{i-1}, m_i^{(0)}, m_i^{(1)}, pk_{i+1}, m_{i+1}, \ldots, pk_n, m_n, \tau) \leftarrow \mathcal{A}^{\mathsf{KUpd},\mathsf{Corrupt},\mathsf{Dec}}(pk)$
$pk_i := pk$
If $(m_1, \ldots, m_i^{(0)}, m_i^{(1)}, \ldots, m_n) \notin \mathcal{M}^{n+1}$ or there is an invalid public key $\mathsf{KVfy}(pk_j) = \perp$
or there is a collision in the public keys with $pk_k = pk_\ell$ for $k \neq \ell$ or $\tau \notin [0..T-1]$ return $\perp$
$m_i := m_i^{(b)}$
$r_1, s_1, \ldots, r_m, s_m \xleftarrow{\$} \mathbb{Z}_p$
$c_1 := \mathsf{Enc}_1(pk_1, m_1, \ldots, pk_n, m_n; r_1, s_1, \ldots, r_m, s_m)$
Parse $\tau = \tau_1 \ldots \tau_{\lambda_T}$
$\tau_{\lambda_T+1} \ldots \tau_\lambda := H(pk_1, \ldots, pk_n, c_1, \tau)$
$c_2 := \mathsf{Enc}_2(\tau_1, \ldots, \tau_\lambda; r_1, s_1, \ldots, r_m, s_m)$
$c := (c_1, c_2)$
$b' \leftarrow \mathcal{A}^{\mathsf{KUpd},\mathsf{Corrupt},\mathsf{Dec}}(c)$
If $\tau_{\mathsf{Corrupt}} \leq \tau$ or $(c, \tau) \in Q$ return $\perp$
If $b = b'$ return $\top$, else return $\perp$
```

KUpd	$\mathsf{Corrupt}$	$\mathsf{Dec}(i, c, \tau)$
Fetch the last dk_τ	Fetch the last $dk_{\tau'}$	Fetch the last $dk_{\tau'}$
$dk_{\tau+1} \leftarrow \mathsf{KUpd}(dk_\tau)$	$\tau_{\mathsf{Corrupt}} := \tau$	$Q := Q \cup \{(c, \tau)\}$
Store $dk_{\tau+1}$	Return dk_τ	Return $\mathsf{Dec}(i, dk_{\tau'}, c, \tau)$
Ignore further calls if $\tau + 1 = T$	Ignore further calls	

Observe that in the random oracle model, the adversary cannot tell the difference from us first picking $\tau_{\lambda_T+1} \ldots \tau_\lambda$ uniformly at random and then later pretending (in essence programming the random oracle) after c_1 is chosen that whenever $\mathcal{A}$ makes an oracle query on (c_1, τ) the hash function returns $\tau_{\lambda_T+1} \ldots \tau_\lambda$. So we see the advantage is unchanged if instead we run the following modified experiment

```
$\mathbf{Exp}'_b(\mathcal{A})$
Query the random oracle for an output $\tau_{\lambda_T+1},\dots,\tau_\lambda \xleftarrow{\$} \{0,1\}^\lambda$
$(pk, dk_0) \leftarrow \mathsf{KGen}$
Store $dk_0$
$(pk_1, m_1, \dots, pk_{i-1}, m_{i-1}, m_i^{(0)}, m_i^{(1)}, pk_{i+1}, m_{i+1}, \dots, pk_n, m_n, \tau) \leftarrow \mathcal{A}^{\mathsf{KUpd},\mathsf{Corrupt},\mathsf{Dec}}(pk)$
$pk_i := pk$
If $(m_1, \dots, m_i^{(0)}, m_i^{(1)}, \dots, m_n) \notin \mathcal{M}^{n+1}$ or there is an invalid public key $\mathsf{KVfy}(pk_j) = \bot$
or there is a collision in the public keys with $pk_k = pk_\ell$ for $k \neq \ell$ or $\tau \notin [0..T-1]$ return $\bot$
$m_i := m_i^{(b)}$
$r_1, s_1, \dots, r_m, s_m \xleftarrow{\$} \mathbb{Z}_p$
$c_1 := \mathsf{Enc}_1(pk_1, m_1, \dots, pk_n, m_n; r_1, s_1, \dots, r_m, s_m)$
Parse $\tau = \tau_1 \dots \tau_{\lambda_T}$
Program the random oracle to $H(pk_1, \dots, pk_n, c_1, \tau) := \tau_{\lambda_T+1} \dots \tau_\lambda$
$c_2 := \mathsf{Enc}_2(\tau_1, \dots, \tau_\lambda; r_1, s_1, \dots, r_m, s_m)$
$c := (c_1, c_2)$
$b' \leftarrow \mathcal{A}^{\mathsf{KUpd},\mathsf{Corrupt},\mathsf{Dec}}(c)$
If $\tau_{\mathsf{Corrupt}} \leq \tau$ or $(c, \tau) \in Q$ return $\bot$
If $b = b'$ return $\top$, else return $\bot$
```

KUpd	$\mathsf{Corrupt}$	$\mathsf{Dec}(i, c, \tau)$
Fetch the lastdk_τ	Fetch the last $dk_{\tau'}$	Fetch the last $dk_{\tau'}$
$dk_{\tau+1} \leftarrow \mathsf{KUpd}(dk_\tau)$	$\tau_{\mathsf{Corrupt}} := \tau$	$Q := Q \cup \{(c, \tau)\}$
Store $dk_{\tau+1}$	Return dk_τ	Return $\mathsf{Dec}(i, dk_{\tau'}, c, \tau)$
Ignore further calls if $\tau + 1 = T$	Ignore further calls	

The only small difference may be if $\mathcal{A}$ tries to program the random oracle and hits the same output or has accidentally hit the same input before, which is unlikely due to the entropy in the ciphertext.

Next, observe we can with probability $1/T$ guess τ in advance. So pick $\tau_1 \dots \tau_{\lambda_T} \xleftarrow{\$} \{0,1\}^\lambda$ and in case $\mathcal{A}$ outputs another epoch return a uniformly random bit b'. This means we have ϵ/T advantage.

We can now use $\mathcal{A}$ in a black-box manner in the modified experiment to construct an rleaf-IND-CPA adversary $\mathcal{A}'$ that has the same advantage and only uses moderately more resources. We expect $\mathcal{A}'$ to have at least the same access to the random oracle as $\mathcal{A}$, i.e., if $\mathcal{A}$ can program the random oracle so can $\mathcal{A}'$. The main idea in the construction is similar to the [BCHK07] CCA transformation of IBE, namely as long as the decryption key for the leaf matching the challenge ciphertext c and challenge epoch τ remains secret, we may in principle reveal the decryption keys for all other leaves and still have indistinguishability, and these other leaves suffice to answer the decryption challenges. To see this, note that if the decryption oracle is asked on a challenge $c' = (c_1, c_2'), \tau$ matching the challenge ciphertext $c = (c_1, c_2)$ in the first part and the challenge epoch τ, then because they uniquely define the second part c_2, we have $c = c'$ since c_2 is uniquely determined by c_1 and τ, and therefore the success condition $(c, \tau) \notin Q$ no longer holds.

```
$\mathcal{A}'(pk, \tau_1, \ldots, \tau_\lambda, \{dk_{\tau_1 \ldots \tau_{j-1}(1-\tau_j)}\}_{j \in [1..\lambda]})$
$(pk_1, m_1, \ldots, m_i^{(0)}, m_i^{(1)}, \ldots, pk_n, m_n, \tau') \leftarrow \mathcal{A}^{\mathsf{KUpd},\mathsf{Corrupt},\mathsf{Dec}}(pk)$
If $\tau' \neq \tau$ return $\perp$
    Where on a query $\mathsf{KUpd}, \mathcal{A}'$ keeps track of the increment $\tau \to \tau + 1$
    Where on a query $\mathsf{Corrupt}$ the tree-decryption keys suffice to construct
    the decryption key $dk_{\tau_{\mathsf{Corrupt}}}$ if $\tau_{\mathsf{Corrupt}} > \tau$
    Where on a query $\mathsf{Dec}(i, c, \tau)$ the tree-decryption key for the leaf suffices to decrypt.
Return $(m'_0, m'_1) := (m_i^{(0)}, m_i^{(1)})$
```

```
$\mathcal{A}'(c)$
Return $\mathcal{A}(c)$
```

With probability $1/T$ we guess the epoch correctly and preserve the advantage from there, so overall we have at least ϵ/T advantage in breaking our underpinning multi-receiver TBE scheme. $\square$

6 Non-interactive zero-knowledge proofs

6.1 Random oracle model

We use the Fiat-Shamir heuristic to create NIZK proofs, which means the prover will use a hash function to create challenges in the proof. Our proof systems will be secure in the random oracle model, where the hash function is modeled as a random function when proving soundness and zero knowledge. I.e., instead of being a deterministic hash function, we allow the function to return a uniformly random value in the codomain.

Let $H : \{0,1\}^* \to \mathcal{Y}$ be a hash function with codomain $\mathcal{Y}$. Looking ahead, in our NIZK proofs we will use codomains of the form $\mathcal{Y} = \mathbb{Z}_p$, $\mathcal{Y} = \{0,1\}^{\lambda_e}$ for variable sizes λ_e, and as we have already used in the signatures $\mathcal{Y} = \mathbb{G}_1$. When we need to distinguish between hash functions with different codomains, we write $H_{\mathcal{Y}}$. We will restrict programmability to guarantee y is uniformly random, which limits the simulator's capabilities and makes our results stronger with respect to zero knowledge. To capture this benignly programmable random oracle model we therefore demand that algorithms using the hash function do it through the following oracles.

$\mathcal{O}$	$\mathcal{O}_{\mathsf{prog}}$	$\mathcal{O}_{\neg Q_{\mathsf{prog}}}$

On input x :
  If $\mathcal{O}(x) = y \neq \perp$ return y
  $y \leftarrow H(x)$
  $\mathcal{O}(x) := y$
  $Q := Q \cup \{x\}$
  Return y
	On activation return $y \xleftarrow{\$} \mathcal{Y}$
	  and wait for an input
	On awaited input x :
	  If $x \in Q \cup Q_{\mathsf{prog}}$ return $\perp$
	  $Q_{\mathsf{prog}} := Q_{\mathsf{prog}} \cup \{x\}$
	  $\mathcal{O}(x) := y$; return $\top$
		On input x
		  If $x \in Q_{\mathsf{prog}}$
		  return $\perp$
		  Else call
		  $y \leftarrow \mathcal{O}(x)$
		  and return y

The convention we use is that $\mathcal{O}(x) = \bot$ whenever the oracle's value has not been defined yet. It is convenient to start with $Q = Q_{\mathsf{prog}} = \emptyset$ but note that the parameter generation our schemes use may be entangled with usage of the hash function, so this may be a false assumption. What we can hope for is that not too many queries have been made, i.e., $q = |Q| + |Q_{\mathsf{prog}}|$ is modest and that the inputs have sufficient min-entropy to be unlikely to hit one of the previous queries. As this will be the case for our concrete NIZK proofs, we will for simplicity in our analysis assume to start out with $Q = Q_{\mathsf{prog}} = \emptyset$ and $q = 0$.

We will build schemes, where oracle calls must be performed in sequence. If the output of one oracle call is y_i, we query on $H(y_i, \text{other input})$ to get y_{i+1}. Since y_i is unpredictable, this guarantees the call that created y_{i+1} came after the call that created y_i. We formalize this guarantee in the following lemma (omitting the easy Birthday paradox style proof).

Lemma 1. *Let $\mathcal{A}^{\mathcal{O},\mathcal{O}_{\mathsf{prog}}}$ be an adversary that creates up to q query-response pairs $\mathcal{O}(x_1) = y_1, \ldots, \mathcal{O}(x_q) = y_q$, whether by making direct queries or by programming the oracle. The probability that $\mathcal{A}$ uses $x_i = (y_j, *)$ for some $i < j$ is less than $\frac{q^2}{2|\mathcal{Y}|}$.*

As a consequence of Lemma 1, we can with little difference in probability distribution, assume we are working with 'chain-restricted' adversaries that do not create chained hash values out of order.

6.2 NIZK proofs

We define a non-interactive zero-knowledge proof for an efficiently decidable binary relation $\mathcal{R}$ (which may depend on system-wide parameters) through three efficient algorithms. The algorithms employ hash functions, which we in the security proofs will treat as random oracles.

$\mathsf{Prove}^{\mathcal{O}}(\mathsf{instance}, \mathsf{witness}) \to \pi$: Randomized algorithm that on an instance and a witness returns a proof π (or error symbol $\bot$)

$\mathsf{PVfy}^{\mathcal{O}}(\mathsf{instance}, \pi) \to b$: Deterministic algorithm that on an instance and a proof π returns $\top$ if the proof is to be considered valid, and otherwise returns $\bot$. If the verification algorithm gets $\bot$ as input, whether in the instance, the proof, or from the oracle, it returns $\bot$.

$\mathsf{Simulate}^{\mathcal{O},\mathcal{O}_{\mathsf{prog}}}(\mathsf{instance}) \to \pi$: Randomized algorithm that on an instance returns a simulated proof π.

We say the NIZK proof system is perfectly complete if for all $(\mathsf{instance}, \mathsf{witness}) \in \mathcal{R}$

$$\Pr[\pi \leftarrow \mathsf{Prove}^{\mathcal{O}}(\mathsf{instance}, \mathsf{witness}) : \mathsf{PVfy}^{\mathcal{O}}(\mathsf{instance}, \pi) = \top] = 1.$$

More generally, we define the advantage of a completeness adversary $\mathcal{A}$ as

$$\Pr\left[\begin{array}{c}(\mathsf{instance}, \mathsf{witness}) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{O}_{\mathsf{prog}}}; \pi \leftarrow \mathsf{Prove}^{\mathcal{O}}(\mathsf{instance}, \mathsf{witness}) : \\ (\mathsf{instance}, \mathsf{witness}) \in \mathcal{R} \text{ and } \mathsf{PVfy}^{\mathcal{O}}(\mathsf{instance}, \pi) = \bot\end{array}\right].$$

In our chunking proof later on, we restrict the set of witnesses to be in a relation $\mathcal{R}_{\text{complete}} \subseteq \mathcal{R}$.

For a zero-knowledge adversary $\mathcal{A}$, we define its advantage as

$$|2\Pr[b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}^{\mathsf{Prove}_b, \mathcal{O}, \mathcal{O}_{\mathsf{prog}}} : b = b'] - 1|,$$

where on $(\mathsf{instance}, \mathsf{witness}) \in \mathcal{R}$ the oracle acts as $\mathsf{Prove}_0(\mathsf{instance}, \mathsf{witness}) = \mathsf{Prove}^{\mathcal{O}}(\mathsf{instance}, \mathsf{witness})$ and $\mathsf{Prove}_1(\mathsf{instance}, \mathsf{witness}) = \mathsf{Simulate}^{\mathcal{O}, \mathcal{O}_{\mathsf{prog}}}(\mathsf{instance})$, while on $(\mathsf{instance}, \mathsf{witness}) \notin R$ it returns $\perp$ whether $b = 0$ or $b = 1$.

For a simulation-soundness adversary $\mathcal{A}$, we define its advantage over par as

$$\Pr[(\mathsf{instance}, \pi) \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{O}_{\mathsf{prog}}} : \mathsf{instance} \notin L_{\mathcal{R}} \text{ and } \mathsf{PVfy}^{\mathcal{O}_{\neg Q_{\mathsf{prog}}}}(\mathsf{instance}, \pi) = \top].$$

Please observe that giving the adversary access to $\mathcal{O}_{\mathsf{prog}}$ means it can run $\mathsf{Simulate}^{\mathcal{O}_{\mathsf{prog}}}$ and therefore simulate proofs on arbitrary instances. However, we only consider a proof on a fake instance $\mathsf{instance} \notin L_{\mathcal{R}}$ problematic if the proof verifies independently of the points where the random oracle was programmed, i.e., it was not simulated.

For a simulation-extractability adversary $\mathcal{A}$, we define its advantage compared to a black-box extractor $\mathsf{Extract}$ as

$$\Pr\left[\begin{array}{c}(\mathsf{instance}, \pi) \leftarrow \mathcal{A}^{\mathcal{O}, \mathcal{O}_{\mathsf{prog}}}; \mathsf{witness} \leftarrow \mathsf{Extract}^{\mathcal{A}^{\mathcal{O}, \mathcal{O}_{\mathsf{prog}}}, \mathsf{Transcript}}(\mathsf{instance}, \mathsf{witness}) : \\ (\mathsf{instance}, \mathsf{witness}) \notin \mathcal{R} \text{ and } \mathsf{PVfy}^{\mathcal{O}_{\neg Q_{\mathsf{prog}}}}(\mathsf{instance}, \pi) = \top\end{array}\right],$$

where the extractor sees the transcript of all oracle queries $\mathcal{A}$ made and is allowed to rewind $\mathcal{A}$ and try again with fresh randomness in the random oracle again seeing all queries and responses.

6.3 Proof of discrete logarithm

We use a standard Schnorr proof for knowledge of a discrete logarithm. We want to use that it is simulation extractable.

NIZK proof for knowledge of a discrete logarithm

Setup: Group $\mathbb{G}_1$ of known prime order p with generator g_1. An oracle $\mathcal{O}$ that in the implementation will be instantiated with a hash function $H_{\mathbb{Z}_p}$ but in the security proofs is modeled as a random oracle.
Instance: $y \in \mathbb{G}_1$
Statement: Knowledge of the unique discrete logarithm of y
Witness: $x \in \mathbb{Z}_p$ such that $y = g_1^x$
$\mathsf{Prove}^{\mathcal{O}}(\mathsf{instance}, \mathsf{witness})$**:**

- Pick $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $a := g_1^r$
- Compute $e := \mathcal{O}(y, a)$
- Compute $z := ex + r \bmod p$

- Let the proof be $\pi = (a, z) \in \mathbb{G}_1 \times \mathbb{Z}_p$
- Return π (and delete intermediate information created during proof)

$\mathsf{PVfy}^{\mathcal{O}}(\mathsf{instance}, \pi)$**:**

- Check that the instance and proof are correctly formatted with group elements $y, a \in \mathbb{G}_1$ and field element $z \in \mathbb{Z}_p$ as expected
- Compute $e := \mathcal{O}(y, a)$
- Return $\top$ if all checks pass and
$$y^e a = g_1^z,$$
else reject by returning $\bot$

Security

Theorem 13. *The proof system for knowledge of discrete logarithm is complete for any choice of oracle, and simulation extractable and zero knowledge in the random oracle model.*

Proof. Perfect completeness: Follows from $y^e a = (g_1^x)^e (g_1)^r = g_1^{ex+r} = g_1^z$.

Zero knowledge: The simulator ask the programmable random oracle for a challenge $e \xleftarrow{\$} \mathbb{Z}_p$. Then the simulator picks $z \xleftarrow{\$} \mathbb{Z}_p$ and sets $a := g_1^z y^{-e}$. The simulator finalizes the simulated proof by programming the random oracle to return e on input (y, a). Both in a real proof and in a simulation, z is is uniformly random as is e from the random oracle. Given these two value, the value a is determined by the verification equation, so real proofs and simulated proofs have identical probability distributions.

Simulation extractability. Follows along the lines of Groth [Gro02]. □

6.4 Proof of correct secret sharing

Looking ahead, we will be construction non-interactive distributed key generation schemes where the dealers encrypt secret sharings with our forward secure multi-receiver encryption scheme. The multi-receiver ciphertext group elements $R_j, C_{i,j}$ has uniquely defined discrete logarithms $R_j = g_1^{r_j}$ and $C_{i,j} = y_i^{r_j} g_1^{s_{i,j}}$.From such a ciphertext, anybody can compute combined group elements

$$R = \prod_{i=1}^{m} R_i^{B^{j-1}}, \quad C_1 = \prod_{i=1}^{m} C_{1,j}^{B^{j-1}}, \ldots, \quad C_n = \prod_{j=1}^{m} C_{n,j}^{B^{j-1}}.$$

These group elements uniquely define $r \in \mathbb{Z}_p$ and $s_1, \ldots, s_n \in \mathbb{Z}_p$ such that $R = g_1^r, C_i = y_i^r \cdot g_1^{s_i}$ with the relationship $r = \sum_{j=1}^{m} r_j B^{j-1}$ and $s_i = \sum_{j=1}^{m} s_{i,j} B^{j-1}$.

In a correct dealing, each $s_i = a(i) = \sum_{k=0}^{t-1} a_k i^k$. We now present an NIZK proof for the statement that this equality holds for each $i = 1, \ldots, n$. The idea is to use the hash function to compute a challenge $x \in \mathbb{Z}_p$, which we use to compress the instance to

$$C := \prod_{i=1}^{n} C_i^{x^i} = \prod_{i=1}^{n} y_i^{x^i} \cdot g_1^{\sum_{i=1}^{n} s_i x^i}.$$

If the statement is true, we have

$$\sum_{i=1}^{n} s_i x^i = \sum_{i=1}^{n} \left(\sum_{k=0}^{t-1} a_k i^k \right) x^i = \sum_{k=0}^{t-1} a_k \left(\sum_{i=1}^{n} i^k x^i \right)$$

and if the statement is false, i.e., there is any $s_i \neq a(i)$, then there are at most n possible values of x where the equality accidentally holds. After the compression step we can therefore do a standard Schnorr-style proof with a second random oracle challenge x' that the compressed $s = \sum_{i=1}^{n} s_i x^i$ is identical to the discrete logarithm of $\prod_{i=1}^{n} \left(\prod_{k=0}^{t-1} A_k^{i^k} \right)^{x^i}$.

NIZK proof for correct secret sharing

Setup: Groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of known prime order p with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and with generators $g_1, g_2, e(g_1, g_2)$. Group element $h \in \mathbb{G}_2 \setminus \{1\}$. An oracle $\mathcal{O}$ that in the implementation will be instantiated with a hash function $H_{\mathbb{Z}_p}$ but in the security proofs is modeled as a random oracle.
Instance: $y_1, \ldots, y_n \in \mathbb{G}_1$, $A_0 = g_2^{a_0}, \ldots, A_{t-1} = g_2^{a_{t-1}}$ and[8]

$$R = g_1^r \quad , \quad C_1 = y_1^r \cdot g_1^{s_1}, \quad \ldots, \quad C_n = y_n^r \cdot g_1^{s_n}$$

Statement: The discrete logarithms in the instance satisfy for $i = 1, \ldots, n$

$$s_i = \sum_{k=0}^{t-1} a_k i^k \bmod p$$

Witness: $r, s_1, \ldots, s_n \in \mathbb{Z}_p$ satisfying $s_i = a(i)$, where $a(i) = \sum_{k=0}^{t-1} a_k i^k \bmod p$.[9]
$\mathsf{Prove}^{\mathcal{O}}(\mathsf{instance}, \mathsf{witness})$**:**
- Compute $x := \mathcal{O}(\mathsf{instance})$
- Generate random $\alpha, \rho \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$F = g_1^{\rho} \quad , \quad\quad , \quad A = g_2^{\alpha} \quad , \quad Y = \left(\prod_{i=1}^{n} y_i^{x^i} \right)^{\rho} \cdot g_1^{\alpha}$$

[8] The instance just specifies the $2n + t$ group elements, the exponents are not part of the instance but uniquely defined by the group elements and indicated for later reference.

[9] The witness can be verified by using exponentiations to check R matches r, each pair y_i, C_i matches r, s_i, and each $\prod_{k=0}^{t-1} A_k^{i^k}$ matches s_i.

- Compute $x' := \mathcal{O}(x, F, A, Y)$
- Compute

$$z_r = rx' + \rho \bmod p \qquad , \qquad z_a = x' \sum_{i=1}^{n} s_i x^i + \alpha \bmod p$$

- Let the proof be $\pi = (F, A, Y, z_r, z_a) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{Z}_p^2$
- Return π (and delete intermediate information created during proof)

$\mathsf{PVfy}^{\mathcal{O}}(\mathsf{instance}, \pi)$**:**

- Check that the instance and proof are correctly formatted with group elements $y_1, \ldots, y_n, R, C_1, \ldots, C_n, F, Y \in \mathbb{G}_1$, $A_0, \ldots, A_{t-1}, A \in \mathbb{G}_2$ and field elements $z_r, z_a \in \mathbb{Z}_p$ as expected
- Compute $x := \mathcal{O}(\mathsf{instance})$ and $x' := \mathcal{O}(x, F, A, Y)$
- Verify

$$R^{x'} \cdot F = g_1^{z_r} \qquad , \qquad \left(\prod_{k=0}^{t-1} A_k^{\sum_{i=1}^{n} i^k x^i} \right)^{x'} \cdot A = g_2^{z_a}$$

and

$$\left(\prod_{i=1}^{n} C_i^{x^i} \right)^{x'} \cdot Y = \prod_{i=1}^{n} \left(y_i^{x^i} \right)^{z_r} \cdot g_1^{z_a}$$

- Return $\top$ if all checks pass, else reject by returning $\bot$

Cost of the proof system for correct secret sharing

Communication: The proof size, not counting the challenges, is 2 group elements in $\mathbb{G}_1$, 1 group element in $\mathbb{G}_2$ and 2 field elements in $\mathbb{Z}_p$

Prover computation: The prover computation is dominated an n-wide multi-exponentiation in $\mathbb{G}_1$.

Verifier computation: The verifier computation is dominated by a t-wide multi exponentiation in $\mathbb{G}_2$, two n-wide multi-exponentiation in $\mathbb{G}_1$, and computing t n-wide inner products in $\mathbb{Z}_p$ (which for large t and n can potentially be reduced using FFTs).

Security

Theorem 14. *The proof system for correct secret sharing is complete for any choice of oracle, and simulation sound*[10] *and zero knowledge in the random oracle model.*

[10] Victor points out regular soundness may suffice

Proof. Perfect completeness: On a well-formed instance, we see that the prover indeed computes a well-formatted proof with elements in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{Z}_p$ as expected. For any oracle outputs $x \in \mathbb{Z}_p$ and $x' \in \mathbb{Z}_p$ writing out the three verification equations, we see that indeed they are satisfied by an honestly constructed proof, i.e.,

$$(g_1^r)^{x'} \cdot g_1^{\rho} = g_1^{rx'+\rho} \qquad , \qquad \left(\prod_{k=0}^{t-1} (g_2^{a_k})^{\sum_{i=1}^n i^k x^i}\right)^{x'} \cdot g_2^{\alpha} = g_2^{x' \sum_{i=1}^n s_i x^i + \alpha}$$

using in the second equality that the witness has $s_i = \sum_{k=0}^{t-1} a_k i^k$, and

$$\left(\prod_{i=1}^n (y_i^r g_1^{s_i})^{x^i}\right)^{x'} \cdot \left(\prod_{i=1}^n y_i^{x^i}\right)^{\rho} \cdot g_1^{\alpha} = \left(\prod_{i=1}^n y_i^{x^i}\right)^{rx'+\rho} \cdot g_1^{x' \sum_{i=1}^n s_i x^i + \alpha}.$$

Statistical zero knowledge in the programmable random oracle model: The simulator first calls $\mathcal{O}(\mathsf{instance})$ to get a challenge $x \in \mathbb{Z}_p$. It then calls $\mathcal{O}_{\mathsf{prog}}$ to get a second challenge $x' \in \mathbb{Z}_p$. The simulator picks uniformly at random $z_r, z_a \xleftarrow{\$} \mathbb{Z}_p$ and computes the unique group elements $F, Y \in \mathbb{G}_1$, $A \in \mathbb{G}_2$ satisfying the three verification equations. It finalizes the programming of the oracle by calling $\mathcal{O}_{\mathsf{prog}}(x, F, A, Y)$, which programs the oracle to have $\mathcal{O}(x, F, A, Y) = x'$, unless (x, F, A, Y) has been used before, in which case the programming fails and returns $\perp$.

Let us now argue that the simulated proof is indistinguishable from a real proof. First observe that in both cases we get uniformly random $x \in \mathbb{Z}_p$ and $x' \in \mathbb{Z}_p$. Now define

$$\rho = z_r - x'r \bmod p \qquad , \qquad \alpha = z_a - x' \sum_{i=1}^n a(i) x^i \bmod p.$$

Since z_r and z_a are chosen uniformly random in the simulation, both real proofs and simulated proofs have uniformly random ρ and α. Now given those values, F, A and Y are uniquely defined by the three verification equations. So real proofs and simulated proofs have exactly the same distribution, the only problem being if (x, F, A, Y) has been queried before since then the simulator fails when trying to program the oracle. However, the random choice of $z_r, z_a \xleftarrow{\$} \mathbb{Z}_p$ means that F, A are uniformly and independently random, so the risk of stumbling across a previous query is at most $\frac{|Q \cup Q_{\mathsf{prog}}|}{p^2}$.

Statistical simulation soundness in the random oracle model: Consider an adversary that tries to produce an instance $y_1, \ldots, C_n$, for which there is an $s_i \neq a(i)$, and a valid proof π. If the instance or proof are malformed, the proof will be rejected, so we may without loss of generality assume the adversary returns a well-formed instance and proof $\pi = (F, A, Y, z_r, z_a) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{Z}_p^2$. If the adversary has programmed the oracle on $\mathsf{instance}$ or (x, F, Y, A) the definition does not consider it a successful forgery, it could just be a simulated proof, so let us assume x, x' stem from queries $\mathcal{O}(\mathsf{instance})$ and $\mathcal{O}(x, F, A, Y)$.

Now, the adversary may try three strategies. The first strategy is to be lucky and make a query $x' \leftarrow \mathcal{O}(x, F, A, Y)$ before the first query $x \leftarrow \mathcal{O}(\mathsf{instance})$. However, this means the adversary has to guess x in advance and by Lemma 1 the adversary's chance of finding a reverse order query pair is less than $\frac{q_{\mathcal{O}}^2}{2 \cdot p}$.

The second strategy is to hope for an x such that $\sum_{i=1}^n s_i x^i = \sum_{i=1}^n a(i) x^i$ even though there is an $s_i \neq a(i)$. Since $s_1, \ldots, s_n$ and the polynomial $a(i)$ are defined by the instance, in each query $\mathcal{O}(\mathsf{instance})$ the Schwartz-Zippel lemma bounds the chance of success to at most n/p. The adversary's chance of succeeding of getting such a pair of instance and challenge x is therefore bounded by $\frac{q_{\mathcal{O}} \cdot n}{p}$.

Finally, it may be that $\sum_{i=1}^n s_i x^i \neq \sum_{i=1}^n a(i) x^i$ but the adversary after choosing F, A, Y gets a random challenge $x' = \mathcal{O}(x, F, A, Y)$ it can answer with z_r, z_a so the proof is valid. Taking discrete logarithms of the first two verification equations we see that $z_r = x'r + \rho$ and $z_a = x' \sum_{i=1}^n a(i) x^i + \alpha$. Letting β so $Y = \prod_{i=1}^n (y_i^{x_i})^\rho \cdot g_1^\beta$ the last verification equation therefore tells us

$$\prod_{i=1}^n (y_i^r g_1^{s_i})^{x^i x'} \cdot \prod_{i=1}^n y_i^{x^i \rho} \cdot g_1^\beta = \prod_{i=1}^n y_i^{x^i (x'r + \rho)} \cdot g^{z_a}.$$

The parts with y_i cancel out, leaving us with

$$\prod_{i=1}^n g_1^{s_i x^i x'} \cdot g_1^\beta = g_1^{z_a}.$$

Taking discrete logarithms base, we get

$$x' \sum_{i=1}^n s_i x^i + \beta = x' \sum_{i=1}^n a(i) x^i + \alpha,$$

which only has a $1/p$ chance over the choice of x' to be true. With up to $q_{\mathcal{O}}$ attempts, the adversary has less than $\frac{q_{\mathcal{O}}}{p}$ probability of success.

In total, an adversary making up to $q_{\mathcal{O}}$ oracle queries has a total chance of defeating simulation soundness that is bounded by $\frac{q_{\mathcal{O}}^2}{2 \cdot p} + \frac{q_{\mathcal{O}} \cdot n}{q} + \frac{p_{\mathcal{O}}}{p} < \frac{q_{\mathcal{O}}^2}{p}$, when with little loss of generality $q_{\mathcal{O}} > n$. $\square$

6.5 Proof of correct chunking

A dealer must provide evidence that it is possible to decrypt ciphertexts in a dealing so the receivers can recover their shares of the signing key. We use the multi-receiver forward secure encryption scheme, where the ciphertexts are for the most part publicly verifiable. The only problem is that plaintexts are supposed to be chunked into small pieces and to extract the chunks the receiver needs to compute discrete logarithms. A receiver would therefore have a problem if the chunks it is supposed to extract are too large and we need an NIZK proof system that can ensure all chunks are of modest size.

We start by observing that we need not consider the full ciphertexts, only parts of them are critical for demonstrating that the encrypted chunks have correct size. Each receiver i sees a set of ElGamal ciphertexts $(R_1, C_{i,1}), \dots, (R_m, C_{i,m})$ purportedly containing m chunks of her plaintext. We would like to show that the ciphertexts are valid ElGamal encryptions to public key y_i of modest size plaintext chunks $s_{i,1}, \dots, s_{i,m}$ so the receiver can extract them and compute her full plaintext $s_i = \sum_{j=1}^{m} s_{i,j} B^{j-1} \bmod p$. Each ElGamal ciphertext $(R_j, C_{i,j})$ can be uniquely written as $(g_1^{r_j}, y_i^{r_j} g_1^{s_{i,j}})$. In our encryption schemes, we use pairings on the ciphertexts in the decryption process to compute $e(g_1, g_2)^{s_{i,j}}$. If the dealer is honest, then $s_{i,j} \in [0..B-1]$ and the receiver can do a brute force search for $s_{i,j}$.

We want to avoid a dishonest dealer using $s_{i,j}$, which cannot be brute force extracted. One strategy for doing this would be to use a range-proof demonstrating indeed $s_{i,j} \in [0..B-1]$. Exact range proofs are expensive to do over prime-order groups though. Instead we aim for a relaxed range-like proof, which will show there is a small $\Delta_{i,j}$ such that $\Delta_{i,j} s_{i,j}$ belongs to a modest size range. This suffices to show $s_{i,j}$ can be extracted, since the receiver can now do a brute force search for a suitable $\Delta_{i,j}$ that takes us inside the range. Due to the multiplicative factor $\Delta_{i,j}$ and the increase in the range, the brute force search is not as efficient as in an honest dealing, but if there is modest difference between the ranges it is still feasible.

The idea behind the proof of correct chunking is to do many small chunking proofs in parallel. Each of the sub-proofs has modest soundness but jointly they leave the prover with a negligible chance of cheating. Each sub-proof will use a challenge $e_{1,1}, \dots, e_{n,m} \leftarrow [0..E-1]$. Taking the linear combination

$$\prod_{i=1}^{n} \prod_{j=1}^{m} C_{i,j}^{e_{i,j}} = \prod_{i=1}^{n} y_i^{\sum_{j=1}^{m} e_{i,j} r_j} \cdot g_1^{\sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j} s_{i,j}}$$

we can derive the matching encoded randomness $\prod_{j=1}^{m} R_j^{e_{i,j}} = g_1^{\sum_{j=1}^{m} e_{i,j} r_j}$ for each y_i, so we have a uniquely determined $g_1^{\sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j} s_{i,j}}$. Now, we could (without worrying about zero knowledge for the moment) ask the prover to reveal $z_s = \sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j} s_{i,j}$. If the prover is honest $s_{i,j} \in [0..B-1]$ and therefore z_s is in the range $[0..S]$, where $S \geq nm(E-1)(B-1)$. Now, what about a dishonest prover providing $z_s \in [0..S]$? Well, if the prover has $\epsilon > \frac{1}{E}$ chance of doing so, then there exists for each (i,j) two sets of challenges $(e_{1,1}, \dots, e_{i,j}, \dots, e_{n,m})$ and $(e_{1,1}, \dots, e'_{i,j}, \dots, e_{n,m})$ differing only in the (i,j) entry, where the prover reveals z_s and z'_s in the range $[0..S]$. If the prover is revealing the right z_s, z'_s this means $(e_{i,j} - e'_{i,j}) s_{i,j} = \sum_{(i^*,j^*)} e_{i^*,j^*} s_{i^*,j^*} - \sum_{(i^*,j^*)}^{m} e'_{i^*,j^*} s_{i^*,j^*} = z_s - z'_s$, so (when wlog $e_{i,j} \geq e''_{i,j}$) with $\Delta_{i,j} = e_{i,j} - e'_{i,j} \in [1; E-1]$ we have that $\Delta_{i,j} s_{i,j} \in [-S..S]$. Repeating ℓ times, reduces the risk of fraud on entry (i,j) to $\epsilon \leq E^{-\ell}$.

The next question is how to force the prover in each parallel run, numbered k, to reveal the correct $z_{s,k} = \sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j,k} s_{i,j}$. To verify this in a communication-efficient manner, we use a challenge x to batch the ℓ parallel runs together to show correctness of the $z_{s,k}$'s in one go.

We also have to avoid revealing the secret values in the witness. So we make the proof zero knowledge by adding blinding factors σ_k before revealing

$$z_{s,k} = \sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j,k} s_{i,j} + \sigma_k.$$

Now, if σ_k is too large though, we end up with $z_{s,k}$ being too large, which means we can no longer guarantee it is possible to do brute force search in the exponent. On the other hand, if σ_k is too small, perhaps it does not hide the sum very well. To get around this problem we use rejection sampling [Gro05, Lyu09]. Consider choosing σ_k at random from $[-S; Z-1]$. The resulting $z_{s,k}$ belongs to the range $[-S; Z+S-1]$. We can split this range into two disjoint parts $[0..Z-1]$ and $[-S; Z+S-1]\backslash$. In the range $[0..Z-1]$ the random choice of σ_k makes each $z_{s,k}$ equally likely. Moreover, each possible sum $\sum_{i=1}^{n}\sum_{j=1}^{n} e_{i,j} s_{i,j}$ has exactly the same probability that the random σ_k lands $z_{s,k}$ inside $[0, Z-1]$. So the idea is for the prover to check each $z_{s,k}$ is within the range $[0..Z-1]$, and if not the prover restarts the entire proof with fresh randomness and tries again. Restarting does not leak information, since any sum $\sum_{i=1}^{n}\sum_{j=1}^{m} e_{i,j,k} s_{i,j}$ has equal probability of resulting in a $z_{s,k}$ outside the permitted range, and is invisible in the Fiat-Shamir heuristic since it all happens locally on the prover's side.[11] The risk of landing outside the range in run k is at most $\frac{S}{Z}$, which means over ℓ runs it is at most $\frac{\ell S}{Z}$. By choosing the parameter Z carefully, we can ensure the risk of restarting is low enough that the prover on expectation has few restarts, yet also the range $[0..Z-1]$ is small enough that $\Delta_{i,j} s_{i,j} \in [1-Z..Z-1]$ can be found by brute force when given $e(g_1, g_2)^{\Delta_{i,j} s_{i,j}}$.

NIZK proof for chunking

Setup: The parameters specify group $\mathbb{G}_1$ of prime order p with generator g_1.
The parameters include security parameter λ and positive integers n, m, ℓ, B, E, S, Z such that $E = 2^{\lceil \lambda/\ell \rceil}$, $S = nm(B-1)(E-1)$ and $2\ell S \leq Z < p 2^{-\lambda/\ell}$ ($Z = 2\ell S$, or we can choose larger Z if we want to reduce rejection risk below 1/2), and $\lambda_e = nm\ell\lceil \lambda/\ell \rceil$.
We let $\mathcal{O}$ be an oracle that in the implementation will be instantiated as a hash or extended output function H and in the security proofs will be modeled as a random oracle.

Instance: Group elements in $\mathbb{G}_1$

$$y_1, \ldots, y_n, R_1 = g_1^{r_1}, \ldots, R_m = g_1^{r_m}, C_{1,1} = y_1^{r_1} g_1^{s_{1,1}}, \ldots, C_{n,m} = y_n^{r_m} g_1^{s_{n,m}}.$$

The discrete logarithms are not part of the instance, but they are uniquely determined by the group elements and indicated for later reference.

[11] While the chance of rejection is the same for all sums, it is possible some sums would require slightly faster or slower computation, so we may use constant time algorithms to prevent timing leaks, though in practice, since we only terminate once per statement it is unlikely that enough information will leak to give an adversarial advantage.

Statement: The discrete logarithms of the instance satisfy for all $i = 1, \dots, n$ and $j = 1, \dots, m$ that there is $\Delta_{i,j} \in [1; E-1]$ such that

$$\Delta_{i,j} s_{i,j} \in [1 - Z..Z - 1]$$

Witness: Discrete logarithms $r_1, \dots, r_m, s_{1,1}, \dots, s_{n,m} \in \mathbb{Z}_p$ satisfying the constraint that all $s_{i,j} \in [0..B-1]$

$\mathsf{Prove}^{\mathcal{O}}(\mathsf{instance}, \mathsf{witness})$**:**

- Pick $y_0 \xleftarrow{\$} \mathbb{G}_1$, $\sigma_1, \dots, \sigma_\ell \xleftarrow{\$} [-S; Z-1]$, $\beta_1, \dots, \beta_\ell \xleftarrow{\$} \mathbb{Z}_p$
- Compute

$$B_1 = g_1^{\beta_1} \quad , \quad C_1 = y_0^{\beta_1} g_1^{\sigma_1}, \quad \dots, \quad B_\ell = g_1^{\beta_\ell} \quad , \quad C_\ell = y_0^{\beta_\ell} g_1^{\sigma_\ell}$$

- Query $\mathcal{O}(\mathsf{instance}, y_0, B_1, C_1, \dots, B_\ell, C_\ell; \lambda_e)$, and parse the output as $e_{1,1,1}, \dots, e_{m,n,\ell} \in [0..E-1]$
- Compute

$$z_{s,1} = \sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j,1} s_{i,j} + \sigma_1, \quad \dots, \quad z_{s,\ell} = \sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j,\ell} s_{i,j} + \sigma_\ell$$

  and check whether they belong to the range $[0..Z-1]$. If they do not, pick fresh $\sigma_1, \dots, \sigma_\ell \leftarrow [-S; Z-1]$ and try again for a maximum of λ attempts. If λ attempts fail, abort by returning $\pi = \perp$.
- Pick $\delta_0, \dots, \delta_n \xleftarrow{\$} \mathbb{Z}_p$
- Compute

$$D_0 = g_1^{\delta_0}, \quad \dots, \quad D_n = g_1^{\delta_n} \quad , \quad Y = \prod_{i=0}^{n} y_i^{\delta_i}$$

- Query $\mathcal{O}(e_{1,1,1}, \dots, e_{n,m,\ell}, z_{s,1}, \dots, z_{s,\ell}, D_0, \dots, D_n, Y)$ to get $x \in \{0,1\}^\lambda$
- Compute

$$z_{r,1} = \sum_{j=1}^{m} \sum_{k=1}^{\ell} e_{1,j,k} r_j x^k + \delta_1, \quad \dots, \quad z_{r,n} = \sum_{j=1}^{m} \sum_{k=1}^{\ell} e_{n,j,k} r_j x^k + \delta_n, \quad z_\beta = \sum_{k=1}^{\ell} \beta_k x^k + \delta_0$$

- Let the proof be $\pi = (y_0, B_1, C_1, \dots, B_\ell, C_\ell, D_0, \dots, D_n, Y, z_{s,1}, \dots, z_{s,\ell}, z_{r,1}, \dots, z_{r,n}, z_\beta)$
- Erase all intermediate information created during the proof and return π

$\mathsf{PVfy}^{\mathcal{O}}(\mathsf{instance}, \pi)$**:**

- Check that the instance belongs to $\mathbb{G}_1^{n+m+nm}$ and parse $\pi = (y_0, \dots, z_\beta) \in \mathbb{G}_1^{2\ell+n+2} \times \mathbb{Z}_p^{\ell+n+1}$
- Check $z_{s,1} \dots, z_{s,\ell} \in [0..Z-1]$
- Compute $e_{1,1,1}, \dots, e_{n,m,\ell}$ and x by querying $\mathcal{O}$ as done by the prover

- Verify

$$\prod_{j=1}^{m} R_j^{\sum_{k=1}^{\ell} e_{1,j,k}x^k} \cdot D_1 = g_1^{z_{r,1}}, \quad \ldots, \quad \prod_{j=1}^{m} R_j^{\sum_{k=1}^{\ell} e_{n,j,k}x^k} \cdot D_n = g_1^{z_{r,n}}, \quad \prod_{k=1}^{\ell} B_k^{x^k} \cdot D_0 = g_1^{z_\beta}$$

  and

$$\prod_{k=1}^{\ell} \left(\prod_{i=1}^{n} \prod_{j=1}^{m} C_{i,j}^{e_{i,j,k}} \right)^{x^k} \cdot \prod_{k=1}^{\ell} C_k^{x^k} \cdot Y = \prod_{i=1}^{n} y_i^{z_{r,i}} \cdot y_0^{z_\beta} \cdot g_1^{\sum_{k=1}^{\ell} z_{s,k}x^k}$$

- If all checks pass accept by returning $\top$, else reject by returning $\perp$

Proof size, prover computation, and verifier computation

Communication: A proof consists of $2\ell + n + 3$ group elements and $n + \ell + 1$ field elements, of which ℓ belong to a smaller range $[0..Z - 1]$.

Prover computation: The prover computes $\ell + n$ single exponentiations of g_1, ℓ exponentiations of y_i's (ignoring the smaller exponentiations to σ_k, which on expectation are repeated less than 2 times), and $n\ell$ full size field multiplications plus $nm\ell$ field multiplications with the smaller $e_{i,j,k}$ elements.

Verifier computation: The verifier cost is dominated by ℓ nm-wide multi-exponentiations to the $e_{i,j,k}$-sized exponents (equivalent to an nm-wide multi-exponentiation to full sized exponents) and n m-wide multi-exponentiations (ignoring roughly $2\ell+n$ exponentiations and $nm\ell$ field multiplications with $e_{i,j,k}$-sized elements on full sized elements).

Security

Theorem 15. *The non-interactive proof system is complete, sound and zero knowledge.*

Proof. Statistical completeness in the random oracle model: Since all $e_{i,j,k} \in [0..E-1]$ and in the witness all $s_{i,j,k} \in [0..B-1]$, we have for all $k = 1, \ldots, \ell$ that $\sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j,k}s_{i,j} \in [0..nm(E-1)(B-1)] \subseteq [0..S]$. There is much entropy in the values $y_0, B_1, \ldots, B_\ell$ that the prover chooses, so the probability of the first oracle query having been made before is at most $q_{\mathcal{O}}/p^{\ell+1}$. In each retry, the prover picks new $C_1, \ldots, C_\ell$ and the risk of a collision among the up to λ queries is bounded by $\frac{\lambda^2}{(S+Z)^\ell}$. Assuming no collision with a prior input happens, the query returns uniformly random values $e_{i,j,k}$ that are independent of the choice of $\sigma_1, \ldots, \sigma_\ell$. The prover chooses all σ_k uniformly at random from $[-S; Z-1]$, which means

$$\Pr\left[z_{s,k} := \sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j,k}s_{i,j} + \sigma_k \notin [0..Z-1] \right] = \frac{S}{Z}.$$

With ℓ such choices, the prover has less than $\frac{\ell \cdot S}{Z} \le \frac{1}{2}$ risk of restarting. After λ such runs, the probability of restarting is therefore bounded by $2^{-\lambda}$ plus the negligible risk of encountering a collision.

Now, inspecting the proof resulting from a successful run by the prover, we see that the instance and proof have the correct format with group elements in $\mathbb{G}_1$ and $\mathbb{Z}_p$. Moreover, by definition of a successful run $z_{s,1}, \ldots, z_{s,\ell} \in [0..Z-1]$. Plugging the proof elements into the verification equations we see that they are all satisfied

$$\prod_{j=1}^{m} (g_1^{r_j})^{\sum_{k=1}^{\ell} e_{1,j,k} x^k} \cdot g_1^{\delta_1} = g_1^{z_{r,1}}, \quad \ldots, \quad \prod_{j=1}^{m} (g_1^{r_j})^{\sum_{k=1}^{\ell} e_{n,j,k} x^k} \cdot g_1^{\delta_n} = g_1^{z_{r,n}}, \quad \prod_{k=1}^{\ell} (g_1^{\beta_k})^{x^k} \cdot g_1^{\delta_0} = g_1^{z_\beta}$$

and

$$\prod_{k=1}^{\ell} \left(\prod_{i=1}^{n} \prod_{j=1}^{m} (y_i^{r_j} g_1^{s_{i,j}})^{e_{i,j,k}} \right)^{x^k} \cdot \prod_{k=1}^{\ell} (y_0^{\beta_k} g_1^{\sigma_k})^{x^k} \cdot \prod_{i=0}^{n} y_i^{\delta_i} = \prod_{i=1}^{n} y_i^{z_{r,i}} \cdot y_0^{z_\beta} \cdot g_1^{\sum_{k=1}^{\ell} z_{s,k} x^k}$$

Computational zero knowledge in the random oracle model based on the DDH assumption in $\mathbb{G}_1$: The simulator picks uniformly at random $y_0, B_1, C_1, \ldots, B_\ell, C_\ell \xleftarrow{\$} \mathbb{G}_1$. Then queries $\mathcal{O}(\mathsf{instance}, y_0, B_1, C_1, \ldots, B_\ell, C_\ell; \lambda_e)$ to get challenges $e_{1,1,1}, \ldots, e_{n,m,\ell}$. Next, the simulator queries $\mathcal{O}_{\mathsf{prog}}(\lambda)$ to get a challenge x. Then it picks $z_{s,1}, \ldots, z_{s,\ell} \xleftarrow{\$} [0..Z-1]$ and $z_{r,1}, \ldots, z_{r,n}, z_\beta \xleftarrow{\$} \mathbb{Z}_p$. Now it computes the unique group elements $D_0, \ldots, D_n, Y$ that can satisfy the $n+2$ verification equations. Finally, it calls $\mathcal{O}_{\mathsf{prog}}(e_{1,1,1}, \ldots, e_{n,m,\ell}, z_{s,1}, \ldots, z_{s,\ell}, D_0, \ldots, D_n, Y)$ to finish programming the random oracle to have $\mathcal{O}(e_{1,1,1}, \ldots, e_{n,m,\ell}, z_{s,1}, \ldots, z_{s,\ell}, D_0, \ldots, D_n, Y; \lambda) = x$ (unless the programming fails).

To see the simulation is good, first observe that due to the large amount of entropy in the random elements, we have less than $\frac{q_\mathcal{O}}{p^{\ell+1}} + \frac{q_\mathcal{O}}{p^{n+1}}$ risk of hitting a previous oracle query and hitting an earlier query or failing in programming the random oracle. Assuming we avoid hitting a previous query point, the elements $e_{i,j,k}$ are uniformly random and so is x, which is also likely the case in a real proof.

Suppose now there is a distinguisher that has advantage ϵ in telling a real proof on a real witness apart from a simulated proof on the same instance. Consider the following hybrid prover, who knows the witness but acts mostly like a simulator. For each $k = 1, \ldots, \ell$ the hybrid prover selects $z_{s,k} \leftarrow [0..Z-1]$ and computes $\sigma_k = z_{s,k} - \sum_{i=1}^{n} \sum_{j=1}^{m} e_{i,j,k} s_{i,j}$, after which she proceeds as a real prover. We observe that both in real proofs and in the hybrid proofs $z_{s,k}$ ends up being uniformly random, since in the real proofs each set of $e_{i,j,k}$-challenges and $z_{s,k} \in [0..Z-1]$ has a single valid $\sigma_k \in [-S; Z-1]$ that will make it happen (and there is no sampling bias since in real proofs any collection of $e_{i,j,k} \in [0..E-1]$ and valid witness has equal risk of being rejected due to $z_{s,k}$ being out of bounds). Next, we observe that the only difference between hybrid proofs and simulated proofs is in the choice of ciphertexts (B_k, C_k) that in simulated proofs encrypt random group elements and in hybrid proofs encrypt $g_1^{\sigma_k}$. A standard hybrid argument then shows that a distinguisher between simulated proofs and hybrid proofs can be used to build an ElGamal (DDH) distinguisher with advantage ϵ/n.

Statistical simulation soundness in the random oracle model: A cheating prover creating a valid proof must provide a well-formed instance and proof with the correct number of group elements in $\mathbb{G}_1$ and field elements in $\mathbb{Z}_p$, where the $z_{s,1}, \ldots, z_{s,\ell} \in [0..Z-1]$. Moreover, since the verifier rejects if the value has been programmed, the two queries used in the fake proof must be made with queries to $\mathcal{O}$.

The prover now has three options it can try: it may use correct $z_{s,k} = \sum_{i=1}^{n}\sum_{j=1}^{m} e_{i,j,k}s_{i,j} + \sigma_k$, it may hope to make out of order queries so it learns x before the first round, or it may try using $z_{s,k} \neq \sum_{i=1}^{n}\sum_{j=1}^{m} e_{i,j,k}s_{i,j} + \sigma_k$ in a proof where queries are made in order. We know from Lemma 1 that the middle option has probability bounded by $\frac{q_{\mathcal{O}}}{2^\lambda}$, so let us now look at the first and last options.

Suppose the adversary is at the stage where it is about to query $\mathcal{O}$ for the first time on $y_0, B_1, C_1, \ldots, B_\ell, C_\ell$ to get challenges $e_{1,1,1}, \ldots, e_{n,m,\ell}$. If it has probability greater than $E^{-\ell}$ of using correct $z_{s,1}, \ldots, z_{s,k}$ of the form $z_{s,k} = \sum_{i=1}^{n}\sum_{j=1}^{m} e_{i,j,k}s_{i,j} + \sigma_k$ in a valid proof, it must also be the case that $z_{s,1}, \ldots, z_{s,\ell} \in [0..Z-1]$. Now, we know the challenges $e_{i,j,k,}$ are uniformly random, so if the probability is higher than $E^{-\ell}$, it means for each pair (i,j) there are two challenges tuples $(e_{1,1,k}, \ldots, e_{i,j,k}, \ldots, e_{n,m,k})$ and $(e_{1,1,k}, \ldots, e'_{i,j,k}, \ldots, e_{n,m,k})$ where it can succeed using correct $z_{s,k}$ values. Without loss of generality, let us assume $e_{i,j,k} > e'_{i,j,k}$ and the corresponding answers are $z_{s,k}, z'_{s,k} \in [0..Z-1]$. This means for that index (i,j) there is an index k such that $(e_{i,j,k} - e'_{i,j,k})s_{i,j} = z_{s,k} - z'_{s,k} \in [1-Z..Z-1]$.

Finally, suppose the adversary uses in order queries in the proof and at least one incorrect $z_{s,k} \neq \sum_{i=1}^{n}\sum_{j=1}^{m} e_{i,j,k}s_{i,j} + \sigma_k$. Let us analyze the probability of being able to make a valid proof when it calls $\mathcal{O}(e_{1,1,1}, \ldots, e_{n,m,\ell}, z_{s,1}, \ldots, z_{s,\ell}, D_0, \ldots, D_n, Y; \lambda)$ to get a challenge x. Taking discrete logarithms of the first $n+1$ verification equations we get

$$z_{r,1} = \sum_{k=1}^{\ell}\sum_{j=1}^{m} r_j e_{1,j,k}x^k + \delta_1 , \ldots, \; z_{r,n} = \sum_{k=1}^{\ell}\sum_{j=1}^{m} r_j e_{n,j,k}x^k + \delta_n \text{ and } z_\beta = \sum_{k=1}^{\ell} \beta_k x^k + \delta_0.$$

Looking at the exponents of $y_1, \ldots, y_n$ and y_0 in the last verification equation, we see that they are equal to $y_1^{z_{r,1}}, \ldots, y_n^{z_{r,n}}$ and $y_0^{z_\beta}$ on both sides of the equality and hence cancel out. What remains in a valid proof is the equality

$$g_1^{\sum_{k=1}^{\ell}\sum_{i=1}^{n}\sum_{j=1}^{m} e_{i,j,k}s_{i,j}x^k} \cdot g_1^{\sum_{k=1}^{\ell}\sigma_k x^k} = g_1^{\sum_{k=1}^{\ell} z_{s,k}x^k}.$$

By the Schwartz-Zippel lemma, the probability of this equality holding for a random choice of x is at most $\frac{n}{2^\lambda}$. With up to $q_{\mathcal{O}}$ queries for the adversary we upper bound the probability of this type of fraud to be at most $\frac{q_{\mathcal{O}} \cdot n}{2^\lambda}$. □

7 Non-interactive distributed key generation and key resharing with forward secrecy

A distributed key generation protocol enables a set of parties to come together to generate a public key together with shares of the secret key. The DKG protocol is run by a set of dealers, and their goal is to generate a public key and provide a set of receivers with matching secret shares of the secret key. The set of participants that act as dealers and the set of participants acting as receivers may be identical, overlapping or disjoint.

We want the DKG protocol to be non-interactive, i.e., the dealers just create dealings and do not interact further with the receivers or each other. The receivers and other parties can combine public key material provided in a set of dealings to get the public keys for the threshold signature scheme. The receivers also retrieve their secret shares of the signing key from the set of dealings. Beyond looking at a the set of dealings the receivers do not interact with other participants.

If a public key has already been generated, we may want to preserve it, but reshare the secret key. In this case, we assume the dealers already have shares of the secret key, but they want to run a distributed resharing protocol to provide a set of receivers with new shares of the secret key. We incorporate both possibilities into our system; when the dealer wants to create a fresh dealing they call the dealing algorithm with no input '-' to indicate they do not already have a share, while in redistribution they call the dealing algorithm with their secret key.

We have two reasons for being interested in resharing. The first reason is that sometimes the set of share holders may change and then we need a method for the present share holders to give a secret sharing to the incoming future set of share holders. The second reason is proactive security, where participants holding shares of a secret key periodically refresh those shares to prevent the occasional leakage of shares to accumulate over time and lead to system compromise.

Setup: The parameters specify a set of possible indices, which we for simplicity will assume is $[1..N]$ and a maximum number of epochs T.

$\mathsf{KGen} \to (pk, dk_0)$: Randomized key generation algorithm that returns a public encryption key and a private decryption key initialized for epoch $\tau = 0$.

$\mathsf{KVfy}(pk) \to b$: Deterministic key verification algorithm that returns $\top$ if the public key is to be considered valid and $\perp$ otherwise.

$\mathsf{KUpd}(dk_\tau) \to dk_{\tau+1}$: Takes as input a decryption key for epoch τ (the decryption key uniquely determines the relevant epoch) and updates it to a decryption key for epoch $\tau + 1$. In case the decryption key given as input is for $\tau = T - 1$, the update call returns $\perp$ to indicate the epochs have reached the limit.

$\mathsf{Deal}(?sk, t, pk_1, \ldots, pk_n, \tau) \to d$: Randomized dealing algorithm that given a threshold and a set of public keys with $n \leq N$ produces a dealing for a given epoch. It takes as optional input a secret key sk to be used in a resharing dealing or if omitted creates a fresh dealing.

$\mathsf{DVfy}(?shvk, t, pk_1, \ldots, pk_n, \tau, d) \rightarrow b$**:** Deterministic dealing verification algorithm that returns $\top$ if the dealing d is to be considered valid and $\perp$ otherwise. It takes as optional input a share verification key $shvk$. The intention is that $shvk$ can be used to test a resharing dealing, while it is omitted when testing a fresh dealing.
It is natural to sanity check inputs, so we assume the dealing verification algorithm can only return $\top$ on positive integers $t \leq n \leq N$ and $\tau \in [0..T-1]$. We also require the consistency property that inclusion of an optional share verification key makes dealing verification as strict or stricter than dealing verification without a share verification key.

$\mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell) \rightarrow (vk, shvk_1, \ldots, shvk_n)$**:** Deterministic algorithm that given a set I of distinct indices $i_1 < \ldots < i_\ell$ and corresponding dealings returns a public verification key vk and share-verification keys $shvk_1, \ldots, shvk_n$.

$\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) \rightarrow b$**:** Deterministic algorithm that given a threshold t and a set of verification keys returns $\top$ if the keys are to be considered valid, and otherwise returns $\perp$. The algorithm can only return $\top$ on positive integers $t \leq n \leq N$.

$\mathsf{SKRetrieve}(j, dk_{\tau'}, I, d_1, \ldots, d_\ell, \tau) \rightarrow sk$**:** Deterministic algorithm that given a decryption key, an index set I of size ℓ, and matching dealings $d_1, \ldots, d_\ell$ for an epoch τ returns a secret share-signing key sk for a given index j.

$\mathsf{SKVfy}(sk, shvk) \rightarrow b$**:** Deterministic secret key verification algorithm that given a secret share-signing key returns $\top$ if it is to be considered valid with respect to a share-verification key $shvk$, and otherwise $\perp$.

Correctness. The protocol is correct if

– Key generation produces valid public keys

$$\Pr[(pk, dk_0) \leftarrow \mathsf{KGen} : \mathsf{KVfy}(pk) = \top] = 1$$

– Dealings made over valid public keys are valid. More precisely, if $1 \leq t \leq n \leq N$ and $\tau \in [0..T-1]$ and $pk_1, \ldots, pk_n$ are valid public keys so that $\mathsf{KVfy}(pk_i) = \top$ and $\mathsf{SKVfy}(sk, shvk) = \top$ or alternatively $(sk, shvk) = (-, -)$ then

$$\Pr[d \leftarrow \mathsf{Deal}(?sk, t, pk_1, \ldots, pk_n, \tau) : \mathsf{DVfy}(?shvk, t, pk_1, \ldots, pk_n, \tau, d) = \top] \approx 1.$$

– Dealing verification is stricter when an optional share verification key is provided. If $\mathsf{DVfy}(shvk, t, pk_1, \ldots, pk_n, d) = \top$ then $\mathsf{DVfy}(-, t, pk_1, \ldots, pk_n, \tau, d) = \top$.

– Valid dealings result in valid verification keys. If $t, pk_1, \ldots, pk_n, \tau$ and dealings $d_1, \ldots, d_\ell$ all satisfy $\mathsf{DVfy}(-, t, pk_1, \ldots, pk_n, \tau, d_k) = \top$ and index set $I \subseteq [1..N]$ has size ℓ, then

$$\Pr\left[\begin{array}{c}(vk, shvk_1, \ldots, shvk_n) \leftarrow \mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell) : \\ \mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_{n'}) = \top\end{array}\right] = 1.$$

– An honest receiver should be able to retrieve a valid secret key for herself from a set of valid dealings. We define a retrieval adversary $\mathcal{A}$'s advantage to be

$$\Pr\left[\begin{array}{c}(pk, dk_0) \leftarrow \mathsf{KGen}; (j, I, pk_1, \ldots, pk_n, d_1, \ldots, d_\ell, \tau) \leftarrow \mathcal{A}^{\mathsf{KUpd}}(pk, dk_0)\\ (vk, shvk_1, \ldots, shvk_n) \leftarrow \mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell)\\ sk \leftarrow \mathsf{SKRetrieve}(j, dk_{\tau'}, I, d_1, \ldots, d_\ell, \tau) :\\ I \subset [1..n] \text{ and } |I| = \ell \text{ and } pk_j = pk \text{ and all dealings are valid, i.e.,}\\ \mathsf{DVfy}(-, t, pk_1, \ldots, pk_n, \tau, d_i) = \top \text{ and } \tau' \leq \tau, \text{ yet } \mathsf{SKVfy}(sk, shvk_j) \neq \top\end{array}\right],$$

where on each call to KUpd the oracle sets $dk_{\tau'+1} := \mathsf{KUpd}(dk_{\tau'})$ and $\tau' := \tau'+1$, and stops reacting to further calls once $\tau'+1 = T$. The oracle responds to the call by sending the decryption key to the adversary.[12]

Verification-key preservation. The protocol preserves the verification key if after a resharing we still have the same verification key. Formally, for positive integers $t \leq n \leq N, t' \leq n' \leq N$, verification keys $vk, shvk_1, \ldots, shvk_n$ with $\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) = \top$, index set I containing $i_1 < \ldots, i_t \leq n$, epoch $\tau \in [0..T-1]$, public keys $pk_1, \ldots, pk_{n'}$ and valid dealings $d_1, \ldots, d_t$ with $\mathsf{DVfy}(shvk_{i_k}, t', pk_1, \ldots, pk_{n'}, \tau, d_k) = \top$, we have

$$\Pr[(vk', shvk'_1, \ldots, shvk'_{n'}) \leftarrow \mathsf{VKCombine}(t', n', I, d_1, \ldots, d_t) : vk' = vk] = 1.$$

7.1 Construction.

We now present a non-interactive distributed key generation and key redistribution protocol with forward secrecy. It builds on an implicit fs-CCA-secure encryption scheme, from which we get the key generation, key verification and key update algorithms $\mathsf{KGen}, \mathsf{KVfy}$ and KUpd. It also is intended for use with BLS threshold signatures, which use the $\mathsf{VKVfy}, \mathsf{SKVfy}$ algorithms to verify the generated keys. We present the full protocol in a self-contained manner here including those algorithms.

Setup: Includes groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of known prime order p with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and generators $g_1, g_2, e(g_1, g_2)$. The BLS threshold signature scheme uses a hash function $H_{\mathbb{G}_1} : \{0,1\}^* \to \mathbb{G}_1$.
The parameters define a maximal number of epochs $T = 2^{\lambda_T}$ and a bound $N < p$ defining the set of indices $[1..N]$ the protocol may assign to participants, which means N is the maximal set of receivers we can have in a single dealing.
The setup also specifies group elements $f_0, \ldots, f_\lambda, h \in \mathbb{G}_2$, which are used in our CCA-secure encryption scheme with forward secrecy. Implicitly the group elements define a function $f : \mathbb{Z}_p^\lambda \to \mathbb{G}_2$ given by $f(\tau_1, \ldots, \tau_\lambda) :=$

[12] We assume all dealings operate with respect to the same target epoch. This can be relaxed, it is conceivable the dealings are for different epochs, but the definitions have to be tweaked a bit in that case.

$f_0 \cdot \prod_{i=1}^{\lambda} f_i^{\tau_i}$ that we will make frequent use of.[13] As part of the parameters for the encryption scheme there is a chunk size $B \geq 2$. We let $m := \lceil \log_B(p) \rceil$, which implies $B^m \geq p$. The encryption scheme makes use of a hash function $H_{\lambda_H} : \{0,1\}^* \to \{0,1\}^{\lambda_H}$ and the parameter λ should satisfy $\lambda = \lambda_T + \lambda_H$. The encryption scheme uses the simulation-extractable NIZK proof of knowledge of a discrete logarithm of an element in $\mathbb{G}_1$ from Section 6.
The construction makes use of the simulation-sound NIZK proofs for correct secret sharing and correct chunking that we presented in Section 6. The NIZK proofs rely on a hash function $H_{\mathbb{Z}_p} : \{0,1\}^* \to \mathbb{Z}_p$. The NIZK proof for chunking includes functions to compute additional parameters ℓ, E, S, Z, λ_e. The NIZK proof for chunking also uses a family of hash functions $H_{\lambda_e} : \{0,1\}^* \to \{0,1\}^{\lambda_e}$, where the length of the output λ_e may be chosen by the user.

$\mathsf{KGen} \to (pk, dk_0)$: Pick $x \xleftarrow{\$} \mathbb{Z}_p$ and set

$$y := g_1^x.$$

Construct a proof of knowledge of the discrete logarithm $\pi_{\mathsf{dlog}} \leftarrow \mathsf{Prove}_{\mathsf{dlog}}(y; x)$. Set $pk := (y, \pi_{\mathsf{dlog}})$.
Pick $\rho \xleftarrow{\$} \mathbb{Z}_p$. Set

$$dk := (g_1^\rho, g_2^x f_0^\rho, f_1^\rho, \ldots, f_\lambda^\rho, h^\rho) \in \mathbb{G}_1 \times \mathbb{G}_2^{\lambda+2}$$

and $dk_0 := (0, dk)$.
Erase intermediate information and return (pk, dk_0).

$\mathsf{KVfy}(pk) \to b$: Parse $pk = (y, \pi_{\mathsf{dlog}})$ and if $y \in \mathbb{G}_1$ return $\mathsf{PVfy}_{\mathsf{dlog}}(y, \pi_{\mathsf{dlog}}) = \top$, else return $\bot$.

$\mathsf{KUpd}(dk_\tau, k) \to dk_{\tau+k}$: Before describing the update procedure, let us give the high level structure of a decryption key dk_τ.
The forward secure encryption scheme builds on a tree encryption scheme, for a binary tree of size 2^λ. Messages are encrypted to leaves of the tree and it should be the case that a decryption key for an internal node allows you to derive decryption keys for all nodes in the subtree below that node.
The structure of a decryption key for a public key with $y = g_1^x$ and a node $\tau_1 \ldots \tau_\ell$ at height $\ell \leq \lambda$ in the binary tree is

$$\begin{aligned} dk_{\tau_1\ldots\tau_\ell} &= (\tau_1 \ldots \tau_\ell, a, b, d_{\ell+1}, \ldots, d_\lambda, e) \\ &= \left(\tau_1 \ldots \tau_\ell, g_1^\rho, g_2^x \left(f_0 \prod_{i=1}^{\ell} f_i^{\tau_i}\right)^\rho, f_{\ell+1}^\rho, \ldots, f_\lambda^\rho, h^\rho\right) \in \{0,1\}^\ell \times \mathbb{G}_1 \times \mathbb{G}_2^{\lambda-\ell+2}. \end{aligned}$$

If the decryption key has been generated with the algorithms from the protocol, ρ will have been chosen uniformly at random from $\mathbb{Z}_p$.[14]

[13] The group elements $f_0, \ldots, f_\lambda, h$ may be generated with a hash function $H_{\text{setup}} : \{0,1\}^* \to \mathbb{G}_2^{\lambda+2}$ to convince third parties that we have nothing up our sleeve. In the random oracle model, this gives us a set of random group elements (up to grinding).

[14] In the key generation algorithm, dk is a decryption key for the root of the binary tree of this form.

From a decryption key $dk_{\tau_1\ldots\tau_\ell}$ it is possible to derive a perfectly randomized decryption key $dk_{\tau_1\ldots\tau_{\ell+\ell'}}$ for any node in the subtree by picking $\delta \xleftarrow{\$} \mathbb{Z}_p$ and setting

$$dk_{\tau_1\ldots\tau_{\ell+\ell'}} := \left(\tau_1\ldots\tau_{\ell+\ell'}, a\cdot g_1^\delta, b\cdot\prod_{i=\ell+1}^{\ell+\ell'} d_i^{\tau_i}\cdot(f_0\prod_{i=1}^{\ell+\ell'} f_i^{\tau_i})^\delta, d_{\ell+\ell'+1}\cdot f_{\ell+\ell'+1}^\delta,\ldots,d_\lambda\cdot f_\lambda^\delta, e\cdot h^\delta\right).$$

The new decryption key has randomness $\rho+\delta$, which is uniformly random in $\mathbb{Z}_p$.
In the encryption scheme with forward secrecy the prefix $\tau_1\ldots\tau_{\lambda_T}$ indicates the epoch the decryption key works for. The decryption key dk_τ with $\tau \in [0..T-1]$ must therefore allow us to derive the key $dk_{\tau_1\ldots\tau_{\lambda_T}}$ and also enable us to derive keys for all subsequent leaves in the height λ_T subtree. For any $\tau \in [0..T-1]$ let $\mathcal{T}_\tau$ be the minimal set of nodes $\tau_1\ldots\tau_\ell$ (with $\ell \le \lambda_T$) such that their subtrees are disjoint and cover all the leaves in $[\tau..T-1]$. A decryption key dk_τ is of the form

$$dk_\tau := \left(\tau, \{dk_{\tau_1\ldots\tau_\ell}\}_{\tau_1\ldots\tau_\ell\in\mathcal{T}_\tau}\right).[15]$$

The key update algorithm works given dk_τ and k such that $\tau+k<T$. From the set of tree decryption keys $\{dk_{\tau_1\ldots\tau_\ell}\}_{\tau_\ell}\}_{\tau_1\ldots\tau_\ell\in\mathcal{T}_\tau}$ it uses the relevant keys in the subtree to derive randomized decryption keys $dk_{\tau_1\ldots\tau_\ell}$ for all $\tau_1\ldots\tau_\ell \in \mathcal{T}_{\tau+k}\setminus\mathcal{T}_\tau$. It then erases intermediate data and returns

$$dk_{\tau+k} = (\tau+k, \{dk_{\tau_1\ldots\tau_\ell}\}_{\tau_1\ldots\tau_\ell\in\mathcal{T}_{\tau+k}}).$$

In case a decryption key is malformed or $k \notin [1..T-\tau-1]$ the update algorithm returns $\perp$.[16]

$\mathsf{Deal}(?sk, t, pk_1,\ldots,pk_n,\tau) \to d$**:**

Parse the input as $?sk = -$ or $sk \in \mathbb{Z}_p$, $t\in[1..n]$ with $n\le N$, $pk_i=(y_i,\pi_i)$ with $y_i\in\mathbb{G}_1$, $\tau\in[0..T-1]$ and if it fails return $\perp$.[17]

- If sk is not present, pick at random $sk \xleftarrow{\$} \mathbb{Z}_p$
- Parse $\tau=\tau_1\ldots\tau_{\lambda_T}$ in binary.
- Set $a_0 := sk$ and pick random $a_1,\ldots,a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$
- Compute $s_1,\ldots,s_n$ as $s_i=\sum_{k=0}^{t-1} a_k i^k \bmod p$
- Write each s_i in B-ary notation, i.e., $s_i=\sum_{j=1}^m s_{i,j}B^{j-1}$ with $s_{i,j}\in[0..B-1]$
- Pick randomness $r_1,s_1,\ldots,r_m,s_m \xleftarrow{\$} \mathbb{Z}_p$

[15] In the key generation algorithm, it can be seen that dk_0 has this form.

[16] Because decryption keys are perfectly randomized running $dk_{\tau+k}\leftarrow\mathsf{KUpd}(dk_\tau,k)$ is equivalent to $dk_{\tau+1}\leftarrow\mathsf{KUpd}(dk_\tau,1),\ldots,dk_{\tau+k}\leftarrow\mathsf{KUpd}(dk_{\tau+k-1},1)$ so in the security definitions we just use one-step updates and omit the jump k by defining $\mathsf{KUpd}(dk_\tau)=\mathsf{KUpd}(dk_\tau,1)$.

[17] The dealing algorithm does not explicitly check the proofs π_i of knowledge of discrete logarithms of y_i assuming such a check has already been performed elsewhere.

- Compute $C_{1,1}, \ldots, C_{n,m}, R_1, S_1, \ldots, R_m, S_m$ as

$$C_{i,j} := y_i^{r_j} \cdot g_1^{s_{i,j}} \qquad R_j := g_1^{r_j} \qquad S_j := g_1^{s_j}$$

- Compute $\tau_{\lambda_T+1} \ldots \tau_{\lambda_H} := H_{\lambda_H}(pk_1, \ldots, pk_n, C_{1,1}, \ldots, C_{n,m}, R_1, S_1, \ldots, R_m, S_m, \tau)$
- Compute $f := f(\tau_1 \ldots \tau_\lambda)$
- Compute $Z_1, \ldots, Z_m$ as $Z_j := f^{r_j} h^{s_j}$
- Compute $A_0 := g_2^{a_0}, \ldots, A_{t-1} := g_2^{a_{t-1}}$
- Compute $r := \sum_{j=1}^{m} r_j B^{j-1} \bmod p$
- Compute $R := g_1^r$ and $C_1 := y_1^r \cdot g_1^{s_1}, \ldots, C_n := y_n^r \cdot g_1^{s_n}$
- Construct a correct secret sharing proof

  $\pi_{\text{share}} \leftarrow \mathsf{Prove}_{\text{share}}(y_1, \ldots, y_n, A_0, \ldots, A_{t-1}, R, C_1, \ldots, C_n; r, s_1, \ldots, s_n)$

- Construct a correct chunking proof

  $\pi_{\text{chunk}} \leftarrow \mathsf{Prove}_{\text{chunk}}(y_1, \ldots, y_n, R_1, \ldots, R_m, C_{1,1}, \ldots, C_{n,m}; r_1, \ldots, r_m, s_{1,1}, \ldots, s_{n,m})$

- Erase intermediate information and return the dealing

$$d := \begin{pmatrix} C_{1,1}, \ldots, C_{n,m}, R_1, S_1 \ldots, R_m, S_m \\ Z_1, \ldots, Z_m, A_0, \ldots, A_{t-1}, \pi_{\text{share}}, \pi_{\text{chunk}} \end{pmatrix}$$

$\mathsf{DVfy}(?shvk, t, pk_1, \ldots, pk_n, \tau, d)$**:**

Parse the input as $?shvk = -$ or $?shvk \in \mathbb{G}_2$, $t \in [1..n]$ with $n \leq N$, $pk_i = (y_i, \pi_i)$ with distinct $y_i \in \mathbb{G}_1$, $\tau \in [0..T-1]$ and if the parsing fails return $\perp$.[18]

- Check the dealing is of the form

$$d := \begin{pmatrix} C_{1,1}, \ldots, C_{n,m}, R_1, S_1, \ldots, R_m, S_m \\ Z_1, \ldots, Z_m, A_0, \ldots, A_{t-1}, \pi_{\text{share}}, \pi_{\text{chunk}} \end{pmatrix}$$

  with $C_{i,j}, R_j, S_j \in \mathbb{G}_1$ and $Z_j, A_k \in \mathbb{G}_2$
- If the optional *shvk* is present, check $shvk = A_0$
- Set $\tau_{\lambda_T+1} \ldots \tau_\lambda := H_{\lambda_H}(pk_1, \ldots, pk_n, C_{1,1}, \ldots, C_{n,m}, R_1, S_1, \ldots, R_m, S_m, \tau)$
- Set $f := f(\tau_1, \ldots, \tau_\lambda)$
- Verify for each triple $(R_1, S_1, Z_1), \ldots, (R_m, S_m, Z_m)$ that $e(g_1, Z_j) = e(R_j, f) \cdot e(S_j, h)$
- Verify the proof for correct secret sharing by checking

$$\mathsf{PVfy}_{\text{share}}(y_1, \ldots, y_n, A_0, \ldots, A_{t-1}, \prod_{j=1}^{m} R_j^{B^{j-1}}, \prod_{j=1}^{m} C_{1,j}^{B^{j-1}}, \ldots, \prod_{j=1}^{m} C_{n,j}^{B^{j-1}}; \pi_{\text{share}}) = \top$$

- Verify the proof for chunking by checking

$$\mathsf{PVfy}_{\text{chunk}}(y_1, \ldots, y_n, R_1, \ldots, R_m, C_{1,1}, \ldots, C_{n,m}; \pi_{\text{chunk}}) = \top$$

[18] The dealing verification algorithm assumes the proofs π_i of knowledge of the discrete logarithms of y_i have already been done elsewhere.

– Return $\top$ if all checks pass, else return $\bot$

$\mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell)$**:** Given $1 \le t \le n \le N$, and a set I of ℓ indices $1 \le i_1 < \ldots < i_\ell \le n$ and a set of dealings

$$d_j := (\ldots, A_{j,0}, \ldots, A_{j,t-1}, \ldots)$$

with all $A_{j,k} \in \mathbb{G}_2$ compute $A_0, \ldots, A_{t-1}$ as

$$A_k := \prod_{j=1}^{\ell} A_{j,k}^{L^I_{i_\ell}(0)}$$

– Set $vk := A_0$
– Compute $shvk_1, \ldots, shvk_n$ as

$$shvk_j := \prod_{k=0}^{t-1} A_k^{j^k}$$

– If all works return $(vk, shvk_1, \ldots, shvk_n)$, else return $\bot$.

$\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n)$**:** We recap the validity condition for public key material in the BLS threshold signature scheme. Check $1 \le t \le n \le N$ and $vk, shvk_1, \ldots, shvk_n \in \mathbb{G}_2$. Set $shvk_0 := vk$ and $J = \{0, \ldots, t-1\}$. For $i = t, \ldots, n$ check whether

$$shvk_i = \prod_{j \in J} shvk_j^{L^J_j(i)}.$$

Return $\top$ if all checks pass, else return $\bot$

$\mathsf{SKRetrieve}(i, dk_{\tau'}, K, d_1, \ldots, d_\ell, \tau)$**:** Parse each dealing as

$$d_k = (C_{k,1,1}, \ldots, C_{k,n,m}, \ldots, C_{k,i,1}, R_{k,1}, S_{k,1}, \ldots, R_{k,m}, S_{k,m}, Z_{k,1}, \ldots, Z_{k,m}, \ldots),$$

with $C_{k,i,j}, R_{k,j}, S_{k,j} \in \mathbb{G}_1$ and $Z_{k,j} \in \mathbb{G}_2$. Check $1 \le i \le n \le N$.
For $k = 1, \ldots, \ell$ define $\tau_{k,1} = \tau_1, \ldots, \tau_{k,\lambda_T} = \tau_{\lambda_T}$ and compute

$$\tau_{k,\lambda_T+1} \ldots \tau_{k,\lambda} := H_{\lambda_H}(pk_1, \ldots, pk_n, C_{k,1,1}, \ldots, C_{k,n,m}, R_{k,1}, S_{k,1}, \ldots, R_{k,m}, S_{k,m}, \tau).$$

Let $f_k := f(\tau_{k,1}, \ldots, \tau_{k,\lambda})$.
Assuming $\tau' \le \tau$ derive as described in decryption key update from $dk_{\tau'}$ a BTE decryption key

$$dk_{\tau_{k,1},\ldots,\tau_{k,\lambda}} = (\tau_{k,1} \ldots \tau_{k,\lambda}, a_k, b_k, e_k) \in \{0,1\}^\lambda \times \mathbb{G}_1 \times \mathbb{G}_2^2$$

for $k = 1, \ldots, \ell$.[19]

[19] For computational efficiency, the randomization step may be omitted in the key derivation, i.e., the key derivation may use $\delta = 0$. When $\mathsf{SKRetrieve}$ is run on dealings that verify as correct, we have $e(g_1, Z_{k,j}) = e(R_{k,j}, f) \cdot e(S_{k,j}, h)$ and it follows that any choice of δ in the key randomization yields the same $M_{k,j}$ values below. Assuming the derived key and intermediate data is erased immediately after use, the retrieval algorithm therefore returns the same result and does not leave a trace that could be used to patch together several nonrandomized derived keys to give the adversary an advantage.

For each $k = 1, \ldots, \ell$ and $j = 1, \ldots, m$ compute

$$M_{k,j} := e(C_{k,i,j}, g_2) \cdot e(R_{k,j}, b_k^{-1}) \cdot e(a_k, Z_{k,j}) \cdot e(S_{k,j}, e_k^{-1}).$$

Then do a brute force search with the Baby-Step Giant-Step algorithm for $s_{k,j} \in \{z/\Delta | \Delta \in [1..E-1], z \in [1-Z; Z-1]\}$ such that $M_{k,j} = e(g_1, g_2)^{s_{k,j}}$ Compute

$$s_k := \sum_{j=1}^{m} s_{k,j} B^{j-1} \bmod p.$$

Parse $K \subset [1..n]$ as distinct indices $k_1 < \ldots < k_\ell$ and compute

$$s_i := \sum_{j=1}^{\ell} s_{k,j} L_{k_j}^K(0).$$

Erase intermediate data and if everything went well return $sk := s_i$ and otherwise return $\perp$.

$\mathsf{SKVfy}(sk, shvk)$**:** We recap the validity condition for secret share-signing keys in the BLS threshold signature scheme. If $sk \in \mathbb{Z}_p$ and $shvk = g_2^{sk}$ erase intermediate data and return $\top$, else return $\perp$.

Theorem 16. *The construction is correct.*

Proof. Key correctness. We inherit key correctness from the underlying fs-CCA secure encryption scheme: Key generation produces keys of the form $pk = (y, \pi_{\mathsf{dlog}})$ with a proof of knowledge of the discrete logarithm x such that $y = g_1^x$. Key verification checks the proof of knowledge of the discrete logarithm of y, and due to its perfect completeness always accepts. This means key generation returns valid keys, $\Pr[(pk, dk_0) \leftarrow \mathsf{KGen} : \mathsf{KVfy}(pk) = \top] = 1$.

Dealings made over valid public keys are valid. More precisely, if $t \le n \le N$ and $\tau \in [0..T-1]$ and $pk_1, \ldots, pk_n$ are valid public keys so that $\mathsf{KVfy}(pk_i) = \top$ and $\mathsf{SKVfy}(sk, shvk) = \top$ or alternatively $(sk, shvk) = (-, -)$ then we want

$$\Pr[d \leftarrow \mathsf{Deal}(?sk, t, pk_1, \ldots, pk_n, \tau) : \mathsf{DVfy}(?shvk, t, pk_1, \ldots, pk_n, \tau, d) = \top] = 1.$$

First, observe that if $\mathsf{SKVfy}(sk, shvk) = \top$ then $sk \in \mathbb{Z}_p$ and $shvk = g_2^{sk}$. Including sk in the dealing algorithm makes $A_0 = g_2^{sk}$ and including $shvk$ in the dealing verification adds the check $A_0 = shvk$, which is true. So on a valid share-signing key pair $(sk, shvk)$ the probability of a dealing being valid is not affected. What remains is to ensure the dealing is valid when we omit $shvk$ in the verification.

The precondition $\mathsf{KVfy}(pk_i)$ ensures the public keys are of the form $pk_i = (y_i, \pi_i)$ with $y_i \in \mathbb{G}_1$. Taken with the other preconditions $t \le n \le N, \tau \in [0..T-1]$ and the optional sk, which must belong to $\mathbb{Z}_p$ to have $\mathsf{SKVfy}(sk, shvk) = \top$, we see that the inputs to the dealing algorithm are as expected. Going through the steps of the dealing algorithm, the completeness of the NIZK proofs (except with small completeness error in the chunking proof) means it terminates without either proof being $\perp$ since we give valid witnesses for correct secret sharing and chunking. The resulting dealing is of the form

$$d = \begin{pmatrix} C_{1,1}, \ldots, C_{n,m}, R_1, S_1, \ldots, R_m, S_m \\ Z_1, \ldots, Z_m, A_0, \ldots, A_{t-1}, \pi_{\text{share}}, \pi_{\text{chunk}} \end{pmatrix}.$$

Looking at the dealing verification algorithm, the preconditions $1 \leq t \leq n \leq N$ and $\tau \in [0..T-1]$ and $\mathsf{KVfy}(pk_i) = \top$ ensure that it parses the inputs without problems. Then it checks the format of the dealing and sees that indeed the format given above is correct with respect to the given parameters.

The dealing verification algorithm has all the values needed to compute $\tau_{\lambda_T+1} \ldots \tau_\lambda$ and then compute f. By construction of the ciphertexts in the dealing algorithm we see for each triple (R_j, S_j, Z_j) that the check $e(g_1, Z_j) = e(R_j, f) \cdot e(S_j, h)$ holds since $Z_j = f^{r_j} h^{s_j}$ and $R_j = g_1^{r_j}$ and $S_j = g_1^{s_j}$.

Finally, the correct secret sharing proof has perfect completeness and verifies as being correct. The same goes for the chunking proof except for the small completeness error. The dealing verification algorithm therefore returns $\top$.

Valid dealings result in valid verification keys. If $t, pk_1, \ldots, pk_n, \tau$ and dealings $d_1, \ldots, d_\ell$ all satisfy $\mathsf{DVfy}(-, t, pk_1, \ldots, pk_n, \tau, d_k) = \top$ and $I \subseteq [1..N]$ has size ℓ then

$$\Pr[(vk, shvk_1, \ldots, shvk_n) \leftarrow \mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell) : \mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) = \top] = 1.$$

To see this is true, first observe that it follows from the dealing verification that the parameters are meaningful, i.e., $1 \leq t \leq N$ and each dealing d_k includes elements $A_{k,0}, \ldots, A_{k,t-1} \in \mathbb{G}_2$. Since the index set $I \subset [1..N]$ has size ℓ as expected, the combine algorithm computes $A_0, \ldots, A_k \in \mathbb{G}_2$ successfully. It then sets $vk = A_0$, which we will think of as $shvk_0$, and $shvk_j := \prod_{k=0}^{t-1} A_k^{j^k} = g_2^{a(j)}$ for the implicit degree $t-1$ polynomial $a(j)$ and $j = 1, \ldots, n$. This means that the discrete logarithms of the share verification keys lie on a degree $t-1$ polynomial and the verification succeeds since by the properties of Lagrange interpolation polynomials

$$a(i) = \sum_{j \in J} a(j) L_j^J(i),$$

where $J = \{0, \ldots, t-1\}$.

An honest receiver can retrieve a valid secret key for herself from a set of valid dealings. The retrieval adversary $\mathcal{A}$'s advantage is

$$\Pr\left[\begin{array}{c} (pk, dk_0) \leftarrow \mathsf{KGen}; (i, I, pk_1, \ldots, pk_n, d_1, \ldots, d_\ell, \tau) \leftarrow \mathcal{A}^{\mathsf{KUpd}}(pk, dk_0) \\ (vk, shvk_1, \ldots, shvk_n) \leftarrow \mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell) \\ sk \leftarrow \mathsf{SKRetrieve}(j, dk_\tau, I, d_1, \ldots, d_\ell) : \\ I \subset [1..n] \text{ and } |I| = \ell \text{ and } pk_i = pk \text{ and all dealings are valid, i.e.,} \\ \mathsf{DVfy}(-, t, pk_1, \ldots, pk_n, \tau, d_k) = \top \text{ and } \tau' \leq \tau, \text{ yet } \mathsf{SKVfy}(sk, shvk_i) \neq \top \end{array}\right],$$

where on each call to KUpd the oracle sets $dk_{\tau'+1} := \mathsf{KUpd}(dk_{\tau'})$ and $\tau' := \tau'+1$, and stops reacting to further calls once $\tau'+1 = T$. The definition says the oracle should give the decryption key to the adversary, but in our scheme this is a moot point since the adversary already has dk_0 and derived keys are perfectly randomized.

Since the dealings are valid with the same parameters t and keys $pk_1, \ldots, pk_n$ and epoch τ, they all have the same format

$$d_k = \begin{pmatrix} C_{k,1,1}, \ldots, C_{k,n,m}, R_{k,1}, S_{k,1}, \ldots, R_{k,m}, S_{k,m} \\ Z_{k,1}, \ldots, Z_{k,m}, A_{k,0}, \ldots, A_{k,t-1}, \pi_{k,\text{share}}, \pi_{k,\text{chunk}} \end{pmatrix}.$$

Since $pk = pk_i$ it has been generated properly by the key generation algorithm, is of the form $pk_i = (y_i, \pi_i)$ with $y_i \in \mathbb{G}_1$, and the decryption key $dk_{\tau'}$ for this index is well defined in the definition. The perfect correctness of the encryption scheme also ensures that $dk_{\tau'}$ is able to decrypt the ciphertexts $C_{k,1,1}, \ldots, Z_{k,m}$ with respect to index i if the ciphertexts are correctly generated.

The dealings must all verify as valid in the success condition in the probability. The chunking proofs in the dealing therefore guarantee up to their soundness error that they have $R_{k,j} = g_1^{r_{k,j}}$ and $C_{k,i,j} = y_i^{r_{k,j}} g_1^{s_{k,i,j}}$ with valid chunks $s_{k,i,j}$. Assuming chunking is correct, the other checks on the ciphertexts imply that they are well formed, i.e., valid outputs of the encryption algorithm for some choice of randomness $r_{k,j}, s_{k,j} \in \mathbb{Z}_p$. The retrieval algorithm therefore succeeds in learning the chunks $s_{k,j,1}, \ldots, s_{k,j,m}$ from the dealings.

Combining the chunks, $\mathsf{SKRetrieve}$ obtains $s_{1,j}, \ldots, s_{t,j} \in \mathbb{Z}_p$ from each deal. For each deal, the share proof $\pi_{k,\text{share}}$ guarantees up to its soundness error that $s_{k,i} = a_k(i)$, where $a_k(X)$ is the degree $t-1$ polynomial defined by the discrete logarithms of $A_{k,0}, \ldots, A_{k,t-1}$. As a result, since both the verification key combination algorithm and the share retrieval algorithm use the same Lagrange interpolation defined by the set I, we get $s_i = \sum_{k=1}^{t} s_{k,i} L_{i_k}^{I}(0)$ and $shvk_i = g_2^{\sum_{k=1}^{t} a_k(i) L_{i_k}^{I}(0)}$ satisfy $shvk_i = g_2^{s_i}$. Consequently, with $sk := s_i$ we have $\mathsf{SKVfy}(sk, shvk_i) = \top$. □

Theorem 17. *The construction has perfect verification-key preservation.*

Proof. Suppose $\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) = \top$. Since the check guarantees $shvk_t, \ldots, shvk_n$ can be derived with Lagrange interpolation in the exponent from vk and $shvk_1, \ldots, shvk_{t-1}$ it ensures that there is a degree $t-1$ polynomial $a(i)$ so that $vk = g_2^{a(0)}$ and $shvk_i = g_2^{a(i)}$. Therefore for any index set I containing $1 \leq i_1 < \ldots < i_t \leq n$ we have $vk = \prod_{j=1}^{t} shvk_{i_j}^{L_{i_j}^{I}(0)}$.

Now, given t dealings $d_1, \ldots, d_t$ that are valid with respect to $shvk_{i_1}, \ldots, shvk_{i_t}$ respectively, we know from the validity checks $\mathsf{DVfy}(shvk_{i_k}, t', pk_1, \ldots, pk_{n'}, \tau, d_k)$ that they use $A_{1,0} = shvk_{i_1}, \ldots, A_{t,0} = shvk_{i_t}$. Since $\mathsf{VKCombine}$ computes $A_0 = \prod_{j=1}^{t} A_{j,0}^{L_{i_j}^{I}(0)}$ we therefore have $A_0 = vk$. And since the new $vk' = A_0$ we see $vk' = vk$ and the verification key is preserved. □

8 Security

A threshold signature scheme with a matching distributed key generation protocol consists of the following algorithms $\mathsf{KGen}, \mathsf{KVfy}, \mathsf{KUpd}, \mathsf{Deal}, \mathsf{DVfy}, \mathsf{VKCombine}, \mathsf{VKVfy}, \mathsf{SKRetrieve}, \mathsf{SKVfy}, \mathsf{SigShare}, \mathsf{SigShVfy}, \mathsf{SigShCombine}, \mathsf{SigVfy}$. We have already defined correctness, uniqueness and key-preservation earlier and assume the scheme has those properties.

We want to avoid unauthorized signatures. This means if we see a signature on a message it should be because a threshold of share holders were involved in producing it. A participant can be willingly involved in a signature by creating a signature share, but also by being corrupt and doing the adversary's bidding, or by failing to do timely erasure of her share-signing key or being negligent in updating her decryption key such that it falls in the wrong hands.

Going into the security definition, we imagine a world where honest parties may generate fresh keys that are made public, i.e., known to the adversary. Their matching decryption keys are kept secret from the adversary unless the party is corrupted. We will assume that public keys have enough entropy to be unique, such that a party can be uniquely identified by its public key, and we will let Q_{pk} be the set of public keys generated by honest parties, specifically, they were honest when they created the key. There may be situations where the adversary circulates public keys on behalf of dishonest parties. We cannot say much about those keys except we assume the Internet Computer will check all public keys in circulation are well formed.

Parties can update their decryption key to a new epoch and erase the decryption key for the previous epoch so they no longer have the ability to decrypt messages for past epochs. They should do so whenever they know they will never decrypt any more ciphertext wrt the old epoch. We keep track of the decryption keys in public key records of the form (pk, dk_τ), where dk_τ is pk's decryption key for epoch τ. In the definition of unforgeability we capture this by letting the adversary decide when a party should update the decryption key, as long as it such corruption does not violate the assumptions we want our key management system to work for. As long as the adversary cannot violate those conditions, the threshold signature scheme should remain unforgeable.

The parties may at times create new threshold signature verification keys with matching secret shares of the signing key. We let the adversary decide when this happens but to uniquely identify who is supposed to do what the adversary must provide a configuration of participants with matching threshold and epoch. This corresponds to how we configure the creation of subnet keys on the Internet Computer and a number of sanity checks can be made on the configuration.

We enable the adversary to trigger the creation of an honest fresh dealing, modeling that an honest party runs $\mathsf{Deal}(-, t, pk_1, \ldots, pk_n, \tau)$, where the values are taken from an existing configuration. The set Q_d contains all honest dealings. Later the adversary may combine honest dealings with arbitrary dealings of its own to form a transcript making the Internet Computer recognize a new public key. In the security definition we require the adversary to include at least one honest dealing in a transcript, since if the adversary makes all the dealings they cannot contribute towards security. A transcript, which references an existing configuration, is verified and sanity checked. If the transcript is accepted, the dealings are combined to derive a verification key vk, which is registered in Q_{vk}. We then let all honest receivers extract their secret share-signing keys. We keep track of those secret share-keys in records of the form (pk, id, sk, τ), where pk is an honest party's key, id identifies the configuration, sk is the share-signing key retrieved by the honest node with pk, and τ is the epoch of the batch.

At other times parties want to reshare an existing secret threshold signing key. Here the adversary must also first create a configuration, but this time the configuration must reference a prior configuration and a transcript of existing public keys it is building on. Also for resharing we enable the adversary to trigger an honest dealing, this time referencing (pk, id') for an honest party that holds a relevant share-signing key recorded as $(pk, id', sk, \cdot)$. The honest node then runs $\mathsf{Deal}(sk, t, pk_1, \ldots, pk_n, \tau)$ and the honest dealing is added to Q_d. Again the adversary may combine honest dealings with arbitrary dealings of its own making to provide a transcript that proscribes a new secret sharing of the existing verification key. Whenever such a resharing takes place, the receivers should verify that the resharing builds on an existing prior transcript, and we keep track of key evolution in transcript records of the form $(vk, id, t, pk_1, shvk_1, \ldots, pk_n, shvk_n, \tau)$.

The receivers may use the share-signing keys to provide signature shares on arbitrary message. In our model, we give the adversary control over when this happens and which party sign which messages.

On the Internet Computer an honest party erases a share-signing key when it is obsolete. Of course, erasure does not make much sense as long as the party has a decryption key that will permit it to recover the share-signing key again. We therefore only enable the adversary to ask for erasure of a share-signing key that can no longer be recovered by the honest party.

Finally, our security model is intended to capture dynamic corruption, so we enable the adversary to corrupt parties. Whenever the adversary corrupts a party, she learns all interesting data the party has, i.e., its decryption key and all its unerased shares. Afterwards the records pertaining to that party are deleted, since the model only keeps track of honest parties and the adversary now has the power to act on the party's behalf.

We define the advantage of a forging adversary $\mathcal{A}$ to be

$$\Pr\left[\begin{array}{c} (vk, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{KGen},\mathsf{KUpd},\mathsf{Config},\mathsf{Deal},\mathsf{Transcript},\mathsf{Erase},\mathsf{SigShare},\mathsf{Corrupt}} : \\ vk \in Q_{vk} \text{ and } \mathsf{SigVfy}(vk, m, \sigma) = \top \\ \text{and for all } id \text{ with a transcript record } (vk, id, t, pk_1, shvk_1, \ldots, pk_n, shvk_n, \tau) : \\ \#\left\{ i \;\middle|\; \begin{array}{c} \text{there is a corruption record } (C, pk_i, \tau') \text{ with } \tau' \leq \tau, \text{ or with } \tau' > \tau \\ \text{and an un-erased share-signing key record } (pk_i, id, sk, \tau), \\ \text{or } \; pk_i \notin Q_{pk}, \text{ or there is a signature share record } (id, pk_i, m) \end{array} \right\} < t \end{array}\right].$$

To win the adversary must produce a valid signature under a recognized verification key, i.e, a verification key from a transcript with at least one honest dealing. The adversary may get help in creating such a signature by acting with a share-signing key it knows because it corrupted an honest party, by controlling a party that was malicious from the outset, or by seeing signature shares on the message from honest parties. We say the adversary wins if for all indexed configurations and transcripts pertaining to the verification key, the adversary does not have help exceeding the threshold t that pertains to the configuration. The oracles the definition refers to are as follows:

```
$\mathsf{KGen}$
$(pk, dk_0) \leftarrow \mathsf{KGen}$
If $pk \in Q_{pk}$ let experiment return $\top$
$Q_{pk} := Q_{pk} \cup \{pk\}$
Store a public key record $(pk, dk_0)$
Return $pk$
```

```
$\mathsf{KUpd}(pk)$
Find stored public key $(pk, dk_\tau)$ and if there is no such record or $\tau = T$ return $\perp$
Set $dk_{\tau+1} \leftarrow \mathsf{KUpd}(dk_\tau)$ and update the stored public key record to $(pk, dk_{\tau+1})$
Return $\top$
```

```
$\mathsf{Config}(?id', t, pk_1, \ldots, pk_n, \tau)$
If there is already a configuration record $(*, *, t, pk_1, \ldots, pk_n, \tau)$ return $\perp$
If $id' = -$ let $\tau' := -1$
Else find the transcript record $(vk, id', , \ldots, \tau')$ and if no such record exists return $\perp$
If $t \notin [1 \ldots n]$ or $\tau \notin [\tau' + 1 \ldots T - 1]$ return $\perp$
For $i = 1, \ldots, n$ if $\mathsf{KVfy}(pk_i) = \perp$ return $\perp$
If there is a collision in public keys $pk_i = pk_j$ for $i \neq j$ return $\perp$
Store a configuration record $(id, id', t, pk_1, \ldots, pk_n, \tau)$        // On initialization when not yet used $id = 0$
Set $id := id + 1$ and return $\top$
```

```
$\mathsf{Deal}(?pk, id)$
Find the configuration record $(id, id', t, pk_1, \ldots, pk_n, \tau)$ and if there is none return $\perp$
If $pk = -$ let $sk = -$ and if $id' \neq -$ return $\perp$
Else find the share-signing key record $(pk, id', sk, \tau')$ and if there is none return $\perp$
$d \leftarrow \mathsf{Deal}(sk, t, pk_1, \ldots, pk_n, \tau)$
Let $Q_d := Q_d \cup \{d\}$ and return $d$
```

```
$\mathsf{Transcript}(id, ?I, d_1, \ldots, d_\ell)$
If there is already a transcript record of the form $(vk, id, t, \ldots)$ return $\bot$
If a dealing repeats $d_i = d_j$ with $i \neq j$ return $\bot$
Find the configuration record $(id, id', t, pk_1, \ldots, pk_n, \tau)$ and if there is none return $\bot$
If there is no honest dealing $d_i \in Q_d$ return $\bot$
If the optional index set $?I$ is $-$ (not included)
  Only allow it if the configuration does not build on a prior configuration, if $id' \neq -$ return $\bot$
  If there is a invalid dealing $d_i$ where $\mathsf{DVfy}(-, t, pk_1, \ldots, pk_n, \tau, d_i) = \bot$ return $\bot$
  Set $I := \{1, \ldots, \ell\}$
Else
  Find the transcript record $(vk', id', t', pk'_1, shvk'_1, \ldots, pk'_\nu, shvk'_\nu, \tau')$ and if there is none or $\ell \neq t'$ return $\bot$
  Parse $I$ as a set of indices $1 \leq i_1 < \ldots i_\ell \leq \nu$ and if it fails return $\bot$
  If there is some $i_j \in I$ where $\mathsf{DVfy}(shvk'_{i_j}, t, pk_1, \ldots, pk_n, \tau, d_j) = \bot$ return $\bot$
$(vk, shvk_1, \ldots, shvk_n) \leftarrow \mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell)$
$Q_{vk} := Q_{vk} \cup \{vk\}$
Store a transcript record $(vk, id, t, pk_1, shvk_1, \ldots, pk_n, shvk_n, \tau)$
For $i = 1, \ldots, n$ if there is a public key record $(pk_i, dk_{\tau_i})$ with $\tau_i \leq \tau$
  $sk_i \leftarrow \mathsf{SKRetrieve}(i, dk_{\tau_i}, I, d_1, \ldots, d_\ell)$
  Store a share-signing key record $(pk_i, id, sk_i, \tau)$
Return $(vk, shvk_1, \ldots, shvk_n)$
```

```
$\mathsf{Erase}(pk, id)$
Find a stored public key record $(pk, dk_\tau)$ and if there is none return $\bot$
Find a stored share-signing key record $(pk, id, sk, \tau')$ and if there is none return $\bot$
Delete the share-signing key record $(pk, id, sk, \tau')$ and return $\top$
```

```
$\mathsf{SigShare}(id, pk, m)$
Find a stored share-signing key record $(pk, id, sk, \tau)$ and if there is none return $\bot$
$sh \leftarrow \mathsf{SigShare}(sk, m)$
Store a signature share record $(id, pk, m)$ and return $sh$
```

```
$\mathsf{Corrupt}(pk)$
Locate the stored public key record $(pk, dk_\tau)$ and if there is none return $\bot$
Store a corruption record $(C, pk, \tau)$
Return the public key record $(pk, dk_\tau)$ and all share-signing key records $(pk, id, sk, \tau')$
    and delete the public key record
```

Correctness properties in the security definition The oracles preserve correctness properties, e.g., KGen will produce valid keys pk. Looking at the Transcript oracle, we note in particular that it always produces valid verification keys. I.e., if a transcript record $(vk, id, t, pk_1, shvk_1, \ldots, pk_n, shvk_n, \tau)$ is stored, which can only happen through a call to $\mathsf{Transcript}$, then vk and the share verification keys $shvk_i$ are constructed with a call $(vk, shvk_1, \ldots, shvk_n) \leftarrow \mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell)$. The transcript oracle ensures there is a ν such that $1 \leq \ell \leq \nu \leq N$ and I contains indices $1 \leq i_1 < \ldots < i_\ell \leq \nu$ and all dealings are valid, i.e., $\mathsf{DVfy}(-, t, pk_1, \ldots, pk_n, \tau, d_{i_j}) = \top$. Correctness therefore means $\mathsf{Transcript}$ always produces valid public keys, i.e., $\mathsf{VKVfy}(t, vk, shvk_1, \ldots, shvk_n) = \top$.

Configuration records are created by the Config oracle. Configuration records can be organized into trees, where we label nodes with the id. A configuration record $(id, -, \ldots)$ is a root of a tree. While a configuration record $(id, id', \ldots)$ means id is a child of id'. When the protocol has verification key preservation, all nodes in a configuration tree will get the same verification key as the root (or still not have been assigned a verification key) when $\mathsf{Transcript}$ is called.

Lemma 2. *If the protocol has verification key preservation, then when* $\mathsf{Transcript}$ *stores a transcript record $(vk, id, \ldots)$ building on a configuration $(id, id', \ldots)$ it must be the case a transcript record $(vk', id', \ldots)$ has already been stored and $vk = vk'$.*

Proof. Transcript records can only be stored through a call to the $\mathsf{Transcript}$ oracle. For a transcript record $(vk, id, \ldots)$ to be stored, there must already be a configuration record $(id, id', \ldots)$. If $id' = -$ there is nothing to prove since it is the root of a configuration tree, but otherwise we must argue there is already a record $(vk, id', \ldots)$.

The $\mathsf{Transcript}$ oracle indeed checks there is an earlier transcript record $(vk', id', \ell, pk'_1, shvk'_1, \ldots, pk'_\nu, shvk'_\nu, \tau')$. As noted before, since it is a transcript record we must have $\mathsf{VKVfy}(\ell, vk', shvk'_1, \ldots, shvk'_\nu) = \top$. It also follows from the checks in Config and $\mathsf{Transcript}$ that $1 \leq \ell \leq \nu \leq N$ and $1 \leq t \leq n \leq N$. The oracle checks $1 \leq i_1 < \ldots < i_\ell \leq \nu$ and in the configuration record $(id, \ldots, \tau)$ we can only have $\tau \in [0..T-1]$. Finally, the transcript oracle checks for each d_j that $\mathsf{DVfy}(shvk_{i_j}, t, pk_1, \ldots, pk_n, \tau, d_j) = \top$. This means all conditions for verification key preservation are satisfied, and therefore when it calls $(vk, shvk_1, \ldots, shvk_n) \leftarrow \mathsf{VKCombine}(t, n, I, d_1, \ldots, d_\ell)$ we get $vk = vk'$. $\square$

Relaxed dynamic security While our distributed key generation may be secure against dynamic adversaries, our security proofs use a simulation argument that runs into selective opening attack problems. Namely, we would like to use the fs-CCA security of the encryption scheme to hide the shares honest receivers get in honest dealings. But if we try to use indistinguishability of the encryption scheme and encrypt a different dummy share, then upon corruption of the receiver the adversary will learn that the share is wrong and does not match the ciphertexts in the dealings and not the share verification key either. This forces us to encrypt the right shares and now we get stuck in the simulation in the security proof. The way we get around this problem is to restrict the adversary such that we do not have to provide those shares. Specifically, we restrict the adversary such that it cannot corrupt an honest receiver while the decryption key is at an epoch where it is possible to decrypt a simulated dealing. Moreover, it should not be possible to corrupt a party with an unerased share.

Restricting the adversary in this way, we introduce an honesty lock: a set H that specifies parties that cannot be corrupted until a certain epoch. We define the advantage of a relaxed forging adversary $\mathcal{A}$ to be

$$\Pr\left[\begin{array}{c} (vk, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{KGen},\mathsf{KUpd},\mathsf{Config},\mathsf{Deal},\mathsf{Transcript},\mathsf{Erase},\mathsf{SigShare},\mathsf{Corrupt}} : \\ vk \in Q_{vk} \text{ and } \mathsf{SigVfy}(vk, m, \sigma) = \top \\ \text{and for all } id \text{ with a transcript record } (vk, id, t, pk_1, shvk_1, \ldots, pk_n, shvk_n, \tau) : \\ \text{there is a configuration record } (id, \ldots, H) \text{ with } |H| > n - t \text{ and} \\ \#\left\{ i \;\middle|\; \begin{array}{c} \text{there is a corruption record } (C, pk_i, \tau') \text{ with } \tau' \leq \tau, \text{ or with } \tau' > \tau \\ \text{and an un-erased share-signing key record } (pk_i, id, sk, \tau), \\ \text{or } \; pk_i \notin Q_{pk}, \text{ or there is a signature share record } (id, pk_i, m) \end{array} \right\} < t \end{array}\right],$$

where we modify the configuration and corruption oracles

```
$\mathsf{Config}(?id', t, pk_1, \ldots, pk_n, \tau, H)$
If there is already a configuration record $(*, *, t, pk_1, \ldots, pk_n, \tau)$ return $\bot$
If $id' = -$ let $\tau' := -1$
Else find a prior transcript record $(vk, id', \ldots, \tau')$ and if there is none return $\bot$
If $t \notin \{1, \ldots, n\}$ or $\tau \notin \{\tau' + 1, \ldots, T - 1\}$ return $\bot$
For $i = 1, \ldots, n$ if $\mathsf{KVfy}(pk_i) = \bot$ return $\bot$
If there is a collision in public keys $pk_i = pk_j$ for $i \neq j$ return $\bot$
Parse $H = \{pk_{i_1}, \ldots, pk_{i_\ell}\}$ with $1 \leq i_1 < \ldots < i_\ell \leq n$
If there is $pk_{i_j}$ with no public key record $(pk_{i_j}, dk_{\tau_{i_j}})$ return $\bot$
Store honesty records $(H, pk_{i_1}, \tau), \ldots, (H, pk_{i_\ell}, \tau)$
Store the configuration record $(id, id', t, pk_1, \ldots, pk_n, \tau, H)$          // On initialization $id = 0$
Set $id := id + 1$ and return $\top$              // Wlog sequential identifiers
```

```
$\mathsf{Corrupt}(pk)$
For all honesty records $(H, pk, \tau)$
    If there is a stored public key record $(pk, dk_{\tau'})$ with $\tau' \leq \tau$ return $\bot$
    If there is a stored share-signing key record $(pk, id, sk, \tau')$ with $\tau' \leq \tau$ return $\bot$
Locate the public key record $(pk, dk_\tau)$ and if there is none return $\bot$
Store a corruption record $(C, pk, \tau)$
Return the public key record $(pk, dk_\tau)$ and all share-signing key records $(pk, id, sk, \tau')$
    and delete the public key record
```

How should we interpret this relaxed security definition? It says the adversary when invoking a dealing (without loss of generality we can assume the dealing comes right after the configuration call) must supply a list H of public keys for parties that will remain honest until after their decryption key has been updated past this epoch and all prior secret share-signing keys have been erased, but is that realistic? It depends on the type of corruptions we believe are realistic in the world. If the adversary can corrupt an arbitrary party anywhere at a moments notice, then no, but in that case why would we believe a specific is realistic? If on the other hand, we believe the adversary at any moment in time has a set of parties in mind it could realistically corrupt (e.g. a vulnerability on a specific type of machine, a data center operator known to be susceptible to bribes) within the next few epochs, then this is a very realistic security model on the spectrum between static corruption and instantaneous dynamic corruption. The shorter the delay from dealings to usage being over and parties upgrading their decryption keys to new epochs and erasing old threshold shares, the more realistic the relaxed security model is. This is not to say it is realistic in all worlds: perhaps every data center operator has a price at which they will be bribed and act quickly in collaboration with the adversary; in which case the adversary could potentially target any node even if not being able to afford corrupting so many. And of course the adversary may also try a DoS attack to halt the system and extend the window in which she can corrupt somebody.

Theorem 18. *Our protocol for threshold BLS signatures has relaxed dynamic security.*

Sketch of proof. As argued earlier, and this holds also for the relaxed security definitions, configurations can be organized into trees, where each configuration is a root of a tree or references a transcript for an earlier configuration. Moreover, we have proved our protocol has verification key preservation, so all transcripts in the same configuration tree have the same verification key as the the root configuration's transcript. It can also be shown that for verification keys created with the involvement of at least one honest dealer, they have negligible probability of becoming the verification key in different configuration tree also created by an honest dealer. So we only need to concern ourselves with the case where the adversary forges a signature with respect to a uniquely defined configuration tree.

Another useful observation is that there is negligible probability of finding a dealing that is valid for two different configurations. The checks in the Config oracle ensures no tuple $(t, pk_1, \dots, pk_n, \tau)$ appears in two configurations. Counting the number of elements $A_0, \dots, A_{t-1}$ in the dealing, we see that t must be the same in all configurations it is valid for. Moreover, the ciphertext in the dealing uniquely determines $\tau_1, \dots, \tau_\lambda$, which in turn uniquely determines τ and the message digest $\tau_{\lambda_T+1}, \dots, \tau_\lambda$. Since the message digest is recomputed in the ciphertext verification, collision resistance means $pk_1, \dots, pk_n$ are uniquely determined.

A third useful observation is that ciphertexts from honest dealings cannot be copied. In an honest dealing, the encrypted secret sharing defines a unique threshold t. And as argued above, it is also tied to $pk_1, \dots, pk_n$ and τ. So the ciphertext created by an honest dealer is tied to a unique configuration. Now, the Transcript oracle specifically checks that all dealings are distinct. This means an adversary can only succeed in copying the ciphertext within the same configuration by tweaking one of the zero-knowledge proofs. However, since they are simulation sound this is not possible for the adversary unless the adversary knows the full secret sharing that was encrypted.

Now, let $\mathcal{A}$ be a forging adversary that creates up to q_{Config} configuration requests of the form $(-,\ldots)$, i.e., for fresh configurations. With probability at least $1/q_{\mathsf{Config}}$ we may guess in advance a root index of a configuration tree with that particular verification key. If $\mathcal{A}$ succeeds in a forgery on the verification key for this particular index we keep it, otherwise we let the experiment fail. Let $\mathbf{Adv}(\mathcal{A})$ be the advantage in the original experiment, and $\mathbf{Adv}'(\mathcal{A})$ be the advantage in the modified experiment where we guess the index. Then $\mathbf{Adv}(\mathcal{A}) \leq q_{\mathsf{Config}} \cdot \mathbf{Adv}'(\mathcal{A})$. So at a tightness loss of q_{Config} we can from now on analyze the advantage $\mathbf{Adv}'(\mathcal{A})$ defined as

$$\Pr\left[\begin{array}{c} id_{\text{guess}} \leftarrow [0..q_{\mathsf{Config}}-1]; (vk, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{KGen,KUpd,Config,Deal,Transcript,Erase,SigShare,Corrupt}}: \\ \text{There are records } (vk, id_{\text{guess}}, *) \text{ and } (id_{\text{guess}}, -, *) \text{ and } \mathsf{SigVfy}(vk, m, \sigma) = \top \\ \text{and for all } id \text{ with a transcript record } (vk, id, t, pk_1, shvk_1, \ldots, pk_n, shvk_n, \tau) \\ \text{there is a configuration record } (id, \ldots, H) \text{ with } |H| > n-t \text{ and} \\ \# \left\{ i \;\middle|\; \begin{array}{c} \text{there is a corruption record } (C, pk_i, \tau') \text{ with } \tau' \leq \tau, \text{ or with } \tau' > \tau \\ \text{and an un-erased share-signing key record } (pk_i, id, sk, \tau), \\ \text{or } \; pk_i \notin Q_{pk}, \text{ or there is a signature share record } (id, pk_i, m) \end{array} \right\} < t \end{array}\right],$$

observing that due to the construction of the $\mathsf{Transcript}$ oracle the record $(vk, id', *)$ implies $vk \in Q_{vk}$ so that earlier condition is redundant and therefore omitted.

We start the analysis of the modified experiment by changing the way honest parties retrieve their shares from honest dealings. Recall, the Deal oracle runs an honest dealing $\mathsf{Deal}(sk, t, pk_1, \ldots, pk_n, \tau)$ to create a recorded $d \in Q_d$. This dealing builds on an honest secret sharing and chunked encryption with forward secrecy of the individual shares to the public keys $pk_1, \ldots, pk_n$. Dealing only happens if there is a prior registered configuration with those public keys, and the registration of a configuration implies all the public keys are valid. For each honestly generated public key $pk_i \in Q_{pk}$ the conditions for correctness of the encryption scheme are satisfied. It follows that an honest dealing that gets decrypted with an arbitrarily updated decryption key matching an honestly generated public key $pk_i \in Q_{pk}$ that for that public key you can decrypt correctly (because the chunks have the right size) and by doing so you get the secret key matching $s_i = \sum_{k=0}^{t-1} a_k i^k$ used in this particular dealing (by construction of the plaintexts that get chunked). We can therefore modify the experiment such that honest dealings created by Deal store their ciphertexts together with the matching plaintexts, which happen to be a secret sharing matching $A_0, \ldots, A_{t-1}$ in the dealing. And correspondingly, whenever $\mathsf{SKRetrieve}(i, dk_{\tau_i}, I, d_1, \ldots, d_\ell)$ inside $\mathsf{Transcript}$ encounters such a ciphertext addressed to an honest and uncorrupted public key $pk \in Q_{pk}$ it just looks up the matching plaintext instead of running the decryption algorithm. Perfect correctness ensures that this lookup always matches a decryption and therefore the advantage is unchanged.

Next, let us modify the dealing oracle Deal to simulate proofs for the ciphertext it creates in its honest dealings whenever it is referencing a node in the tree with root id_{guess}. The new dealing oracle is

```
$\mathsf{Deal}'(?pk, id)$
$\quad$Find the record $(id, id', t, pk_1, \ldots, pk_n, \tau, H)$ and if there is none return $\perp$
$\quad$If $pk = -$ let $sk = -$ and if $id' \neq -$ return $\perp$
$\quad$Else find the record $(pk, id', sk, \tau')$ and if there is none return $\perp$
$\quad$If $id$ is not in the subtree rooted in $id_{\text{guess}}$
        $d \leftarrow \mathsf{Deal}(sk, t, pk_1, \ldots, pk_n, \tau)$
$\quad$Else
        $d \leftarrow \mathsf{Deal}(sk, t, pk_1, \ldots, pk_n, \tau)$ with simulated proofs $\pi_{\text{share}}, \pi_{\text{chunk}}$
$\quad$Let $Q_d := Q_d \cup \{d\}$ and return $d$
```

It follows from the zero knowledge property of the proof systems that this change gives negligible difference in adversary's advantage.

Shifting focus to adversarial dealings, let us look at how the Transcript oracle deals with them on behalf of honest receivers. The Transcript oracle checks all dealings are valid, which means the adversarial dealings must have correctly formatted ciphertexts and accepting NIZK proofs for correct secret sharing and chunking. Now, let us modify the Transcript oracle to let the experiment return $\perp$ in case there is a ciphertext not created by the honest dealing oracle but with valid NIZK proofs, such that decryption for an honest $pk \in Q_{pk}$ fails or the resulting plaintext does not match what was expected given $A_0, \ldots, A_{t-1}$ in the dealing. Since the chunking proof implies our encryption scheme with forward secrecy has a correctly generated ciphertext and the encryption scheme has perfect correctness, failure to decrypt means we have violated soundness of the NIZK proof for chunking. Similarly, if the plaintext does not match a secret share as defined by $A_0, \ldots, A_{t-1}$, then we have violated soundness of the NIZK proof for correct secret sharing. It follows from simulation soundness that the risk of this happening is negligible and therefore the adversary's advantage remains almost the same.

As argued before the adversary cannot make copies of ciphertexts from honest dealings so we can think of share-signing key retrieval from an adversarial dealing as using a decryption oracle. All honest receivers can decrypt the adversarial dealings, and by the correct secret sharing proof their share matches the public share verification key for that receiver. By definition of the adversary's advantage $\mathbf{Adv}'(\mathcal{A})$ there can be at most t corrupt dealers and therefore are $n - t > t$ honest receivers. From the honest receivers' shares it is therefore possible to use Lagrange interpolation to reconstruct a degree $t - 1$ polynomial matching the shares underpinning the public share verification keys. Using the coefficients from the Lagrange interpolation used when combining the share verification keys, we now know a degree $t - 1$ polynomial that represents the contribution towards the shares from adversarially created dealings. We can use that to cancel out any contribution the adversary makes to key generation.

Now, let us model the usage of the hash function $H_{\mathbb{G}_1}(m)$ in the BLS signature scheme as a random oracle. We can see every query to the hash function as returning a group element g_1^{μ} with $\mu \leftarrow \mathbb{Z}_p$. If we program the random oracle to give us such values, we can think of μ as random known field elements. We can now change the way we share-sign messages on behalf of honest-locked parties. For a $pk \in H$ that must share sign a message, instead of using its secret share $s_i = a(i)$ to sign, we can think of the signature as using $g_1^{a(i)}$ and returning $(g_1^{a(i)})^\mu$. So instead of keeping track of the plaintexts for the honesty-locked parties, instead we can keep track of the exponentiation of their shares $g_1^{a(i)}$.

At this stage, the encryption scheme and the NIZK proofs no longer play a role. Neither do the adversarial dealings, since we can extract the full secret sharing and cancel it out. Moreover, we may without loss of generality assume there is a single honest dealing in each transcript, simply acting as an honest "adversary" in the other dealings. What remains is an adversary that in each transcript sees one honest dealing with group elements $A_0, \ldots, A_{t-1}$ and a subset of the secret shares $s_{i_1}, \ldots, s_{i_c}$ for corrupted or corruptible receivers. While for honest-locked parties the adversary only sees $g_1^{s_{j_1}}, \ldots, g_1^{s_{j_{n-c}}}$ but uses its control over the random oracle to create their signature shares. We will show that we can embed a one-more DH problem instance in this scenario such that a successful attacker forging signatures can be used to break the 1MDH instance. The 1MDHP setup in Type III pairings can be formulated as giving random group elements g_1^μ and $g_1^{\gamma_0}, g_2^{\gamma_0}, \ldots, g_1^{\gamma_q}, g_2^{\gamma_q}$ to the adversary. The attacker may now adaptively ask for linear combinations of the form $(r_0, \ldots, r_\ell) \in \mathbb{Z}_p^{\ell+1}$ receiving as response $g_1^{\mu \cdot \sum_{i=0}^{q} r_i \gamma_i}$. The attacker wins, if after q queries she is able to compute $g_1^{\mu \cdot \gamma_0}, \ldots, g_1^{\mu \cdot \gamma_q}$, i.e., she needs to compute one more Diffie-Hellman group element than the number of queries she is allowed to ask.

Ok, let us see how we can embed the 1MDH problem into our system so a successful attacker can be used to break the 1MDH problem. We guess in advance the index of the random oracle query the attacker will use to create a successful forgery, aborting the attack if we guess wrong at a bounded loss in tightness in the reduction. We then get the 1MDH challenge g_1^μ and $g_1^{\gamma_0}, g_2^{\gamma_0}, \ldots, g_1^{\gamma_q}, g_2^{\gamma_q}$ where q is an upper bound on the number of honesty-locked parties in honest dealings with the relevant configuration tree. Now, for the guessed configuration we pick $g_2^{\gamma_0}$ as the honest dealer's contribution to the verification key, which will be modified by the adversarial dealings to give $vk = g_2^{\gamma_0 + a_0}$ for some known a_0. In each honest dealing there will be some corruptible parties with indices $i_1, \ldots, i_c$, where we just picked random shares $s_{i_1}, \ldots, s_{i_c}$. Then after having chosen these, we will for the first $m = t - c - 1$ honesty-locked indices $j_1, \ldots, j_m$ pick previously unused elements $g_1^{\gamma_{k+1}}, \ldots, g_1^{\gamma_{k+m}}$ and define the shares of those parties in this dealing to be the unknown $s_{j_1} = \gamma_{k+1}, \ldots, s_{j_m} = \gamma_{k+m}$. From our 1MDH challenge we know the matching public key material $g_2^{\gamma_{k+1}}, \ldots, g_2^{\gamma_{k+m}}$). Now, for the forgery message m we program the oracle to return g^μ from the 1MDH challenge. We can use the 1MDH oracle to answer all share-signing queries for this message. For all other messages, we just program the random oracle to give us a random exponent, enabling us to sign all share-signing queries. And agin, we can easily cancel out the contributions from adversarial dealings. So we have now used the 1MDH oracle to get a perfect simulation of the protocol. At the end of the day, the adversary may now return a forged signature σ such that

$$e(\sigma, g_2) = e(H(M), vk) = e(g_1^{\mu}, g_2^{\gamma_0 + a_0}) = e(g_1^{\mu\cdot\gamma_0} \cdot g_1^{\mu\cdot a_0}, g_2).$$

Since a_0 is known, this means we can compute

$$\sigma \cdot (g_1^{\mu})^{-a_0} = g_1^{\mu\cdot\gamma_0}.$$

All share-signing queries are linearly independent of γ_0, so computing this value leads to a solution to the one-more DH instance. □

8.1 Proactive security.

With public keys remaining in use for a long time, it is reasonable to assume each share holder carries significant risk of compromise during this period, whether that be from a glitch or malware. If the secret shares are also long lived, they may leak one by one and if an adversary collects enough of them she can sign arbitrary messages. To counter this effect it is natural to put defenses in place such as monitoring the behavior of parties to detect deviation from normal behavior or periodic resets to flush out undetected malware. We model this kind of faulty or malicious behavior using a mobile adversary model, in which the adversary in any given epoch may compromise parties but cannot control too many parties at the same time. I.e., the adversary may compromise some parties in an epoch, but if she has already reached the corruption threshold, she must also relinquish control over some parties that then revert back to being honest. In the mobile adversary model, every single party may be compromised at some point in time, just not too many at the same time.

If we model a compromise as the adversary getting full control over a party, then there is not much hope for proactive security protecting against it. As soon as the adversary learns a decryption key, it can decrypt all future dealings where the party is a receiver. But we can use proactive security to protect against attacks such as accidental leakage of secret shares due to glitches, malicious behavior due to non-severe attacks that do not affect the decryption keys, or systems where long-term decryption keys are protected by secure hardware. To protect against such a mobile adversary, the share holders can use the non-interactive distributed key resharing protocol to periodically refresh their shares and delete their old shares. Once enough shares for a given epoch have been deleted, the remaining shares for that epoch will never suffice to reconstruct the secret key. This way we can reduce the attack surface against a mobile adversary to the duration of a single epoch.

Now, while we do not give a mobile attacker access to the decryption key, the malicious behavior could still involve obtaining decryptions of different ciphertexts; for instance even if the decryption key is protected by secure hardware presumably the mobile adversary may request the decryption of some ciphertexts. So we change the security model to be that the adversary may compromise a participant to learn a share-signing key and also gain access to a decryption oracle. Formally, we define the advantage of a forging adversary $\mathcal{A}$ with access to a decryption oracle as

$$\Pr\left[\begin{array}{c} (vk, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{KGen},\mathsf{KUpd},\mathsf{Config},\mathsf{Deal},\mathsf{Transcript},\mathsf{Erase},\mathsf{SigShare},\mathsf{Dec},\mathsf{Corrupt}} : \\ vk \in Q_{vk} \text{ and } \mathsf{SigVfy}(vk, m, \sigma) = \top \\ \text{and for all } id \text{ with a transcript record } (vk, id, t, pk_1, shvk_1, \ldots, pk_n, shvk_n, \tau) : \\ \# \left\{ i \;\middle|\; \begin{array}{c} \text{there is a corruption record } (C, pk_i, \tau') \text{ with } \tau' \leq \tau, \text{ or with } \tau' > \tau \\ \text{and an un-erased share-signing key record } (pk_i, id, sk, \tau), \\ \text{or } \; pk_i \notin Q_{pk}, \text{ or there is a signature share record } (id, pk_i, m) \end{array} \right\} < \; t \end{array}\right].$$

The security proof can be adapted to cover this stronger form of security. Therefore, we get proactive security against a mobile adversary with CCA-access to the encryption scheme who may also able to learn some share-signing keys for a given epoch.

Acknowledgment

Many people have guided design choices, suggested improvements to the article and worked on implementation of the protocols on the Internet Computer, we would especially like to thank Jan Camenisch, Andrea Cerulli, David Derler, Manu Drijvers, Maria Dubovitskaya, Ben Lynn, Maximillian Murphy, Gregory Neven, Franz-Stefan Preiss, Bartosz Przydatek, Victor Shoup and Björn Tackmann.

References

BBG05. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

BCHK07. Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.

BLS04. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

Fel87. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437. IEEE Computer Society, 1987.

GPS08. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discret. Appl. Math.*, 156(16):3113–3121, 2008.

Gro02. Jens Groth. Evaluating security of voting schemes in the universal composability framework. Cryptology ePrint Archive, Report 2002/002, 2002. `https://eprint.iacr.org/2002/002`.

Gro05. Jens Groth. A verifiable secret shuffle of homomorphic encryptions. Cryptology ePrint Archive, Report 2005/246, 2005.

Lyu09. Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer, 2009.

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