Flood and Submerse: Distributed Key Generation and Robust Threshold Signature from Lattices

Thomas Espitau[1], Guilhem Niot[1,2], and Thomas Prest[1]

[1] PQShield
thomas@espitau.com, guilhem@gniot.fr, thomas.prest@pqshield.com
[2] Univ Rennes, CNRS, IRISA

Abstract. We propose a new framework based on *random submersions*—that is projection over a random subspace blinded by a small Gaussian noise—for constructing verifiable short secret sharing and showcase it to construct efficient threshold lattice-based signatures in the hash-and-sign paradigm, when based on *noise flooding*. This is, to our knowledge, the first hash-and-sign lattice-based threshold signature.
Our threshold signature enjoys the very desirable property of *robustness*, including at key generation. In practice, we are able to construct a robust hash-and-sign threshold signature for threshold and provide a typical parameter set for threshold $T = 16$ and signature size 13kB. Our constructions are provably secure under standard MLWE assumption in the ROM and only require basic primitives as building blocks. In particular, we do not rely on FHE-type schemes.

This is the full version of a paper appearing in the proceedings of the 44th Annual International Cryptology Conference (CRYPTO 2024). This version includes additional security proofs that were omitted in the proceedings version.

Table of Contents

1 Introduction and State-of-the-Art . . . 3
- 1.1 Post-Quantum Surge and Multiparty Protocols . . . 3
- 1.2 Distributed Key Generation and Verifiable Secret Sharing . . . 3
- 1.3 Our Proposal: Full-Fledged (publicly) V3S, synchronous DKG and Threshold Signatures from MLWE and MSIS in the ROM . . . 4

2 Technical Overview . . . 6
- 2.1 A lattice verifiable short secret sharing proposal . . . 6
- 2.2 A Proposal for Robust Secret Sharing and Robust DKG . . . 8
- 2.3 Robust Threshold Lattice-Based Signature . . . 10
- 2.4 Some open problems and directions . . . 12

3 Preliminaries . . . 12
- 3.1 Notations . . . 12
- 3.2 Distributions . . . 12
- 3.3 Shamir Secret Sharing over Modules . . . 13
- 3.4 Hardness Assumptions . . . 14
- 3.5 Generalization of Hint-MLWE . . . 14

4 Verifiable Short Secret Sharing . . . 16
- 4.1 Security notions . . . 16
- 4.2 Formal definition of our VSSS . . . 17
- 4.3 Towards Applications . . . 22

5 Pelican: A Robust Threshold Signature Scheme . . . 23
- 5.1 Robust Distributed Key Generation (DKG) . . . 24
- 5.2 Robust Distributed Signing Procedure . . . 26

6 Parameter Selection and Instantiation . . . 29
- 6.1 Reminder: Parameter Selection in Plover . . . 30
- 6.2 Parameter Selection for the Signing Procedure of Pelican . . . 30
- 6.3 Distributed Key Generation . . . 33
- 6.4 Selected parameter sets . . . 33

A Proof of security of the V3S . . . 36

B Security proof of Pelican . . . 41
- B.1 Robustness . . . 42
- B.2 Unforgeability . . . 46

C Security proof of Pelican with DKG . . . 50
- C.1 Robustness . . . 50
- C.2 Unforgeability . . . 51
- C.3 A note on the distributed generation of the salt . . . 54

D Proof of MatrixHint-MLWE . . . 54

E Distribution of submersion matrices $\mathbf{R}$ . . . 55

1 Introduction and State-of-the-Art

1.1 Post-Quantum Surge and Multiparty Protocols

The field of post-quantum cryptography has experienced significant growth, underscored by numerous standardization efforts over the past decade. Lattice-based cryptosystems have demonstrated their versatility and efficiency. A key area of development is threshold cryptography, which enables secure collaborative generation of information (e.g., secrets, signatures, encryptions). For example, a T-out-of-N threshold signature scheme [Des90,DF90] allows any group of $T \leqslant N$ participants to jointly sign a message, ensuring that groups with strictly fewer than T members cannot produce a valid signature. The security of these protocols is nuanced, requiring mechanisms to address potential misbehavior of parties—including those acting dishonestly or those who are *honest but curious* [PMB], merely observing the protocol's execution. These aspects are often referred to as *robustness* properties. While extensively studied in the classical context, the exploration of post-quantum solutions for these issues is relatively recent. Notably, NIST is set to announce a call for proposals on multiparty threshold schemes [BP23], with submissions expected by the first half of 2025.

1.2 Distributed Key Generation and Verifiable Secret Sharing

Classical DKGs... A core concept in threshold cryptography is *distributed key generation* (DKG). This protocol allows n participants to collaboratively generate a public key while distributing its corresponding secret key among them. Once completed, this setup enables a subset of T or more members to perform operations requiring the secret key, such as decryption or signing, without a central trusted authority.

... from VSS. The subproblem of *secret sharing* extends beyond mere secret distribution to include robustness through verification, allowing any party to verify the secret. This led to the development of *publicly verifiable secret sharing schemes* (pVSS), introduced by Stadler [Sta96], although the initial VSS concept by Chor et al.[CGMA85] already incorporated verifiability. Several VSS have been proposed over the years [BGW88,Ped92,ABCP23,KGS23], see [ABCP23,KGS23] for excellent comparative introductions. These VSS are easily instantiated in the discrete logarithm setting.

Threshold Signatures, Old and New. Robust threshold protocols based on RSA [GRJK00] and DSS [GJKR96] have been proposed, with most robustness-oriented works employing a reliable broadcast model. More recently, ROAST [RRJ$^+$22] build over the the FROST [KG20] proposal and offers a generic framework for enhancing semi-interactive threshold signatures with robustness[3].

[3] It requires only a semi-trusted coordinator instead of broadcast channels but necessitating $2(|S| - T) + 3$ asynchronous rounds.

Once again with lattices, achieving practical, let alone robust, post-quantum threshold signatures has been challenging. [BKP13] laid a theoretical foundation for robust threshold signatures within the GPV framework [GPV08]. In a higher-level view, by relying on the universal thresholdizer of [BGG$^+$18], Agrawal et al. [ASY22] introduced a two-round threshold signature based on threshold fully homomorphic encryption (FHE), achieving robustness through homomorphic signatures. This incurs a high cost. Following this initial attempt, Gur et al. [GKS23] optimized this approach and removed homomorphic signatures, but also lost robustness in the process. They obtain a signature size of 11.5kB and a communication cost per party of 1.5MB for signing. [JTZ23] proposed a robust threshold signature scheme based on Fiat-Shamir with aborts, however requiring the participation of all non-corrupted users.

Analogously to the classical setting, it is expected that a VSS with shortness proof such as [GHL22] could be used for concretizing a post-quantum DKG and robust threshold signature scheme. However, adding to the complexity of using zero-knowledge proofs, their current instantiation relies on discrete-log assumptions, and moving it to post-quantum secure assumptions is deemed impractical.

When it comes to constructing post-quantum threshold signature schemes *without robustness*, several works [TPCZ23,PKM$^+$24] managed to have quite efficient instantiations. But, achieving robustness via their mechanisms is deemed highly nontrivial. In short, it seems that we can get *efficient* threshold from lattices at the cost of loosing robustness, or we can get *robust* threshold at the cost of relying on highly inefficient primitives. As such, remains the following interrogation:

Can we reconcile efficiency and robustness in the post-quantum era, using lattices?

1.3 Our Proposal: Full-Fledged (publicly) V3S, synchronous DKG and Threshold Signatures from MLWE and MSIS in the ROM

We study and propose a solution for robust (short) secret sharing, turn it into a robust distributed key generation (DKG), and as a byproduct get a lattice-based threshold signature using noise flooding. Before further presentation, we shed light on that all our primitives are underpinned by *standard lattice assumptions* and do not require advanced additional primitives such as (threshold) FHE.

Lattice-Based Verifiable (Short) Secret Sharing Scheme: We propose a "random submersion" technique for secret sharing, enabling the secure generation and distribution of Learning With Errors (LWE) samples. This approach provides a form of zero-knowledge proof to affirm the shortness of these samples, a critical aspect in maintaining both the integrity and confidentiality of secrets in a distributed environment. Our proposal is inspired by [ABCP23] for its syntax and challenge sampling, and [GHL22] for its compactness proof, employing Johnson-Lindenstrauss-type projections. We adopt a pragmatic approach for our VSS and prove approximate shortness only, with a small loss factor in the norm proven of about 10. This small loss allows us to improve over the practicality of [GHL22] by removing complex zero-knowledge proofs – which are computationally intensive, and large in size – and the use of classical hardness assumptions in their instantiation.

Robust Distributed Key Generation: Building upon our V3S, we build a *robust* DKG. The protocol is *three-round*, *synchronous*, and uses a general complaint round to reach a consensus on the trustable parties.

More precisely, we propose a broadcast protocol where each party performs its own verifiable secret sharing via the V3S protocol highlighted above. Next, each party broadcasts its list of trustees, and the (public) intersection of these lists is computed, effectively forming a public clique of mutually trusted parties. Finally, each party aggregates the shares they received from trusted parties, which yields a secret sharing of a jointly generated short secret key, thanks to the linearity of Shamir's secret sharing.

While this solution has a nonnegligible communication cost (as the proofs are sent to every party by every party, it ensures maximal robustness for a minimal number of rounds.

Robust Threshold Signatures: We can adapt this methodology, almost *verbatim*, to construct threshold signatures. This follows from the fact that in the underlying signature scheme[4], the signing algorithm is essentially the key generation procedure with extra steps and slightly different parameters. As a consequence, the robustness of such signatures follows from similar arguments to the ones used for the DKG protocol. Our approach relies on the *noise-flooding* technique to create signatures; here the *flooding* involves linearly concealing secret values with sufficiently large noise and ensuring security by quantifying the residual statistical leakage. Due to its linearity, this method has proven to be very effective in masking-friendly [dEK+23,EEN+24] and threshold-friendly signatures [PKM+24,ASY22]. In our case, it is also very amenable to Shamir's secret sharing.

We instantiate our framework in Pelican – a threshold signature scheme system featuring a 3-round distributed key generation and a 4-round signing protocol.[5] Pelican boasts a signature size of 12.3kB and a communication overhead of $59.8 + 19N$kB per participant for a threshold $T = 16$ among N parties. It is noteworthy that, while our framework solely depends on LWE in the ROM, it is sufficiently versatile to accommodate thresholding for both the Fiat-Shamir and Hash-And-Sign paradigms.

While Pelican builds upon the digital signature scheme Plover, we emphasize that our robust threshold signature and distributed key generation constructions

[4] In our case this scheme is the hash-then-sign scheme Plover, but the same comment would apply to the Fiat-Shamir signature Raccoon.

[5] We continue the "tradition" of naming lattice-based hash-then-sign schemes after birds: Falcon, Eagle, Robin, Plover, etc. Since our scheme relies on random submersions and noise flooding, we named it after the aquatic, diving species *Pelican*.

are generic and could for instance be applied to the NIST submission Raccoon [dEK+23]. We leave this as future work.

Before training to the details and security proofs of all these protocols, we propose a high-level overview of the main technicalities, caveats, and ideas used further.

2 Technical Overview

We now turn to a high-level introduction of our techniques and protocols. Our first contribution is a proposal for a lattice-based verifiable short secret sharing scheme, for which we can control its leakage very precisely, but which is not technically zero-knowledge. From this essential building framework, we show how to extend it to devise a protocol for robust distributed key generation and robust threshold signatures. All of these are secure under *standard* lattice assumptions and *does not* require more advanced primitives such as FHE.

2.1 A lattice verifiable short secret sharing proposal

Many lattice-based cryptographic schemes hinge on variants of the Learning With Errors (LWE) assumption, which is crucially based on the shortness of some secret elements. In particular, in the context of devising a threshold scheme, the verification of this shortness is critical. Addressing this challenge, we introduce a new technique coined *random submersion*. This method enables the secure generation and distribution of an LWE sample, concurrently providing a zero-knowledge proof to confirm the shortness of the sample. It ensures both the integrity and the confidentiality of the secret and we will leverage the technique to ensure the robustness of our threshold scheme in Section 5.2. Our technique aims for practicality, and only provides approximate shortness bounds in exchange for a lower communication and computational cost than prior work [GHL22]. *Random submersion* relies on a Johnson-Lindenstrauss type lemma, blinded by a Gaussian noise for confidentiality. This type of lemma was already successfully applied for verifiable secret sharing with approximate shortness – relying on rejection sampling – in [GHL22, Section 3.4], but differs in the distribution of its blindings, which removes the need for rejection sampling and provides tighter shortness proofs.

We present a tweakable framework for securely distributing a small secret vector among N parties, with the provision that a maximum of $T < N/2$ among these parties may be untrustworthy. Our approach revolves on the sharing of secrets that are derived from a Gaussian distribution and also accommodates secret vectors chosen by adversaries, provided these vectors have a norm capped at B. We name our secret sharing technique $\mathsf{V3S}$ (Verifiable Short Secret Sharing).

Methodology. The cornerstone of our method is to construct a scheme *as linear as feasible*, to ensure compatibility with lattice-based operations. Our initial step involves defining the most fundamental requirements for our scheme, starting with the secret itself.

1. The secret is represented as a compact vector, which is then distributed linearly into a series of random vectors.
2. The verification of the secret's smallness is achieved through a random linear transformation applied to itself.
3. To prevent excessive leakage of the secret by the output of this transformation, we blind it by incorporating an additive mask within its image space.
4. The choice of the randomness used to construct the transformation shall be verifiable by the parties.

Writing what precedes as an algorithm would yield a blueprint of the following shape, using a Merkle tree to handle the randomness verifiability:

1. The dealer samples an ephemeral Gaussian blinding value $\mathbf{y}$, that will be used within their individual proof to prove the shortness of $\mathbf{x}$ without leaking its value.
2. Then they generate a linear random secret sharing for $\mathbf{x}$ and $\mathbf{y}$ of order T, noted $[\![\mathbf{x}]\!]$, and $[\![\mathbf{y}]\!]$.
3. To generate verifiable randomness, the dealer hashes the shares $[\![\mathbf{x}]\!]_{i\in\{1,\ldots,N\}}$ and $[\![\mathbf{y}]\!]_{i\in\{1,\ldots,N\}}$ into a Merkle tree of hash h. It also produces individual proofs proof_i allowing each party to verify that $[\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i$ belongs to the tree.
4. It derives a challenge matrix $\mathbf{R} = H(h)$ from a suitable distribution, and computes the value $[\![\mathbf{v}]\!] = \mathbf{R} \cdot [\![\mathbf{x}]\!] + [\![\mathbf{y}]\!]$.
5. The broadcast proof π is $(h, [\![\mathbf{v}]\!])$. Individual proofs are $\pi_i = ([\![\mathbf{y}]\!]_i, \mathsf{proof}_i)$.

Any party i, given its share $[\![\mathbf{x}]\!]_i$ and proofs $\pi = (h, [\![\mathbf{v}]\!])$ and $\pi_i = ([\![\mathbf{y}]\!]_i, \mathsf{proof}_i)$ performs the following verifications:

1. proof_i correctly proves that $[\![\mathbf{x}]\!]_i$ and $[\![\mathbf{y}]\!]_i$ are included in Merkle tree h.
2. Derive $\mathbf{R} = H(h)$ and verify that $[\![\mathbf{v}]\!]_i = \mathbf{R} \cdot [\![\mathbf{x}]\!]_i + [\![\mathbf{y}]\!]_i$.
3. Verify that $\|\mathbf{v}\|$ is smaller than some fixed bound B'.

The reconstruction hinges here on the fact that we can see Shamir's secret sharing as from a Reed-Solomon code, which allows to recover $\mathbf{b}$ even if a set of users behaved dishonestly by the robustness of error correction.

Guaranteeing soundness and correctness. While the verification of the randomness is classical thanks to the Merkle tree, the crux of the shortness proof will lie in the following requirements:

small secret implies small proof — The proof $\mathbf{R} \cdot \mathbf{x} + \mathbf{y}$ must be small—with overwhelming probability—if both $\mathbf{x}$ and $\mathbf{y}$ are small, for instance when Gaussian is drawn.

big secret implies big proof — It must become large if $\|\mathbf{x}\|$ is compromised, that is when it is larger than the desired acceptance bound B.

few collisions — Obviously, for correctness, we can not tolerate too many collisions in the values, so we should not have too many pairs $(\mathbf{x}, \mathbf{y})$ being sent to the same value $\mathbf{R}\mathbf{x} + \mathbf{y} \bmod q$.

This means that from a geometric perspective, we are asking the matrix $\mathbf{R}$ to act as a random *pseudo-isometry* in the mod$q$ hypercube. This is very reminiscent of the Johnson-Lindenstrauss type lemma [JLS86], which has been successfully applied to the modular setting in cryptography in [GHL22]. In the following, we say that such distributions satisfy property G (see Definition 8 for a formal definition).

Adding zero-knowledge. Revealing the pair $(\mathbf{R}, \mathbf{R} \cdot \mathbf{x} + \mathbf{y})$ *leaks* some statistical information on $\mathbf{x}$. However, when the dealer is honest, $\mathbf{y}$ is sampled from a Gaussian distribution of sufficiently large variance with regards to $\|\mathbf{R} \cdot \mathbf{x}\|$, so that this information leakage is very mitigated. More precisely, we can even estimate it when generating an MLWE sample from $\mathbf{x}$: after the secret sharing, parties can additionally sample a matrix $\mathbf{A} \in \mathcal{R}^{k \times \ell}$, and a MLWE sample $\mathbf{A} \cdot \mathbf{x}$. From there we get an instance of the recent "MLWE problem with hint" $\mathbf{R} \cdot \mathbf{x} + \mathbf{y}$, which in turn reduces to an MLWE with a smaller standard deviation.

2.2 A Proposal for Robust Secret Sharing and Robust DKG

From $\mathsf{V3S}$ to Distributed Secret Sharing. Consider a scenario where N users aim to collectively generate a secret and each obtains a share of a short vector $\mathbf{s}$ as a secret share. Each user $i \in \{1, \ldots, N\}$ will possess $[\![\mathbf{s}]\!]_i = \boldsymbol{P}(i)$, where $\boldsymbol{P}$ is an interpolation polynomial such that $\boldsymbol{P}(0) = s$. To achieve this, we require each participant to generate their own share of the global secret and transmit it to all other participants. However, for reasons of robustness and security, direct broadcast of their share is inadvisable, as it would allow any eavesdropper to learn all the shares and, by extension, the secret. Instead, each participant divides their share into smaller shares (termed local shares) and distributes these local shares to others. To ensure each local share between parties i and j remains confidential to them, we assume the existence of symmetric encryption between each pair of parties.

This method alone does not guarantee robustness, nor does it prevent potential dishonesty in the generation of local shares. To address this, our approach involves using a $\mathsf{V3S}$ scheme to allow parties to verify the local shares they receive. If a proof of a local share fails – say, if party i detects an incorrect proof from party j – then i broadcasts a complaint against j along with a proof of the error, enabling *all parties* to verify the incorrectness of j's proof to i and exclude j from their list of trusted parties. After addressing these complaints, every honest party knows the other parties that shared a secret honestly, and the correct local shares, allowing them to aggregate the local shares they have received to form their share of the secret. The protocol proceeds as follows:

(V1) **Round 1 (Secret-Share Individual Secret).** Each party i:

a. Generates a short vector $\mathbf{s}_i$.
b. Utilizes the V3S *as dealer* to secret-share $\mathbf{s}_i$ and generate a distributed proof of shortness, denoting $[\![\mathbf{s}_i]\!]$ as the secret-sharing and $\pi_{i\to j}$ as the partial proof for j.
c. For each j, encrypts $[\![\mathbf{s}_i]\!]_j, \pi_{i\to j}$ under a symmetric key known only to i and j.
d. Broadcasts the encrypted proofs and shares.

(V2) **Round 2 (Verify and Complain).** Each party i:
a. Decrypts all secret shares $[\![\mathbf{s}_j]\!]_i$ and partial proofs $\pi_{j\to i}$ sent by others.
b. Only keep secret shares with valid partial V3S proofs. We note valid_i the set of participants j with valid proofs, and $\mathsf{complaints}_i$ as those without.
c. Broadcasts $\mathsf{complaints}_i$ along with proof of incorrectness for the partial V3S proofs.

(V3) **Round 3 (Review Complaints).** Each party i:
a. Receives $\mathsf{complaints}_j$ from user j, including "proof of incorrectness".
b. Removes any user from valid_i if proven invalid by j's complaint.

(V4) **Round 4 (Aggregate Valid Secrets).** Each party i, now with a confirmed list valid_i of valid users:
a. Aggregates the valid secret shares received: $[\![\mathbf{s}]\!]_i = \sum_{j\in\mathsf{valid}_i}[\![\mathbf{s}_j]\!]_i$, thus forming a secret sharing of $\mathbf{s} = \sum_{j\in\mathsf{valid}_i}\mathbf{s}_j$.

When the protocol ends, when a sufficient number of honest users participate, each honest user i holds a share $[\![\mathbf{s}]\!]_i$ of the aggregated secret $\mathbf{s} = \sum_{j\in\mathsf{valid}_i}\mathbf{s}_j$.

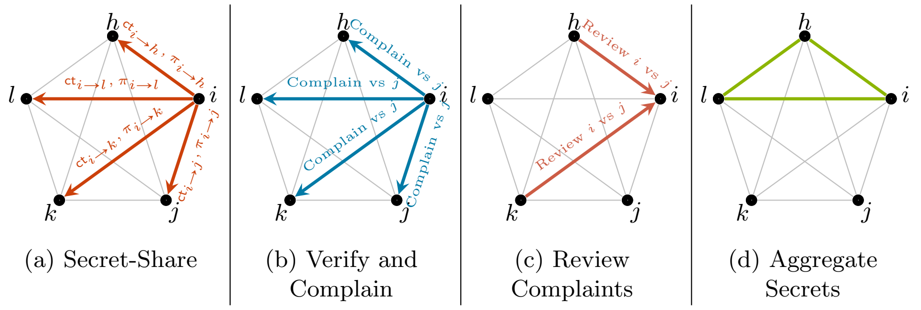
h
l
i
k
j
$\mathsf{ct}_{i\to h}, \pi_{i\to h}$
$\mathsf{ct}_{i\to l}, \pi_{i\to l}$
$\mathsf{ct}_{i\to k}, \pi_{i\to k}$
$\mathsf{ct}_{i\to j}, \pi_{i\to j}$
(a) Secret-Share
Complain vs j
(b) Verify and Complain
Review i vs j
(c) Review Complaints
(d) Aggregate Secrets

Fig. 1: Our blueprint for secret-sharing a jointly generated short secret $\mathbf{x} = \sum_{i\in\mathsf{honest}}\mathbf{x}_i$. This structured underlies the distributed signing (Figs. 6 and 7) and key generation (Fig. 5) protocols of Pelican.

Turning a V3S into a Robust DKG. Leveraging our robust secret sharing protocol, we can readily derive a robust distributed key generation (DKG) protocol. After all, a secret is essentially comprised of the secret itself and a bit of salt. Our protocol is robust and secure as long as $T \leqslant N/3$ for the final reconstruction.

(K1) **Round 1.** Each participant i undertakes the following:
  a. Generate, secret-share, prove, and encrypt the shares and partial proofs of a short secret vector $\mathbf{s}_i$, following the procedure outlined in (V1).
  b. Concurrently, generate an individual salt salt_i and broadcast it.

(K2) **Round 2.** Participant i continues by:
  a. Decrypting and verifying the shares+proofs received, and broadcasting complaints against users who submitted invalid shares/proofs, mirroring the steps in (V2).

(K3) **Round 3.** Participant i proceeds to:
  a. Review the complaints issued by all users, update valid_i accordingly, and compute an aggregate secret share $[\![\mathbf{s}]\!]_i = \sum_{j \in \mathsf{valid}_i} [\![\mathbf{s}_j]\!]_i$, akin to the method described in (V3).
  b. Produce a public salt salt by hashing the salts of all users in valid_i.
  c. Utilize salt to generate a public matrix $\mathbf{A} = \left[\bar{\mathbf{A}}\ \mathbf{I}\right]$, which is identical for all honest users. Calculate a partial public key $[\![\mathbf{b}]\!]_i = \mathbf{A} \cdot [\![\mathbf{s}]\!]_i$.
  d. Broadcast the salt salt and the partial public key $[\![\mathbf{b}]\!]_i$.

(K4) **Round 4.** Participant i:
  a. Retrieves the salt from the previous round, and in case of conflicting contributions, employ majority voting.
  b. Reconstructs the secrets using the $\mathsf{V3S}$ for a robust reconstruction.

Upon completion, provided a sufficient number of honest users participate, the public key $\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s}$ can be recovered using the error-correcting features of Reed-Solomon codes/Shamir's secret sharing. Each honest participant i retains a share $[\![\mathbf{s}]\!]_i$ of $\mathbf{s}$. Our DKG is formally described and proven in Section 5.1.

2.3 Robust Threshold Lattice-Based Signature

Another application of our robust distributed secret-sharing protocol is threshold signing, wherein the secrets being shared and combined are the shares of the signature itself. Utilizing our technique, we can robustly adapt both signatures in the Hash-and-Sign paradigm and the Fiat-Shamir paradigm. For example, we can develop robust variants of both recent proposals Raccoon and Plover. In this section, we focus on presenting a threshold Hash-and-Sign method based on Plover, marking the first instance of a *robust hash and sign threshold signature* grounded in standard lattice assumptions. We emphasize again the adaptability of our techniques.

Noise-Flooded Hash-and-Sign in a Nutshell. Before delving further into our design, we briefly revisit the concept behind the masking-friendly hash and sign signature Plover. The scheme's core idea is straightforward: within the GPV-like framework, it obviates the need to produce a signature that is zero-knowledge concerning the secret key by employing sufficiently large noise to independently conceal the secret. This approach involves replacing the traditional choice of Gaussian distribution, which relies solely on the (public) lattice and not on the short secret key, with a substantial sum of uniform noise. The leak is mitigated through the Hint-RLWE problem, which encapsulates this scenario: a public RLWE sample accompanied by a leak.

Key Generation. The process initiates by sampling a public polynomial a, derived from a seed. The second component of the public key is essentially an RLWE sample b offset by β, an integer power of two.

Signing Procedure. The procedure begins by hashing the provided message msg to a target polynomial u. It then employs its trapdoor to locate a short pre-image $\mathbf{z} = (z_1, z_2, c_1)$ such that $\mathbf{A} \cdot \mathbf{z} \equiv z_1 + a \cdot z_2 + b \cdot c_1 = u - c_2 \bmod q$ for a short c_2 and $\mathbf{A} \equiv \begin{bmatrix} 1 & a & b \end{bmatrix}$. To prevent the trapdoor's disclosure, a noise vector $\mathbf{p}$ is sampled and added to the pre-image $\mathbf{z}$. The actual signature is z_2, as $z_1 + c_2 = u - a \cdot z_2 - b \cdot c_1$ is recoverable in the verification process.

Verification. The verification commences with the recovery of $z_1' := u - a \cdot z_2 - b \cdot c_1$ (equivalent to $z_1 + c_2$), followed by verifying the shortness of (z_1', z_2). This concise description lays the algorithmic groundwork for our methodology.

Towards $\mathsf{Pelican}$: A Robust Threshold Hash-and-Sign. Now presenting our third application of the technique, we opt for a more condensed exposition. The principal idea remains unchanged from the distributed key generation: signers are tasked with generating and locally sharing their portion of what will become the perturbation vector $\mathbf{p}$ – akin to the noise from Plover, employing the $\mathsf{V3S}$. Upon reaching a consensus on the perturbation share (via a four-round protocol that identifies dishonest participants through a complaint round), each portion of the final signature is derived from the perturbation share, given the linear nature of the Plover signing operations. We propose the following protocol:

(S1) **Round 1.** Each user i performs the following:
  a. Generate, secret-share, prove, and encrypt the shares and partial proofs of a short perturbation vector $\mathbf{p}_i$, akin to (V1).

(S2) **Round 2.** Each user i performs the following:
  a. Decrypt and verify the shares and partial proofs received, broadcasting complaints against users who sent invalid shares/proofs, as in (V2). Complaints are also raised if user i fails to receive another user's share.

(S3) **Round 3.** Each user i performs the following:
  a. Review complaints from all users, adjust valid_i accordingly, and compute an aggregate secret share $[\![\mathbf{p}]\!]_i = \sum_{j \in \mathsf{valid}_i} [\![\mathbf{p}_j]\!]_i$, as in (V3).
  b. Compute a partial commitment $[\![w]\!]_i = \mathbf{A} \cdot [\![\mathbf{p}]\!]_i$.
  c. Broadcast valid_i and $[\![w]\!]_i$.

(S4) **Round 4.** Each user i reconstructs $w = \mathbf{A} \cdot \mathbf{p}$ from the shares $[\![w]\!]_i$, derives a salt from w which acts as a source of randomness, and concludes the signature as in the non-threshold scenario. The sole distinction lies in the necessity to calculate shares of $\mathbf{z}$ using the shares $[\![\mathbf{s}]\!]_i$ and $[\![\mathbf{p}]\!]_i$.

At the protocol's conclusion, if a sufficient number of honest users are present, a valid vector $\mathbf{z} = \mathbf{p} + c \cdot \mathbf{s}$ can be assembled using the error-correcting properties of Reed-Solomon codes. Consequently, $\mathsf{sig} = (\mathsf{salt}, c, \mathbf{z})$ constitutes a valid signature for the message msg under a Plover public key.

2.4 Some open problems and directions

The protocol we constructed is synchronous as we must wait for each round to be completed, in particular for the complaint round. It is in particular quite easy to perform a forking-like attack web we do not have all complaints.

3 Preliminaries

3.1 Notations

Sets, functions, and distributions. For an integer $N > 0$, we note $[N] = \{0, \ldots, N-1\}$. To denote the *assign* operation, we use $y := f(x)$ when f is deterministic and $y \leftarrow f(x)$ when randomized. When S is a finite set, we note $\mathcal{U}(S)$ the uniform distribution over S, and shorthand $x \overset{\$}{\leftarrow} S$ for $x \leftarrow \mathcal{U}(S)$.

Linear algebra. Throughout the work, for a fixed power-of-two n, we note $\mathcal{K} = \mathbb{Q}[x]/(x^n+1)$ and $\mathcal{R} = \mathbb{Z}[x]/(x^n+1)$ the associated cyclotomic field and cyclotomic ring. We also note $\mathcal{R}_q = \mathcal{R}/(q\mathcal{R})$. Given $\mathbf{x} \in \mathcal{K}^\ell$, we abusively note $\|\mathbf{x}\|$ the Euclidean norm of the $(n\,\ell)$-dimensional vector of the coefficients of $\mathbf{x}$. By default, vectors are treated as *column* vectors (unless specified otherwise).

Rounding. Let $\beta \in \mathbb{N}, \beta \geqslant 2$ be a power-of-two. Any integer $x \in \mathbb{Z}$ can be decomposed uniquely as $x = \beta \cdot x_1 + x_2$, where $x_2 \in \{-\beta/2, \ldots, \beta/2 - 1\}$. In this case, $|x_1| \leqslant \left\lceil \frac{x}{\beta} \right\rceil$, where $\lceil \cdot \rfloor$ denote rounding up to the nearest integer. For odd q, we note $\mathsf{Decompose}_\beta : \mathbb{Z}_q \to \mathbb{Z} \times \mathbb{Z}$ the function which takes as input $x \in \mathbb{Z}_q$, takes its unique representative in $\bar{x} \in \{-(q-1)/2, \ldots, (q-1)/2\}$, and decomposes $\bar{x} = \beta \cdot x_1 + x_2$ as described above and outputs (x_1, x_2). We extend $\mathsf{Decompose}_\beta$ to polynomials in $\mathbb{Z}_q[x]$, by applying the function to each of its coefficients. For $c \overset{\$}{\leftarrow} \mathbb{Z}_q$ and $(c_1, c_2) := \mathsf{Decompose}_\beta(c)$, we have $|c_1| \leqslant \left\lceil \frac{q-1}{2\beta} \right\rceil$, $\mathbb{E}[c_1] = 0$ and $\mathbb{E}[c_1^2] \leqslant \frac{M^2-1}{12}$ for $M = 2\left\lceil \frac{q-1}{2\beta} \right\rceil + 1$.

Polynomials. Given a finite commutative ring R, we note $R[x]$ the set of univariate polynomials over R. We also note $R_{<T}[x]$ the subset of polynomials of degree $< T$.

3.2 Distributions

Definition 1 (Discrete Gaussians). *Given a positive definite $\Sigma \in \mathbb{R}^{m \times m}$, we note $\rho_{\sqrt{\Sigma}}$ the Gaussian function defined over $\mathbb{R}^m$ as $\rho_{\sqrt{\Sigma}}(\mathbf{x}) = \exp\left(-\frac{\mathbf{x}^t \cdot \Sigma^{-1} \cdot \mathbf{x}}{2}\right)$.*

The above definition may be extended to $(\Sigma, \mathbf{x}) \in \mathcal{K}^{m \times m} \times \mathcal{K}^m$ by treating Σ as a block-circulant matrix in $\mathbb{R}^{nm \times nm}$ and $\mathbf{x}$ as a vector in $\mathbb{R}^{nm}$.

We may note $\rho_{\sqrt{\Sigma},\mathbf{c}}(\mathbf{x}) = \rho_{\sqrt{\Sigma}}(\mathbf{x} - \mathbf{c})$. When Σ is of the form $\sigma \cdot \mathbf{I}_m$, where $\sigma \in \mathcal{K}^{++}$ and $\mathbf{I}_m$ is the identity matrix, we note $\rho_{\sigma,\mathbf{c}}$ as shorthand for $\rho_{\sqrt{\Sigma},\mathbf{c}}$.

For any countable set $S \subset \mathcal{K}^m$, we note $\rho_{\sqrt{\Sigma},\mathbf{c}}(S) = \sum_{\mathbf{x}\in\mathcal{K}^m} \rho_{\sqrt{\Sigma},\mathbf{c}}(\mathbf{x})$ whenever this sum converges. Finally, when $\rho_{\sqrt{\Sigma},\mathbf{c}}(S)$ converges, the discrete Gaussian distribution $D_{S,\mathbf{c},\sqrt{\Sigma}}$ is defined over S by its probability distribution function:

$$D_{S,\sqrt{\Sigma},\mathbf{c}}(\mathbf{x}) = \frac{\rho_{\sqrt{\Sigma},\mathbf{c}}(\mathbf{x})}{\rho_{\sqrt{\Sigma},\mathbf{c}}(S)}. \tag{1}$$

3.3 Shamir Secret Sharing over Modules

Let $\mathbb{F}$ be a finite field. Given $a \in \mathbb{F}$, $1 \leqslant T$ and $\mathcal{S} \subseteq \mathbb{F}^*$, a $(T, \mathcal{S})$-sharing of a is obtained by doing the following:

1. Generate $P \in \mathbb{F}_{<T}[x]$ uniformly at random, conditioned to $P(0) = a$.
2. For $s \in \mathcal{S}$, compute $[\![a]\!]_s^P = P(s)$.
3. The output is the indexed tuple $[\![a]\!]_{\mathcal{S}}^P = ([\![a]\!]_s^P)_{s\in\mathcal{S}}$.

Given $(a, P, \mathcal{S})$, $[\![a]\!]_{\mathcal{S}}^P$ is uniquely defined. We say that an indexed tuple $[\![a]\!]$ is a valid T-sharing of a if there exists a polynomial $P \in \mathbb{F}_{<T}[x]$ and an evaluation set $\mathcal{S} \subseteq \mathbb{F}^*$ such that $[\![a]\!] = [\![a]\!]_{\mathcal{S}}^P$. If $|\mathcal{S}| \geqslant T$, then for any indexed tuple $b = (b_s)_{s\in\mathcal{S}}$ there exists at most a single pair (a, P) such that $b = [\![a]\!]_{\mathcal{S}}^P$.

When P and/or $\mathcal{S}$ are clear from context, we may omit them and note $[\![a]\!]^P$, $[\![a]\!]_{\mathcal{S}}$ or $[\![a]\!]$. The set of valid $(T, \mathcal{S})$-sharings is a $\mathbb{F}$-linear space of dimension T. Indeed, given $\lambda, \mu \in \mathbb{F}$:

$$\lambda\, [\![a]\!]_{\mathcal{S}}^P + \mu\, [\![b]\!]_{\mathcal{S}}^Q = [\![\lambda\, a]\!]_{\mathcal{S}}^{\lambda\, P} + [\![\mu\, b]\!]_{\mathcal{S}}^{\mu\, Q} = [\![\lambda\, a + \mu\, b]\!]_{\mathcal{S}}^{\lambda\, P + \mu\, Q} \tag{2}$$

Given a T-sharing $[\![a]\!]_{\mathcal{S}}$ where $|\mathcal{S}| = T$, we can recover a using Lagrange interpolation. Note that this recovery process is independent of P.

$$a = \sum_{s\in\mathcal{S}} \lambda_{s,\mathcal{S}}\, [\![a]\!]_s, \quad \text{where} \quad \lambda_{s,\mathcal{S}} = \prod_{s'\in\mathcal{S}\backslash s} \frac{s'}{s'-s}. \tag{3}$$

Generalizations. We can generalize secret-sharing as follows:

- **Composite rings.** We can generalize secret-sharing to any finite commutative ring R. Instead of $\mathcal{S} \subseteq F\backslash\{0\}$, this requires the stronger condition $\{\forall s, s' \in \mathcal{S}, (s - s') \in R^\times\}$, where $R^\times$ is the multiplicative group of invertible elements of R.
  This allows us to secret-share over $\mathbb{Z}_q$, where $q = q_1 \cdot q_2$ and $q_1 < q_2$ are two prime numbers. Over this ring, we can perform secret-sharing for any set $\mathcal{S} \subseteq \{1, \ldots, q_1 - 1\}$.
- **Vectors.** We can perform secret-sharing for vector secrets $\mathbf{a} = (a_i)_{i\in[k]} \in R^k$. This is done by secret-sharing each coefficient independently: $\forall i \in [k]$ we secret-share a_i with a distinct polynomial $P_i(x) = \sum_{j<T} p_{i,j}\, x^j \in R[x]$.
  If we note $[\![\mathbf{a}]\!]_{\mathcal{S}} = ([\![\mathbf{a}]\!]_s)_{s\in\mathcal{S}}$ the tuple obtained, and $\mathbf{p}_j = (p_{i,j})_{i\in[k]} \in R^k$, we can see that $\forall s \in \mathcal{S}$, $[\![\mathbf{a}]\!]_s = \sum_{j<T} s^j \cdot \mathbf{p}_j$. Abusing notation, we note $\boldsymbol{P}(x) = \sum_{j<T} x^j \cdot \mathbf{p}_j$ and will refer to $\boldsymbol{P}(x)$ as the interpolation polynomial of $[\![\mathbf{a}]\!]_{\mathcal{S}}$, which we may also note $[\![\mathbf{a}]\!]_{\mathcal{S}}^{\boldsymbol{P}}$. This notation is abusive since R^k is not a ring but a module when $k > 1$, however, it is helpful for our purposes.

The set of polynomials over the module R^k, which we note $R^k[x]$, is a R-module. In particular, operations such as addition and multiplication by a scalar $a \in R$ are well-defined over $R^k[x]$. Given a matrix $\mathbf{M} \in R^{\ell \times k}$, one can check that:

$$\mathbf{M} \cdot [\![\mathbf{a}]\!]_{\mathcal{S}}^{\boldsymbol{P}} = [\![\mathbf{M} \cdot \mathbf{a}]\!]_{\mathcal{S}}^{\mathbf{M}\cdot\boldsymbol{P}} \tag{4}$$

3.4 Hardness Assumptions

We recall the Ring-SIS (RSIS) assumption.

Definition 2 (RSIS)**.** *Let ℓ, q be integers and $B_2 > 0$ be a real number. The advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{RSIS}}(\kappa)$ of an adversary $\mathcal{A}$ against the Ring Short Integer Solutions problem $\mathsf{RSIS}_{q,\ell,B_2}$ is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{RSIS}}(\kappa) = \Pr\left[\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^{\ell},\, \mathbf{z} \leftarrow \mathcal{A}(\mathbf{a}) \,:\, 0 < \|\mathbf{z}\| \leqslant B_2 \,\wedge \left[1 \ \mathbf{a}^{\top}\right]\mathbf{z} = \mathbf{0} \bmod q\right].$$

The $\mathsf{RSIS}_{q,\ell,B_2}$ assumption states that any efficient adversary $\mathcal{A}$ has a negligible advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{RSIS}}(\kappa)$.

Another assumption of interest is the $\mathsf{Hint\text{-}MLWE}$ problem which was introduced recently in [KLSS23a] and enjoys a dimension-preserving reduction to RLWE.

Recall however from the technical overview that our DKG and threshold signatures reveal matricial hints $\mathbf{R} \cdot \mathbf{s} + \mathbf{y}_1$ and $\mathbf{R}' \cdot \mathbf{p} + \mathbf{y}_2$ through the use of our $\mathsf{V3S}$, and then computes combined hints of the form $c \cdot \mathbf{s} + \mathbf{p}$. The $\mathsf{Hint\text{-}MLWE}$ problem doesn't capture the use of matrices in hints, nor leakage on noises.

The following section thus introduces a generalized variant of $\mathsf{Hint\text{-}MLWE}$.

3.5 Generalization of Hint-MLWE

We introduce a generalized version of $\mathsf{Hint\text{-}MLWE}$, allowing matrix hints and variable perturbation standard deviations. We call it $\mathsf{MatrixHint\text{-}MLWE}$. Although we do not make full use of them in this paper, we believe they are of independent interest. Like $\mathsf{Hint\text{-}MLWE}$, $\mathsf{MatrixHint\text{-}MLWE}$ reduces efficiently to MLWE.

Definition 3 ($\mathsf{MatrixHint\text{-}MLWE}$)**.** *Let k, ℓ, q, Q be integers, $\mathcal{D}_{\mathsf{sk}}, (\mathcal{D}_{\mathbf{p}}^{(i)})_{i \in [Q]}$ be probability distributions over $\mathbb{Z}^{k+\ell}$, and $\mathcal{M}$ be a distribution of challenge matrices over $\left(\mathbb{Z}^{n(k+\ell)\times n(k+\ell)}\right)^Q$. The advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MatrixHint\text{-}MLWE}}(\kappa)$ of an adversary $\mathcal{A}$ against $\mathsf{MatrixHint\text{-}MLWE}_{k,\ell,q,Q,\mathcal{D}_{\mathsf{sk}},(\mathcal{D}_{\mathbf{p}}^{(i)})_i,\mathcal{M}}$ — Matrix Hint Module Learning with Errors problem — is defined as:*

$$\left|\Pr\left[1 \leftarrow \mathcal{A}\left(\mathbf{A}, \left[\mathbf{I}_k \ \mathbf{A}\right] \cdot \mathbf{s}, (\mathbf{M}_i, \mathbf{z}_i)_{i \in [Q]}\right)\right] - \Pr\left[1 \leftarrow \mathcal{A}\left(\mathbf{A}, \mathbf{u}, (\mathbf{M}_i, \mathbf{z}_i)_{i \in [Q]}\right)\right]\right|,$$

where $(\mathbf{A}, \mathbf{u}) \xleftarrow{\$} \mathcal{R}_q^{k\times\ell} \times \mathcal{R}_q^{k}$, $\mathbf{s} \leftarrow \mathcal{D}_{\mathsf{sk}}$ and for $i \in [Q]$: $\mathbf{M}_i \leftarrow \mathcal{M}$, $\mathbf{y}_i \leftarrow \mathcal{D}_{\mathbf{p}}^{(i)}$, and $\mathbf{z}_i = \mathbf{M}_i \cdot \mathbf{s} + \mathbf{y}_i$. The $\mathsf{MatrixHint\text{-}MLWE}_{k,\ell,q,Q,\mathcal{D}_{\mathsf{sk}},(\mathcal{D}_{\mathbf{p}}^{(i)})_i,\mathcal{M}}$ assumption states that any efficient adversary $\mathcal{A}$ has a negligible advantage.

Note that in the product $\mathbf{M}_i \cdot \mathbf{s}$, we consider $\mathbf{s}$ as a vector of $n(k+\ell)$ scalars in $\mathbb{Z}$. We can recover the polynomial version of Hint-MLWE *by taking $\mathbf{M}_i$ to be a matrix in blocs with negacyclic matrices of size $n \times n$ corresponding to the desired polynomial on the diagonal. We also recover the classical* RLWE *problem by taking $Q = 0$:* $\mathsf{RLWE}_{q,\mathcal{D}_{\mathsf{sk}}} = \mathsf{MatrixHint\text{-}MLWE}_{q,Q=0,\mathcal{D}_{\mathsf{sk}},(\mathcal{D}^{(i)}_{\mathsf{pert}})_i,\mathcal{M}}$.

The spectral norm $s_1(\mathbf{M})$ of a matrix $\mathbf{M}$ is defined as the value $\max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{M}\,\mathbf{x}\|}{\|\mathbf{x}\|}$. We recall that if $\mathbf{M}$ is symmetric, then its spectral norm is also its largest eigenvalue.

Theorem 1 (Hardness of MatrixHint-MLWE). *Let k, ℓ, q, Q, be positive integers, and $\mathcal{M}$ be a distribution over $\mathbb{Z}^{n(k+\ell)\times n(k+\ell)\times Q}$. For $\sigma_{\mathsf{sk}}, (\sigma_y^{(i)})_{i\in[Q]} > 0$, let $\sigma > 0$ be a real number defined as $\frac{1}{\sigma^2} = 2(\frac{1}{\sigma_{\mathsf{sk}}^2} + \sum_{i\in[Q]} \frac{s_1(\mathbf{M}_i^\top \mathbf{M}_i)}{(\sigma_y^{(i)})^2})$. If $\sigma \geqslant \sqrt{2}\cdot\eta_\varepsilon(\mathbb{Z}^{n(k+\ell)})$ for some $\varepsilon \in (0, 1/2]$, then there exists an efficient reduction from* $\mathsf{MLWE}_{k,\ell,q,\sigma}$ *to* $\mathsf{MatrixHint\text{-}MLWE}_{k,\ell,q,Q,\sigma_{\mathsf{sk}},(\sigma_y^{(i)})_i,\mathcal{M}}$ *that reduces the advantage by at most 2ε.*

Note that we recover the Hardness of MatrixHint-MLWE *again by taking $\mathbf{M}_i$ equal to a matrix in blocs with the negacyclic matrix of some polynomial γ on the diagonal.*

When the distributions $\mathcal{D}_{\mathsf{sk}}$ and $(\mathcal{D}_{\mathbf{p}}^{(i)})_i$ are discrete Gaussians, the proof relies as in [KLSS23a] on the fact that the distribution of $\mathbf{s}$ conditioned on the values $(\mathbf{M}_i \cdot \mathbf{s} + \mathbf{y}_i)_i$ still follows a discrete Gaussian distribution. We provide a proof of Lemma 1 in Appendix D, and refer to the proof of Theorem 2 for Theorem 1.

Lemma 1. *Let $Q > 0$ be an integer, and $\sigma_{\mathsf{sk}}, (\sigma_y^{(i)})_{i\in[Q]}$ be reals > 0. Take $\mathbf{M}_0, \ldots, \mathbf{M}_{Q-1} \in \mathbb{Z}^{n(k+\ell)\times n(k+\ell)}$. We define $\mathbf{\Sigma}_0 := (\frac{1}{\sigma_{\mathsf{sk}}^2}\cdot\mathbf{I} + \sum_{i\in[Q]} \frac{1}{\sigma_{y,i}^2}\cdot\mathbf{M}_i^\top\mathbf{M}_i)^{-1}$. Then the following two distributions over $\mathcal{R}\times\mathbb{Z}^{n(k+\ell)\cdot Q}$ are statistically identical.*

1. $\left\{(\mathbf{s}, \mathbf{z}_0, \ldots, \mathbf{z}_{Q-1}) \mid \mathbf{s} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_{\mathsf{sk}}}, \mathbf{y}_i \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_y^{(i)}}, \mathbf{z}_i = \mathbf{M}_i \cdot \mathbf{s} + \mathbf{y}_i\right\}$
2. $\left\{(\hat{\mathbf{s}}, \mathbf{z}_0, \ldots, \mathbf{z}_{m-1}) \middle| \begin{array}{c} \mathbf{s} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_{\mathsf{sk}}}, \mathbf{y}_i \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_y^{(i)}}, \mathbf{z}_i = \mathbf{M}_i \cdot \mathbf{s} + \mathbf{y}_i, \\ \mathbf{c} = \mathbf{\Sigma}_0 \cdot \sum_{i\in[Q]} \frac{1}{(\sigma_y^{(i)})^2}\mathbf{M}_i^\top \mathbf{z}_i, \hat{\mathbf{s}} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\mathbf{c},\sqrt{\mathbf{\Sigma}_0}} \end{array}\right\}$

Lemma 1 actually suffices to prove results in this paper. Intuitively, when $\mathbf{s}$ follows a Gaussian distribution parameterized with $\mathbf{\Sigma}_0$, we can reexpress it as the sum of an isotropic Gaussian of variance equal to the minimal radius of $\mathbf{\Sigma}_0$, plus some smaller Gaussian. When $\mathbf{y}$ has sufficiently large deviation, the minimal radius of $\mathbf{\Sigma}_0$ is close to σ_{sk}^2. This lemma allows us to extract some "good randomness" from the $\mathbf{s}_i$ and perturbations $\mathbf{p}_i$ in Pelican even after revealing hints on them.

Elliptic MatrixHint-MLWE. Interestingly, it is possible to obtain a more general reduction of MatrixHint-MLWE covering the case of leakage on the perturbations. We give a quick introduction to it here, although we decided to not make use of it in our proofs and rather rely on Lemma 1 as it doesn't lead to a substancial gain in parameters in our case.

Theorem 2 (Hardness of Elliptic MatrixHint-MLWE**).** *Let k, ℓ, Q, q, be positive integers, and $\mathcal{M}$ be a distribution over $\mathbb{Z}^{n(k+\ell)\times n(k+\ell)\times Q}$. For $\mathbf{\Sigma}_{\mathsf{sk}}, \mathbf{\Sigma}_{y,i}$ symmetric positive definite matrices, vectors $\mathbf{c_s}, (\mathbf{c_y}^{(i)})_i \in \mathbb{Z}^{n(k+\ell)}$. Let $\sigma > 0$ be a real number defined as $\frac{1}{\sigma^2} = 2 \cdot (s_1(\mathbf{\Sigma}_{\mathsf{sk}}^{-1}) + \sum_{i\in[Q]} s_1(\mathbf{M}_i^\top \mathbf{M}_i) s_1(\mathbf{\Sigma}_{y,i}^{-1}))$. If $\sigma \geqslant \sqrt{2} \cdot \eta_\varepsilon(\mathbb{Z}^{n(k+\ell)})$ for some $\varepsilon \in (0, 1/2]$, then there exists an efficient reduction from $\mathsf{MLWE}_{k,\ell,q,\sigma}$ to $\mathsf{MatrixHint\text{-}MLWE}_{k,\ell,q,Q,\mathcal{D}_{\mathsf{sk}}:=\mathcal{D}_{\mathbf{c_s},\sqrt{\mathbf{\Sigma}_1}},(\mathcal{D}_{\mathsf{pert}}^{(i)}:=\mathcal{D}_{\mathbf{c}_i,\sqrt{\mathbf{\Sigma}_{y,i}}})_{i\in[Q]},\mathcal{M}}$ that reduces the advantage by at most 2ε.*

We include a proof overview in Appendix D.

In our case, secrets and perturbations are of the form $\mathbf{r} = \sum_i \mathbf{r}_i$, and the reveal of hints $\mathbf{R}_i \cdot \mathbf{r}_i + \mathbf{y}_i$ modifies their distribution. We can see with Lemma 1 that the resulting distribution has covariance matrix:

$$\mathbf{\Sigma}_{\mathbf{r}} := \sum_i \left(\frac{1}{\sigma_{\mathbf{r}}^2} \mathbf{I} + \frac{1}{\sigma_{\mathbf{y}}^2} \mathbf{R}_i^\top \mathbf{R}_i \right)^{-1}$$

If B is an overwhelming bound on $s_1(\mathbf{R}_i^\top \mathbf{R}_i)$, then the minimum eigenvalue of $\mathbf{\Sigma}_{\mathbf{r}}$ is at least $\sum_i (\frac{1}{\sigma_{\mathbf{r}}^2} + \frac{B}{\sigma_{\mathbf{y}}^2})^{-1}$. When taking such a sum with k elements for both the secret and perturbations, we obtain a reduction to MLWE with σ verifying:

$$\frac{1}{\sigma^2} = 2 \cdot \frac{1}{k} \left(\frac{1}{\sigma_{\mathsf{sk}}^2} + \frac{B}{\sigma_{y,\mathsf{sk}}^2} + \sum_{i\in[Q]} s_1(c_i c_i^*) \cdot \left(\frac{1}{\sigma_{\mathbf{p}}^2} + \frac{B}{\sigma_{y,\mathbf{p}}^2} \right) \right)$$

4 Verifiable Short Secret Sharing

We start with our new proposal for verifiable (short) secret sharing. As presented in Section 2.1, we introduce a new *random submersion* technique allowing one to distribute and prove the shortness of a secret among N parties. This shortness is crucial in the context of lattice-based cryptographic schemes based on variants of the Learning With Errors (LWE) assumption, which is based on the shortness of secret elements. Importantly, our technique controls the leakage on the secret and can be leveraged to sample short secrets in threshold schemes, as will be formalized for a threshold robust signature scheme, and a distributed key generation protocol in Section 5.

4.1 Security notions

We first define in Definition 4 the syntax of a verifiable short secret sharing, as well as standard security properties in Definitions 5 and 6.

Definition 4 (Verifiable Short Secret Sharing). *Let $\mathcal{D}_{\mathbf{x}}$ be a distribution (corresponding to* honestly generated secrets*), and V be a subset of its support* Sec *(the acceptable secrets). Let $N > T$ be two nonnegative integers. Assume that we have N parties communicating and yet another distinguished party called* dealer*. A Verifiable Short Secret Sharing (*$\mathsf{V3S}$*) is defined as a tuple of functions:*

- $\mathsf{V3S.Share}(N, T, \mathbf{x}) \to (\llbracket \mathbf{x} \rrbracket_1, ..., \llbracket \mathbf{x} \rrbracket_N, \pi, \pi_1, ..., \pi_N) \in \mathit{Sec}^N \times (\{0,1\}^*)^N$, *run by the dealer.*
- $\mathsf{V3S.Verify}_i(\llbracket \mathbf{x} \rrbracket_i, \pi, \pi_i) \to (\mathsf{true} \mid \mathsf{false})$*: run by the party* i *if sufficiently enough parties accept the proof from the dealer, they are guaranteed to have shares from an acceptable secret, i.e. belonging to* V*.*
- $\mathsf{V3S.Reconstruct}(\llbracket \mathbf{x} \rrbracket_{i_1}, ..., \llbracket \mathbf{x} \rrbracket_{i_k}) \to (\mathbf{x} \mid \perp)$*: reconstruct the secret from* $k \geqslant T$ *shares. Returns* $\perp$ *in case of failure.*
- $\mathsf{V3S.RobustReconstruct}(\llbracket \mathbf{x} \rrbracket_{i_1}, ..., \llbracket \mathbf{x} \rrbracket_{i_k}) \to \mathbf{x}$*: reconstruct the secret from* $k \geqslant T$ *shares in the presence of up to* $(k-T)/2$ *invalid shares.*

Definition 5 **(V3S Correctness).** *A* V3S *scheme is said to be* correct *if when* V3S.Share() *is honestly executed with a secret* $\mathbf{x}$ *sampled from the honest distribution* $\mathcal{D}_{\mathbf{x}}$*, then verification correctly passes for all parties with overwhelming probability, and* V3S.Reconstruct() *correctly recovers the secret.*

Definition 6 **(V3S Soundness).** *A* V3S *scheme is said* sound *if for any* $S \subseteq \{1, ..., N\}$ *of cardinal at least* T*, and any sharing* $\llbracket \mathbf{x} \rrbracket$*, with corresponding proofs* π, π_i*, verification will fail with overwhelming probability if either:*

- *shares are* inconsistent *among* S*, i.e. reconstructions over shares of* S *returns* $\perp$*. Then, at least one party in* S *will fail verification.*
- *or the secret is* invalid*, i.e. the secret reconstructed by parties in* S *does not belong to* V*. Then verification will fail for at least one party in* S*.*

This is formalized as requiring any efficient adversary $\mathcal{A}$ *to win Game* V3S-sound *defined in Figure 2 with negligible probability.*

We say that a V3S has the fragmentary knowledge property if knowledge of at most $T-1$ shares and proofs leaks only partial information on the secret key, and keeps sufficient entropy. This is formalized with two simulators as follows:

Definition 7 **(Computational/Statistical V3S Fragmentary Knowledge).** *A* V3S *has the fragmentary knowledge property if there exists two simulators* $\mathsf{SimProof}(S) \to ((\llbracket \mathbf{x} \rrbracket_i)_{i \in S}, \pi, (\pi_i)_{i \in S})$*, defined for subsets* $S \subset \{1, ..., N\}$ *of cardinality at most* $T-1$*, and* $\mathsf{SimSecret}(\pi, (\pi_i)_{i \in S})$ *such that the output distribution* $(\mathbf{x}, (\llbracket \mathbf{x} \rrbracket_i)_{i \in S}, \pi, (\pi_i)_{i \in S})$ *of the two processes*

- $((\llbracket \mathbf{x} \rrbracket_i)_{i \in S}, \pi, (\pi_i)_{i \in S}) \leftarrow \mathsf{SimProof}(S)$*, and* $\mathbf{x} \leftarrow \mathsf{SimSecret}(\pi, (\pi_i)_{i \in S})$
- *or* $\mathbf{x} \leftarrow \mathcal{D}_{\mathbf{x}}$*,* $((\llbracket \mathbf{x} \rrbracket_i)_{i \in \{1,...,N\}}, \pi, (\pi_i)_{i \in \{1,...,N\}}) \leftarrow \mathsf{V3S.Share}(N, T, \mathbf{x})$

are (computationally/statistically) indistinguishable. This is formalized as requiring the winning advantage of any efficient adversary $\mathcal{A}$ *to win Game* V3S-fk *defined in Figure 3 to be at a negligible distance from 1/2.*

4.2 Formal definition of our VSSS

The protocol drafted in the technical overview translates to pseudo-code quite straightforwardly and is given in Figure 4. The syntax and formalization are inspired by the recent work of [ABCP23], as well as the use of hash functions to commit on secret shares and derive a challenge for verification. We formalize the requirements on the submersion matrices which were hinted in Section 2.1.

```
Game_{V3S-sound}
1: $L_H := \varnothing$
2: $N, T, S \leftarrow \mathcal{A}()$        ▷ The adversary $\mathcal{A}$ chooses a subset $S$ of parties to target
3: assert{ $S \subset \{1, \ldots, N\} \wedge |S| \geqslant T$ }        ▷ $S$ must be large enough to allow reconstruction
4: $(s_i)_{i\in\{1,\ldots,N\}}, \pi, (\pi_i)_{i\in\{1,\ldots,N\}} \leftarrow \mathcal{A}^{H(\cdot)}(N, T, S)$        ▷ $\mathcal{A}$ produces a $T$-sharing among N parties
5: if $\forall i \in S$, $\mathsf{V3S.Verify}(s_i, \pi, \pi_i) =$ false then
6:     return 0        ▷ If a party in $S$ fails verification, $\mathcal{A}$ loses
7: $\mathbf{x} = \mathsf{V3S.Reconstruct}((s_i)_{i\in S})$
8: if $\mathbf{x} = \perp$ then        ▷ Verification passes but shares are inconsistent in $S$
9:     return 1
10: if $\mathbf{x} \notin V$ then        ▷ Verification passes but the secret is invalid
11:     return 1
12: return 0        ▷ The sharing chosen by the adversary is valid

    $H(\mathsf{str}, \mathsf{digest})$
    1: assert{ $\mathsf{str} \in \mathsf{pp}.\mathrm{HashParams}$ }        ▷ Check domain string
    2: if $\exists r.(\mathsf{str}, \mathsf{digest}, r) \in L_H$ then
    3:     return $r$
    4: else
    5:     Sample $r$ uniformly
    6:     $L_H := L_H \cup \{(\mathsf{str}, \mathsf{digest}, r)\}$
    7:     return $r$
```

Fig. 2: Soundness game for a V3S. $\mathcal{A}$ wins if the game V3S-sound returns 1.

Definition 8 (Property G). *A distribution of matrices* $\mathbf{R}$ *is said to satisfy the property* G_{p_1,p_2,p_3} *if there exists two bounds* B, B' *such that we have:*

separation *if* $(\mathbf{x}, \mathbf{y}) \neq (\mathbf{x}', \mathbf{y}')$, $\Pr_{\mathbf{R}\leftarrow\mathcal{D}_\mathbf{R}}[\mathbf{R}\mathbf{x} + \mathbf{y} \bmod q = \mathbf{R}\mathbf{x}' + \mathbf{y}' \bmod q] = p_1$ *with* $p_1 = \mathsf{negl}(\kappa)$*: the matrices* $\mathbf{R}$ *send different secrets to different points.*

large norm detection *if* $\|\mathbf{x}\| > B$*, for any* $\mathbf{y}$, $\Pr_{\mathbf{R}\leftarrow\mathcal{D}_\mathbf{R}}[\|\mathbf{R}\mathbf{x} + \mathbf{y} \bmod q\| > B'] = 1 - p_2$ *with* $p_2 = \mathsf{negl}(\kappa)$*: intuitively, if* $\mathbf{x}$ *is large, we want the challenge to also be large with overwhelming probability.*

honest execution $\Pr_{\mathbf{x}\leftarrow\mathcal{D}^n_{\sigma_x},\mathbf{y}\leftarrow\mathcal{D}_{\sigma_y},\mathbf{R}\leftarrow\mathcal{D}_\mathbf{R}}[\|\mathbf{R}\mathbf{x} + \mathbf{y} \bmod q\| \leqslant B'] = 1 - p_3$ *with* $p_3 = \mathsf{negl}(\kappa)$*: in case of honest generation of the secret, the challenge is small.*

Assuming this crucial property in the construction, we show that our V3S construction is secure for the notions introduced in Section 4.1. We classically work in the ROM and assume that hash functions are modeled as random oracles with output size 2κ.

Theorem 3 (Security of V3S in the ROM). *Let* Q_H *be the maximum number of queries allowed to the random oracle. For the* V3S *in Fig. 4, taking as set of valid secrets* $V = \{\mathbf{x} \mid \|\mathbf{x}\| \leqslant B\}$*, if the distribution of matrices* $\mathbf{R}$ *satisfies property* G_{p_1,p_2,p_3}*, then we have:*

```
Game_{V3S-fk^b}, with $b \in \{0,1\}$
1: $L_H := \varnothing$
2: $N, T, S \leftarrow \mathcal{A}()$        ▷ The adversary chooses a subset $S$ of parties to target
3: assert{ $S \subset \{1,\ldots,N\} \wedge |S| = T-1$ }
4: if $b = 0$ then
5:     $\mathbf{x} \leftarrow \mathcal{D}_{\mathbf{x}}$
6:     $(\llbracket x \rrbracket_i)_{i\in\{1,\ldots,N\}}, \pi, (\pi_i)_{i\in\{1,\ldots,N\}} \leftarrow \mathsf{V3S.Share}(\mathbf{x})$        ▷ Honest sharing of $\mathbf{x}$
7: else
8:     $(\llbracket x \rrbracket_i)_{i\in\{1,\ldots,N\}}, \pi, (\pi_i)_{i\in\{1,\ldots,N\}} \leftarrow \mathsf{SimProof}(S)$        ▷ Simulation of the sharing transcript
9:     $\mathbf{x} \leftarrow \mathsf{SimSecret}(\pi, (\pi_i)_{i\in S})$
10: $b' \leftarrow \mathcal{A}^H(\mathbf{x}, (\llbracket x \rrbracket_i)_{i\in S}, \pi, (\pi_i)_{i\in S})$
11: return $b'$

    $H(\mathsf{str}, \mathsf{digest})$
    1: assert{ $\mathsf{str} \in \mathsf{pp}.\text{HashParams}$ }        ▷ Check domain string
    2: if $\exists r.(\mathsf{str}, \mathsf{digest}, r) \in L_H$ then
    3:     return $r$
    4: else
    5:     Sample $r$ uniformly
    6:     $L_H := L_H \cup \{(\mathsf{str}, \mathsf{digest}, r)\}$
    7:     return $r$
```

Fig. 3: Fragmentary Knowledge game for a VSSS. We consider the distinguishing advantage of an adversary $\mathcal{A}$ between the games with $b = 0$, and $b = 1$.

– ***Correctness*** *with probability* $1 - p_3$.
– ***Soundness*** *with an advantage of at most:* $\frac{Q_H^2}{2^{2\kappa-1}} + Q_H \cdot (p_1 + p_2)$
– ***Fragmentary knowledge*** *with an advantage of at most* $N \cdot \frac{Q_H}{2^{2\kappa}}$*, where for a set* $|S| = T - 1$ *the simulators are defined in Algorithms 5 and 6.*

Algorithm 5 $\mathsf{SimProof}(S)$

```
1: $\forall i \in S, \llbracket \mathbf{x} \rrbracket_i, \llbracket \mathbf{y} \rrbracket_i \xleftarrow{\$} \mathbb{Z}_q,\ r_i \xleftarrow{\$} \{0,1\}^{2\kappa}$
2: Generate Merkle tree $h$ containing $(\llbracket \mathbf{x} \rrbracket_i, \llbracket \mathbf{y} \rrbracket_i, r_i)_{i\in S}$ and complete the hashes of
   missing shares $j \notin S$ by uniform i.i.d values in $\{0,1\}^{2\kappa}$.
3: Produce $\mathsf{proof}_i$ for shares in $S$.
4: $\mathbf{x}, \mathbf{y} \leftarrow \mathcal{D}^n_{\sigma_x}, \mathcal{D}^d_{\sigma_y}$
5: Compute $\llbracket v \rrbracket$ such that:
     – for $i \in S$, $\llbracket \mathbf{v} \rrbracket_i = \mathbf{R} \cdot \llbracket \mathbf{x} \rrbracket_i + \llbracket \mathbf{y} \rrbracket_i$
     – $\llbracket \mathbf{v} \rrbracket_0 = \mathbf{R} \cdot \mathbf{x} + \mathbf{y}$
6: return $(\llbracket \mathbf{x} \rrbracket_i)_{i\in S}, \pi = (h, \llbracket \mathbf{v} \rrbracket), (\pi_i)_{i\in S} = (\llbracket \mathbf{y} \rrbracket_i, r_i, \mathsf{proof}_i)_{i\in S}$
```

Algorithm 1 V3S.Share$(N, T, \mathbf{x})$

```
1: $\mathbf{y} \leftarrow \mathcal{D}^d_{\sigma_y}$; $(r_i)_{i\in\{1,\ldots,N\}} \xleftarrow{\$} \{0,1\}^{N\cdot 2\kappa}$
2: Generate $T$-out-of-$N$ sharings $[\![\mathbf{x}]\!] = [\![\mathbf{x}]\!]_{\{1,\ldots,N\}}$, $[\![\mathbf{y}]\!] = [\![\mathbf{y}]\!]_{\{1,\ldots,N\}}$   ▷ Section 3.3
3: $h$ := root of Merkle tree Tree with leaves $\mathsf{leaf}_i = ([\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i, r_i)$ for $i \in \{1,\ldots,N\}$
4: for $i \in \{1,\ldots,N\}$ do
5:     $\mathsf{proof}_i$ := co-path of $\mathsf{leaf}_i$ in the tree Tree   ▷ Proves $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i, r_i)$ is in $h$
6:     $\pi_i := ([\![\mathbf{y}]\!]_i, \mathsf{proof}_i)$
7: $\mathbf{R} := H_{\mathbf{R}}(h)$   ▷ Hash $h$ to obtain a random matrix from $\mathcal{D}_{\mathbf{R}}$
8: $[\![\mathbf{v}]\!] := \mathbf{R}\cdot[\![\mathbf{x}]\!] + [\![\mathbf{y}]\!]$   ▷ Johnson-Lindenstrauss
9: $\pi := (h, [\![\mathbf{v}]\!])$   ▷ Challenge polynomial and Merkle tree root
10: return $([\![\mathbf{x}]\!], \pi, (\pi_i)_{i\in\{1,\ldots,N\}})$
```

Algorithm 2 V3S.Verify$([\![\mathbf{x}]\!]_i, \pi, \pi_i)$

```
1: Parse $\pi := (h, [\![\mathbf{v}]\!])$ and $\pi_i := ([\![\mathbf{y}]\!]_i, r_i, \mathsf{proof}_i)$
2: Recover $\mathbf{v}$ from $[\![\mathbf{v}]\!]$   ▷ See Section 3.3
3: if ($\mathsf{proof}_i$ is not a valid proof that $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i, r_i)$ is in Tree) then
4:     return false
5: $\mathbf{R} := H_{\mathbf{R}}(h)$
6: if $([\![\mathbf{v}]\!]_i \neq \mathbf{R}\cdot[\![\mathbf{x}]\!]_i + [\![\mathbf{y}]\!]_i) \vee (\|\mathbf{v}\| > B')$ then
7:     return false ▷ Check the consistency of shares, and shortness of secret vector
8: return true
```

Algorithm 3 V3S.Reconstruct$([\![\mathbf{x}]\!]_I)$, with $|I| \geqslant T$

```
1: $J$ = {the first $T$ indices in $I$}   ▷ Reconstruct over a set of size $T$
2: Compute the unique $P \in (\mathbb{Z}_q)_{<T}[X]$ such that $\forall j \in J, P(j) = [\![\mathbf{x}]\!]_j$
3: if $\exists i \in I \backslash J$ s.t. $P(i) \neq [\![\mathbf{x}]\!]_i$ then
4:     return $\perp$
5: return $\mathbf{x} := P(0)$
```

Algorithm 4 V3S.RobustReconstruct$(([\![\mathbf{x}]\!]_i)_{i\in I})$, with $|I| \geqslant T$

```
1: Interpret $([\![x]\!]_i)_{i\in I}$ as a Reed-Solomon codeword of $\begin{cases}\text{block length } T\\ \text{message length } |I|\end{cases}$
2: Run error correction on $([\![\mathbf{x}]\!]_i)_{i\in I}$   ▷ Fixes up to $(|I| - T)/2$ errors
3: return V3S.Reconstruct$(([\![\mathbf{x}]\!]_i)_{i\in I})$
```

Fig. 4: Algorithms for our Verifiable Short Secret Sharing (V3S). Shamir's sharing modulo q is done using standard Lagrange interpolation. Secrets space is $\mathbb{Z}^n$, and matrices $\mathbf{R}$ are sampled in $\mathbb{Z}^{d\times n}$ following distribution $\mathcal{D}_{\mathbf{R}}$. The support for the honest secrets $\mathbf{x}$ is $\mathcal{D}_{\mathbf{x}} = \mathcal{D}^n_{\sigma_x}$, and the support for the blinding $\mathbf{y}$ is $\mathcal{D}^d_{\sigma_y}$.

Algorithm 6 $\mathsf{SimSecret}(\pi \equiv (h, [\![\mathbf{v}]\!]), (\pi_i \equiv ([\![\mathbf{y}]\!]_i, r_i, \mathsf{proof}_i))_{i \in S})$

```
1: Compute $\mathbf{R} = H_{\mathbf{R}}(h)$.
2: $\mathbf{c} \equiv \Sigma_0 \cdot \frac{1}{\sigma_y^2} \cdot \mathbf{R}^\top \cdot \mathbf{z}$
3: $\Sigma_0 \equiv \left(\frac{1}{\sigma_x^2} \cdot \mathbf{I} + \frac{1}{\sigma_y^2} \cdot \mathbf{R}^\top \mathbf{R}\right)^{-1}$
4: $\mathbf{z} = [\![\mathbf{v}]\!]_0$.
5: return $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z}^n, \mathbf{c}, \sqrt{\Sigma_0}}$
```

We demonstrate in Section 6 how to efficiently construct such a desirable distribution.

The proof of this theorem is quite lengthy and relies on a certain number of hybrids, we defer it to the appendix: we let the reader refer to Appendix A. For completeness purposes, we still give here the main sketch and ideas upon which the proofs are built.

Correctness. V3S.Share() correctly constructs a Shamir's sharing and a corresponding Merkle tree for secure sharing and verification. The V3S.Reconstruct function can accurately reconstruct the original value for any subset of shares of size at least T, and the V3S.Verify() function consistently passes its checks by the *honest execution* property of the distribution of $\mathbf{R}$.

Soundness. The proof employs a hybrid argument approach, consisting of three main steps:

- The first game is the actual soundness game.
- Hybrid_2 is a tweak of the soundness game ensuring that the matrix $\mathbf{R}$ is sampled *after* the shares are chosen, which is the crux for applying *separation and large norm detection properties* of $\mathcal{D}_{\mathbf{R}}$. This hybrid aims to maintain the integrity of the Merkle tree hashes and the random oracle's programming, penalizing the adversary for any inconsistency or attempts to exploit the hash function's collision or pre-image resistance.
- Hybrid_3: modifies the conditions under which an adversary can win, specifically targeting inconsistencies and too large norms in the reconstructed secret during the verification process.
- Probability of Winning: Rely on the *separation and large norm detection* to limit the probability of an adversary's success.

Fragmentary Knowledge. The proof also goes by three hybrids.

- Hybrid_1 : Adversary observes the real distribution of transcripts, corresponding to $b = 0$ in the V3S-fk game.
- Hybrid_2 : Introduces uniform random strings in place of the hashes of shares not in set S, arguing the adversary's view changes negligibly if they had not queried these shares (basically, the view of the adversary differs *only if* it did query the random oracle on one of the shares $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}_i]\!], r_i)$ with $i \notin S$ in Hybrid_1)
- Hybrid_3 : Further modifies by replacing shares in S with uniformly random vectors and simulating the adversary's view under $b = 1$. It uses the correctness of the secret sharing and the indistinguishability proved in Lemma 1 to argue the adversary's advantage remains unchanged from Hybrid 2.

4.3 Towards Applications

Our definition of $\mathsf{V3S}$ is agnostic to the communication means used. We describe at a high level in this section two generic ways for using our $\mathsf{V3S}$ in protocols to obtain different guarantees and cost tradeoffs. Further, we demonstrate concrete usage and security proofs in Section 5.

VSSS protocol with detection of malicious behavior. The first protocol we introduce allows us to detect misbehavior during the protocol execution and abort in that case. It requires lightweight communication assumptions, i.e. confidential and authenticated pairwise channels, additionally reliable for correctness when all the parties are honest. However, the counterpart is that as soon as at least one of the parties is dishonest, the protocol may abort with no possibility of recovery. We assume from now on that there are at least T honest parties. The protocol proceeds in several rounds:

1. In the first round, the dealer samples $\mathbf{x} \leftarrow \mathcal{D}_{\mathbf{x}}$ and runs $(\llbracket \mathbf{x} \rrbracket_i)_{i \in \{1,\ldots,N\}}$, π, $(\pi_i)_{i \in \{1,\ldots,N\}} \leftarrow \mathsf{V3S.Share}(\mathbf{x})$. It then sends $(\llbracket \mathbf{x} \rrbracket_i, \pi, \pi_i)$ to each other party i over the pairwise channel between them.
2. In the second round, everyone checks the data provided by the dealer. They run $b = \mathsf{V3S.Verify}(\llbracket \mathbf{x} \rrbracket_i, \pi, \pi_i)$. After that, they send b to each other party, along with a hash of π to prove they had the same common proof.
3. Finally, each party accepts the sharing only if it receives a positive response from all other parties, along with the same hash as theirs.

From the good properties of our $\mathsf{V3S}$ proposal, we can easily show that this protocol has also:

- *correctness*: in case all parties behave honestly, then the protocol terminates and all honest parties accept the sharing.
- *soundness*: in case the dealer sends inconsistent shares to any honest party, or reconstructing to an invalid secret, it will be detected with overwhelming probability and the protocol will abort.
- *fragmentary knowledge*: in case the dealer is honest, and the protocol terminates, the transcript is indistinguishable from the one simulated by the fragmentary knowledge property of our VSSS.

Robust $\mathsf{V3S}$ protocol. We propose a second protocol providing guaranteed delivery in case the dealer is honest, and, in case the dealer is dishonest and misbehaves, all honest parties simultaneously abort. This protocol assumes the existence of an *authenticated reliable* (even for corrupted parties) *non-ordered broadcast channel*, and that a majority of the parties are honest.

It works with an $\mathsf{IND\text{-}CPA}$ symmetric encryption scheme SKE to implement non-repudiable pairwise communication. We assume that pairwise keys $\mathsf{sk}_i^{\mathsf{SKE}}$ are shared prior to the protocol execution to communicate with the dealer, along with a signature $\mathsf{sig}_i = \mathsf{Sign}(\mathsf{sk}, (i, \mathsf{sk}_{i,j}))$ signed by the dealer for a verified reveal of the key. Note, that requirements on these prior steps could be reduced by using non-repudiable and publicly verifiable signcryption. We could then simply assume that their public keys are pre-shared, and the verified reveal can be replaced by proofs that a given ciphertext is invalid, or corresponds to a given message using the non-repudiability property.

Our robust protocol also works in three rounds:

1. The dealer samples $\mathbf{x}$, and runs $\mathsf{V3S.Share}(\mathbf{x})$. It broadcasts the proof π along with individual encryptions produced with the SKE for each other party of $[\![\mathbf{x}]\!]_i, \pi_i$.
2. In the second round, each party decrypts the message from the dealer and runs $\mathsf{V3S.Verify}()$ on its share. If decryption or verification fails, the party broadcasts a complaint containing the secret key of its communication channel with the dealer $\mathsf{sk}_i^{\mathsf{SKE}}$, along with sig_i.
3. The third round consists of opening and verifying all the complaints. If any of the complaints is valid, it means that the dealer behaved dishonestly, if no complaint is valid, the sharing is valid, and the party can accept it.

In this protocol, we achieve:

– *unframeability:* if the dealer is honest, no corrupted party will be able to make honest parties reject. Indeed, as the signature scheme is unforgeable, only a complaint with the correct SKE key can be produced, and then the message sent by the dealer will be correctly decrypted.
– *accountability:* if the dealer is corrupted, the use of a broadcast channel ensures that all honest parties will receive the same set of complaints and will conclude identically. If they accept the shares, the shares of all honest parties pass verification, which means that the shares are consistent, and the secret is valid.

We could reduce the communication of our protocol by exchanging ciphertexts on pairwise channels instead of broadcasting them. We would then add an extra round after the round of complaints to allow dealers to answer complaints by broadcasting the share of the plaintiff in case there was a complaint.

5 Pelican: A Robust Threshold Signature Scheme

We make the following assumptions in our communication model:

– **Authenticated broadcast.** All contributions by user i are broadcast on a reliable public channel. In addition, they are signed by i and therefore authenticated. Concretely, i has a signing key sk_i and signs all their contributions: $\mathsf{SIG.Sign}(\mathsf{sk}_i, \mathsf{contrib}_j[i])$, where SIG is an $\mathsf{UF\text{-}CMA}$ standard signature scheme $\mathsf{SIG} = \{\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify}\}$.
These signatures are checked by other parties upon reception, and the contribution is declared invalid if the signature is invalid. For conciseness, this is omitted from the descriptions of the distributed key generation and distributed signing protocols.

- **Signed pairwise keys.** For any ordered pair of users (i, j), i and j share a pairwise symmetric key $\mathsf{K}_{i\to j}$ that has been signed by i. This secret key is used by i to send encrypted data to i using an $\mathsf{IND\text{-}CPA}$ symmetric key encryption scheme $\mathsf{SKE} = \{\mathsf{Encrypt}, \mathsf{Decrypt}\}$.
  Encrypted messages are sent over the broadcast channel in order to authenticate the sender. When j files a complaint against i, they reveal the $\mathsf{K}_{i\to j}$ along with the signature. This binds i to the data they have sent to j, and in case of misbehavior, revealing $\mathsf{K}_{i\to j}$ allows other parties to acknowledge the misbehavior.
  Concretely, signed pairwise keys can be easily established in a setup phase. It suffices that i generates a KEM ciphertext i encapsulating a KEM symmetric key, and uses this KEM symmetric key to encrypt $\mathsf{K}_{i\to j}$ and a signature $\mathsf{sig}_{i\to j} \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}_i, \{i, j, \mathsf{K}_{i\to j}\})$.

Our protocols are round-based, with one algorithm per round. Rounds are run sequentially and synchronously, with the returned value being posted to the broadcast channel.

5.1 Robust Distributed Key Generation (DKG)

We present a first application of our framework, allowing one to verifiably sample and share a short secret among parties, assuming that 2 out of 3 participants are honest. We apply it to the key generation of $\mathsf{Pelican}$ and prove that the resulting scheme remains robust and unforgeable.

In environments where the key generation cannot be entitled to a single trusted entity, it is important to provide the possibility to distribute the key generation process among several actors.

Distributed key generation description Our Distributed Key Generation was informally presented in Section 2.2. We now provide a formal description of its algorithms in Figure 5. Note that usual rounding techniques on the public key apply. We omit them in our description for simplicity. Introducing rounding only incurs a small loss in the SIS bound B_2, and requires introducing an infinite norm bound on c in the verification procedure.

As our DKG allows the adversary to partly bias the distribution of the secret key, it makes it very complex to define a secrecy notion for our DKG that is independent of the underlying primitive.

While the literature introduced zero-knowledge notions [KGS23] to abstract the unbiased behavior of a DKG, and leverage it in any threshold scheme, this does not easily generalize to our setting where the key generation allows some bias. It would be possible to define a generic notion with several simulators similar to $\mathsf{V3S\text{-}fk}$, but we concluded that the added complexity in the proofs was not worth the small gain in generality. Instead, we introduce in Theorem 4 new robustness and unforgeability notions for a signature scheme used in conjunction with a DKG. The unforgeability and robustness of our scheme is stated in Theorem 5 (resp. Theorem 4), which are proven in Appendix C.

Algorithm 7 $\mathsf{Pelican.ShareKeygen}_1(\mathsf{state}_i)$

```
1: assert{ $\mathsf{state.rnd} = \varnothing$ }; $\mathsf{state.rnd} = 1$
2: $\mathsf{salt}_i \xleftarrow{\$} \{0,1\}^\kappa$
3: $\mathbf{s}_i \leftarrow \mathcal{D}^2_{\sigma_t}$
4: $([\![\mathbf{s}_i]\!], \pi_i, (\pi_{i,j})_{j\in\{1,\ldots,N\}}) \leftarrow \mathsf{V3S.Share}(N, T, \mathbf{s}_i)$    ▷ Share $[\![\mathbf{s}_i]\!]$ and make proofs
5: if $\{\exists j$ s.t. $\mathsf{V3S.Verify}([\![\mathbf{s}_i]\!]_j, \pi_i, \pi_{i\to j}) = \mathsf{false}\}$ then {restart}    ▷ Invalid proof
6: for $j \in \{1,\ldots,N\}$ do
7:     $\mathsf{pt}_j := ([\![\mathbf{s}_i]\!]_j, \pi_{i,j})$
8:     $\mathsf{ct}_{i,j} \leftarrow \mathsf{SKE.Encrypt}(\mathsf{K}_{i\to j}, \mathsf{pt}_j)$    ▷ Encrypt (share, proof) to each party
9: return $\mathsf{contrib}_1[i] := \mathsf{salt}_i, \pi_i, (\mathsf{ct}_{i,j})_{j\in\{1,\ldots,N\}}$
```

Algorithm 8 $\mathsf{Pelican.ShareKeygen}_2(\mathsf{state}_i, \mathsf{contrib}_1)$

```
1: assert{ $\mathsf{state.rnd} = 1$ }; $\mathsf{state.rnd} = 2$;
2: $\mathsf{complaints}_i := \{\}$
3: for $(j \in \mathsf{contrib}_1)$ do    ▷ Set of round 1 contributors
4:     $\mathsf{salt}_j, \pi_j, (\mathsf{ct}_{j\to k})_{k\in\{1,\ldots,N\}} := \mathsf{contrib}_1[j]$
5:     $[\![\mathbf{s}_j]\!]_i, \pi_{j\to i} := \mathsf{SKE.Decrypt}(\mathsf{K}_{j\to i}, \mathsf{ct}_{j\to i})$
6:     if ($\mathsf{SKE.Decrypt}$ failed) or ($\mathsf{V3S.Verify}([\![\mathbf{s}_j]\!]_i, \pi_j, \pi_{j\to i}) = \mathsf{false}$) then
7:         $\mathsf{complaints}_i[j] = \{\mathsf{K}_{j\to i}, \mathsf{sig}_{j\to i}\}$    ▷ $i$ complains against $j$
8: $\mathsf{valid}_i = \{j \in \mathsf{contrib}_1\} \backslash \mathsf{complaints}_i$
9: $\mathsf{state}_i.\mathsf{session.valid} := \mathsf{valid}_i$
10: $\mathsf{state}_i.\mathsf{session.contrib}_1 := \mathsf{contrib}_1$
11: $\mathsf{state}_i.\mathsf{session.shares} := ([\![\mathbf{s}_j]\!]_i)_{j\in\mathsf{valid}_i}$
12: return $\mathsf{contrib}_2[i] := \mathsf{complaints}_i$
```

Algorithm 9 $\mathsf{Pelican.ShareKeygen}_3(\mathsf{state}_i, \mathsf{contrib}_2)$

```
1: assert{ $\mathsf{state.rnd} = 2$ }; $\mathsf{state.rnd} = 3$
2: $\mathsf{valid}_i := \mathsf{state}_i.\mathsf{session.valid} \cap \{j \in \mathsf{contrib}_2\}$
3: for $j \in \{1,\ldots,N\}$ do
4:     for $k \in \mathsf{complaints}_j$ do    ▷ Lines 5 to 9 study $j$'s complaint against $k$
5:         $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\} := \mathsf{complaints}_j[k]$
6:         if $\{\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}) = \mathsf{false}\}$ then {continue}
7:         $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j})$
8:         if ($\mathsf{SKE.Decrypt}$ failed) or $\mathsf{V3S.Verify}([\![\mathbf{s}_k]\!]_j, \pi_k, \pi_{k\to j}) = \mathsf{false}$ then
9:             $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$    ▷ If $j$'s complaint against $k$ is valid, invalidate $j$
10: $\mathsf{state}_i.\mathsf{session.salt} := H_{\mathsf{salt}}((\mathsf{salt}_j)_{j\in\mathsf{valid}_i})$
11: $a := H_a(\mathsf{state}_i.\mathsf{session.salt}) \in \mathcal{R}_q$    ▷ Derive the public key $a \in \mathcal{R}_q$ from a seed
12: $[\![\mathbf{s}]\!]_i := \sum_{j\in\mathsf{valid}_i} [\![\mathbf{s}_j]\!]_i$    ▷ $\mathbf{s} = \sum_{j\in\mathsf{valid}} \mathbf{s}_j$
13: $[\![b]\!]_i := \beta - \left[1\ a\right] \cdot [\![\mathbf{s}]\!]_i$
14: Store $[\![\mathbf{s}]\!]_i$ in $\mathsf{state}_i$
15: return $\mathsf{contrib}_3[i] := (\mathsf{state}_i.\mathsf{session.salt}, [\![b]\!]_i)$
```

Algorithm 10 $\mathsf{Pelican.CombineKey}(\mathsf{contrib}_3)$

```
1: Retrieve salt from $\mathsf{contrib}_3$    ▷ If contradictory contributions use majority vote
2: $b := \mathsf{V3S.RobustReconstruct}([\![b]\!]_i, \ldots, [\![b]\!]_N)$
3: assert{ $b \neq \perp$ }
4: return $\mathsf{vk} := (\mathsf{salt}, b)$
```

Fig. 5: Algorithms for the distributed Keygen. For conciseness, we omit the parsing of $(\mathsf{contrib}_i)_{i\in\{1,2,3\}}$ in Algorithms 8 to 10.

Definition 9 (Unforgeability and robustness with DKG). *A threshold signature scheme with DKG* $(\mathsf{ShareKeygen}_i)_{i\in\{1,\dots,\mathsf{rnd}_{\mathsf{Keygen}}\}}$, $(\mathsf{ShareSign}_i)_{i\in\{1,\dots,\mathsf{rnd}_{\mathsf{sig}}\}}$, CombineKey, Combine *is unforgeable (resp. robust) if all probabilistic polynomial-time adversaries* $\mathcal{A}$ *win the game* $\mathsf{Game}_{\mathsf{DKG\text{-}TH\text{-}UF}}$ *(resp.* $\mathsf{Game}_{\mathsf{DKG\text{-}TH\text{-}RB}}$*) from Figure 30 with negligible probability.*

Theorem 4. *Consider* $\mathcal{K}$ *a distribution of keys of the form* $\sum_j \mathbf{s}_j$ *such that:*
1. *at most* $(T-1)$ *vectors* $\mathbf{s}_j$ *have norm bounded by* B*;*
2. *the other vectors* $\mathbf{s}_j$ *are sampled from* $\mathcal{D}^2_{\mathsf{sk}}$.

Assume that we have $\|(z_1, z_2, c_1)\| \leqslant B_2$ *with overwhelming probability* p *for any set* valid, $\mathbf{p} = \sum_{j\in\mathsf{valid}} \mathbf{p}_j$ *with (i) at most* $T-1$ *arbitrary perturbations of norm bounded by* B*, (ii) the others sampled from Gaussians of standard deviation* $\sigma_{\mathbf{p}}$*, and over any distribution of keys* $\mathcal{K}$.

The advantage of any polynomial-time adversary $\mathcal{A}$ *in the game* DKG-TH-RB *from Figure 29 reduces to the robustness of* Pelican *with a small advantage loss. Formally, there exist adversaries* $\mathcal{B}_1$ *against the* UF-CMA *security of* SIG, $\mathcal{B}_2$ *against the* V3S-sound *security of* V3S, $\mathcal{B}_3$ *against the* Leak-TH-RB *security of* Pelican*, with running time* $\mathcal{T}_{\mathcal{B}_1} \approx \mathcal{T}_{\mathcal{B}_2} \approx \mathcal{T}_{\mathcal{B}_3} \approx \mathcal{T}_{\mathcal{A}}$.

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DKG\text{-}TH\text{-}RB}}(\kappa) \leqslant & N \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{V3S\text{-}sound}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{Leak\text{-}TH\text{-}RB}}(\kappa) \\ & + N \cdot (1 - \Pr[\mathsf{V3S}\ \textit{correctness}])\end{aligned}$$

Theorem 5. *Define* σ'_{sk} *as* $\frac{1}{\sigma'^2_{\mathsf{sk}}} = 2\left(\frac{1}{\sigma^2_{\mathsf{sk}}} + \frac{B}{\sigma^2_y}\right)$*, with* B *such that* $\Pr\left[s_1(\mathbf{R}^\top\mathbf{R}) \leqslant B\right]$ *with overwhelming probability and* $\sigma'_{\mathsf{sk}} \geqslant \sqrt{2}\eta_\varepsilon(\mathbb{Z}^{2n})$.

The advantage of any polynomial-time adversary $\mathcal{A}$ *in the game* DKG-TH-UF *from Figure 30 reduces to the unforgeability of* Pelican *with a small advantage loss. Formally, there exist adversaries* $\mathcal{B}_1$ *against the* UF-CMA *security of* SIG, $\mathcal{B}_2$ *against the* IND-CPA *security of* SKE, $\mathcal{B}_3$ *against the* V3S-sound *security of* V3S, $\mathcal{B}_4$ *against the* V3S-fk *security of* V3S, $\mathcal{B}_5$ *against the* Leak-TH-UF *security of* Pelican *with* $\sigma_t = \sigma'_{\mathsf{sk}}$*, with running time* $\mathcal{T}_{\mathcal{B}_i} \approx \mathcal{T}_{\mathcal{A}}$ *for* $i \in \{1, \dots, 5\}$.

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DKG\text{-}TH\text{-}UF}}(\kappa) \leqslant & N \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa) + N^2 \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{IND\text{-}CPA}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{V3S\text{-}sound}}(\kappa) \\ & + N \cdot \mathsf{Adv}_{\mathcal{B}_4}^{\mathsf{V3S\text{-}fk}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_5}^{\mathsf{Leak\text{-}TH\text{-}UF}}(\kappa) \\ & + 4N \cdot \varepsilon + N \cdot \Pr[\mathsf{V3S}\ \textit{correctness}]\end{aligned}$$

5.2 Robust Distributed Signing Procedure

We present as a second application of our result an efficient lattice-based robust threshold signature based on standard assumptions . It provides additional guarantees over the existing threshold schemes such as threshold Raccoon [PKM+24] which provides unforgeability, but no guarantee of termination.

Assuming the existence of an authenticated reliable *broadcast channel*, and that at least 2/3 of the parties are honest, it ensures that the signature protocol will produce a valid signature.

5.2.1 Description of Pelican Pelican was drafted in the Technical Overview, in Section 2.3. It relies on a symmetric encryption scheme SKE, a signature scheme SIG, and a Verifiable Short Secret Sharing V3S.

Our informal description of Pelican easily translates to formal algorithms, described in Figures 6 and 7.

Robustness of Pelican. The *robustness* property of Pelican lies in its ability to generate signatures even in the presence of malicious signers. We prove it when Pelican is used over a broadcast channel by following the methodology of [KGS23] with successive rounds. The robustness of Pelican is stated in Theorem 6, which is proven in Appendix B.1.

Theorem 6. *Assume that selected parameters are such that we have $\|(z_1, z_2, c_1)\| \leqslant B_2$ with overwhelming probability p for any set* valid, $\mathbf{p} = \sum_{j \in \mathsf{valid}} \mathbf{p}_j$ *with (i) at most $T-1$ arbitrary perturbations of norm bounded by B, (ii) the others sampled from Gaussians of standard deviation $\sigma_{\mathbf{p}}$, and over the distribution of keys. Recall that* $(z_1, z_2, c_1) = \begin{bmatrix} c_1 \cdot \mathbf{s} + \mathbf{p} \\ c_1 \end{bmatrix} + \begin{bmatrix} c_2 \\ 0 \\ 0 \end{bmatrix}$.

The advantage of any polynomial-time adversary $\mathcal{A}$ making at most Q_s signing queries in the game TH-RB *from Figure 16 is bounded by*

$$N \cdot \mathsf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{B}_1}(\kappa) + \mathsf{Adv}^{\mathsf{V3S\text{-}sound}}_{\mathcal{B}_2}(\kappa) + NQ_s \cdot (1 - \Pr[\mathsf{V3S}\ \textit{correctness}]) + (1-p)$$

where $\mathcal{B}_1$ is an adversary against the UF-CMA *security of* SIG, *$\mathcal{B}_2$ is an adversary against the* V3S-sound *security of* V3S, *with running time $\mathcal{T}_{\mathcal{B}_1} \approx \mathcal{T}_{\mathcal{B}_2} \approx \mathcal{T}_{\mathcal{A}}$.*

Unforgeability of Pelican. Importantly, we also want our scheme to remain unforgeable in the presence of dishonest signers. The unforgeability of our scheme is stated in Theorem 7, which is proven in Appendix B.2.

Theorem 7. *Define $\sigma'_{\mathbf{p}}$ as $\frac{1}{\sigma'^2_{\mathbf{p}}} = 2\left(\frac{1}{\sigma^2_{\mathbf{p}}} + \frac{B}{\sigma^2_y}\right)$, with B such that $s_1(\mathbf{R}^\top \mathbf{R}) \leqslant B$ with overwhelming probability and $\sigma'_{\mathbf{p}} \geqslant \sqrt{2}\eta_\varepsilon(\mathbb{Z}^{2n})$.*

Pelican *is* TH-UF *secure in the random oracle model.*

Formally, let $\mathcal{A}$ be an adversary against the TH-UF *security game from Figure 21 starting at most Q_s signing sessions, and making at most Q_H random oracles queries.*

Then there exists adversaries $\mathcal{B}_1$ against the UF-CMA *security of* SIG, *$\mathcal{B}_2$ against the* IND-CPA *security of* SKE, *$\mathcal{B}_3$ against the* V3S-sound *security of* V3S, *$\mathcal{B}_4$ against the* V3S-fk *security of* V3S, *$\mathcal{B}_5$ against* $\mathsf{Hint\text{-}RLWE}_{q,Q_{\mathsf{Sign}},\mathcal{D}_{\sigma_t},\mathcal{D}_{\sigma'_{\mathbf{p}}},\mathcal{C}_1}$, *$\mathcal{B}_6$ against* $\mathsf{RLWE}_{q,\mathcal{U}([-B_2/\sqrt{2n},B_2/\sqrt{2n}]^n)^2}$, *$\mathcal{B}_7$ against* $\mathsf{RSIS}_{q,2,2B_2}$, *and running time $\mathcal{T}_{\mathcal{B}_i} \approx \mathcal{T}_{\mathcal{A}}$ for $i \in \{1, \ldots, 7\}$, such that*

Algorithm 11 $\mathsf{Pelican.ShareSign}_1(\mathsf{state}_i, \mathsf{sid}, \mathsf{msg})$

```
assert{ $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$ does not exist }    ▷
$\mathbf{p}_i \leftarrow \mathcal{D}_{\mathbf{p}}^2$
$[\![\mathbf{p}_i]\!], \pi_i, (\pi_{i\to j})_{j\in\{1,\ldots,N\}} \leftarrow \mathsf{V3S.Share}(N, T, \mathbf{p}_i)$
if $\exists j$ s.t. $\mathsf{V3S.Verify}([\![\mathbf{p}_j]\!]_i, \pi_j, \pi_{j\to i}) = \mathsf{false}$ then
    restart    ▷ Restart if the V3S produced an invalid proof
for $j \in \{1, \ldots, N\}$ do
    $\mathsf{ct}_{i\to j} \leftarrow \mathsf{SKE.Encrypt}(\mathsf{K}_{i\to j}, ([\![\mathbf{p}_i]\!]_j, \pi_{i\to j}))$
$\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{\mathsf{rnd} = 1, \mathsf{msg}, [\![\mathbf{p}_i]\!], \varnothing, \varnothing\}$
return $\mathsf{contrib}_1[i] := (\pi_i, (\mathsf{ct}_{i\to j})_{j\in\{1,\ldots,N\}})$
```

Algorithm 12 $\mathsf{Pelican.ShareSign}_2(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_1)$

```
Fetch $(\mathsf{rnd}, \mathsf{msg})$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
assert{ $\mathsf{rnd} = 1$ }    ▷
$\mathsf{complaints}_i := \{\}$
for $(j \in \mathsf{contrib}_1)$ do
    $\pi_j, (\mathsf{ct}_{j\to k})_{k\in\{1,\ldots,N\}} := \mathsf{contrib}_1[j]$
    $[\![\mathbf{p}_j]\!]_i, \pi_{j\to i} := \mathsf{SKE.Decrypt}(\mathsf{K}_{j\to i}, \mathsf{ct}_{j\to i})$
    if ($\mathsf{SKE.Decrypt}$ failed) or ($\mathsf{V3S.Verify}([\![\mathbf{p}_j]\!]_i, \pi_j, \pi_{j\to i}) = \mathsf{false}$) then
        $\mathsf{complaints}_i[j] = \{\mathsf{K}_{j\to i}, \mathsf{sig}_{j\to i}\}$    ▷ $j$'s ciphertext or proof is invalid
$\mathsf{valid}_i = \{j \in \mathsf{contrib}_1\} \backslash \mathsf{complaints}_i$
$\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{\mathsf{rnd} = 2, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1\}$
return $\mathsf{contrib}_2[i] := \mathsf{complaints}_i$
```

Algorithm 13 $\mathsf{Pelican.ShareSign}_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$

```
1:  Fetch $(\mathsf{rnd}, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1)$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
2:  assert{ $\mathsf{rnd} = 2$ }
3:  $\mathsf{valid}_i := \mathsf{valid}_i \cap \{j \in \mathsf{contrib}_2\}$
4:  for $j \in \{1, \ldots, N\}$ do
5:      for $k \in \mathsf{complaints}_j$ do    ▷ Lines 6 to 11 study $j$'s complaint against $k$
6:          $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\} := \mathsf{complaints}_j[k]$
7:          if $\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}) = \mathsf{false}$ then    ▷ See Section 5
8:              continue
9:          $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j})$
10:         if ($\mathsf{SKE.Decrypt}$ failed) or $\mathsf{V3S.Verify}([\![\mathbf{p}_k]\!]_j, \pi_k, \pi_{k\to j}) = \mathsf{false}$ then
11:             $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$
12: $[\![\mathbf{p}]\!]_i := \sum_{j\in\mathsf{valid}_i} [\![\mathbf{p}_j]\!]_i$    ▷ $\mathbf{p} = (p_1, p_2)$
13: $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{3, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1\}$
14: return $\mathsf{contrib}_3[i] := ([\![w]\!]_i := \mathbf{A} \cdot [\![\mathbf{p}]\!]_i)$    ▷ $\mathbf{A} = [1\ a\ b]$
```

Fig. 6: Algorithms for robust threshold signature, part 1/2. For conciseness, we omit the parsing of $(\mathsf{contrib}_i)_{i\in\{1,2\}}$ in Algorithms 12 and 13.

Algorithm 14 $\mathsf{Pelican.ShareSign}_4(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_3)$

```
1: Fetch $(\mathsf{rnd}, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j \in \mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1)$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
2: assert{ $\mathsf{rnd} = 3$ }
3: Parse $\mathsf{contrib}_3 = ([\![w]\!]_j)_j$
4: $w := \mathsf{V3S.RobustReconstruct}(([\![w]\!]_j)_{j \in \mathsf{valid}_i})$    ▷ $w = \mathbf{A} \cdot \mathbf{p}$, where $\mathbf{p} = \sum_{j \in \mathsf{valid}_i} \mathbf{p}_j$
5: $\mathsf{salt} := H_{\mathsf{salt}}(w)$
6: $u := H_u(\mathsf{vk}, \mathsf{salt}, \mathsf{msg}) \in \mathcal{R}_q$    ▷ Hash the message msg to a random $u \in \mathcal{R}_q$
7: $c := u - w$
8: $(c_1, c_2) := \mathsf{Decompose}_\beta(c)$    ▷ Recall $c = \beta \cdot c_1 + c_2$
9: $[\![z]\!]_i := c_1 \cdot [\![s]\!]_i + [\![p_2]\!]_i$    ▷ Recall $\mathbf{s} = (e, s)$ and $\mathbf{p} = (p_1, p_2)$
10: $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \varnothing$
11: return $\mathsf{contrib}_4[i] = (\mathsf{salt}, [\![z]\!]_i, c_1)$
```

Algorithm 15 $\mathsf{Pelican.Combine}(\mathsf{vk}, \mathsf{msg}, \mathsf{contrib}_4)$

```
1: Parse $\mathsf{contrib}_4 = (\mathsf{salt}_j, [\![z]\!]_j, c_1)_j$
2: Retrieve $\mathsf{salt}, c_1$ from $\mathsf{contrib}_3$    ▷ If contradictory contributions: majority vote
3: $u = H_u(\mathsf{vk}, \mathsf{salt}, \mathsf{msg}) \in \mathcal{R}_q$
4: $z := \mathsf{V3S.RobustReconstruct}(([\![z]\!]_j)_{j \in \mathsf{contrib}_4})$
5: return $\mathsf{sig} := (\mathsf{salt}, z, c_1)$
```

Algorithm 16 $\mathsf{Pelican.Verify}(\mathsf{vk} = (\mathsf{seed}, b), \mathsf{msg}, \mathsf{sig} = (\mathsf{salt}, z, c_1))$

```
1: $a := H_a(\mathsf{seed}) \in \mathcal{R}_q$
2: $u := H_u(\mathsf{vk}, \mathsf{salt}, \mathsf{msg}) \in \mathcal{R}_q$
3: $z_2 := z$
4: $z_1 := u - a \cdot z_2 - b \cdot c_1$
5: return $||(z_1, z_2, c_1)|| \leqslant B_2$
```

Fig. 7: Algorithms for robust threshold signature, part 2/2. For conciseness, we omit the parsing of $(\mathsf{contrib}_i)_{i \in \{1,3,4\}}$, vk and sig in Algorithms 14 to 16.

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{TH\text{-}UF}}_{\mathcal{A}}(\kappa) \leqslant & N \cdot \mathsf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{B}_1}(\kappa) + N^2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{B}_2}(\kappa) + \mathsf{Adv}^{\mathsf{V3S\text{-}sound}}_{\mathcal{B}_3}(\kappa) \\
& + Q_s \cdot N \cdot \mathsf{Adv}^{\mathsf{V3S\text{-}fk}}_{\mathcal{B}_4}(\kappa) + N \cdot Q_s \cdot (1 - \Pr[\mathsf{V3S}\ \textit{correctness}]) \\
& + \mathsf{Adv}^{\mathsf{Hint\text{-}RLWE}}_{\mathcal{B}_5}(\kappa) + Q_H \cdot \mathsf{Adv}^{\mathsf{RLWE}}_{\mathcal{B}_6}(\kappa) + \mathsf{Adv}^{\mathsf{RSIS}}_{\mathcal{B}_7}(\kappa) \\
& + \frac{Q_s}{2^{2\kappa}} + 4 Q_s N \cdot \varepsilon + Q_s \cdot 2^{-n+2} + p_c
\end{aligned}$$

for some $p_c \leqslant 2^{-n(2\log_2(2B_2/\sqrt{2n}) - \log_2(q))}$.

6 Parameter Selection and Instantiation

We now move on to instantiation of our constructions. The main component required is a proper distribution of matrices $\mathbf{R}$ verifying property G from Definition 8. We chose to leverage matrices $\mathbf{R} \in \{0, \pm 1\}^{256 \times 2n}$ where each coefficient of $\mathbf{R}$ is 0 with probability 1/2, and ± 1 with probability 1/4. They have strong distribution properties, and have been already successfully applied in [GHL22,Ngu22]. We note this distribution $\mathcal{D}_{\mathbf{R}}$. We will rely on Lemmas 2 to 4. Proofs are included in Appendix E.

Lemma 2 (Large Norm Detection, Lemma 3.2.5 from [Ngu22]). *Fix $n, q \in \mathbb{N}$, and a bound $b \leqslant q/(82n)$, and let $\mathbf{s} \in [\pm q/2]^{2n}$, with $\|\mathbf{s}\|_2 \geqslant b$. Let $\mathbf{y} \in [\pm q/2]^{256}$. Then $\Pr_{\mathbf{R} \leftarrow \mathcal{D}_{\mathbf{R}}} \left[\|\mathbf{R} \cdot \mathbf{s} + \mathbf{y} \bmod q\|_2 < \frac{1}{2} b\sqrt{26} \right] < 2^{-128}$.*

Lemma 3 (separation). *Fix $n, q \in \mathbb{N}$, and $(\mathbf{s}, \mathbf{y}) \neq (\mathbf{s}', \mathbf{y}') \in [\pm q/2]^{2n+256}$. Then $\Pr_{\mathbf{R} \leftarrow \mathcal{D}_{\mathbf{R}}} [\mathbf{R} \cdot \mathbf{s} + \mathbf{y} = \mathbf{R} \cdot \mathbf{s}' + \mathbf{y}' \bmod q] \leqslant 2^{-256}$.*

Lemma 4 (small spectral norm). *Fix $n, q \in \mathbb{N}$ and $\mathbf{R} \in \{0, \pm 1\}^{256 \times 2n}$. $s_1(\mathbf{R}^\top \mathbf{R}) \leqslant 512 \cdot n$. For fixed values of n, we can obtain better average bounds. For $n = 2048$,*

$$\Pr_{\mathbf{R} \leftarrow \mathcal{D}_{\mathbf{R}}} \left[s_1(\mathbf{R}^\top \mathbf{R}) < 20096 \right] \geqslant 1 - 2^{-142}$$

6.1 Reminder: Parameter Selection in Plover

In the (standard) signature scheme Plover, a signature is essentially a pair $(\mathsf{salt}, \mathbf{z} = (z_1, z_2, c_1)^t)$, where $\mathbf{z}$ is of the form:

$$\mathbf{z} = \begin{bmatrix} e \\ s \\ 1 \end{bmatrix} \cdot c_1 + \begin{bmatrix} p_1 \\ p_2 \\ 0 \end{bmatrix} + \begin{bmatrix} c_2 \\ 0 \\ 0 \end{bmatrix}$$

One can see that $\begin{bmatrix} 1 & a & b \end{bmatrix} \cdot \mathbf{z} = H(\mathsf{vk}, \mathsf{salt}, \mathsf{msg})$. Following the analysis of [EEN+24, Section 3.5], and assuming $\sigma_{\mathsf{sk}} = \frac{\beta^2}{q\sqrt{2\,n}}$:

$$\mathbb{E}\left[\|\mathbf{z}\|^2\right] \approx n\left(2\,\sigma_{\mathbf{p}}^2 + \frac{\beta^2}{12} + \frac{q^2\,n}{6\,\beta^2}\,\sigma_{\mathsf{sk}}^2\right) \approx 2\,n\left(\sigma_{\mathbf{p}}^2 + \frac{\beta^2}{12}\right)$$

Following the security reduction of [EEN+24], Plover is unforgeable under the $\mathsf{RSIS}_{q,\ell,B_2}$ and $\mathsf{Hint\text{-}RLWE}_{q,Q_{\mathsf{Sign}},\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathbf{p}},\mathcal{C}_1}$ assumptions. In our case, all distributions are Gaussians, so that $\mathsf{Hint\text{-}RLWE}_{q,Q_{\mathsf{Sign}},\sigma_{\mathsf{sk}},\sigma_{\mathbf{p}},\mathcal{C}_1} \geqslant \mathsf{RLWE}_{q,\sigma_{\mathsf{red}}}$, where $\frac{1}{\sigma_{\mathsf{red}}^2} = 2\left(\frac{1}{\sigma_{\mathsf{sk}}^2} + \frac{B_{\mathsf{HRLWE}}}{\sigma_{\mathbf{p}}^2}\right) \approx \frac{2B_{\mathsf{HRLWE}}}{\sigma_{\mathbf{p}}^2}$, where $B_{\mathsf{HRLWE}} \approx \frac{n\,Q_{\mathsf{Sign}}\,q^2}{12\,\beta^2}$, see Lemma 2 in [EEN+24]. If $\sigma_{\mathbf{p}} = o(\beta\sqrt{Q_{\mathsf{Sign}}})$, then $\sigma_{\mathsf{sk}} = \omega(\frac{\sigma_{\mathbf{p}}}{\sqrt{B_{\mathsf{HRLWE}}}})$ and therefore $\sigma_{\mathsf{red}} \sim \frac{\sigma_{\mathbf{p}}}{\sqrt{2 \cdot B_{\mathsf{HRLWE}}}}$.

6.2 Parameter Selection for the Signing Procedure of Pelican

Parameter selection for Pelican is much more involved than for Plover. Note that a parameter set defined for a threshold T also supports any $1 \leqslant T' \leqslant T$ with the same security and verification procedure. We thus limit our analysis to an upper bound T on supported thresholds, while ensuring that Pelican is

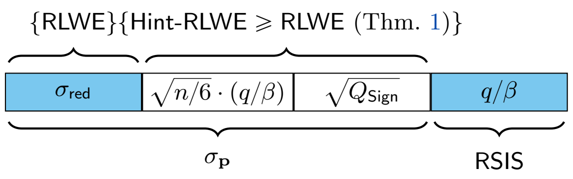
{RLWE}{Hint-RLWE ⩾ RLWE (Thm. 1)}
σ_{red}
$\sqrt{n/6} \cdot (q/\beta)$
$\sqrt{Q_{\mathsf{Sign}}}$
q/β
$\sigma_{\mathbf{p}}$
RSIS

Fig. 8: Constraints on the modulus q in Plover [EEN+24]. We represent q in logarithmic scale. Each constraint adds an overhead to the mimimum size of q. As a rule of thumb, dimensions must scale in $\tilde{O}(\log q)$ for security, and as a result the cryptosystem's bitsizes scale in $\tilde{O}((\log q)^2)$.

interchangeable[6]. It is achieved by increasing σ_{sk} and $\sigma_{\mathbf{p}}$ by a factor $\sqrt{T/T'}$. Parameter selection for Pelican can be fairly delicate since we need to balance several (sometimes conflicting) conditions. Therefore we present in Table 1 the relationship between parameters (modulus, dimensions, etc.) and security metrics (RLWE, RSIS, etc.).

Parameter	RLWE	{Hint-RLWE ⩾ RLWE}	RSIS	V3S Slack	Public key bitsize	Signature bitsize
q	↘	≈	↗	≈	↘	≈
n	↗	↘	↗	≈	↘	↘
σ_{sk}	↗	≈	≈	≈	↗	≈
$\sigma_{\mathbf{p}}$	↗	↗	↘	≈	≈	↘
q/β	≈	↘	↗	≈	≈	↘
T	≈	↗	↗	↗	≈	↘
Q_{Sign}	≈	↘	≈	≈	≈	↘

Table 1: Relationship between the parameters of Pelican and the associated security and efficiency metrics. For the cell in the X-th row and Y-th column, we use the symbol ↗ (resp. ↘, resp. ≈) to indicate that increasing the parameter X has a positive (resp. negative, resp. limited) influence on the metric Y. Note that the signature size increases logarithmically in the maximum values of T and Q_{Sign}, but the public key size can be made essentially independent of them using appropriate bit-dropping, see Section 6.4.

Let us fix an upper bound T on supported thresholds. For simplicity, we assume here that the set of signers S satisfies $|S| = 3T$;[7], our analysis is easily

[6] Interchangeability guarantees that a threshold scheme has static public parameters for any supported threshold and number of parties.

[7] If $|S| < 2T$, then we still have unforgeability but the V3S can no longer detect dishonest users. Taking $|S| > 3T$ is also of limited interest since it cannot detect more than $T - 1$ dishonest users.

extended to other values of $|S|$. For our choice of S, at least $2/3$ of the users are honest, so that the vector $\mathbf{p} = \sum_{i \in S} \mathbf{p}_i$ contains at least $2\,T$ vectors that are unknown to the set of corrupted users; as a benefit, the Hint-RLWE reduction can support $\sqrt{2\,T}$ more queries.

The signing procedure of Pelican (Figs. 6 and 7) provides, for each $(\mathbf{p}_i)_{i \in [Q_{\mathsf{Sign}}]}$, a V3S proof which contains $\mathbf{v}_i = \mathbf{R} \cdot \mathbf{p}_i + \mathbf{y}_i$. This has two consequences:

1. The V3S proves a slightly looser bound on the norm of $\mathbf{p}_i$.
2. The pair $(\mathbf{R}_i, \mathbf{v}_i = \mathbf{R} \cdot \mathbf{p}_i + \mathbf{y}_i, i)$ biases the conditional distribution of $\mathbf{p}_i$.

Slack of the V3S. Lemma 2 tells us that $\frac{\sqrt{26}}{2}\|\mathbf{p}_i\| \leqslant \|\mathbf{v}_i\|$ with overwhelming probability. On the other hand, [Ngu22, Lemma 3.2.4] tells us:

$$\|\mathbf{v}_i\| \leqslant \|\mathbf{R}_i \cdot \mathbf{p} + \mathbf{y}\| \leqslant \sqrt{337} \cdot \|\mathbf{p}\| + \|\mathbf{y}\|, \tag{5}$$

where τ is a value such that $\mathbb{P}_{\mathbf{x} \leftarrow \mathcal{D}^{256}_{\sigma_\mathbf{p}}}\left[\|\mathbf{x}\| \geqslant \tau \sigma_\mathbf{p} \sqrt{256}\right] \leqslant 2^{-128}$, here $\tau \leqslant 1.4$. If we set $\sigma_\mathbf{y} = \sqrt{337} \cdot \sigma_\mathbf{p}$ and note $\textsc{Slack}_{\mathsf{V3S}} = \tau\sqrt{337}\,\frac{2}{\sqrt{26}} \approx 10$, our V3S proves:

$$\|\mathbf{p}_i\| \leqslant \textsc{Slack}_{\mathsf{V3S}} \cdot (\sigma_\mathbf{p}\sqrt{2\,n}) \tag{6}$$

There is a gap $\textsc{Slack}_{\mathsf{V3S}}$ between the expected norm of $\mathbf{p}_i$, which is $\sigma_\mathbf{p}\sqrt{2\,n}$, and the norm that is actually proven, which is the one in Eq. (6). While $\textsc{Slack}_{\mathsf{V3S}}$ is constant due to the lemmas we use, increasing the security parameter κ will also increase $\textsc{Slack}_{\mathsf{V3S}}$. In order to have robustness, we need to use the latter norm in our verification procedure. In addition, S perturbations $\mathbf{p}_i$ are added in the signature and this needs to be taken into account. Therefore we pick this bound on the norm in the verification procedure:

$$B_2 = |S| \cdot \textsc{Slack}_{\mathsf{V3S}} \cdot (\sigma_\mathbf{p}\sqrt{2\,n}) \tag{7}$$

Leakage from the V3S. The pair $(\mathbf{R}_i, \mathbf{v}_i = \mathbf{R}_i \cdot \mathbf{p}_i + \mathbf{y}_i)$ is a hint on the value of $\mathbf{p}_i$. More precisely, Lemma 1 tells us that the conditional distribution of $\mathbf{p}_i$ conditioned on $(\mathbf{R}_i, \mathbf{v}_i)$ follows a (non-spherical) Gaussian $D_{\mathcal{R}^2, \mathbf{c}, \sqrt{\mathbf{\Sigma}_0}}$, where:

$$\mathbf{\Sigma}_0^{-1} := \sigma_\mathbf{p}^{-2} \cdot \mathbf{I} + \sigma_\mathbf{y}^{-2} \cdot \mathbf{R}_i^\top \mathbf{R}_i \tag{8}$$

Theorem 7 tells us that we can safely use an isotropic Gaussian from each $\mathbf{p}_i$ of standard deviation $\sigma'_\mathbf{p}$ verifying $\frac{1}{\sigma'^2_\mathbf{p}} = \frac{1}{\sigma^2_\mathbf{p}} + \frac{s_1(\mathbf{R}_i^\top \mathbf{R}_i)}{\sigma_y^2}$ (we ignore the factor 2 which could be removed in the proof using Theorem 2).

Recall that according to Lemma 4, for $n = 2048$, we have $s_1(\mathbf{R}_i^\top \mathbf{R}_i) < 20096$ with overwhelming probability. Combining this with Eq. (8) and the fact that $\sigma_\mathbf{y} = \sqrt{337} \cdot \sigma_\mathbf{p}$, the usable part of perturbations $\mathbf{p}_i$ for $n = 2048$ has a standard deviation $\sigma'_\mathbf{p} = \Theta(\sigma_\mathbf{p})$ since:

$$\sigma'_\mathbf{p} := \left(\sigma_\mathbf{p}^{-2} + \sigma_\mathbf{y}^{-2} \cdot 20096\right)^{-1/2} \approx \sqrt{\frac{337}{20096}} \cdot \sigma_\mathbf{p} \approx \frac{1}{\sqrt{60}} \cdot \sigma_\mathbf{p} \tag{9}$$

Reduction from Hint-RLWE *to* RLWE. Hints on the secrets are masked with the sum of the honest perturbations. The resulting perturbation has standard deviation $\sqrt{|\mathsf{honest}|} \cdot \sigma'_{\mathbf{p}}$, where $|\mathsf{honest}| \geqslant \frac{2}{3}|S| = 2T$. The analysis of Plover thus adapts with $\sigma_{\mathsf{red}} \sim \frac{\sqrt{T} \cdot \sigma'_{\mathbf{p}}}{\sqrt{2 \cdot B_{\mathsf{HRLWE}}}}$.

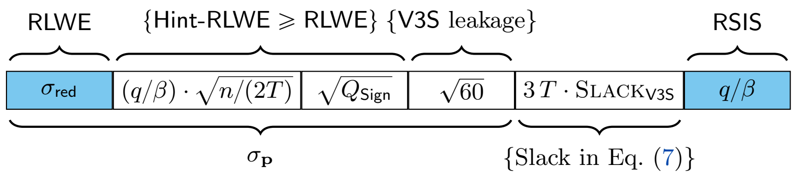

Fig. 9: Constraints on the modulus q in Pelican. We represent q in logarithmic scale. Each constraint adds an overhead to the mimimum size of q.

6.3 Distributed Key Generation

Parameter selection for our DKG is analogous to the signing procedure of Pelican. Similarly, the resulting σ_{sk} usable in the Hint-MLWE reduction after revealing $\mathbf{R}_i \cdot \mathbf{s}_i + \mathbf{y}_i$ is $\sigma'_{\mathsf{sk}} := \frac{\sqrt{2T}}{\sqrt{60}} \sigma_{\mathsf{sk}}$.

However, since $\sigma_{\mathbf{p}}$ was selected so that $s_1(\mathbf{R}) \cdot \sigma_{\mathsf{sk}}$ is negligible compared to $\sigma_{\mathbf{p}}$, the norm of $\mathbf{R} \cdot \mathbf{s}$ remains negligible compared to the norm of $\mathbf{p}$ even with the slack of the V3S on the bound on $\|\mathbf{s}\|$, and the norm of the signatures produced by our scheme is not affected by our V3S. Hence, we simply need to ensure that Hint-MLWE remains hard with σ'_{sk}.

6.4 Selected parameter sets

We rely on the lattice estimator [APS15] – an open-source tool available at `https://github.com/malb/lattice-estimator` – for estimating the concrete hardness of RLWE. We provide three possible parameter sets in Table 2, providing different tradeoffs between security, size, and the maximum supported threshold.

Note that in order to use Lemma 2, we need to fulfill the condition $b \leqslant q/(82n)$. It would be inefficient to increase q in the entire scheme for this, and we instead introduce q_{V3S} a multiple of q, for use during V3S secret sharing. Local shares are later reduced mod q before use in round 3 and 4.

We express the communication cost per party in round 1 as a function of T. It increases linearly with T as our V3S broadcasts an entire sharing. The total communication cost of our protocol hence evolves quadratically T. We recall that the bytesizes of vk and sig are computed as:

$$8 \cdot |\mathsf{vk}| = 2 \cdot \kappa + n \lceil \log_2 q \rceil \tag{10}$$

$$8 \cdot |\mathsf{sig}| \approx 2 \cdot \kappa + n \lceil \log_2 (T \cdot \sigma_{\mathbf{p}}) \rceil + n \lceil \log_2 (q/\beta) \rceil \tag{11}$$

This omits the optimization that consists of performing bit-dropping in vk. The formula for $|\mathsf{sig}|$ is approximate since it depends on the exact encoding used for the vector z.

κ	n	$\log \beta$	$\lfloor q_{\mathsf{V3S}} \rfloor$	T	$\sigma_{\mathbf{p}}$	Q_{Sign}	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{sig}\rvert$	Hint-RLWE C/Q	RSIS C/Q	comm. per party rnd_1	rnd_3	rnd_4
128	2048	43	2^{65}	16	2^{32}	2^{36}	12.8	12.3	113/99	113/99	$56T$	12.8	14
192	4096	43	2^{68}	1024	2^{31}	2^{48}	25.6	26.4	227/199	222/195	$115T$	25.6	28.2
256	4096	45	2^{70}	64	2^{33}	2^{48}	25.6	25.1	251/220	249/219	$119T$	25.6	27.4

Table 2: Parameter sets for Pelican. We showcase parameter sets for different security levels κ, and value of T. We take $N = 3T$, $q \approx 2^{50}$ and $\sigma_{\mathsf{sk}} \approx 2^{20}$. All sizes are in kilobytes (kB). We indicate the core-SVP hardness of Hint-RLWE and RSIS, a metric which ignores several polynomial factors. We also give approximate communication cost per participant in each round of key generation and threshold signature, excluding authentication of broadcast communication. Note that round 4 happens only for signature generation.

References

ABCP23. Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. VSS from distributed ZK proofs and applications. In *ASIACRYPT (1)*, volume 14438 of *Lecture Notes in Computer Science*, pages 405–440. Springer, 2023.

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

ASY22. Shweta Agrawal, Damien Stehlé, and Anshu Yadav. Round-optimal lattice-based threshold signatures, revisited. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl, July 2022.

BGG$^+$18. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

BKP13. Rikke Bendlin, Sara Krehbiel, and Chris Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 218–236. Springer, Heidelberg, June 2013.

BP23. Luís T. A. N. Brandão and Rene Peralta. Nist first call for multi-party threshold schemes. NIST Internal Report (IR) 8214C, National Institute of Standards and Technology, January 2023. Initial Public Draft.

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

dEK$^+$23. Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

Des90. Yvo Desmedt. Abuses in cryptography and how to fight them. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 375–389. Springer, Heidelberg, August 1990.

DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990.

EEN$^+$24. Muhammed Esgin, Thomas Espitau, Guilhem Niot, Thomas Prest, Amin Sakzad, and Ron Steinfeld. Plover: Masking-friendly hash-and-sign lattice signatures. In *EUROCRYPT*, 2024.

GHL22. Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 458–487. Springer, Heidelberg, May / June 2022.

GJKR96. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 354–371. Springer, Heidelberg, May 1996.

GKS23. Kamil Doruk Gur, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice signatures from threshold homomorphic encryption. Cryptology ePrint Archive, Paper 2023/1318, 2023. https://eprint.iacr.org/2023/1318.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

GRJK00. Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology*, 13(2):273–300, March 2000.

JLS86. William B Johnson, Joram Lindenstrauss, and Gideon Schechtman. Extensions of lipschitz maps into banach spaces. *Israel Journal of Mathematics*, 54(2):129–138, 1986.

JTZ23. Yunfeng Ji, Yang Tao, and Rui Zhang. Robust (t, n)-threshold lattice signature. 2023.

KG20. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Heidelberg, October 2020.

KGS23. Chelsea Komlo, Ian Goldberg, and Douglas Stebila. A formal treatment of distributed key generation, and new constructions. Cryptology ePrint Archive, Report 2023/292, 2023. https://eprint.iacr.org/2023/292.

KLSS23a. Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580. Springer, Heidelberg, August 2023.

KLSS23b. Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-mlwe. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 549–580, Cham, 2023. Springer Nature Switzerland.

Ngu22. Ngoc Khanh Nguyen. *Lattice-Based Zero-Knowledge Proofs Under a Few Dozen Kilobytes.* PhD thesis, ETH Zurich, Zürich, Switzerland, 2022.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

PKM$^+$24. Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions, 2024.

PMB. Andrew Paverd, Andrew Martin, and Ian Brown. Modelling and automatically analysing privacy properties for honest-but-curious adversaries.

RRJ$^+$22. Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022.

Sta96. Markus Stadler. Publicly verifiable secret sharing. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 190–199. Springer, Heidelberg, May 1996.

TPCZ23. Guofeng Tang, Bo Pang, Long Chen, and Zhenfeng Zhang. Efficient lattice-based threshold signatures with functional interchangeability. *IEEE Transactions on Information Forensics and Security*, 18:4173–4187, 2023.

A Proof of security of the V3S

Proof (of Correctness). By construction, the function V3S.Share() constructs a valid Shamir's sharing based on Lagrange interpolations for $[\![\mathbf{x}]\!]$ and $[\![\mathbf{y}]\!]$, in addition to a (valid) corresponding Merkle tree. Thus, V3S.Reconstruct reconstructs the expected value $\mathbf{x}$ for any subset of size T by definition. Now remark that in V3S.Verify(), the Merkle tree proof verification, and equality check $[\![\mathbf{v}]\!]_i = \mathbf{R} \cdot [\![\mathbf{x}]\!]_i + [\![\mathbf{y}]\!]_i$ always pass. Additionally, when the execution is honest, we have $\mathbf{v} = \mathbf{R} \cdot \mathbf{x} + \mathbf{y}$, where $\mathbf{R}, \mathbf{x}, \mathbf{y}$ are independently sampled from $\mathcal{D}_{\mathbf{R}}, \mathcal{D}_{\sigma_x}, \mathcal{D}_{\sigma_y}$. By the *honest execution* property of the distribution $\mathcal{D}_{\mathbf{R}}$, we have $\|\mathbf{v}\| = \|\mathbf{R} \cdot \mathbf{x} + \mathbf{y}\| \leqslant B'$ with a loss of at most $p_3 = \mathsf{negl}(\kappa)$. □

Proof (of Soundness). We follow a hybrid proof approach where we define:

Hybrid_1. First hybrid corresponds to the real V3S-sound game of Figure 2.

```
Hybrid_2
1: $L_H$, Programmed$[\cdot]$ $:= \varnothing$
2: $N, T, S \leftarrow \mathcal{A}()$    ▷ The adversary chooses a subset $S$ of parties to target
3: assert{ $S \subset \{1, ..., N\} \wedge |S| \geqslant T$ }    ▷ $S$ must be large enough to allow reconstruction
4: $(s_i)_{i \in \{1,...,N\}}, \pi, (\pi_i)_{i \in \{1,...,N\}} \leftarrow \mathcal{A}^{H(\cdot)}(N, T, S)$    ▷ The adversary produces a $T$-sharing among N parties
5: Parse $\pi = (h, [\![v]\!]), (\pi_i)_{i \in S} = ([\![\mathbf{y}]\!]_i, r_i, \mathsf{proof}_i)_{i \in S}$
6: if Programmed$[h] \neq (s_i, [\![\mathbf{y}]\!]_i)_{i \in S}$ then
7:     return 0
8: if $\forall i \in S$, V3S.Verify$(s_i, \pi, \pi_i)$ = false then
                                   ▷ In case a party in $S$ fails verification, the adversary loses
9:     return 0
10: $\mathbf{x}$ = V3S.Reconstruct$((s_i)_{i \in S})$
11: if $x = \perp$ then    ▷ Verification passes but shares are inconsistent in $S$
12:     return 1
13: if $\mathbf{x} \notin V$ then    ▷ Verification passes but the secret is invalid
14:     return 1
15: return 0    ▷ The sharing chosen by the adversary is valid

    $H_{\mathbf{R}}(h)$
    1: if $\exists r.(h, "\mathbf{R}", r) \in L_H$ then
    2:     return $r$
    3: else if $\exists([\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j, r_j)_{j \in S}$ s.t. they are in the Merkle tree described by $h$ then
    4:     $\mathbf{R} \leftarrow \mathcal{D}_{\mathbf{R}}$
    5:     Programmed$[h] = (([\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j)_{j \in S})$
    6:     $L_H := L_H \cup \{(h, "\mathbf{R}", \mathbf{R})\}$
    7:     return $\mathbf{R}$
    8: else
    9:     $\mathbf{R} \leftarrow \mathcal{D}_{\mathbf{R}}$
    10:    $L_H := L_H \cup \{(h, "\mathbf{R}", \mathbf{R})\}$
    11:    return $\mathbf{R}$
```

Fig. 10: The second hybrid of the security proof of the soundness of our VSSS construction. Difference with the previous hybrid are highlighted.

Hybrid$_2$. This hybrid ensures that the matrix $\mathbf{R}$ is sampled *after* the shares $([\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j)_{j \in S}$ are chosen by checking the RO calls. It is given in Figure 10. This will allow us to apply *separation* and *large norm detection* properties of $\mathcal{D}_{\mathbf{R}}$ in the next hybrids as then $\mathbf{R}$ will be independent of the tested value. The added check makes the adversary lose in several cases:

– $(\llbracket \mathbf{x} \rrbracket_j, \llbracket \mathbf{y} \rrbracket_j, r_j)_{j \in S}$ do not correctly hash in the Merkle tree h. But then, at least one of the proofs proof_i fails, and one of the calls to V3S.Verify would have failed. Hence, this case does not introduce an advantage loss.
– The shares correctly hash to h but Programming$[h] = \perp$ at the time of programming of the random oracle, one of the intermediary hash of the Merkle tree hadn't been queried yet. This may only happen if the adversary is able to break the pre-image resistance of the hash function. This happens with probability at most $Q_H^2/2^{2\kappa}$.
– The shares given by the adversary correctly hash to h but Programming$[h]$ is equal to a different set of shares. This can only happen if the adversary breaks the collision resistance of the hash function. This happens with probability at most $Q_H^2/(2^{2\kappa})$.

Overall,

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_1}(\kappa)| \leqslant \frac{Q_H^2}{2^{2\kappa-1}}$$

Hybrid_3. This hybrid ensures that in case the random oracle is programmed on shares $\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket$, then if they are inconsistent or if the reconstructed secret is large, then $\mathbf{R} \cdot \mathbf{x} + \mathbf{y}$ will be inconsistent or large. In case one of these checks, we consider that the adversary wins, and remove the previous winning checks. This is formalized in Figure 11.

These modifications allow the adversary to win in more cases. Indeed, in Hybrid_2, the adversary could win in two ways:

– If reconstruction over S fails, but V3S.Verify() returns true over S. But then, it means in particular that shares of $\mathbf{R} \cdot \llbracket \mathbf{x} \rrbracket + \llbracket \mathbf{y} \rrbracket$ were all consistent over S, and that it correctly reconstructs. In Hybrid_3, this case is covered by the first assertion added in $H_{\mathbf{R}}(h)$ that makes the adversary in that case.
– If the reconstructed secret $\mathbf{x}$ has a norm larger than B, but $\mathbf{R} \cdot \mathbf{x} + \mathbf{y}$ has a norm smaller than B'. Similarly, the second assertion added in $H_{\mathbf{R}}$ would fail when called on h, and allow the adversary to win.

As such:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa) \leqslant \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa)$$

Probability of winning Hybrid_3. First, consider the event where some call to $H_{\mathbf{R}}$ works on inconsistent shares for $\llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket$ over S but $\mathbf{R} \cdot \llbracket \mathbf{x} \rrbracket + \llbracket \mathbf{y} \rrbracket$ are all consistent. Noting $(\mathbf{x}, \mathbf{y}) \neq (\mathbf{x}', \mathbf{y}')$ the secrets reconstructed from I, I' two subsets of S of cardinal T, as $\mathbf{R}$ is independent of these vectors, we can apply the *separation* property of $\mathbf{R}$ to bound its probability:

$$\Pr\left[\mathbf{R}\mathbf{x} + \mathbf{y} \bmod q = \mathbf{R}\mathbf{x}' + \mathbf{y}' \bmod q\right] = p_1$$

By the union-bound the probability that the first assertion succeeds for all the calls to the random oracles is at most $Q_H \cdot p_1$. Similarly, for the second assertion,

```
Hybrid_3
1: $L_H, \text{Programmed}[\cdot] := \varnothing$
2: $N, T, S \leftarrow \mathcal{A}()$    ▷ The adversary chooses a subset $S$ of parties to target
3: assert{ $S \subset \{1, \ldots, N\} \wedge |S| \geqslant T$ }    ▷ $S$ must be large enough to allow reconstruction
4: $(s_i)_{i \in \{1,\ldots,N\}}, \pi, (\pi_i)_{i \in \{1,\ldots,N\}} \leftarrow \mathcal{A}^{H(\cdot)}(N, T, S)$    ▷ The adversary produces a $T$-sharing among N parties
5: Parse $\pi = (h, [\![v]\!]), (\pi_i)_{i \in S} = ([\![\mathbf{y}]\!]_i, r_i, \mathsf{proof}_i)_{i \in S}$
6: if $\text{Programmed}[h] \neq (s_i, [\![\mathbf{y}]\!]_i)_{i \in S}$ then
7:     return 0
8: if $\forall i \in S, \mathsf{V3S.Verify}(s_i, \pi, \pi_i) = \mathsf{false}$ then
       ▷ In case a party in $S$ fails verification, the adversary loses
9:     return 0
10: return 0    ▷ The sharing chosen by the adversary is valid

    $H_{\mathbf{R}}(h)$
    1: if $\exists r.(h, "\mathbf{R}", r) \in L_H$ then
    2:     return $r$
    3: else if $\exists([\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j)_{j \in S}$ s.t. they are in the Merkle tree described by $h$ then
    4:     $\mathbf{R} \leftarrow \mathcal{D}_{\mathbf{R}}$
    5:     $\text{Programmed}[h] = (([\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j)_{j \in S})$
    6:     if $\mathsf{V3S.Reconstruct}(([\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j)_{j \in S}) = \perp$ then
               ▷ If shares of $\mathbf{x}, \mathbf{y}$ are inconsistent over $S$
    7:         assert{ $\mathsf{V3S.Reconstruct}((\mathbf{R} \cdot [\![\mathbf{x}]\!]_j + [\![\mathbf{y}]\!]_j)_{j \in S}) = \perp$ }
               ▷ Then the shares of $\mathbf{R} \cdot \mathbf{x} + \mathbf{y}$ are also inconsistent
    8:     else
    9:         Reconstruct $\mathbf{x}, \mathbf{y}$ from the shares in $S$
    10:        if $\|\mathbf{x}\| > B$ then
    11:            assert{ $\|\mathbf{R} \cdot \mathbf{x} + \mathbf{y}\| > B'$ }
    12:    $L_H := L_H \cup \{(h, "\mathbf{R}", \mathbf{R})\}$
    13:    return $\mathbf{R}$
    14: else
    15:    $\mathbf{R} \leftarrow \mathcal{D}_{\mathbf{R}}$
    16:    $L_H := L_H \cup \{(h, "\mathbf{R}", \mathbf{R})\}$
    17:    return $\mathbf{R}$
```

Fig. 11: The third hybrid of the security proof of the soundness of our VSSS construction. Difference with the previous hybrid are highlighted. In case one of the assertions in $H_{\mathbf{R}}$ fails, consider that the adversary wins.

the matrix $\mathbf{R}$ is independent of the reconstructed value and the *large norm detection* property of $\mathcal{D}_{\mathbf{R}}$ allows us to bound the probability of failure by $Q_H \cdot p_2$.

We can conclude:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa) \leqslant Q_H \cdot (p_1 + p_2).$$

Summing all the intermediary advantage losses gives the final result. □

Proof (of fragmentary knowledge). We proceed once again with a series of hybrids.

Hybrid_1. The first hybrid corresponds to the case $b = 0$ of the V3S-fk game, i.e. where the adversary observes the real distribution of transcripts.

```
Hybrid_2
$L_H := \varnothing$
$N, T, S \leftarrow \mathcal{A}()$    ▷ The adversary chooses a subset $S$ of parties to target
assert{ $S \subset \{1, ..., N\} \wedge |S| = T - 1$ }
$\mathbf{x} \leftarrow \mathcal{D}_{\mathbf{x}}$
▹▹▹ Content of V3S.Share() ◃◃◃
$\mathbf{y} \leftarrow \mathcal{D}^d_{\sigma_y}, (r_i) \xleftarrow{\$} \{0,1\}^{N \cdot 2\kappa}$
Sample random sharings $[\![\mathbf{x}]\!], [\![\mathbf{y}]\!]$ of order $T$
$h$ := hash of Merkle tree containing $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i, r_i)$ for $i \in S$, and replacing hashes of other shares by random values.
for $i \in S$ do
    $\mathsf{proof}_i$ := proof that $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i, r_i)$ is in Merkle tree $h$
    $\pi_i := ([\![\mathbf{y}]\!]_i, r_i, \mathsf{proof}_i)$
$\mathbf{R} := H_{\mathbf{R}}(h)$    ▷ Hash $h$ to obtain a random matrix from $\mathcal{D}_{\mathbf{R}}$
$[\![\mathbf{v}]\!] := \mathbf{R} \cdot [\![\mathbf{x}]\!] + [\![\mathbf{y}]\!]$
$\pi := (h, [\![\mathbf{v}]\!])$    ▷ Publish challenge polynomial and Merkle tree hash
$b' \leftarrow \mathcal{A}^H(\mathbf{x}, ([\![x]\!]_i)_{i \in S}, \pi, (\pi_i)_{i \in S})$
return $b'$
```

Fig. 12: The second hybrid of the security proof of the fragmentary knowledge of our VSSS construction. Difference with the previous hybrid are highlighted.

Hybrid_2. The second hybrid replaces the hashes of shares not in S in the Merkle tree by uniform strings. This change is depicted in Figure 12.

The view of the adversary differs *only if* it did query the random oracle on one of the shares $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}_i]\!], r_i)$ with $i \notin S$ in Hybrid_1 and the Merkle tree hash does not correspond in Hybrid_2. However, since each share $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}_i]\!], r_i)_{i \in S}$ has min-entropy at least $\mathcal{H}(r_i) = 2\kappa$, this happens for each share with probability at most $Q_H/2^{2\kappa}$. By union bound for all shares we have:

$$|\mathsf{Adv}^{\mathsf{Hybrid}_2}_{\mathcal{A}}(\kappa) - \mathsf{Adv}^{\mathsf{Hybrid}_1}_{\mathcal{A}}(\kappa)| \leqslant N \cdot \frac{Q_H}{2^{2\kappa}}$$

```
Hybrid_3
1:  $L_H := \varnothing$
2:  $N, T, S \leftarrow \mathcal{A}()$          ▷ The adversary chooses a subset $S$ of parties to target
3:  assert{ $S \subset \{1, \ldots, N\} \wedge |S| = T - 1$ }
4:  $\mathbf{x} \leftarrow \mathcal{D}_{\mathbf{x}}$
    ▷▷▷ Content of V3S.Share() ◁◁◁
5:  $\mathbf{y} \leftarrow \mathcal{D}^{d}_{\sigma_y}$
6:  $[\![\mathbf{x}]\!]_{i \in S}, [\![\mathbf{y}]\!]_{i \in S} \xleftarrow{\$} \mathbb{Z}_q^{n+d}$          ▷ Sample observed shares randomly
7:  $h :=$ hash of Merkle tree containing $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i)$ for $i \in S$, and replacing hashes
    of other shares by random values.
8:  for $i \in S$ do
9:      $\mathsf{proof}_i :=$ proof that $([\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i)$ is in Merkle tree $h$
10:     $\pi_i := ([\![\mathbf{y}]\!]_i, \mathsf{proof}_i)$
11: $\mathbf{R} := H_{\mathbf{R}}(h)$          ▷ Hash $h$ to obtain a random matrix from $\mathcal{D}_{\mathbf{R}}$
12: Compute $[\![\mathbf{v}]\!]$ s.t. $[\![\mathbf{v}]\!]_0 = \mathbf{R} \cdot \mathbf{x} + \mathbf{y}$, and for $i \in S$, $[\![\mathbf{v}]\!]_i = \mathbf{R} \cdot [\![\mathbf{x}]\!]_i + [\![\mathbf{y}]\!]_i$
13: $\pi := (h, [\![\mathbf{v}]\!])$          ▷ Publish challenge polynomial and Merkle tree hash
14: $\hat{\mathbf{x}} \leftarrow \mathcal{D}_{\mathbb{Z}^n, \mathbf{c}, \sqrt{\Sigma_0}}$
15: $b' \leftarrow \mathcal{A}^H(\hat{\mathbf{x}}, ([\![x]\!]_i)_{i \in S}, \pi, (\pi_i)_{i \in S})$
16: return $b'$
```

Fig. 13: The third hybrid of the security proof of the fragmentary knowledge of our VSSS construction. Difference with the previous hybrid are highlighted.

Hybrid$_3$. This hybrid replaces the shares of $\mathbf{x}$ in S by uniformly drawn vectors of $\mathbb{Z}_q^n$, and the $\mathbf{x}$ send to the adversary by a Gaussian variable following $\mathcal{D}_{\mathbb{Z}^n, \mathbf{c}, \sqrt{\Sigma_0}}$. This is formalized in 13. We observe that this corresponds to the case $b = 1$ of the V3S-fk game, and implements perfectly the simulators SimProof, SimSecret we previously described. The computation of $[\![\mathbf{v}]\!]$ as a function of $\mathbf{R} \cdot x + \mathbf{y}$ and $(\mathbf{R} \cdot [\![\mathbf{x}]\!]_i, [\![\mathbf{y}]\!]_i)_{i \in S}$ is equivalent to the one done in Hybrid$_2$ due to the correctness of the sharing of $\mathbf{x}$, and as we only need T points of $[\![\mathbf{v}]\!]$ to recover its full value. Hence, this change does not induce an advantage loss. We also note that the adversary observes outputs computed from exactly $T - 1$ shares of $[\![\mathbf{x}]\!]$ in Hybrid$_3$ and they are thus randomly distributed independently from $\mathbf{x}$. We can sample them directly with no advantage loss. Finally, applying Lemma 1, we have that $(\mathbf{x}, \mathbf{R} \cdot \mathbf{x} + \mathbf{y})$ follows the same distribution as $(\hat{\mathbf{x}}, \mathbf{R} \cdot \mathbf{x} + \mathbf{y})$. We conclude that: $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa)$. □

B Security proof of Pelican

We define in Figure 15 variants of the key generation from Figure 14 coined LeakyKeygen$_{\mathsf{RB}}$ and LeakyKeygen$_{\mathsf{UF}}$. LeakyKeygen allows an adversary to bias the distribution of the key. In this section, we prove stronger unforgeability and robustness results using LeakyKeygen. It will be used to prove the security of our distributed key generation.

```
Pelican.Setup(pp, $N$)
1: for $i \in \{1, ..., N\}$ do
2:    $\mathsf{sk}_i^{\mathsf{SIG}}, \mathsf{pk}_i^{\mathsf{SIG}} \leftarrow \mathsf{SIG.Keygen(pp)}$
3:    for $j \in \{1, ..., N\}$ do
4:       $\mathsf{K}_{i \to j} \leftarrow \mathsf{SKE.Keygen(pp)}$
5:       $\mathsf{sig}_{i \to j} = \mathsf{SIG.Sign}(\mathsf{sk}_i, \{i, k, \mathsf{K}_{i \to j}\})$
6: return $(\mathsf{pk}_i^{\mathsf{Setup}} := \mathsf{pk}_i^{\mathsf{SIG}})_{i \in \{1,...,N\}}, (\mathsf{sk}_i^{\mathsf{Setup}} := (\mathsf{K}_{j \to k}, \mathsf{sig}_j k)_{j=i \vee k=i})_{i \in \{1,...,N\}}$
Pelican.Keygen(pp, $T$, $N$)
1: $(\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,...,N\}} \leftarrow \mathsf{Pelican.Setup}(\mathsf{pp}, N)$
2: $\mathbf{s} \leftarrow \mathcal{D}_{\sigma_t}^2$
3: Sample $P \in \mathcal{R}_q^2[X]$ of degree $< T$ such that $P(0) = \mathbf{s}$
4: $\mathsf{salt} \leftarrow \{0,1\}^\kappa$
5: $a := H_a(\mathsf{salt})$
6: $b := \beta - \begin{bmatrix} 1 & a \end{bmatrix} \cdot \mathbf{s} \bmod q$
7: return $\mathsf{vk} = (a, b), (\mathsf{sk}_i := (P(i), (\mathsf{pk}_j^{\mathsf{Setup}})_{j \in \{1,...,N\}}, \mathsf{sk}_i^{\mathsf{Setup}}))_{i \in \{1,...,N\}}$
```

Fig. 14: Algorithms for the key generation of Pelican.

B.1 Robustness

Proof (of Theorem 6). We start by proving a slightly more general variant of Theorem 6, with keys generated with $\mathsf{LeakyKeygen_{RB}}$ and allowing the adversary to bias generated keys. This result will be leveraged for the robustness proof of Pelican used with our DKG in Appendix C.1.

Lemma 5. *Take a polynomial-time adversary $\mathcal{A}$ in the game* Leak-TH-RB *from Figure 17. Consider $\mathcal{K}$ the distribution of the keys generated by* $\mathsf{LeakyKeygen_{RB}}$ *with the adversarial input from $\mathcal{A}$. Assume that over the distribution of keys $\mathcal{K}$, we have $\|(z_1, z_2, c_1)\| \leqslant B_2$ with overwhelming probability p for any set* valid*, $\mathbf{p} = \sum_{j \in \mathsf{valid}} \mathbf{p}_j$ with (i) at most $T-1$ arbitrary perturbations of norm bounded by B, (ii) the others sampled from Gaussians of standard deviation $\sigma_{\mathbf{p}}$.*

The advantage of $\mathcal{A}$ in the game Leak-TH-RB *is then bounded by*

$$N \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{V3S\text{-}sound}}(\kappa) + N \cdot Q_s \cdot (1 - \Pr[\mathsf{V3S}\ \textit{correctness}]) + Q_s \cdot (1 - p)$$

where $\mathcal{B}_1$ is an adversary against the UF-CMA *security of* SIG*, $\mathcal{B}_2$ is an adversary against the* V3S-sound *security of* V3S*, with running time $\mathcal{T}_{\mathcal{B}_1} \approx \mathcal{T}_{\mathcal{B}_2} \approx \mathcal{T}_{\mathcal{A}}$.*

```
$\mathsf{Pelican.LeakyKeygen_{RB}}(\mathsf{pp}, T, N, \mathsf{aux} = ((\llbracket \mathbf{s}' \rrbracket, \mathsf{salt}), (\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,\dots,N\}})$
1: $\mathbf{s}' := \mathsf{V3S.Reconstruct}((\llbracket \mathbf{s}' \rrbracket_i)_i)$
2: assert{ $\mathbf{s}' \neq \bot$ }                 ▷ Corrupted contribution is a valid sharing
3: $a := H_a(\mathsf{salt})$
4: $b := \beta - \begin{bmatrix} 1 & a \end{bmatrix} \cdot \mathbf{s}' \bmod q$
5: return $\mathsf{vk} = (a, b), (\mathsf{sk}_i := (\llbracket \mathbf{s}' \rrbracket_i, (\mathsf{pk}_j^{\mathsf{SKE}})_{j \in \{1,\dots,N\}}, \mathsf{sk}_i^{\mathsf{SKE}}))_{i \in \{1,\dots,N\}}$
```

```
$\mathsf{Pelican.LeakyKeygen_{UF}}(\mathsf{pp}, T, N, \mathsf{aux} = ((\llbracket \mathbf{s}' \rrbracket, \mathsf{salt}), (\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,\dots,N\}})$
1: $\mathbf{s}' := \mathsf{V3S.Reconstruct}((\llbracket \mathbf{s}' \rrbracket_i)_i)$
2: assert{ $\mathbf{s}' \neq \bot$ }                 ▷ Corrupted contribution is a valid sharing
3: $\mathbf{s} \leftarrow \mathcal{D}_{\sigma_t}^2$
4: Sample $P \in \mathcal{R}^2[X]$ of degree $< T$ such that $P(0) = \mathbf{s}$
5: $a := H_a(\mathsf{salt})$
6: $b := \beta - \begin{bmatrix} 1 & a \end{bmatrix} \cdot (\mathbf{s} + \mathbf{s}') \bmod q$
7: return $\mathsf{vk} = (a, b), (\mathsf{sk}_i := (P(i) + \llbracket \mathbf{s}' \rrbracket_i, (\mathsf{pk}_j^{\mathsf{SKE}})_{j \in \{1,\dots,N\}}, \mathsf{sk}_i^{\mathsf{SKE}}))_{i \in \{1,\dots,N\}}$
```

Fig. 15: Algorithms for the leaky key generation of Pelican, used respectively for robustness proofs, and for unforgeability proofs. These algorithms are used to model bias that can be introduced by the adversary in generated keys when using our distributed key generation. Note that we can recover Keygen from Fig. 14 by calling $\mathsf{LeakyKeygen_{UF}}$ (resp. $\mathsf{LeakyKeygen_{RB}}$)with $\mathsf{aux} = \llbracket 0 \rrbracket^0$ (resp. $\mathsf{aux} \leftarrow \llbracket \mathcal{D}_{\sigma_t}^2 \rrbracket$), $\mathsf{salt} \xleftarrow{\$} \{0,1\}^\kappa$ and $(\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,\dots,N\}} \leftarrow \mathsf{Pelican.Setup}(\mathsf{pp})$. LeakyKeygen is useful for proofs but not used in any actual implementation.

Theorem 6 is directly implied by Lemma 5. We construct an efficient adversary $\mathcal{B}$ for the game Leak-TH-RB from the adversary $\mathcal{A}$ for the game TH-RB:

- $\mathcal{B}$ samples and return a random $\mathsf{salt} \xleftarrow{\$} \{0,1\}^\kappa$.
- It samples $\mathsf{aux} = \llbracket \mathcal{D}_{\sigma_t} \rrbracket$.
- It runs $\mathcal{A}$ normally after obtaining the output of $\mathsf{LeakyKeygen_{RB}}$.

With the above bias aux, and a uniform salt, $\mathcal{B}$ recovers the behavior of the trusted Keygen, and the view of $\mathcal{A}$ is the same as in the game TH-RB. The assumption in Theorem 6 also ensures that for the distribution of keys from Keygen, we have $\|(z_1, z_2, c_1)\| \leqslant B_2$ with overwhelming probability.

The advantage of $\mathcal{B}$ in Leak-TH-RB, is the same as the advantage of $\mathcal{A}$ in TH-RB.

$\square$

Proof (of Lemma 5).

We proceed with a series of hybrids starting with the robustness game from Figure 17.

Hybrid_2. The second hybrid asserts that all honest parties agree on the same valid set at the end of round 3. This is formalized in Figure 18.

```
$\mathsf{Game}_{\mathsf{TH\text{-}RB}}$
$L_H, L_{\mathsf{sid}} := \varnothing$
$\mathsf{vk}_0, \mathsf{sk}_0 := \varnothing$
$(N, T, \mathsf{corrupt}) \leftarrow \mathcal{A}(\mathsf{pp}, 1^\kappa)$
assert{ $\mathsf{corrupt} \subseteq \{1, ..., N\} \wedge T \leqslant N/3$ }
assert{ $|\mathsf{corrupt}| < T$ }
$\mathsf{honest} := \{1, ..., N\} \backslash \mathsf{corrupt}$
$(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \{1,...,N\}}) \leftarrow \mathsf{Sign.Keygen}(\mathsf{pp}, T, N)$
for $i \in \mathsf{honest}$ do
  $\mathsf{state}_i.\mathsf{sk} := \mathsf{sk}_i$
  $\mathsf{state}_i.\mathsf{vk} := \mathsf{vk}$
$\mathsf{sid}, (\mathsf{out}_i)_{i \in \mathsf{corrupt}} \leftarrow \mathcal{A}^{H, (\mathsf{OPerformRound}_i(\cdot))_{i \in [\mathsf{rnd}_{\mathsf{Sign}}]}}(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \mathsf{corrupt}})$
Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
if $\mathsf{rnd} \neq \mathsf{rnd}_{\mathsf{Sign}}$ then
  return 0                          ▷ The adversary did not finish the protocol
$\mathsf{status}, \mathsf{sig} := \mathsf{Sign.Combine}(\mathsf{vk}, \mathsf{msg}, (\mathsf{out}_i)_{i \in \{1,...,N\}})$
if $\mathsf{status} = \mathbf{ok} \wedge \mathsf{Sign.Verify}(\mathsf{vk}, \mathsf{sig})$ then
  return 0                          ▷ The protocol produced a valid signature
return 1

$\mathsf{OPerformRound}_1(\mathsf{sid}, \mathsf{msg})$
assert{ $\mathsf{sid} \notin L_{\mathsf{sid}}$ }
for $i \in \mathsf{honest}$ do
  $\mathsf{out}_i := \mathsf{Sign.ShareSign}_1(\mathsf{state}_i, \mathsf{sid}, \mathsf{msg})$
$L_{\mathsf{sid}}[\mathsf{sid}] = \{1, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

$\mathsf{OPerformRound}_k((\mathsf{in}_i)_{i \in \mathsf{corrupt}})$, for $k \in \{2, ..., \mathsf{rnd}_{\mathsf{Sign}}\}$
Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{in}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
assert{ $\mathsf{rnd} = k - 1$ }
for $i \in \mathsf{honest}$ do
  $\mathsf{out}_i := \mathsf{Sign.ShareSign}_k(\mathsf{state}_i, \mathsf{sid}, (\mathsf{in})_{i \in \{1,...,N\}})$
$L_{\mathsf{sid}}[\mathsf{sid}] = \{k, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

$H(\mathsf{str}, \mathsf{digest})$
assert{ $\mathsf{str} \in \mathsf{pp.HashParams}$ }          ▷ Check domain string
if $\exists r.(\mathsf{str}, \mathsf{digest}, r) \in L_H$ then
  return $r$
else
  Sample $r$ uniformly
  $L_H := L_H \cup \{(\mathsf{str}, \mathsf{digest}, r)\}$
  return $r$
```

Fig. 16: Robustness game. The adversary $\mathcal{A}$ wins if the game TH-RB returns 1, i.e. if it is able to prevent the protocol from producing a valid signature.

As we rely on a broadcast channel, all the honest parties will observe the same messages in round 2 and round 3. In particular, they will observe empty contributions for the same parties in round 2 and will read the same complaints in round 3. As the algorithms $\mathsf{Decrypt}()$ and $\mathsf{V3S.Verify}()$ are deterministic, all honest parties will simultaneously accept or reject the complaints.

So,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_1}(\kappa)$$

Hybrid_3. In this hybrid, we ensure that honest parties do not raise or accept complaints against other honest parties. We additionally assert that shares from honest parties correctly reconstruct. This is formalized in Figure 19.

First, since honest contributions failing verification are rejected in round 1, honest shares received in round 2 will always pass verification and honest parties won't complain about other honest parties.

As for complaints coming from corrupted parties, recall that the contributions sent by honest parties always pass verification by construction of the protocol. So the only way a corrupted complaint may be accepted is if they provide an incorrect key in their complaint. But then it means that they forged a signature of $\{i, j, \mathsf{K}_{i\to j}\}$ for some $i \in \mathsf{honest}$. This reduces the advantage by at most $|\mathsf{honest}|\cdot \mathsf{Adv}^{\mathsf{UF\text{-}CMA}}(\kappa)$.

The check $\mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ finally automatically passes as honest parties correctly send contributions in round 1 and round 2, and as we ensured no complaints against them are accepted.

Hence, there exists and adversary $\mathcal{B}_1$ against the $\mathsf{UF\text{-}CMA}$ security of SIG such that

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa)| \leqslant N \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa)$$

Hybrid_4. This hybrid removes the restart in round 1 in case verification fails, and asserts that honest shares are consistent. It also asserts that accepted shares coming from corrupted parties reconstruct to valid secrets. This is formalized in Figure 20.

By the *correctness* of the $\mathsf{V3S}$, during each signing session, restart in round 1 will happen with probability bounded by $|\mathsf{honest}| \cdot (1 - \Pr[\mathsf{V3S}\text{ correctness}])$. The VSSS *correctness* also ensures that honest shares correctly reconstruct.

As for the shares coming from corrupted parties, this directly reduces to the soundness of the $\mathsf{V3S}$. If the new assertion fails, it gives us an invalid sharing that passes verification over the set of honest parties honest which has a cardinal larger than T.

Thus, for an adversary $\mathcal{B}_2$ against the $\mathsf{V3S\text{-}sound}$ security of $\mathsf{V3S}$, we have

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa)| \leqslant NQ_s \cdot (1 - \Pr[\mathsf{V3S}\text{ correctness}]) + \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{V3S\text{-}sound}}(\kappa)$$

Conclusion. We are then able to conclude on the robustness of all opened signing sessions.

At the end of round 3, honest parties agree on the same set valid, and only use contributions that correctly reconstruct to perturbations $(\mathbf{p}_j)_j \in \mathsf{valid}$. In particular, the shares $([\![\mathbf{p}]\!]_i)_{i \in \mathsf{honest}}$ correctly reconstruct to $\mathbf{p} = \sum_{j \in \mathsf{valid}} \mathbf{p}_j$.

In round 4, parties will thus receive at least $|\mathsf{honest}|$ correct shares of $w = \mathbf{A} \cdot \mathbf{p}$, and as we assumed $N \geqslant 3T$, V3S.RobustReconstruct will recover its value correctly.

Honest parties will also correctly compute shares of $z = c_1 \cdot s + p_2$. Same as previously, as $N \geqslant 3T$, the value of z will be correctly recovered in Pelican.Combine.

Since we assumed that over the distribution of keys, and honest perturbations, we have $\|(z_1, z_2, c_1)\| \leqslant B_2$ with overwhelming probability p when there are at most $T - 1$ corrupted perturbations all in the set V, we conclude that a valid signature will be produced with probability at least p in each signing session.

Finally, we conclude with the probability that all signing sessions succeed,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4}(\kappa) \leqslant Q_s \cdot (1 - p)$$

□

B.2 Unforgeability

Proof (of Theorem 7). As in our proof of robustness, we wish to prove a more general variant of unforgeability for our scheme. Namely, we allow the adversary to bias keys by generating them with $\mathsf{LeakyKeygen}_{\mathsf{UF}}$ from Fig. 15. This result will be leveraged for the unforgeability proof of Pelican used with our DKG in Appendix C.2.

Lemma 6. *Define* $\sigma'_{\mathbf{p}}$ *as* $\frac{1}{\sigma'^2_{\mathbf{p}}} = 2\left(\frac{1}{\sigma^2_{\mathbf{p}}} + \frac{B}{\sigma^2_y}\right)$, *with* B *such that* $s_1(\mathbf{R}^\top \mathbf{R}) \leqslant B$ *with overwhelming probability and* $\sigma'_{\mathbf{p}} \geqslant \sqrt{2}\eta_\varepsilon(\mathbb{Z}^{2n})$.

Pelican *instanciated with* $(\mathsf{Setup}, \mathsf{LeakyKeygen}_{\mathsf{UF}})$ *as defined in Figures 14 and 15 is* Leak-TH-UF *secure in the random oracle model.*

Formally, let $\mathcal{A}$ *be an adversary against the* Leak-TH-UF *security game from Figure 22 starting at most* Q_s *signing sessions, making at most* Q_H *random oracles queries, and* Q_a *queries to the random oracle* H_a.

Then there exists adversaries $\mathcal{B}_1$ *against the* UF-CMA *security of* SIG, $\mathcal{B}_2$ *against the* IND-CPA *security of* SKE, $\mathcal{B}_3$ *against the* V3S-sound *security of* V3S, $\mathcal{B}_4$ *against the* V3S-fk *security of* V3S, $\mathcal{B}_5$ *against* $\mathsf{Hint\text{-}RLWE}_{q,Q_{\mathsf{Sign}},\mathcal{D}_{\sigma_t},\mathcal{D}_{\sigma'_{\mathbf{p}}},\mathcal{C}_1}$, $\mathcal{B}_6$ *against* $\mathsf{RLWE}_{q,\mathcal{U}([-B_2/\sqrt{2n},B_2/\sqrt{2n}]^n)^2}$, $\mathcal{B}_7$ *against* $\mathsf{RSIS}_{q,2,2B_2}$, *and running time* $\mathcal{T}_{\mathcal{B}_i} \approx \mathcal{T}_{\mathcal{A}}$ *for* $i \in \{1, \ldots, 7\}$, *such that*

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Leak\text{-}TH\text{-}UF}}(\kappa) \leqslant & N \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa) + N^2 \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{IND\text{-}CPA}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{V3S\text{-}sound}}(\kappa) \\ & + Q_s \cdot N \cdot \mathsf{Adv}_{\mathcal{B}_4}^{\mathsf{V3S\text{-}fk}}(\kappa) + N \cdot Q_s \cdot (1 - \Pr[\mathsf{V3S}\ \mathit{correctness}]) \\ & + Q_a \cdot \mathsf{Adv}_{\mathcal{B}_5}^{\mathsf{Hint\text{-}RLWE}}(\kappa) + Q_H \cdot \mathsf{Adv}_{\mathcal{B}_6}^{\mathsf{RLWE}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_7}^{\mathsf{RSIS}}(\kappa) \\ & + \frac{Q_s}{2^{2\kappa}} + 4Q_s N \cdot \varepsilon + Q_s \cdot 2^{-n+2} + p_c\end{aligned}$$

for some $p_c \leqslant 2^{-n(2\log_2(2B_2/\sqrt{2n}) - \log_2(q))}$.

The proof of Theorem 7 directly derives from Lemma 6. Indeed, we can recover the behavior of Keygen by taking the leaks in LeakyKeygen equal to 0, i.e. $\mathsf{aux} = (\llbracket 0 \rrbracket^0)$, and sample the salt from $\{0,1\}^\kappa$. Only one query to H_a is performed and we can thus take $Q_a = 1$. □

Proof (of Lemma 6). Let $\mathcal{A}$ be an adversary against the unforgeability of the threshold Pelican signature scheme with leaky keygen. We use a series of hybrid games where Hybrid_1 corresponds to the original unforgeability game defined in Figure 22. The final hybrid is designed such that a reduction $\mathcal{B}'$ can simulate it given a RSIS instance, and extract a solution to this problem from a successful adversary.

Hybrid_1. This is the unforgeability security game.

Hybrid_2, Hybrid_3, *and* Hybrid_4. The hybrid 2, 3, and 4 perform the same changes as for the robustness game. They ensure respectively that (i) all parties agree on the same valid set, (ii) no complaint is raised or accepted against honest parties, and (iii) that accepted contributions are consistent, and that no restart is performed in round 1. They are formalized in Figures 18, 19, 20.

The advantage loss is for some adversaries $\mathcal{B}_1, \mathcal{B}_3$

$$\begin{aligned}|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_1}(\kappa)| \leqslant & N \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa) + \mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{V3S\text{-}sound}}(\kappa) \\ & + Q_s \cdot N \cdot \Pr[\mathsf{V3S}\ \text{correctness}]\end{aligned}$$

Hybrid_5. In this hybrid, we replace the plaintexts in round 1 destined to other honest parties using the SKE with $\varnothing$. This is formalized in Figure 23.

Previous hybrids made sure no decryption of ciphertexts between honest parties is performed, and keys $\mathsf{K}_{i\to j}$ are never sent to the adversary for $i, j \in \mathsf{honest}$. Hence, this reduces to the IND-CPA security of the underlying SKE scheme.

We can define an adversary $\mathcal{B}_2$ against the IND-CPA security of SKE verifying

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_5}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4}(\kappa)\right| \leqslant N^2 \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{IND\text{-}CPA}}(\kappa)$$

Hybrid_6. In this hybrid, we remove the dependency on honest shares of the secret $\mathbf{s}$ and of honest perturbations $\mathbf{p}_i$. This is formalized in Figure 24.

Recall that Lagrange interpolation allows us to recover the value in 0 of any polynomial of degree less than T given T of its images. Specifically, for given images over the set $S \cup \{i\}$ for $i \in \mathsf{honest}$, there exist coefficients $(\lambda_k^i)_{k \in S \cup \{i\}}$ such that for any polynomial P of degree less than T, we have $P(0) = \sum_{k \in S \cup \{i\}} \lambda_k^i P(k)$.

Due to the correctness of the Fiat Shamir sharing of honest parties, we have $\mathbf{s} = \sum_{k \in S \cup \{i\}} \lambda_k^i [\![\mathbf{s}]\!]_k$ and for any $i, j \in \mathsf{honest}$, $\mathbf{p}_j = \sum_{k \in S \cup \{i\}} \lambda_k^i [\![\mathbf{p}_j]\!]_k$, or equivalently $[\![\mathbf{s}]\!]_i = (\mathbf{s} - \sum_{k \in S} \lambda_k^i [\![\mathbf{s}]\!]_k) \cdot (\lambda_i^i)^{-1}$ and $[\![\mathbf{p}_j]\!]_i = (\mathbf{p}_j - \sum_{k \in S} \lambda_k^i [\![\mathbf{p}_j]\!]_k) \cdot (\lambda_i^i)^{-1}$.

This is exactly the expression used to replace $[\![\mathbf{s}]\!]_i$ and $[\![\mathbf{p}_j]\!]_i$ in $\mathsf{ShareSign}_3$ and $\mathsf{ShareSign}_4$, so there is no advantage loss.

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_6}(\kappa) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_5}(\kappa)$$

Hybrid_7. In this hybrid, we replace $\mathsf{V3S}$ proofs and shares observed by the attacker using $\mathsf{V3S}$ fragmentary knowledge simulators. This is formalized in Figure 25.

In the previous hybrid, the adversary's view does no longer depend on $[\![\mathbf{p}_i]\!]_j, \pi_{i \to j}$ for $i, j \in \mathsf{honest}$, which implies that it only depends on at most $T - 1$ shares. So we can simulate proofs and shares observed by the adversary and directly reduce the indistinguishability of doing so to the advantage of an attacker against the fragmentary knowledge property of the underlying $\mathsf{V3S}$. Since we want to simulate $Q_s \cdot |\mathsf{honest}| \leqslant N$ transcripts, we lose at most an advantage $Q_s \cdot N \cdot \mathsf{Adv}^{\mathsf{V3S\text{-}fk}}$.

So, for an adversary $\mathcal{B}_4$ efficiently derived from $\mathcal{A}$,

$$\left| \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_7}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_6}(\kappa) \right| \leqslant Q_s \cdot N \cdot \mathsf{Adv}_{\mathcal{B}_4}^{\mathsf{V3S\text{-}fk}}(\kappa)$$

Hybrid_8. In this hybrid, we extract a Gaussian of standard deviation $\sqrt{|\mathsf{honest}|}\sigma'_{\mathbf{p}}$ from the simulated secrets. This is formalized in Figure 26.

Recall that $\sigma'_{\mathbf{p}}$ is defined such that:

$$\frac{1}{\sigma'^2_{\mathbf{p}}} = 2\left(\frac{1}{\sigma^2_{\mathbf{p}}} + \frac{B}{\sigma_y}^2\right)$$

with B s.t. $\Pr\left[s_1(\mathbf{R}^\top \mathbf{R}) < B\right]$ with overwhelming probability. We additionally assume $\sigma'_{\mathbf{p}} \geqslant 2\eta_\varepsilon(\mathbb{Z}^{2n})$.

We perform this extraction by applying a similar strategy as in the proof of Theorem 1. We proceed in two steps:

- We first extract Gaussians of standard deviation $\sigma'_{\mathbf{p}}$ from each Gaussian $\mathcal{D}_{\mathbb{Z}^{2n}, \mathbf{c}_1, \sqrt{\Sigma_0}}$ produced by $\mathsf{SimSecret}$. This incurs a statistical difference 2ε for each replacement: $2Q_s|\mathsf{honest}|\varepsilon$ in total.
- The second step consists in joining all the $\mathcal{D}_{\mathbb{Z}^{2n}, 0, \sigma'_{\mathbf{p}}}$ together. Again this incurs an advantage loss of at most 2ε for each replacement.

Hence,

$$\left| \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_8}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_7}(\kappa) \right| \leqslant 4Q_s N \cdot \varepsilon$$

Hybrid_9. In this hybrid, we program the honest responses z_i in $\mathsf{ShareSign}_3$. This is formalized in Figure 27.

Our changes start by computing the shares $[\![w]\!]_i$ for $i \in \mathsf{honest}$, and reconstruct w the corresponding value. This will always succeed as Hybrid_4 ensured that all shares for perturbations from the valid set are consistent.

The second step consists in programming the random oracle H_{salt} on w. We note that w is of the form $\mathbf{A} \cdot \mathrm{HonestP}[\mathsf{sid}] + Z$, where Z is independent of the honest perturbation. For considered parameters, $\mathbf{A} \cdot \mathbf{p}_{\mathsf{honest}}$ has min-entropy at least $n-1$ with overwhelming probability $1-2^{-n+1}$ and under that condition programming fails with probability at most 2^{-n+1}. We refer to [PKM+24][Lemma B.4] for a detailed proof of this min-entropy bound. As we do at most Q_s signing session, we can bound the probability of programming failure by $Q_s \cdot 2^{-n+2}$.

Programming of H_u on salt fails with probability at most $\frac{Q_s}{2^{2\kappa}}$.

Finally, this hybrid reexpresses equivalently the $[\![w]\!]_i$ and $[\![z]\!]_i$ as functions of $\mathbf{h}_{\mathsf{honest}} := c_1 \cdot \mathbf{s} + \mathrm{HonestP}[\mathsf{sid}]$ and $w_{\mathsf{honest}} := \mathbf{A} \cdot \mathrm{HonestP}[\mathsf{sid}]$.

Summing up everything,

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_9}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_8}(\kappa)\right| \leqslant Q_s \cdot 2^{-n+2} + \frac{Q_s}{2^{2\kappa}}$$

Hybrid_{10}. In this hybrid, we start with the computation of $\mathbf{h}_{\mathsf{honest}}$, and derive other variables from it to preserve the same distribution of the adversary's view. This is formalized in Figure 28.

In Hybrid_9, we had $u := \mathbf{A} \cdot \left(\begin{bmatrix} \mathbf{z} \\ c_1 \end{bmatrix} + \begin{bmatrix} c_2 \\ 0 \\ 0 \end{bmatrix} \right)$ with $\mathbf{z} = c_1 \cdot \mathbf{s} + \mathbf{p}$ and $\mathbf{p}$ reconstructed from the shares $([\![\sum_{j \in \mathsf{valid}} \mathbf{p}]\!]_i)_{i \in \mathsf{honest}}$. And $(c_1, c_2) = \mathsf{Decompose}_\beta(c)$ where $c = u - w$.

We observe that c is uniformly sampled in $\mathcal{R}$ when u is uniform. We can thus exchange computations:

1. We first sample $c \xleftarrow{\$} \mathcal{R}$.
2. We compute $\mathbf{h}_{\mathsf{honest}} = c_1 \cdot \mathbf{s} + \mathrm{HonestP}[\mathsf{sid}]$, and w_{honest} as a function of $\mathbf{h}_{\mathsf{honest}}$ and b.
3. z can also be computed equivalently as it is a function of c_1 and $\mathbf{h}_{\mathsf{honest}}$.
4. Finally, to preserve the adversary's view distribution, we simply need to compute u with the expression given above, as a function of $c_1, c_2, \mathbf{z}$.

Finally,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_{10}}(\kappa) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_9}(\kappa)$$

Conclude. The rest of the proof is analogous to the proof of unmasked Plover [EEN+24, section 3.2] and we just briefly describe it here.

Specifically, our proof concludes in a few more steps:

1. We replace b by a random value, which reduces to $\mathsf{Hint\text{-}MLWE}$. There is however a difference here as we allow the adversary to adversarially choose a seed for generating a. We need an instance of $\mathsf{Hint\text{-}MLWE}$ for each call to H_a which makes us a lose a factor in the advantage.
   The total advantage loss of this step is:
$$Q_a \cdot \mathsf{Adv}^{\mathsf{Hint\text{-}MLWE}}(\kappa)$$
   where Q_a is the number of queries allowed to H_a.
2. Next step, we replace the random oracle H_u by a sample $\mathbf{A} \cdot \mathbf{z}$ with $\mathbf{z} \xleftarrow{\$} \{0\} \times ([-B_2/\sqrt{2n}, B_2/\sqrt{2n}]^n)^2$. This loses an advantage $Q_H \cdot \mathsf{Adv}^{\mathsf{RLWE}}(\kappa)$.
3. After that, we reduce to RSIS with bound $2B_2$ up to an advantage $p_c \leqslant 2^{-n(2\log_2(2B_2/\sqrt{2n})-\log_2(q))}$.

This concludes our proof.

□

C Security proof of Pelican with DKG

In this section, we prove that Pelican remains robust and unforgeable when used in combination with our DKG.

C.1 Robustness

Proof (of Theorem 4). As the robustness proof of Pelican with DKG uses hybrids very similar to the ones introduced to prove the robustness of Pelican, we only give a quick overview of the proof here and refer to other sections for detailed hybrids.

We proceed with a series of hybrids starting from the game $\mathsf{Game}_{\mathsf{DKG\text{-}TH\text{-}RB}}$ from Figure 29.

Hybrid_2. The second hybrid asserts that all parties agree on the same valid set at the end of round 3 of key generation. We refer to 18 for how it is formalized for the robustness of Pelican.

Due to the use of a broadcast channel, all honest parties observe the same messages in round 2 and round 3, and will simultaneously accept or reject complaints.

Hence,
$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DKG\text{-}TH\text{-}RB}}(\kappa) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa)$$

Hybrid_3. In this hybrid, we assert that honest parties do not raise or accept complaints against other honest parties in the key generation.

Since honest contributions failing verification are rejected in both key generation and signature rounds, no honest party will complain about another.

As for complaints coming from corrupted parties, recall that the contributions sent by honest parties always pass verification by construction of the protocol.

So the only way a corrupted complaint may be accepted is if they provide an incorrect key in their complaint in order to alter decryption. But then it means that they forged a signature of $\{i, j, \mathsf{K}_{i\to j}\}$ for some $i \in \mathsf{honest}$. This reduces the advantage by at most $|\mathsf{honest}| \cdot \mathsf{Adv}^{\mathsf{UF\text{-}CMA}}(\kappa)$:

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa)\right| \leqslant N \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa)$$

Hybrid_4. This hybrid removes the restart in round 1 of key generation in case verification fails, and asserts that honest shares are consistent. It also asserts that accepted shares coming from corrupted parties reconstruct to valid secrets. We refer to Figure 20 for an analogous formalization for the robustness of the threshold signature.

By the *correctness* of the V3S, restart in round 1 will happen with probability bounded by $|\mathsf{honest}| \cdot (1 - \Pr[\mathsf{V3S}\text{ correctness}])$. The VSSS *correctness* also ensures that honest shares correctly reconstruct.

As for the shares coming from corrupted parties, this directly reduces to the soundness of the V3S. If the new assertion fails, it gives us an invalid sharing that passes verification over the set of honest parties honest which has a cardinal larger than T.

Thus,

$$|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa)| \leqslant N \cdot (1 - \Pr[\mathsf{V3S}\text{ correctness}]) + \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{V3S\text{-}sound}}(\kappa)$$

Conclusion. We observe at this stage that the key generation necessarily completes with the assertion added, and provides all honest parties with consistent shares of some secret $\mathbf{p} = \sum_{i \in \mathsf{valid}} \mathbf{p}_i$, where (i) at most $T - 1$ $\mathbf{p}_i$ are adversarially chosen and of norm bounded by B, and (ii) the others are sampled from a Gaussian of standard deviation σ_{sk}.

The view of the adversary is then the same as in the game Leak-TH-RB, and we can derive an adversary $\mathcal{B}_3$ playing the Leak-TH-RB game from $\mathcal{A}$. $\mathcal{B}_3$ first runs $\mathcal{A}$until key generation completes, and calls $\mathsf{LeakyKeygen}_{\mathsf{RB}}$ with the secret produced. Then it normally executes the adversary $\mathcal{A}$.

Note that the distribution of keys $\mathcal{K}$ produced by $\mathsf{LeakyKeygen}_{\mathsf{RB}}$ with input generated by $\mathcal{B}_3$ is such that $\|(z_1, z_2, c_1)\| \leqslant B_2$ with overwhelming probability p by assumption of 4, and we can correctly apply 5. □

C.2 Unforgeability

Proof (of Theorem 5). We start with the $\mathsf{Game}_{\mathsf{DKG\text{-}TH\text{-}UF}}$ from Figure 30, and go through a series of hybrids to reduce the advantage of an adversary $\mathcal{A}$ to its advantage against the unforgeability of Pelican with leaky keygen.

For readability, in each hybrid, we omit unmodified functions and directly rewrite the algorithms $\mathsf{Keygen.ShareKeygen}_i$ instead of their associated oracles.

Hybrid_2. In this hybrid, we ensure that no complaint against an honest party is accepted by another honest party. This is formalized in Figure 31.

The only case an honest party will complain against another honest party is if the VSSS verification wrongly fails. However, this happens with negligible probability due to the VSSS correctness. This introduces a loss of advantage $|\mathsf{honest}| \cdot \Pr[\text{VSSS correctness}]$.

As for the possibility that a complaint against an honest party is accepted, there are two possibilities:

– VSSS verification actually fails on some corrupted share. But this is also bounded by the correctness of the VSSS.
– The adversary is able to forge a signature allowing it to complain against an honest party with an invalid pairwise key. This introduces an advantage loss of at most $|\mathsf{honest}| \cdot \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa)$.

Overall,

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}^{\mathsf{DKG\text{-}TH\text{-}UF}}}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_1}(\kappa)\right| \leqslant N \cdot (\Pr[\text{VSSS correctness}] + \mathsf{Adv}_{\mathcal{B}_1}^{\mathsf{UF\text{-}CMA}}(\kappa))$$

Hybrid_3. In this hybrid, we replace the plaintexts in round 1 destined to other honest parties using the SKE with $\varnothing$. This is formalized in Figure 32.

As the previous hybrid made sure $\mathsf{sk}_{i,j}$ is not used in any other place for $i, j \in \mathsf{honest}$, this reduces to the IND-CPA security of the underlying SKE scheme.

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_2}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa)\right| \leqslant N^2 \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{IND\text{-}CPA}}(\kappa)$$

for some efficient adversary $\mathcal{B}_2$ against the IND-CPA security of SKE.

Hybrid_4. In this hybrid, we remove the dependency on honest shares over honest secrets. This is formalized in Figure 33.

Recall that Lagrange interpolation allows us to recover the value in 0 of any polynomial of degree less than T given T of its images. Specifically, for given images over the $S \cup \{i\}$ for $i \in \mathsf{honest}$, there exists coefficients $(\lambda_k^i)_{k \in S \cup \{i\}}$ such that for any polynomial P, we have $P(0) = \sum_{k \in S \cup \{i\}} \lambda_k^i P(k)$.

Due to the correctness of the Fiat Shamir sharing of honest parties, we have for any $i, j \in \mathsf{honest}$ $\mathbf{s}_j = \sum_{k \in S \cup \{i\}} \lambda_k^i [\![\mathbf{s}_j]\!]_k$, or equivalently $[\![\mathbf{s}_j]\!]_i = (\mathbf{s}_j - \sum_{k \in S} \lambda_k^i [\![\mathbf{s}_j]\!]_k) \cdot (\lambda_i^i)^{-1}$.

This is exactly the expression used to replace $[\![\mathbf{s}_j]\!]_i$ in $\mathsf{ShareKeygen}_3$, so there is no advantage loss.

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_3}(\kappa) = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4}(\kappa)$$

Hybrid_5. In this hybrid, we assert that accepted shares from corrupted parties correctly reconstruct among honest parties. This is formalized in Figure 34.

This is enforced by the soundness of the V3S:

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_5}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4}(\kappa)\right| \leqslant \mathsf{Adv}_{\mathcal{B}_3}^{\mathsf{V3S\text{-}sound}}(\kappa)$$

Hybrid_6. In this hybrid, we replace $\mathsf{V3S}$ proofs and shares observed by the attacker using $\mathsf{V3S}$ fragmentary knowledge simulators. This is formalized in Figure 35.

This directly relates to the advantage of an attacker against the fragmentary knowledge property of the underlying $\mathsf{V3S}$. Since we want to simulate $|\mathsf{honest}| \leqslant N$ transcripts, we lose at most an advantage:

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_6}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_5}(\kappa)\right| \leqslant N \cdot \mathsf{Adv}_{\mathcal{B}_4}^{\mathsf{V3S\text{-}fk}}(\kappa)$$

Hybrid_7. In this hybrid, we extract a Gaussian of standard deviation $\sqrt{|\mathsf{honest}|}\sigma'_{\mathsf{sk}}$ from the simulated secrets. This is formalized in Figure 36.

Recall that σ'_{sk} is defined such that:

$$\frac{1}{\sigma'^2_{\mathsf{sk}}} = 2\left(\frac{1}{\sigma^2_{\mathsf{sk}}} + \frac{B}{\sigma_y}^2\right)$$

with B s.t. $s_1(\mathbf{R}^\top\mathbf{R}) < B$ with overwhelming probability. We additionally assume $\sigma'_{\mathsf{sk}} \geqslant 2\eta_\varepsilon(\mathbb{Z}^{2n})$.

We perform this extraction by applying a similar strategy as in the proof of Theorem 1. We proceed in two steps:

- We first extract Gaussians of standard deviation σ'_{sk} from each Gaussian $\mathcal{D}_{\mathbb{Z}^{2n},\mathbf{c}_1,\sqrt{\Sigma_0}}$. This incurs a statistical difference 2ε for each replacement: $2|\mathsf{honest}|\varepsilon$ in total.
- The second step consists in joining all the $\mathcal{D}_{\mathbb{Z}^{2n},0,\sigma'_{\mathsf{sk}}}$ together. Again this incurs an advantage loss of at most 2ε for each replacement.

Hence,

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_7}(\kappa) - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_6}\right| \leqslant 4N \cdot \varepsilon$$

Reduction to the unforgeability of leaky $\mathsf{Pelican}$. We can then reduce the advantage of an adversary against Hybrid_7 to the unforgeability of leaky $\mathsf{Pelican}$. Indeed, we can simulate the view of an adversary in Hybrid_6 by calling $\mathsf{LeakyKeygen}_{\mathsf{UF}}$ in round 3 as a replacement for the sampling of $\mathbf{s}_{\mathsf{honest}}$.

In Hybrid_7, we can equivalently express secret shares as:

$$[\![\mathbf{s}]\!]_i = X \cdot (\lambda_i^i) + \underbrace{\left(\sum_{j \in \mathsf{valid} \cap \mathsf{corrupt}} [\![\mathbf{s}_j]\!]_i + (Y - \sum_{j \in \mathsf{honest} \wedge k \in S} \lambda_k^i [\![\mathbf{s}_j]\!]_k) \cdot (\lambda_i^i)\right)}_{:= [\![\mathbf{s}']\!]_i}$$

for $X = \mathcal{D}_{\mathbb{Z}^{2n},\sqrt{|\mathsf{honest}|}\sigma'_{\mathsf{sk}}}$.

Thus each honest share is shifted by $[\![\mathbf{s}']\!]_i$. We call $\mathsf{LeakyKeygen}_{\mathsf{UF}}$ with $\mathsf{aux} = ([\![\mathbf{s}']\!]_i)_{i \in \mathsf{honest}}$. Note that thanks to the assertion on the reconstructability of accepted corrupted shares, $([\![\mathbf{s}']\!]_i)_{i \in \mathsf{honest}}$ also correctly reconstructs, and the assertion in $\mathsf{LeakyKeygen}_{\mathsf{UF}}$ is fulfilled. □

C.3 A note on the distributed generation of the salt

Although there are several similarities between the distributed key generation (Fig. 5) and distributed signing (Figs. 6 and 7) protocols, they use different methods for generating the salt (signing salt in $\mathsf{ShareSign}_{1,2,3,4}$, and salt for generating a in $\mathsf{ShareKeygen}_{1,2,3}$). A natural question is why distinct methods are used in both cases.

The method used in $\mathsf{ShareKeygen}_{1,2,3,4}$ for generating the salt consists of having each party generate a salt, and of computing the final salt by hashing the individual salts of valid signers. In this setting, this adds a factor Q_H to the $\mathsf{Hint\text{-}RLWE}$ advantage, which we deem acceptable as allowing the adversary to bias a does not reduce security in practice, and distributed key generation would typically be short-lived and only permit a limited number of hash generations. However, using the same method in $\mathsf{ShareSign}_{1,2,3,4}$ would require programming the random oracle on salts corresponding to signatures, which would result in a $\mathsf{Hint\text{-}RLWE}$ with Q_H hints. This would result in an unacceptable degradation of the parameters.

Conversely, in $\mathsf{ShareSign}_{1,2,3}$ the salt is generated by using the randomness inside the commitment w. Transposing this method to the key generation would entail an additional round, which is undesirable.

D Proof of MatrixHint-MLWE

Proof (of Lemma 1).

The proof is analogous to the proof of [KLSS23a, Lemma 7], the main differences being that: (i) we directly consider the full vector $\mathbf{r}$ instead of one of its coordinates, and (ii) we replace polynomials with matrices.

We compute the probability of sampling a value $\mathbf{s} = \mathbf{v}$ with the first distribution conditioned on the values of the $\mathbf{z}_i$:

$$\begin{aligned}
&\Pr\left[\mathbf{s} = \mathbf{v} \wedge \mathbf{M}_i \cdot \mathbf{s} + \mathbf{y}_i = \mathbf{w}_i \mid \mathbf{s} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_1}, \mathbf{y}_i \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_{y,i}}\right] \\
=&\mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_1}(\mathbf{v}) \cdot \prod_{i\in[Q]} \mathcal{D}_{\mathbb{Z}^{n(k+\ell)},\sigma_{y,i}}(\mathbf{w}_i - \mathbf{M}_i \cdot \mathbf{v}) \\
\propto& \exp\left[-\frac{1}{2}\cdot\left(\frac{1}{\sigma_1^2}\mathbf{v}^\top\mathbf{v} + \sum_{i\in[Q]}\frac{1}{\sigma_{y,i}^2}(\mathbf{w}_i - \mathbf{M}_i\cdot\mathbf{v})^\top(\mathbf{w}_i - \mathbf{M}_i\cdot\mathbf{v})\right)\right] \\
=& \exp\left[-\frac{1}{2}\cdot\left((\mathbf{v}-\mathbf{c})^\top\mathbf{\Sigma}_0^{-1}(\mathbf{v}-\mathbf{c}) - \mathbf{c}^\top\mathbf{\Sigma}_0^{-1}\mathbf{c} + \sum_{i\in[Q]}\frac{1}{\sigma_{y,i}^2}\mathbf{w}_i^\top\mathbf{w}_i\right)\right]
\end{aligned}$$

where $\mathbf{c} = \mathbf{\Sigma}_0 \cdot \sum_{i\in[Q]} \frac{1}{\sigma_{y,i}^2}\mathbf{M}_i^\top\mathbf{w}_i$.

We deduce that the conditional probability $\Pr\left[\mathbf{s} = \mathbf{v} \mid \mathbf{M}_i \cdot \mathbf{s} + \mathbf{y}_i = \mathbf{w}_i\right]$ is proportional to $\exp\left[-\frac{1}{2}\cdot(\mathbf{v}-\mathbf{c})^\top\mathbf{\Sigma}_0^{-1}(\mathbf{v}-\mathbf{c})\right] \propto \rho_{\sqrt{\mathbf{\Sigma}_0}}(\mathbf{v}-\mathbf{c})$ for any $\mathbf{w}_0, ..., \mathbf{w}_{m-1} \in \mathcal{R}$. So the two distributions are identical. □

Proof (of Theorem 2).

The proof of this theorem is analogous to the proof of [KLSS23b, Theorem 1].

Let us take $\mathbf{A}, \mathbf{b}$ an MLWE sample for parameters k, ℓ, q, σ.

We start by sampling $\mathbf{s} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)}, \sqrt{\mathbf{\Sigma_1}}, \mathbf{c_s}}, \mathbf{y} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)}, \sqrt{\mathbf{\Sigma}_{y,i}}, \mathbf{c}_i}$.

We then compute for $i \in [Q]$, $\mathbf{z}_i = \mathbf{M}_i \cdot \mathbf{s} + \mathbf{y}_i$, as well as:

- $\mathbf{\Sigma}_0 = (\mathbf{\Sigma}_1^{-1} + \sum_{i \in [Q]} \mathbf{M}_i^\top \mathbf{\Sigma}_{y,i}^{-1} \mathbf{M}_i)^{-1}$.
- $\mathbf{c}_i = \mathbf{\Sigma}_0 \cdot \sum_{i \in [Q]} \mathbf{M}_i^\top \mathbf{\Sigma}_{y,i}^{-1} \mathbf{z}_i$.

.

The smallest eigenvalue λ of $\mathbf{\Sigma}_0$ is equal to the inverse of the largest eigenvalue of $\mathbf{\Sigma}_1^{-1} + \sum_{i \in [Q]} \mathbf{M}_i^\top \mathbf{\Sigma}_{y,i}^{-1} \mathbf{M}_i$. And we can bound it with the spectral norm:

$$\begin{aligned}\lambda &\geqslant s_1(\mathbf{\Sigma}_1^{-1} + \sum_{i \in [Q]} \mathbf{M}_i^\top \mathbf{\Sigma}_{y,i}^{-1} \mathbf{M}_i)^{-1} \\ &\geqslant \left(s_1(\mathbf{\Sigma}_1^{-1}) + \sum_{i \in [Q]} s_1(\mathbf{M}_i^\top \mathbf{M}_i) \cdot s_1(\mathbf{\Sigma}_{y,i}^{-1}) \right)^{-1} = 2\sigma^2\end{aligned}$$

In particular, $\mathbf{\Sigma}_0 - \sigma^2 \mathbf{I}$ is positive semi-definite, and the matrix $\sqrt{\mathbf{\Sigma}_0 - \sigma^2 \mathbf{I}}$ is well defined.

After that, we sample $\mathbf{t} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)}, \mathbf{c}_i, \sqrt{\mathbf{\Sigma}_0 - \sigma^2 \mathbf{I}}}$.

We use those polynomials to transform the MLWE sample into an MatrixHint-MLWE sample as $(\mathbf{A}, \mathbf{b} + \begin{bmatrix} \mathbf{I} & \mathbf{A} \end{bmatrix} \cdot \mathbf{t}, c_0, ..., c_{Q-1}, \mathbf{z}_0, ..., \mathbf{z}_{Q-1})$.

Let us show that is has correct distribution.

- First, if $\mathbf{b}$ is uniformly sampled, then so is $\mathbf{b} + \begin{bmatrix} \mathbf{I} & \mathbf{A} \end{bmatrix} \mathbf{t}$ as $\mathbf{t}$ is independent of $\mathbf{b}$. The distribution of the $c_i, \mathbf{z}_i$ is correct by construction.
- Now consider that $\mathbf{b} = \begin{bmatrix} \mathbf{I} & \mathbf{A} \end{bmatrix} \mathbf{s}'$, with $\mathbf{s}' \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)}, \sigma}$.
  Then, conditioned on the values of the $\mathbf{M}_i, \mathbf{z}_i$, we have that $\mathbf{s}' + \mathbf{t}$ follows the distribution $\mathcal{D}_{\mathbb{Z}^{n(k+\ell)}, \sigma} + \mathcal{D}_{\mathbb{Z}^{n(k+\ell)}, \mathbf{c}, \sqrt{\mathbf{\Sigma}_0 - \sigma^2 \mathbf{I}}}$.
  Same as in [KLSS23b], this distribution is at statistical distance at most 2ε of $\hat{\mathbf{s}} \leftarrow \mathcal{D}_{\mathbb{Z}^{n(k+\ell)}, \mathbf{c}, \sqrt{\Sigma_0}}$. We can thus replace $\mathbf{s}' + \mathbf{t}$ with $\hat{\mathbf{s}}$.
  We can finally apply Lemma 1 – sligthly generalized, in order to replace $(\hat{\mathbf{s}}, c_0, ..., c_{Q-1}, \mathbf{z}_0, ..., \mathbf{z}_{Q-1})$ with $(\mathbf{s}, c_0, ..., c_{Q-1}, \mathbf{z}_0, ..., \mathbf{z}_{Q-1})$.

□

E Distribution of submersion matrices R

Proof (of Lemma 3).

- Case 1: $\mathbf{s} = \mathbf{s}'$ and $\mathbf{y} \neq \mathbf{y}'$.
  Then, $(\mathbf{R}\mathbf{s}' + \mathbf{y}') - (\mathbf{R}\mathbf{s} + \mathbf{y}') = (\mathbf{y} - \mathbf{y}') \neq 0 \bmod q$.
  Equality never happens in that case.

– Case 2: $\mathbf{s} \neq \mathbf{s}'$. Note i such that $s_i \neq s'_i$.
  Then, for each line $\mathbf{r}_j$ of $\mathbf{R}$, we have

$$\langle \mathbf{r}_j, \mathbf{s} \rangle = \langle \mathbf{r}_j, \mathbf{s}' \rangle \bmod q$$
$$\iff r_{j,i}(s_i - s'_i) = \sum_{i' \neq i} r_{j,i'}(s'_{i'} - s_{i'}) \bmod q$$

  The term on the left is independent of the term on the right, and since $s_i - s'_i \neq 0$, the above equality happens for at most one of $r_{j,i} \in \{0, \pm 1\}$. Hence, it happens with probability at most $1/2$.
  Thus the probability that all the coefficients $\langle \mathbf{r}_j, \mathbf{s} \rangle$ are equal to $\langle \mathbf{r}_j, \mathbf{s}' \rangle$ is at most 2^{-256}.

□

Proof (of Lemma 4). We want to evaluate $s_1(\mathbf{R}^\top \mathbf{R}) = s_1(\mathbf{R}\mathbf{R}^\top)$. For $j \in [256]$, let us note $\mathbf{r}_j$ the i-th row of $\mathbf{R}_i$.

We have $\mathbf{R}\mathbf{R}^\top = (\langle \mathbf{r}_j, \mathbf{r}_k \rangle)_{j,k \in [256]}$. Since $\mathbf{R}\mathbf{R}^\top$ is symmetric, its largest eigenvalue is also its spectral norm. For any $j, k \in [256]$, we have $|\langle \mathbf{r}_j, \mathbf{r}_j \rangle| \leqslant 2n$. Therefore via Gershgorin's disc theorem, $s_1(\mathbf{R}\mathbf{R}^\top) \leqslant 512 \cdot n$.

As noted in the theorem's statement this analysis is very coarse, and when fixing n we can perform a more refined average analysis.

One method is to bound $\sum_{j \in [256] \wedge j \neq i_0} |\langle \mathbf{r}_{i_0}, \mathbf{r}_j \rangle|$ with overwhelming probability for any i_0. We observe that when $\mathbf{r}_{i_0}$ is fixed, the worse case is for $\mathbf{r}_{i_0}$ having no zero coordinate. We can actually prove that the distribution of $\sum_{j \in [256] \wedge j \neq i_0} |\langle \mathbf{r}_{i_0}, \mathbf{r}_j \rangle|$ conditioned on the number of zeros k in $\mathbf{r}_{i_0}$ only disperses towards higher value when k increases. It is thus sufficient to overwhelmingly bound $\sum_{j \neq i_0} |\langle \mathbf{1}, \mathbf{r}_j \rangle|$.

The latter is easier to perform as it is a sum of independent variables. For $n = 2048$, a software evaluation of the distribution $\sum_{j \neq i_0} |\langle \mathbf{1}, \mathbf{r}_j \rangle|$ using the security scripts for Kyber[8] tells us that it is lower than 16000 with probability at least $1 - 2^{-142}$. It follows with Gershgorin's disc theorem that $s_1(\mathbf{R}_i \mathbf{R}_i^\top) \leqslant 16000 + 2n = 20096$ with overwhelming probability. □

[8] https://github.com/pq-crystals/security-estimates

```
Game_{Leak-TH-RB}
1: $L_H, L_{\mathsf{sid}} := \varnothing$
2: $\mathsf{vk}_0, \mathsf{sk}_0 := \varnothing$
3: $(N, T, \mathsf{corrupt}) \leftarrow \mathcal{A}(\mathsf{pp}, 1^\kappa)$
4: assert{ $\mathsf{corrupt} \subseteq \{1, ..., N\} \wedge T \leqslant N/3$ }
5: assert{ $|\mathsf{corrupt}| < T$ }
6: $\mathsf{honest} := \{1, ..., N\} \backslash \mathsf{corrupt}$
7: $(\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,...,N\}} \leftarrow \mathsf{Sign.Setup}(\mathsf{pp}, N)$
8: $\mathsf{aux} \leftarrow \mathcal{A}^H((\mathsf{pk}_i^{\mathsf{Setup}})_{i \in \{1,...,N\}}, (\mathsf{sk}_i^{\mathsf{Setup}})_{i \in \mathsf{corrupt}})$
                                        ▷ aux allows the adversary to bias key generationy
9: $(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \{1,...,N\}}) \leftarrow \mathsf{Sign.LeakyKeygen}(\mathsf{pp}, T, N, \mathsf{aux}, (\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,...,N\}})$
10: for $i \in \mathsf{honest}$ do
11:     $\mathsf{state}_i.\mathsf{sk} := \mathsf{sk}_i$
12:     $\mathsf{state}_i.\mathsf{vk} := \mathsf{vk}$
13: $\mathsf{sid}, (\mathsf{out}_i)_{i \in \mathsf{corrupt}} \leftarrow \mathcal{A}^{H, (\mathsf{OPerformRound}_i(\cdot))_{i \in [\mathsf{rnd}_{\mathsf{Sign}}]}}(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \mathsf{corrupt}})$
14: Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
15: if $\mathsf{rnd} \neq \mathsf{rnd}_{\mathsf{Sign}}$ then
16:     return 0                        ▷ The adversary did not finish the protocol
17: $\mathsf{status}, \mathsf{sig} := \mathsf{Sign.Combine}(\mathsf{vk}, \mathsf{msg}, (\mathsf{out}_i)_{i \in \{1,...,N\}})$
18: if $\mathsf{status} = \mathbf{ok} \wedge \mathsf{Sign.Verify}(\mathsf{vk}, \mathsf{sig})$ then
19:     return 0                        ▷ The protocol produced a valid signature
20: return 1

    OPerformRound_1(sid, msg)
    1: assert{ $\mathsf{sid} \notin L_{\mathsf{sid}}$ }
    2: for $i \in \mathsf{honest}$ do
    3:     $\mathsf{out}_i := \mathsf{Sign.ShareSign}_1(\mathsf{state}_i, \mathsf{sid}, \mathsf{msg})$
    4: $L_{\mathsf{sid}}[\mathsf{sid}] = \{1, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

    OPerformRound_k((in_i)_{i∈corrupt}), for $k \in \{2, ..., \mathsf{rnd}_{\mathsf{Sign}}\}$
    1: Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{in}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
    2: assert{ $\mathsf{rnd} = k - 1$ }
    3: for $i \in \mathsf{honest}$ do
    4:     $\mathsf{out}_i := \mathsf{Sign.ShareSign}_k(\mathsf{state}_i, \mathsf{sid}, (\mathsf{in})_{i \in \{1,...,N\}})$
    5: $L_{\mathsf{sid}}[\mathsf{sid}] = \{k, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

    H(str, digest)
    1: assert{ $\mathsf{str} \in \mathsf{pp}.\mathrm{HashParams}$ }          ▷ Check domain string
    2: if $\exists r.(\mathsf{str}, \mathsf{digest}, r) \in L_H$ then
    3:     return $r$
    4: else
    5:     Sample $r$ uniformly
    6:     $L_H := L_H \cup \{(\mathsf{str}, \mathsf{digest}, r)\}$
    7:     return $r$
```

Fig. 17: Robustness game for a leaky threshold signature. The adversary $\mathcal{A}$ wins if the game Leak-TH-RB returns 1, i.e. if it is able to prevent the protocol from producing a valid signature.

```
Hybrid$_2$

$L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
▷▷▷ Other lines are identical ◁◁◁

    ShareSign$_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$

    Fetch $(\mathsf{rnd}, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j \in \mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1)$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
    assert{ $\mathsf{rnd} = 2$ }
    $\mathsf{valid}_i := \mathsf{valid}_i \cap \{j \in \mathsf{contrib}_2\}$
    for $j \in \mathsf{valid}_i$ do
        for $k \in \mathsf{complaints}_j$ do
            $\{\mathsf{K}_{k \to j}, \mathsf{sig}_{k \to j}\} := \mathsf{complaints}_j[k]$
            if $\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k \to j}, \{k, j, \mathsf{K}_{k \to j}\}) = \mathsf{false}$ then
                continue
            $[\![\mathbf{s}_k]\!]_j, \pi_{k \to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k \to j}, \mathsf{ct}_{k \to j})$
            if ($\mathsf{SKE.Decrypt}$ failed) or $\mathsf{V3S.Verify}([\![\mathbf{p}_k]\!]_j, \pi_k, \pi_{k \to j}) = \mathsf{false}$ then
                $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$
    if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
        $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
    assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}]$ }
    $[\![\mathbf{p}]\!]_i := \sum_{j \in \mathsf{valid}_i} [\![\mathbf{p}_j]\!]_i$                ▷ $\mathbf{p} = (p_1, p_2)$
    $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{3, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j \in \mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1\}$
    return $\mathsf{contrib}_3[i] := ([\![w]\!]_i := \mathbf{A} \cdot [\![\mathbf{p}]\!]_i)$
```

Fig. 18: The second hybrid of the security proof of the robustness of Pelican. Difference with the previous hybrid are highlighted.

```
Hybrid$_3$
$L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
▷▷▷ Other lines are identical ◁◁◁

  ShareSign$_2(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_1)$
  Fetch $(\mathsf{rnd}, \mathsf{msg})$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
  assert{ $\mathsf{rnd} = 1$ }
  $\mathsf{complaints}_i := \{\}$
  for $(j \in \mathsf{contrib}_1 \cap \mathsf{corrupt})$ do
    $\pi_j, (\mathsf{ct}_{j\to k})_{k\in\{1,\ldots,N\}} := \mathsf{contrib}_1[j]$
    $[\![\mathbf{p}_j]\!]_i, \pi_{j\to i} := \mathsf{SKE.Decrypt}(\mathsf{K}_{j\to i}, \mathsf{ct}_{j\to i})$
    if ($\mathsf{SKE.Decrypt}$ failed) or ($\mathsf{V3S.Verify}([\![\mathbf{p}_j]\!]_i, \pi_j, \pi_{j\to i}) = \mathsf{false}$) then
      $\mathsf{complaints}_i[j] = \{\mathsf{K}_{j\to i}, \mathsf{sig}_{j\to i}\}$
  $\mathsf{valid}_i = \{j \in \mathsf{contrib}_1\}\backslash\mathsf{complaints}_i$
  $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{\mathsf{rnd} = 2, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1\}$
  return $\mathsf{contrib}_2[i] := \mathsf{complaints}_i$
  ShareSign$_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$
  Fetch $(\mathsf{rnd}, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1)$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
  assert{ $\mathsf{rnd} = 2$ }
  $\mathsf{valid}_i := \mathsf{valid}_i \cap \{j \in \mathsf{contrib}_2\}$
  for $j \in \{1, \ldots, N\}$ do
    for $k \in \mathsf{complaints}_j$ do
      if $k \in \mathsf{honest}$ then
        continue
      $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\} := \mathsf{complaints}_j[k]$
      if $\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}) = \mathsf{false}$ then
        continue
      $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j})$
      if ($\mathsf{SKE.Decrypt}$ failed) or $\mathsf{V3S.Verify}([\![\mathbf{p}_k]\!]_j, \pi_k, \pi_{k\to j}) = \mathsf{false}$ then
        $\mathsf{valid}_i = \mathsf{valid}_i\backslash\{k\}$
  if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
    $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
  assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}] \wedge \mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ }
  $[\![\mathbf{p}]\!]_i := \sum_{j\in\mathsf{valid}_i} [\![\mathbf{p}_j]\!]_i$    ▷ $\mathbf{p} = (p_1, p_2)$
  $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{3, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1\}$
  return $\mathsf{contrib}_3[i] := ([\![w]\!]_i := \mathbf{A}\cdot[\![\mathbf{p}]\!]_i)$
```

Fig. 19: The third hybrid of the security proof of the robustness of Pelican. Difference with the previous hybrid are highlighted.

```
Hybrid$_4$
1: $L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
   ▷▷▷ Other lines are identical ◁◁◁

     ShareSign$_1$(state$_i$, sid, contrib$_2$)
        ▷▷▷ Remove restart in case verification fails ◁◁◁
     ShareSign$_3$(state$_i$, sid, contrib$_2$)
     1: Fetch $(\mathsf{rnd}, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j \in \mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1)$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
     2: assert{ $\mathsf{rnd} = 2$ }
     3: $\mathsf{valid}_i := \mathsf{valid}_i \cap \{j \in \mathsf{contrib}_2\}$
     4: for $j \in \{1, \dots, N\}$ do
     5:   for $k \in \mathsf{complaints}_j$ do
     6:     if $k \in \mathsf{honest}$ then
     7:       continue
     8:     $\{\mathsf{K}_{k \to j}, \mathsf{sig}_{k \to j}\} := \mathsf{complaints}_j[k]$
     9:     if $\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k \to j}, \{k, j, \mathsf{K}_{k \to j}\}) = \mathsf{false}$ then
    10:       continue
    11:     $[\![\mathbf{s}_k]\!]_j, \pi_{k \to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k \to j}, \mathsf{ct}_{k \to j})$
    12:     if (SKE.Decrypt failed) or $\mathsf{V3S.Verify}([\![\mathbf{p}_k]\!]_j, \pi_k, \pi_{k \to j}) = \mathsf{false}$ then
    13:       $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$
    14: if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
    15:   $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
    16: assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}] \wedge \mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ }
    17: for $j \in \mathsf{valid}_i$ do
    18:   if $j \in \mathsf{corrupt}$ then
    19:     assert{ $\mathsf{V3S.Reconstruct}(([\![\mathbf{p}_j]\!]_k)_{k \in \mathsf{honest}}) \in V$ }
    20:   else
    21:     assert{ $\mathsf{V3S.Reconstruct}(([\![\mathbf{p}_j]\!]_k)_{k \in \mathsf{honest}}) \neq \perp$ }
    22: $[\![\mathbf{p}]\!]_i := \sum_{j \in \mathsf{valid}_i} [\![\mathbf{p}_j]\!]_i$                ▷ $\mathbf{p} = (p_1, p_2)$
    23: $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{3, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j \in \mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1\}$
    24: return $\mathsf{contrib}_3[i] := ([\![w]\!]_i := \mathbf{A} \cdot [\![\mathbf{p}]\!]_i)$
```

Fig. 20: The fourth hybrid of the security proof of the robustness of Pelican. Difference with the previous hybrid are highlighted.

```
Game_{TH-UF}
$L_H, L_{\mathsf{sid}}, L_{\mathsf{sig}} := \varnothing$
$(N, T, \mathsf{corrupt}) \leftarrow \mathcal{A}(\mathsf{pp}, 1^\kappa)$
**assert**{ $\mathsf{corrupt} \subseteq \{1, ..., N\} \wedge T \leqslant N/3$ }
**assert**{ $|\mathsf{corrupt}| < T$ }
$\mathsf{honest} := \{1, ..., N\} \backslash \mathsf{corrupt}$
$(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \{1,...,N\}}) \leftarrow \mathsf{Sign.Keygen}(\mathsf{pp}, T, N)$
**for** $i \in \mathsf{honest}$ **do**
    $\mathsf{state}_i.\mathsf{sk} := \mathsf{sk}_i$
    $\mathsf{state}_i.\mathsf{vk} := \mathsf{vk}$
$(\mathsf{msg}, \mathsf{sig}) \leftarrow \mathcal{A}^{H, (\mathsf{OPerformRound}_i(\cdot))_{i \in [\mathsf{rnd}_{\mathsf{Sign}}]}}(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \mathsf{corrupt}})$
**if** $(\mathsf{msg} \in L_{\mathsf{sig}})$ or $\neg\mathsf{Sign.Verify}(\mathsf{vk}, \mathsf{msg}, \mathsf{sig})$ **then**
    **return** 0
**return** 1

    $\mathsf{OPerformRound}_1(\mathsf{sid}, \mathsf{msg})$
    **assert**{ $\mathsf{sid} \notin L_{\mathsf{sid}}$ }
    **for** $i \in \mathsf{honest}$ **do**
        $\mathsf{out}_i := \mathsf{Sign.ShareSign}_1(\mathsf{state}_i, \mathsf{sid}, \mathsf{msg})$
    $L_{\mathsf{sig}} := L_{\mathsf{sig}} \cup \{\mathsf{msg}\}$
    $L_{\mathsf{sid}}[\mathsf{sid}] = \{1, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

    $\mathsf{OPerformRound}_k((\mathsf{in}_i)_{i \in \mathsf{corrupt}})$, for $k \in \{2, ..., \mathsf{rnd}_{\mathsf{Sign}}\}$
    Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{in}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
    **assert**{ $\mathsf{rnd} = k - 1$ }
    **for** $i \in \mathsf{honest}$ **do**
        $\mathsf{out}_i := \mathsf{Sign.ShareSign}_k(\mathsf{state}_i, \mathsf{sid}, (\mathsf{in})_{i \in \{1,...,N\}})$
    $L_{\mathsf{sid}}[\mathsf{sid}] = \{k, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

    $H(\mathsf{str}, \mathsf{digest})$
    **assert**{ $\mathsf{str} \in \mathsf{pp}.\mathrm{HashParams}$ }            ▷ Check domain string
    **if** $\exists r.(\mathsf{str}, \mathsf{digest}, r) \in L_H$ **then**
        **return** $r$
    **else**
        Sample $r$ uniformly
        $L_H := L_H \cup \{(\mathsf{str}, \mathsf{digest}, r)\}$
        **return** $r$
```

Fig. 21: Unforgeability game for a threshold signature. The adversary $\mathcal{A}$ wins if the game $\mathsf{TH\text{-}UF}$ returns 1, i.e. if it forged a new signature.

```
$\mathsf{Game}_{\mathsf{Leak\text{-}TH\text{-}UF}}$
$L_H, L_{\mathsf{sid}}, L_{\mathsf{sig}} := \varnothing$
$(N, T, \mathsf{corrupt}) \leftarrow \mathcal{A}(\mathsf{pp}, 1^\kappa)$
assert{ $\mathsf{corrupt} \subseteq \{1, ..., N\} \wedge T \leqslant N/3$ }
assert{ $|\mathsf{corrupt}| < T$ }
$\mathsf{honest} := \{1, ..., N\} \setminus \mathsf{corrupt}$
$(\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,...,N\}} \leftarrow \mathsf{Sign.Setup}(\mathsf{pp}, N)$
$\mathsf{aux} \leftarrow \mathcal{A}^H((\mathsf{pk}_i^{\mathsf{Setup}})_{i \in \{1,...,N\}}, (\mathsf{sk}_i^{\mathsf{Setup}})_{i \in \mathsf{corrupt}})$
                                        ▷ aux allows the adversary to bias key generation
$(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \{1,...,N\}}) \leftarrow \mathsf{Sign.LeakyKeygen}(\mathsf{pp}, T, N, \mathsf{aux}, (\mathsf{pk}_i^{\mathsf{Setup}}, \mathsf{sk}_i^{\mathsf{Setup}})_{i \in \{1,...,N\}})$
for $i \in \mathsf{honest}$ do
    $\mathsf{state}_i.\mathsf{sk} := \mathsf{sk}_i$
    $\mathsf{state}_i.\mathsf{vk} := \mathsf{vk}$
$(\mathsf{msg}, \mathsf{sig}) \leftarrow \mathcal{A}^{H, (\mathsf{OPerformRound}_i(\cdot))_{i \in [\mathsf{rnd}_{\mathsf{Sign}}]}}(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \mathsf{corrupt}})$
if $(\mathsf{msg} \in L_{\mathsf{sig}})$ or $\neg\mathsf{Sign.Verify}(\mathsf{vk}, \mathsf{msg}, \mathsf{sig})$ then
    return 0
return 1

    $\mathsf{OPerformRound}_1(\mathsf{sid}, \mathsf{msg})$
    assert{ $\mathsf{sid} \notin L_{\mathsf{sid}}$ }
    for $i \in \mathsf{honest}$ do
        $\mathsf{out}_i := \mathsf{Sign.ShareSign}_1(\mathsf{state}_i, \mathsf{sid}, , \mathsf{msg})$
    $L_{\mathsf{sig}} := L_{\mathsf{sig}} \cup \{\mathsf{msg}\}$
    $L_{\mathsf{sid}}[\mathsf{sid}] = \{1, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

    $\mathsf{OPerformRound}_k((\mathsf{in}_i)_{i \in \mathsf{corrupt}})$, for $k \in \{2, ..., \mathsf{rnd}_{\mathsf{Sign}}\}$
    Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{in}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
    assert{ $\mathsf{rnd} = k - 1$ }
    for $i \in \mathsf{honest}$ do
        $\mathsf{out}_i := \mathsf{Sign.ShareSign}_k(\mathsf{state}_i, \mathsf{sid}, (\mathsf{in})_{i \in \{1,...,N\}})$
    $L_{\mathsf{sid}}[\mathsf{sid}] = \{k, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$

    $H(\mathsf{str}, \mathsf{digest})$
    assert{ $\mathsf{str} \in \mathsf{pp}.\mathrm{HashParams}$ }          ▷ Check domain string
    if $\exists r.(\mathsf{str}, \mathsf{digest}, r) \in L_H$ then
        return $r$
    else
        Sample $r$ uniformly
        $L_H := L_H \cup \{(\mathsf{str}, \mathsf{digest}, r)\}$
        return $r$
```

Fig. 22: Unforgeability game for a threshold signature with leaky keygen. The adversary $\mathcal{A}$ wins if the game Leak-TH-UF returns 1, i.e. if it forged a new signature.

```
Hybrid$_5$
1: $L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
   ▷▷▷ Other lines are identical ◁◁◁

   ShareSign$_1$(state$_i$, sid, act, msg)
   1: $\mathbf{p}_i \leftarrow \mathcal{D}_{\mathbf{p}}^2$
   2: $[\![\mathbf{p}_i]\!], \pi_i, (\pi_{i\to j})_{j\in\{1,\ldots,N\}} \leftarrow$ V3S.Share$(N, T, \mathbf{p}_i)$
   3: if $\exists j$ s.t. V3S.Verify$([\![\mathbf{p}_j]\!]_i, \pi_j, \pi_{j\to i}) =$ false then
   4:    restart
   5: for $j \in \{1, \ldots, N\}$ do
   6:    if $j \in$ honest then
   7:       $\mathsf{pt}_j := \varnothing$
   8:    else
   9:       $\mathsf{pt}_j := ([\![\mathbf{p}_i]\!]_j, \pi_{i\to j})$
   10:   $\mathsf{ct}_{i\to j} \leftarrow$ SKE.Encrypt$(\mathsf{K}_{i\to j}, \mathsf{pt}_j)$
   11: state$_i$.session[sid] := {rnd = 1, msg, $[\![\mathbf{p}_i]\!]_i, \varnothing, \varnothing$}
   12: return contrib$_1[i] := (\pi_i, (\mathsf{ct}_{i\to j})_{j\in\{1,\ldots,N\}})$
   ShareSign$_3$(state$_i$, sid, contrib$_2$)
      ▷▷▷ Store honest shares in round 1 and recover them in round 3 ◁◁◁
```

Fig. 23: The fifth hybrid of the security proof of the unforgeability of Pelican. Difference with the previous hybrid are highlighted .

```
Hybrid_6
1: $L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
   $\triangleright\triangleright\triangleright$ Other lines are identical $\triangleleft\triangleleft\triangleleft$

   $\mathsf{ShareSign}_1(\mathsf{state}_i, \mathsf{sid}, \mathsf{act}, \mathsf{msg})$
   1: Fix a set $S$ of cardinal $T-1$ s.t. $\mathsf{corrupt} \subset S$ for the session
   2: $\mathbf{p}_i \leftarrow \mathcal{D}_{\mathbf{p}}^2$
   3: $[\![\mathbf{p}_i]\!], \pi_i, (\pi_{i\to j})_{j\in\{1,\ldots,N\}} \leftarrow \mathsf{V3S.Share}(N, T, \mathbf{p}_i)$
   4: if $\exists j$ s.t. $\mathsf{V3S.Verify}([\![\mathbf{p}_j]\!]_i, \pi_j, \pi_{j\to i}) = \mathsf{false}$ then
   5:     restart
   6: for $j \in \{1, \ldots, N\}$ do
   7:     if $j \in \mathsf{honest}$ then
   8:         $\mathsf{pt}_j := \varnothing$
   9:     else
   10:        $\mathsf{pt}_j := ([\![\mathbf{p}_i]\!]_j, \pi_{i\to j})$
   11:    $\mathsf{ct}_{i\to j} \leftarrow \mathsf{SKE.Encrypt}(\mathsf{K}_{i\to j}, \mathsf{pt}_j)$
   12: $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{\mathsf{rnd} = 1, \mathsf{msg}, [\![\mathbf{p}_i]\!]_i, \varnothing, \varnothing\}$
   13: return $\mathsf{contrib}_1[i] := (\pi_i, (\mathsf{ct}_{i\to j})_{j\in\{1,\ldots,N\}})$

   $\mathsf{ShareSign}_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$
   $\triangleright\triangleright\triangleright$ Other lines are identical $\triangleleft\triangleleft\triangleleft$
   1: if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
   2:     $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
   3: assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}] \wedge \mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ }
   4: for $j \in \mathsf{valid}_i$ do
   5:     if $j \in \mathsf{corrupt}$ then
   6:         assert{ $\mathsf{V3S.Reconstruct}(([\![\mathbf{p}_j]\!]_k)_{k\in\mathsf{honest}}) \in V$ }
   7:     else
   8:         assert{ $\mathsf{V3S.Reconstruct}(([\![\mathbf{p}_j]\!]_k)_{k\in\mathsf{honest}}) \neq \perp$ }
   9: $[\![\mathbf{p}]\!]_i := \sum_{j\in\mathsf{valid}_i} [\![\mathbf{p}_j]\!]_i$          $\triangleright$ $\mathbf{p} = (p_1, p_2)$
   10: $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{3, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1\}$
   11: $[\![w]\!]_i := \mathbf{A} \cdot \sum_{j\in\mathsf{valid}\cap\mathsf{corrupt}} [\![\mathbf{p}_j]\!]_i$
   12: $[\![w]\!]_i = [\![w]\!]_i + \mathbf{A} \cdot \sum_{j\in\mathsf{honest}} (\mathbf{p}_j - \sum_{k\in S} \lambda_k^i [\![\mathbf{p}_j]\!]_k) \cdot (\lambda_i^i)^{-1}$
   13: return $\mathsf{contrib}_3[i] := ([\![w]\!]_i)$

   $\mathsf{ShareSign}_4(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_3)$
   1: Fetch $(\mathsf{rnd}, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1)$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
   2: assert{ $\mathsf{rnd} = 3$ }
   3: Parse $\mathsf{contrib}_3 = (\mathsf{valid}_j, [\![w]\!]_j)_{j\in\mathsf{act}}$
   4: $w := \mathsf{V3S.RobustReconstruct}(([\![w]\!]_j)_{j\in\mathsf{valid}_i})$          $\triangleright$ $w = \mathbf{A} \cdot \mathbf{p}$, where
      $\mathbf{p} = \sum_{j\in\mathsf{valid}_i} \mathbf{p}_j$
   5: $\mathsf{salt} := H_{\mathsf{salt}}(w)$
   6: $u := H_u(\mathsf{vk}, \mathsf{salt}, \mathsf{msg})$
   7: $c := u - w$
   8: $(c_1, c_2) := \mathsf{Decompose}_\beta(c)$
   9: $[\![z]\!]_i := \sum_{j\in\mathsf{valid}_i\cap\mathsf{corrupt}} [\![p_{j,2}]\!]_i$
   10: $[\![z]\!]_i = [\![z]\!]_i + c_1 \cdot (s - \sum_{k\in S} \lambda_k^i [\![s]\!]_k \cdot (\lambda_i^i)^{-1})$
   11: $[\![z]\!]_i = [\![z]\!]_i + \sum_{j\in\mathsf{honest}} (p_{j,2} - \sum_{k\in S} \lambda_k^i [\![p_{j,2}]\!]_k) \cdot (\lambda_i^i)^{-1}$
                                        $\triangleright$ Recall $\mathbf{s} = (s, e)$ and $\mathbf{p}_j = (p_{j,1}, p_{j,2})$
   12: $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \varnothing$
   13: return $\mathsf{contrib}_4[i] = (\mathsf{salt}, [\![z]\!]_i, c_1)$
```

Fig. 24: The sitxh hybrid of the security proof of the unforgeability of Pelican. Difference with the previous hybrid are highlighted .

```
Hybrid$_7$

1: $\mathrm{HonestP}[\cdot] := \varnothing$
2: $L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
   $\triangleright\triangleright\triangleright$ Other lines are identical $\triangleleft\triangleleft\triangleleft$

   ShareSign$_1(\mathsf{state}_i, \mathsf{sid}, \mathsf{act}, \mathsf{msg})$
   1: Fix a set $S$ of cardinal $T-1$ s.t. $\mathsf{corrupt} \subset S$ for the session
   2: $(⟦\mathbf{p}_i⟧_{j\in S}), \pi_i, (\pi_{i\to j})_{j\in S} \leftarrow \mathsf{SimProof}(S)$
   3: for $j \in \{1, ..., N\}$ do
   4:     if $j \in \mathsf{honest}$ then
   5:         $\mathsf{pt}_j := \varnothing$
   6:     else
   7:         $\mathsf{pt}_j := (⟦\mathbf{p}_i⟧_j, \pi_{i\to j})$
   8:     $\mathsf{ct}_{i\to j} \leftarrow \mathsf{SKE.Encrypt}(\mathsf{K}_{i\to j}, \mathsf{pt}_j)$
   9: $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}] := \{\mathsf{rnd} = 1, \mathsf{msg}, ⟦\mathbf{p}_i⟧_i, \varnothing, \varnothing\}$
   10: return $\mathsf{contrib}_1[i] := (\pi_i, (\mathsf{ct}_{i\to j})_{j\in\{1,...,N\}})$
   ShareSign$_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$
       $\triangleright\triangleright\triangleright$ Other lines are identical $\triangleleft\triangleleft\triangleleft$
   1: if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
   2:     $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
   3: assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}] \wedge \mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ }
   4: for $j \in \mathsf{valid}_i$ do
   5:     if $j \in \mathsf{corrupt}$ then
   6:         assert{ $\mathsf{V3S.Reconstruct}((⟦\mathbf{p}_j⟧_k)_{k\in\mathsf{honest}}) \in V$ }
   7:     else
   8:         assert{ $\mathsf{V3S.Reconstruct}((⟦\mathbf{p}_j⟧_k)_{k\in\mathsf{honest}}) \neq \bot$ }
   9: if $\mathrm{HonestP}[\mathsf{sid}] = \bot$ then
   10:     $\mathrm{HonestP}[\mathsf{sid}] \leftarrow \sum_{j\in\mathsf{honest}} \mathsf{SimSecret}(\pi_j, (\pi_{j\to k})_{k\in S})$
   11: $⟦w⟧_i := \mathbf{A} \cdot \sum_{j\in\mathsf{valid}\cap\mathsf{corrupt}} ⟦\mathbf{p}_j⟧_i$
   12: $⟦w⟧_i = ⟦w⟧_i + \mathbf{A} \cdot \left(\mathrm{HonestP}[\mathsf{sid}] - \sum_{j\in\mathsf{honest}} \sum_{k\in S} \lambda_k^i ⟦\mathbf{p}_j⟧_k\right) \cdot (\lambda_i^i)^{-1}$
   13: return $\mathsf{contrib}_3[i] := (⟦w⟧_i)$
   ShareSign$_4(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_3)$
       $\triangleright\triangleright\triangleright$ Replace occurence of $\sum_{j\in\mathsf{honest}} \mathbf{p}_j$ with $\mathrm{HonestP}[\mathsf{sid}]$ $\triangleleft\triangleleft\triangleleft$
```

Fig. 25: The seventh hybrid of the security proof of the unforgeability of Pelican. Difference with the previous hybrid are highlighted .

```
Hybrid$_8$
HonestP$[\cdot] := \varnothing$
$L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
▷▷▷ Other lines are identical ◁◁◁

    ShareSign$_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$
    Fetch $(\mathsf{rnd}, \mathsf{msg}, ([\![\mathbf{p}_j]\!]_i)_{j \in \mathsf{valid}_i}, \mathsf{valid}_i, \mathsf{contrib}_1)$ from $\mathsf{state}_i.\mathsf{session}[\mathsf{sid}]$
    assert{ $\mathsf{rnd} = 2$ }
    $\mathsf{valid}_i := \mathsf{valid}_i \cap \{j \in \mathsf{contrib}_2\}$
    for $j \in \mathsf{valid}_i$ do
      for $k \in \mathsf{complaints}_j$ do
        if $k \in \mathsf{honest}$ then
          continue
        $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\} := \mathsf{complaints}_j[k]$
        if $\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}) = \mathsf{false}$ then
          continue
        $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j})$
        if ($\mathsf{SKE.Decrypt}$ failed) or $\mathsf{V3S.Verify}([\![\mathbf{s}_k]\!]_j, \pi_k, \pi_{k\to j}) = \mathsf{false}$ then
          $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$
    if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
      $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
    assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}] \wedge \mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ }
    for $j \in \mathsf{valid}_i$ do
      if $j \in \mathsf{corrupt}$ then
        assert{ $\mathsf{V3S.Reconstruct}(([\![\mathbf{p}_j]\!]_k)_{k \in \mathsf{honest}}) \in V$ }
      else
        assert{ $\mathsf{V3S.Reconstruct}(([\![\mathbf{p}_j]\!]_k)_{k \in \mathsf{honest}}) \neq \bot$ }
    if HonestP$[\mathsf{sid}] = \bot$ then
      HonestP$[\mathsf{sid}] \leftarrow \mathcal{D}_{\mathbb{Z}^{2n}, \sqrt{|\mathsf{honest}|}\sigma'_{\mathbf{p}}} + Y$
                          ▷ Where $Y$ is a random variable independent of the Gaussian
    $[\![w]\!]_i := \mathbf{A} \cdot \sum_{j \in \mathsf{valid} \cap \mathsf{corrupt}} [\![\mathbf{p}_j]\!]_i$
    $[\![w]\!]_i = [\![w]\!]_i + \mathbf{A} \cdot \Big(\text{HonestP}[\mathsf{sid}] - \sum_{j \in \mathsf{honest}} \sum_{k \in S} \lambda_k^i [\![\mathbf{p}_j]\!]_k\Big) \cdot (\lambda_i^i)^{-1}$
    return $\mathsf{contrib}_3[i] := ([\![w]\!]_i)$
```

Fig. 26: The eigth hybrid of the security proof of the unforgeability of Pelican. Difference with the previous hybrid are highlighted .

```
$\mathsf{Hybrid}_9$
$\mathrm{HonestP}[\cdot], \mathrm{Programmed}[\cdot] := \varnothing$
$L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
▷▷▷ Other lines are identical ◁◁◁

    $\mathsf{ShareSign}_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$
      ▷▷▷ Other lines are identical ◁◁◁
    if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
      $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
    assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}] \wedge \mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ }
    for $j \in \mathsf{valid}_i$ do
      if $j \in \mathsf{corrupt}$ then
        assert{ $\mathsf{V3S.Reconstruct}((\llbracket \mathbf{p}_j \rrbracket_k)_{k \in \mathsf{honest}}) \in V$ }
      else
        assert{ $\mathsf{V3S.Reconstruct}((\llbracket \mathbf{p}_j \rrbracket_k)_{k \in \mathsf{honest}}) \neq \perp$ }
    if $\mathrm{HonestP}[\mathsf{sid}] = \perp$ then
      $\mathrm{HonestP}[\mathsf{sid}] \leftarrow \mathcal{D}_{\mathbb{Z}^{2n}, \sqrt{|\mathsf{honest}|}\sigma'_{\mathbf{p}}} + Y$
      $w_{\mathsf{honest}} = \mathbf{A} \cdot \mathrm{HonestP}[\mathsf{sid}]$
      for $j \in \mathsf{honest}$ do
        $\llbracket w \rrbracket_j := \mathbf{A} \cdot \sum_{j \in \mathsf{valid} \cap \mathsf{corrupt}} \llbracket \mathbf{p}_j \rrbracket_i$
        $\llbracket w \rrbracket_j = \llbracket w \rrbracket_j + \big(w_{\mathsf{honest}} - \mathbf{A} \cdot \sum_{l \in \mathsf{honest}} \sum_{k \in S} \lambda^i_k \llbracket \mathbf{p}_j \rrbracket_k\big) \cdot (\lambda^j_j)^{-1}$
      $w := \mathsf{V3S.Reconstruct}((\llbracket w \rrbracket_i)_{i \in \mathsf{honest}})$
      $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$
      $u \xleftarrow{\$} \mathcal{R}$
      $c := u - w$
      $(c_1, c_2) := \mathsf{Decompose}_\beta(c)$
      $\mathbf{h}_{\mathsf{honest}} = c_1 \cdot \mathbf{s} + \mathrm{HonestP}[\mathsf{sid}]$
      $\llbracket z \rrbracket_i := \sum_{j \in \mathsf{valid} \cap \mathsf{corrupt}} \llbracket p_{j,2} \rrbracket_i$
      $\llbracket z \rrbracket_i = \llbracket z \rrbracket_i + h_{\mathsf{honest},1} \cdot (\lambda^i_i)^{-1} + \sum_{j \in \mathsf{honest}} (\sum_{k \in S} \lambda^i_k (c_1 \cdot \llbracket s \rrbracket_k + \llbracket p_{j,2} \rrbracket_k)) \cdot (\lambda^i_i)^{-1}$
      $\mathrm{Programmed}[\mathsf{sid}] := (\llbracket w \rrbracket_i, \llbracket z \rrbracket_i)_{i \in \mathsf{honest}}$
      $H_{\mathsf{salt}}(w) := \mathsf{salt};\ H_u(\mathsf{vk}, \mathsf{salt}, \mathsf{msg}) = u$        ▷ Program random oracle
    return $\mathsf{contrib}_3[i] := (\mathsf{valid}_i, \mathrm{Programmed}[\mathsf{sid}][0]_i)$
    $\mathsf{ShareSign}_4(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_3)$
      ▷▷▷ Return $\llbracket z \rrbracket_i$ as programmed in $\mathrm{Programmed}[\mathsf{sid}]$ ◁◁◁
```

Fig. 27: The nineth hybrid of the security proof of the unforgeability of Pelican. Difference with the previous hybrid are highlighted . In case the random oracle has previously been queried on programmed values, we consider that the adversary wins.

```
Hybrid$_{10}$
$\text{HonestP}[\cdot], \text{Programmed}[\cdot] := \varnothing$
$L_{\mathsf{sig}}, L_H, L_{\mathsf{valid}} := \varnothing$
$\triangleright\triangleright\triangleright$ Other lines are identical $\triangleleft\triangleleft\triangleleft$

    ShareSign$_3(\mathsf{state}_i, \mathsf{sid}, \mathsf{contrib}_2)$
    $\triangleright\triangleright\triangleright$ Other lines are identical $\triangleleft\triangleleft\triangleleft$
    if $\mathsf{sid} \notin L_{\mathsf{valid}}$ then
        $L_{\mathsf{valid}}[\mathsf{sid}] = \mathsf{valid}_i$
    assert{ $\mathsf{valid}_i = L_{\mathsf{valid}}[\mathsf{sid}] \wedge \mathsf{honest} \subset L_{\mathsf{valid}}[\mathsf{sid}]$ }
    for $j \in \mathsf{valid}_i$ do
        if $j \in \mathsf{corrupt}$ then
            assert{ $\mathsf{V3S.Reconstruct}((\llbracket \mathbf{p}_j \rrbracket_k)_{k \in \mathsf{honest}}) \in V$ }
        else
            assert{ $\mathsf{V3S.Reconstruct}((\llbracket \mathbf{p}_j \rrbracket_k)_{k \in \mathsf{honest}}) \neq \bot$ }
    if $\text{HonestP}[\mathsf{sid}] = \bot$ then
        $c \xleftarrow{\$} \mathcal{R}$
        $(c_1, c_2) := \mathsf{Decompose}_\beta(c)$
        $\text{HonestP}[\mathsf{sid}] \leftarrow \mathcal{D}_{\mathbb{Z}^{2n}, \sqrt{|\mathsf{honest}|}\sigma'_{\mathbf{p}}} + Y$
        $\mathbf{h}_{\mathsf{honest}} = c_1 \cdot \mathbf{s} + \text{HonestP}[\mathsf{sid}]$
        $w_{\mathsf{honest}} = \begin{bmatrix} 1 & a \end{bmatrix} \cdot \mathbf{h}_{\mathsf{honest}} - c_1 \cdot b - c_1 \cdot \beta$        $\triangleright$ Recall $b = a \cdot s + e$
        for $j \in \mathsf{honest}$ do
            $\llbracket w \rrbracket_j := \mathbf{A} \cdot \sum_{j \in \mathsf{valid} \cap \mathsf{corrupt}} \llbracket \mathbf{p}_j \rrbracket_i$
            $\llbracket w \rrbracket_j = \llbracket w \rrbracket_j + \left(w_{\mathsf{honest}} - \mathbf{A} \cdot \sum_{l \in \mathsf{honest}} \sum_{k \in S} \lambda_k^i \llbracket \mathbf{p}_j \rrbracket_k\right) \cdot (\lambda_j^j)^{-1}$
        $w := \mathsf{V3S.Reconstruct}((\llbracket w \rrbracket_i)_{i \in \mathsf{honest}})$
        $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$
        $\llbracket \mathbf{z} \rrbracket_i := \sum_{j \in \mathsf{valid} \cap \mathsf{corrupt}} \llbracket \mathbf{p}_j \rrbracket_i$
        $\llbracket \mathbf{z} \rrbracket_i = \llbracket \mathbf{z} \rrbracket_i + \mathbf{h}_{\mathsf{honest}} \cdot (\lambda_i^i)^{-1} + \sum_{j \in \mathsf{honest}} (\sum_{k \in S} \lambda_k^i (c_1 \cdot \llbracket \mathbf{s} \rrbracket_k + \llbracket \mathbf{p}_j \rrbracket_k)) \cdot (\lambda_i^i)^{-1}$
        $\text{Programmed}[\mathsf{sid}] := (\llbracket w \rrbracket_i, \llbracket \mathbf{z} \rrbracket_i)_{i \in \mathsf{honest}}$
        $\mathbf{z}' := \begin{bmatrix} \mathsf{V3S.Reconstruct}((\llbracket \mathbf{z} \rrbracket_i)_{i \in \mathsf{honest}}) \\ c_1 \end{bmatrix} + \begin{bmatrix} c_2 \\ 0 \\ 0 \end{bmatrix}$
        $u := \mathbf{A} \cdot \mathbf{z}'$
        $H_{\mathsf{salt}}(w) := \mathsf{salt}$; $H_u(\mathsf{vk}, \mathsf{salt}, \mathsf{msg}) = u$        $\triangleright$ Program random oracle
    return $\mathsf{contrib}_3[i] := (\mathsf{valid}, \text{Programmed}[\mathsf{sid}][0]_i)$
```

Fig. 28: The tenth hybrid of the security proof of the unforgeability of Pelican. Difference with the previous hybrid are highlighted .

```
Game_DKG-TH-RB
1: $L_{\mathsf{sig}}, L_H, L_{\mathsf{sid}} := \varnothing$
2: $L_{\mathsf{Keygen}} = \varnothing$
3: $\mathsf{vk}_0, \mathsf{sk}_0 := \varnothing$
4: $(N, T, \mathsf{corrupt}) \leftarrow \mathcal{A}(\mathsf{pp}, 1^\kappa)$
5: assert$\{ \mathsf{corrupt} \subseteq \{1, ..., N\} \wedge |\mathsf{corrupt}| < T \}$
6: $\mathsf{honest} := \{1, ..., N\} \backslash \mathsf{corrupt}$
7: for $i \in \mathsf{honest}$ do
8:    $\mathsf{pk}_i^{\mathsf{in}}, \mathsf{sk}_i^{\mathsf{in}} := \mathsf{Setup}()$
9:    $\mathsf{state}_i.\mathsf{sk} := \mathsf{sk}_i^{\mathsf{in}}$
10: $(\mathsf{pk}_i^{\mathsf{in}})_{i \in \mathsf{corrupt}}, \mathsf{state}_{\mathcal{A}} \leftarrow \mathcal{A}^H((\mathsf{pk}_i^{\mathsf{in}})_{i \in \mathsf{honest}})$
11: for $i \in \mathsf{honest}$ do
12:    $\mathsf{state}_i.\mathsf{pk} := (\mathsf{pk}_i^{\mathsf{in}})_{i \in \{1,\ldots,N\}}$
    ▷▷▷ Run first round ◁◁◁
13: for $i \in \mathsf{honest}$ do
14:    $\mathsf{out}_i := \mathsf{Keygen.ShareKeygen}_1(\mathsf{state}_i)$
15: $L_{\mathsf{Keygen}}.\mathsf{rnd} = 1$; $L_{\mathsf{Keygen}}.\mathsf{out} = \mathsf{out}$
16: $(\mathsf{out}_i^{\mathsf{Keygen}})_{i \in \mathsf{corrupt}}, (\mathsf{out}_i^{\mathsf{Sign}})_{i \in \mathsf{corrupt}}, (\mathsf{msg}, \mathsf{sig}) \leftarrow \mathcal{A}^{\mathrm{oracles}}(\mathsf{state}_{\mathcal{A}}, L_{\mathsf{Keygen}}.\mathsf{out})$
17: if $L_{\mathsf{Keygen}}.\mathsf{rnd} \neq \mathsf{rnd}_{\mathsf{Keygen}}$ then
18:    return 0                         ▷ The adversary must finish key generation
19: $\mathsf{status}, \mathsf{vk} := \mathsf{Keygen.CombineKey}(\mathsf{contrib}_3)$
20: if $\mathsf{status} = $ abort then
21:    return 1                         ▷ The adversary made the DKG fail
22: Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{out}_i^{\mathsf{Sign}})_{i \in \mathsf{honest}}, \mathsf{msg}\}$
23: if $\mathsf{rnd} \neq \mathsf{rnd}_{\mathsf{Sign}}$ then
24:    return 0                         ▷ The adversary did not finish the protocol
25: $\mathsf{status}, \mathsf{sig} := \mathsf{Sign.Combine}(\mathsf{vk}, \mathsf{msg}, (\mathsf{out}_i^{\mathsf{Sign}})_{i \in \{1,\ldots,N\}})$
26: if $\mathsf{status} = $ ok $\wedge\ \mathsf{Sign.Verify}(\mathsf{vk}, \mathsf{sig})$ then
27:    return 0                         ▷ The protocol produced a valid signature
28: return 1

    OPerformRound_k((in_i)_{i∈corrupt}), for $k \in \{2, ..., \mathsf{rnd}_{\mathsf{Keygen}}\}$
    1: assert$\{ L_{\mathsf{Keygen}}.\mathsf{rnd} = k - 1 \}$
    2: for $i \in \mathsf{honest}$ do
    3:    $\mathsf{out}_i := \mathsf{Keygen.ShareKeygen}_k(\mathsf{state}_i, L_{\mathsf{Keygen}}.\mathsf{out})$
    4: $L_{\mathsf{Keygen}}.\mathsf{rnd} = k$; $L_{\mathsf{Keygen}}.\mathsf{out} = (\mathsf{out}_i)_{i \in \mathsf{honest}}$
    5: return $(\mathsf{out}_i)_{i \in \mathsf{honest}}$

    OPerformSignRound_1(sid, msg)
    1: assert$\{ L_{\mathsf{Keygen}}.\mathsf{rnd} = \mathsf{rnd}_{\mathsf{Keygen}} \wedge \mathsf{sid} \notin L_{\mathsf{sid}} \}$
    2: for $i \in \mathsf{honest}$ do
    3:    $\mathsf{out}_i := \mathsf{Sign.ShareSign}_1(\mathsf{state}_i, \mathsf{sid}, \mathsf{msg})$
    4: $L_{\mathsf{sig}} := L_{\mathsf{sig}} \cup \{\mathsf{msg}\}$
    5: $L_{\mathsf{sid}}[\mathsf{sid}] = \{1, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
    6: return $(\mathsf{out}_i)_{i \in \mathsf{honest}}$

    OPerformSignRound_k((in_i)_{i∈corrupt}), for $k \in \{2, ..., \mathsf{rnd}_{\mathsf{Sign}}\}$
    1: Fetch $L_{\mathsf{sid}}[\mathsf{sid}] = \{\mathsf{rnd}, (\mathsf{in}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
    2: assert$\{ \mathsf{rnd} = k - 1 \}$
    3: for $i \in \mathsf{honest}$ do
    4:    $\mathsf{out}_i := \mathsf{Sign.ShareSign}_k(\mathsf{state}_i, \mathsf{sid}, (\mathsf{in})_{i \in \{1,\ldots,N\}})$
    5: $L_{\mathsf{sid}}[\mathsf{sid}] = \{k, (\mathsf{out}_i)_{i \in \mathsf{honest}}, \mathsf{msg}\}$
    6: return $(\mathsf{out}_i)_{i \in \mathsf{honest}}$

    H(str, digest)
    1: assert$\{ \mathsf{str} \in \mathsf{pp}.\mathrm{HashParams} \}$          ▷ Check domain string
    2: if $\exists r.(\mathsf{str}, \mathsf{digest}, r) \in L_H$ then
    3:    return $r$
    4: else
    5:    Sample $r$ uniformly
    6:    $L_H := L_H \cup \{(\mathsf{str}, \mathsf{digest}, r)\}$
    7:    return $r$
```

Fig. 29: Robustness game of threshold signature with DKG. The adversary wins $\mathcal{A}$ if $\mathsf{Game}_{\mathcal{A}}^{\mathsf{DKG\text{-}TH\text{-}RB}}$ returns 1.

```
$\mathsf{Game}_{\mathsf{DKG\text{-}TH\text{-}UF}}$
$L_{\mathsf{sig}}, L_H, L_{\mathsf{sid}} := \varnothing$
$L_{\mathsf{Keygen}} = \varnothing$
$\mathsf{vk}_0, \mathsf{sk}_0 := \varnothing$
$(N, T, \mathsf{corrupt}) \leftarrow \mathcal{A}(\mathsf{pp}, 1^\kappa)$
**assert**$\{$ $\mathsf{corrupt} \subseteq \{1, ..., N\}$ $\}$
**assert**$\{$ $|\mathsf{corrupt}| < T$ $\}$
$\mathsf{honest} := \{1, ..., N\} \backslash \mathsf{corrupt}$
**for** $i \in \mathsf{honest}$ **do**
    $\mathsf{pk}_i^{\mathsf{in}}, \mathsf{sk}_i^{\mathsf{in}} := \mathsf{Setup}()$
    $\mathsf{state}_i.\mathsf{sk} := \mathsf{sk}_i^{\mathsf{in}}$
$(\mathsf{pk}_i^{\mathsf{in}})_{i \in \mathsf{corrupt}}, \mathsf{state}_{\mathcal{A}} \leftarrow \mathcal{A}^H((\mathsf{pk}_i^{\mathsf{in}})_{i \in \mathsf{honest}})$
**for** $i \in \mathsf{honest}$ **do**
    $\mathsf{state}_i.\mathsf{pk} := (\mathsf{pk}_i^{\mathsf{in}})_{i \in \{1,\dots,N\}}$
$\triangleright\triangleright\triangleright$ Run first round $\triangleleft\triangleleft\triangleleft$
**for** $i \in \mathsf{honest}$ **do**
    $\mathsf{out}_i := \mathsf{Keygen.ShareKeygen}_1(\mathsf{state}_i)$
$L_{\mathsf{Keygen}}.\mathsf{rnd} = 1$; $L_{\mathsf{Keygen}}.\mathsf{out} = \mathsf{out}$
$(\mathsf{out}_i)_{i \in \mathsf{corrupt}}, (\mathsf{msg}, \mathsf{sig}) \leftarrow \mathcal{A}^{\mathrm{oracles}}(\mathsf{state}_{\mathcal{A}}, L_{\mathsf{Keygen}}.\mathsf{out})$
**if** $L_{\mathsf{Keygen}}.\mathsf{rnd} \neq \mathsf{rnd}_{\mathsf{Keygen}}$ **then**
    **return** $0$                              $\triangleright$ The adversary must finish key generation
$\mathsf{status}, \mathsf{vk} := \mathsf{Keygen.CombineKey}(\mathsf{contrib}_3)$
**if** $\mathsf{status} = $ **abort then**          $\triangleright$ The adversary must return a valid DKG output
    **return** $0$
**if** $(\mathsf{msg} \in L_{\mathsf{sig}})$ or $\mathsf{Sign.Verify}(\mathsf{vk}, \mathsf{msg}, \mathsf{sig}) = 0$ **then**
    **return** $0$
**return** $1$

$\triangleright\triangleright\triangleright$ Same oracles $(\mathsf{OPerformRound}_i), (\mathsf{OPerformSignRound}_i), H$ as Fig. 29. $\triangleleft\triangleleft\triangleleft$
```

Fig. 30: Unforgeability game of threshold signature with DKG. The adversary wins $\mathcal{A}$ if $\mathsf{Game}_{\mathcal{A}}^{\mathsf{DKG\text{-}TH\text{-}UF}}$ returns 1.

```
$\mathsf{Hybrid}_2$

    $\triangleright\triangleright\triangleright$ Identical to $\mathsf{Hybrid}_1$ $\triangleleft\triangleleft\triangleleft$

    $\mathsf{ShareKeygen}_2(\mathsf{state}_i, \mathsf{contrib}_1)$
    assert{ $\mathsf{state.rnd} = 1$ }; $\mathsf{state.rnd} = 2$; $\mathsf{valid} = \{\}$
    $\mathsf{complaints}_i := \{\}$
    for $(j \in \mathsf{contrib}_1 \cap \mathsf{corrupt})$ do
        $\mathsf{salt}_i, \pi_j, (\mathsf{ct}_{j\to k})_{k\in\{1,\ldots,N\}} := \mathsf{contrib}_1[j]$
        $[\![\mathbf{s}_j]\!]_i, \pi_{j\to i} := \mathsf{SKE.Decrypt}(\mathsf{K}_{j\to i}, \mathsf{ct}_{j\to i})$
        if $(\mathsf{SKE.Decrypt}$ failed$)$ or $(\mathsf{V3S.Verify}([\![\mathbf{s}_j]\!]_i, \pi_j, \pi_{j\to i}) = \mathsf{false})$ then
            $\mathsf{complaints}[j] = \{\mathsf{K}_{j\to i}, \mathsf{sig}_{j\to i}\}$
    $\mathsf{valid}_i = \{j \in \mathsf{contrib}_1\} \backslash \mathsf{complaints}_i$
    $\mathsf{state}_i.\mathsf{session.valid} = \mathsf{valid}_i$
    $\mathsf{state}_i.\mathsf{session.contrib}_1 := \mathsf{contrib}_1$
    $\mathsf{state}_i.\mathsf{session.shares} := ([\![\mathbf{p}_j]\!]_i)_{j\in\mathsf{valid}}$
                                   $\triangleright$ Implicitly retrieve honest shares from round 1
    return $\mathsf{contrib}_2[i] := \mathsf{complaints}_i$
    $\mathsf{ShareKeygen}_3(\mathsf{state}_i, \mathsf{contrib}_2)$
    assert{ $\mathsf{state.rnd} = 2$ }; $\mathsf{state.rnd} = 3$
    $\mathsf{valid}_i := \mathsf{state}_i.\mathsf{session.valid} \cap \{j \in \mathsf{contrib}_2\}$
    for $j \in \mathsf{valid}_i$ do
        for $k \in \mathsf{complaints}_j$ do
            if $k \in \mathsf{honest}$ then
                continue                       $\triangleright$ Ignore complaints against honest parties
            $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\} := \mathsf{complaints}_j[k]$
            if $\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}) = \mathsf{false}$ then
                continue
            $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j})$
            if $(\mathsf{SKE.Decrypt}$ failed$)$ or $\mathsf{V3S.Verify}([\![\mathbf{s}_k]\!]_j, \pi_k, \pi_{k\to j}) = \mathsf{false}$ then
                $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$
    $\mathsf{state}_i.\mathsf{session.salt} := H_{\mathsf{salt}}((\mathsf{salt}_j)_{j\in\mathsf{valid}_i})$
    $a := H_a(\mathsf{state}_i.\mathsf{session.salt})$
    $[\![\mathbf{s}]\!]_i := \sum_{j\in\mathsf{valid}_i} [\![\mathbf{s}_j]\!]_i$                  $\triangleright$ $\mathbf{s} = \sum_{j\in\mathsf{valid}} \mathbf{s}_j$
    $[\![b]\!]_i := [a\ 1] \cdot [\![\mathbf{s}]\!]_i$
    Store $[\![\mathbf{s}]\!]_i$ in $\mathsf{state}_i$
    return $\mathsf{contrib}_3[i] := (\mathsf{state}_i.\mathsf{session.salt}, [\![b]\!]_i)$
```

Fig. 31: The second hybrid of the security proof of the unforgeability of Pelican with DKG. Difference with the previous hybrid are highlighted .

```
Hybrid_3
    ▷▷▷ Identical to Hybrid_1 ◁◁◁

    ShareKeygen_1(state_i)
    assert{ state.rnd = ∅ }; state.rnd = 1
    $\mathsf{salt}_i \xleftarrow{\$} \{0,1\}^\kappa$
    $\mathbf{s}_i \leftarrow \mathcal{D}^2_{\sigma_t}$
    $([\![\mathbf{s}_i]\!], \pi_i, (\pi_{i,j})_{j\in\{1,\ldots,N\}}) \leftarrow \mathsf{V3S.Share}(N, T, \mathbf{s}_i)$
    for $j \in \{1, \ldots, N\}$ do
      if $j \in \mathsf{honest}$ then
        $\mathsf{pt}_j := \varnothing$
      else
        $\mathsf{pt}_j := ([\![\mathbf{s}_i]\!]_j, \pi_{i,j})$
      $\mathsf{ct}_{i,j} \leftarrow \mathsf{SKE.Encrypt}(K_{i,j}, \mathsf{pt}_j)$
    return $\mathsf{contrib}_1[i] := \mathsf{salt}_i, \pi_i, (\mathsf{ct}_{i,j})_{j\in\{1,\ldots,N\}}$
```

Fig. 32: The third hybrid of the security proof of the unforgeability of Pelican with DKG. Difference with the previous hybrid are highlighted .

```
Hybrid$_4$
  ▹▹▹ Identical to Hybrid$_1$ ◃◃◃

  ShareKeygen$_1$(state$_i$)
    assert{ state.rnd $= \varnothing$ }; state.rnd $= 1$
    Fix a set $S \supseteq$ corrupt of cardinal $T - 1$
    $\mathsf{salt}_i \xleftarrow{\$} \{0,1\}^\kappa$
    $\mathbf{s}_i \leftarrow \mathcal{D}^2_{\sigma_t}$
    $([\![\mathbf{s}_i]\!], \pi_i, (\pi_{i,j})_{j\in\{1,\ldots,N\}}) \leftarrow$ V3S.Share$(N, T, \mathbf{s}_i)$
    for $j \in \{1, \ldots, N\}$ do
      if $j \in$ honest then
        $\mathsf{pt}_j := \varnothing$
      else
        $\mathsf{pt}_j := ([\![\mathbf{s}_i]\!]_j, \pi_{i,j})$
      $\mathsf{ct}_{i,j} \leftarrow$ SKE.Encrypt$(K_{i,j}, \mathsf{pt}_j)$
    return $\mathsf{contrib}_1[i] := \mathsf{salt}_i, \pi_i, (\mathsf{ct}_{i,j})_{j\in\{1,\ldots,N\}}$

  ShareKeygen$_3$(state$_i$, contrib$_2$)
    assert{ state.rnd $= 2$ }; state.rnd $= 3$
    $\mathsf{valid}_i :=$ state$_i$.session.valid $\cap \{j \in \mathsf{contrib}_2\}$
    for $j \in \mathsf{valid}_i$ do
      for $k \in \mathsf{complaints}_j$ do
        if $k \in$ honest then
          continue
        $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\} := \mathsf{complaints}_j[k]$
        if SIG.Verify$(\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}) =$ false then
          continue
        $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j} :=$ SKE.Decrypt$(\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j})$
        if (SKE.Decrypt failed) or V3S.Verify$([\![\mathbf{s}_k]\!]_j, \pi_k, \pi_{k\to j}) =$ false then
          $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$
    state$_i$.session.salt $:= H_{\mathsf{salt}}((\mathsf{salt}_j)_{j\in\mathsf{valid}_i})$
    $a := H_a$(state$_i$.session.salt)
    $[\![b]\!]_i := \beta - [1\ a] \cdot \sum_{j\in\mathsf{valid}_i\cap\mathsf{corrupt}} [\![\mathbf{s}_j]\!]_i$
    $[\![b]\!]_i = [\![b]\!]_i - [1\ a] \cdot \sum_{j\in\mathsf{honest}} (\mathbf{s}_j - \sum_{k\in S} \lambda^i_k [\![\mathbf{s}]\!]_k) \cdot (\lambda^i_i)^{-1}$
    Store $[\![\mathbf{s}]\!]_i$ in state$_i$
    return $\mathsf{contrib}_3[i] :=$ (state$_i$.session.salt, $[\![b]\!]_i$)
```

Fig. 33: The fourth hybrid of the security proof of the unforgeability of Pelican with DKG. Difference with the previous hybrid are highlighted .

Hybrid_5

```
▷▷▷ Identical to $\mathsf{Hybrid}_1$ ◁◁◁

$\mathsf{ShareKeygen}_3(\mathsf{state}_i, \mathsf{contrib}_2)$

assert{ $\mathsf{state.rnd} = 2$ }; $\mathsf{state.rnd} = 3$
$\mathsf{valid}_i := \mathsf{state}_i.\mathsf{session.valid} \cap \{j \in \mathsf{contrib}_2\}$
for $j \in \mathsf{valid}_i$ do
  for $k \in \mathsf{complaints}_j$ do
    if $k \in \mathsf{honest}$ then
      continue
    $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\} := \mathsf{complaints}_j[k]$
    if $\mathsf{SIG.Verify}(\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}) = \mathsf{false}$ then
      continue
    $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j} := \mathsf{SKE.Decrypt}(\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j})$
    if ($\mathsf{SKE.Decrypt}$ failed) or $\mathsf{V3S.Verify}([\![\mathbf{s}_k]\!]_j, \pi_k, \pi_{k\to j}) = \mathsf{false}$ then
      $\mathsf{valid}_i = \mathsf{valid}_i \backslash \{k\}$
assert{ $\mathsf{V3S.Reconstruct}\left(\left(\sum_{j\in\mathsf{valid}_i\cap\mathsf{corrupt}}[\![\mathbf{s}_j]\!]_i\right)_{i\in\mathsf{honest}}\right) \neq \perp$ }
$\mathsf{state}_i.\mathsf{session.salt} := H_{\mathsf{salt}}((\mathsf{salt}_j)_{j\in\mathsf{valid}_i})$
$a := H_a(\mathsf{state}_i.\mathsf{session.salt})$
$[\![b]\!]_i := \beta - [1\ a] \cdot \sum_{j\in\mathsf{valid}_i\cap\mathsf{corrupt}}[\![\mathbf{s}_j]\!]_i$
$[\![b]\!]_i = [\![b]\!]_i - [1\ a] \cdot \sum_{j\in\mathsf{honest}}(\mathbf{s}_j - \sum_{k\in S}\lambda_k^i[\![\mathbf{s}]\!]_k) \cdot (\lambda_i^i)^{-1}$
Store $[\![\mathbf{s}]\!]_i$ in $\mathsf{state}_i$
return $\mathsf{contrib}_3[i] := (\mathsf{state}_i.\mathsf{session.salt}, [\![b]\!]_i)$
```

Fig. 34: The fifth hybrid of the security proof of the unforgeability of Pelican with DKG. Difference with the previous hybrid are highlighted .

```
Hybrid$_6$

1: $\mathsf{s}_{\mathsf{honest}} := \varnothing$
   ▷▷▷ Identical to Hybrid$_1$ ◁◁◁

   ShareKeygen$_1$(state$_i$)

    1: assert{ state.rnd $= \varnothing$ }; state.rnd $= 1$
    2: Fix a set $S \supseteq$ corrupt of cardinal $T - 1$
    3: salt$_i \xleftarrow{\$} \{0,1\}^\kappa$
    4: $((\llbracket \mathbf{s}_i \rrbracket_j)_{j \in S}, \pi_i, (\pi_{i,j})_{j \in S}) \leftarrow$ V3S.SimProof$(S)$
    5: for $j \in \{1, \ldots, N\}$ do
    6:    if $j \in$ honest then
    7:       pt$_j := \varnothing$
    8:    else
    9:       pt$_j := (\llbracket \mathbf{s}_i \rrbracket_j, \pi_{i,j})$
   10:    ct$_{i,j} \leftarrow$ SKE.Encrypt$(K_{i,j}, \mathsf{pt}_j)$
   11: return contrib$_1[i] :=$ salt$_i, \pi_i, (\mathsf{ct}_{i,j})_{j \in \{1,\ldots,N\}}$

   ShareKeygen$_3$(state$_i$, contrib$_2$)

    1: assert{ state.rnd $= 2$ }; state.rnd $= 3$
    2: valid$_i :=$ state$_i$.session.valid $\cap \{j \in$ contrib$_2\}$
    3: for $j \in$ valid$_i$ do
    4:    for $k \in$ complaints$_j$ do
    5:       if $k \in$ honest then
    6:          continue
    7:       $\{\mathsf{K}_{k \to j}, \mathsf{sig}_{k \to j}\} :=$ complaints$_j[k]$
    8:       if SIG.Verify$(\mathsf{pk}_k, \mathsf{sig}_{k \to j}, \{k, j, \mathsf{K}_{k \to j}\}) =$ false then
    9:          continue
   10:       $\llbracket \mathbf{s}_k \rrbracket_j, \pi_{k \to j} :=$ SKE.Decrypt$(\mathsf{K}_{k \to j}, \mathsf{ct}_{k \to j})$
   11:       if (SKE.Decrypt failed) or V3S.Verify$(\llbracket \mathbf{s}_k \rrbracket_j, \pi_k, \pi_{k \to j}) =$ false then
   12:          valid$_i$ = valid$_i \setminus \{k\}$
   13: assert{ V3S.Reconstruct $\left(\left(\sum_{j \in \mathsf{valid}_i \cap \mathsf{corrupt}} \llbracket \mathbf{s}_j \rrbracket_i\right)_{i \in \mathsf{honest}}\right) \neq \perp$ }
   14: state$_i$.session.salt $:= H_{\mathsf{salt}}((\mathsf{salt}_j)_{j \in \mathsf{valid}_i})$
   15: $a := H_a(\mathsf{state}_i.\mathsf{session.salt})$
   16: $\llbracket b \rrbracket_i := \beta - [1\ a] \cdot \sum_{j \in \mathsf{valid}_i \cap \mathsf{corrupt}} \llbracket \mathbf{s}_j \rrbracket_i$
   17: if $\mathsf{s}_{\mathsf{honest}} = \varnothing$ then
   18:    $\mathsf{s}_{\mathsf{honest}} := \sum_{j \in \mathsf{honest}}$ V3S.SimSecret$(\pi_j, (\pi_{j \to k})_{k \in S})$
   19: $\llbracket b \rrbracket_i = \llbracket b \rrbracket_i - [1\ a] \cdot \left(\mathsf{s}_{\mathsf{honest}} - \sum_{j \in \mathsf{honest}} \sum_{k \in S} \lambda_k^i \llbracket \mathbf{s}_j \rrbracket_k\right) \cdot (\lambda_i^i)^{-1}$
   20: Store $\llbracket \mathbf{s} \rrbracket_i$ in state$_i$
   21: return contrib$_3[i] :=$ (state$_i$.session.salt, $\llbracket b \rrbracket_i$)
```

Fig. 35: The sixth hybrid of the security proof of the unforgeability of Pelican with DKG. Difference with the previous hybrid are highlighted .

```
Hybrid$_7$

    ▷▷▷ Identical to Hybrid$_6$ ◁◁◁

    ShareKeygen$_3$(state$_i$, contrib$_2$)

    assert{ state.rnd $= 2$ }; state.rnd $= 3$
    valid$_i$ := state$_i$.session.valid $\cap \{j \in$ contrib$_2\}$
    for $j \in$ valid$_i$ do
      for $k \in$ complaints$_j$ do
        if $k \in$ honest then
          continue
        $\{\mathsf{K}_{k\to j}, \mathsf{sig}_{k\to j}\}$ := complaints$_j[k]$
        if SIG.Verify($\mathsf{pk}_k, \mathsf{sig}_{k\to j}, \{k, j, \mathsf{K}_{k\to j}\}$) = false then
          continue
        $[\![\mathbf{s}_k]\!]_j, \pi_{k\to j}$ := SKE.Decrypt($\mathsf{K}_{k\to j}, \mathsf{ct}_{k\to j}$)
        if (SKE.Decrypt failed) or V3S.Verify($[\![\mathbf{s}_k]\!]_j, \pi_k, \pi_{k\to j}$) = false then
          valid$_i$ = valid$_i \setminus \{k\}$
    assert{ V3S.Reconstruct $\left(\left(\sum_{j\in \mathsf{valid}_i \cap \mathsf{corrupt}} [\![\mathbf{s}_j]\!]_i\right)_{i\in\mathsf{honest}}\right) \neq \perp$ }
    state$_i$.session.salt := $H_{\mathsf{salt}}((\mathsf{salt}_j)_{j\in\mathsf{valid}_i})$
    $a := H_a$(state$_i$.session.salt)
    $[\![b]\!]_i := \beta - [1\ a] \cdot \sum_{j\in\mathsf{valid}_i\cap\mathsf{corrupt}} [\![\mathbf{s}_j]\!]_i$
    if $\mathbf{s}_{\mathsf{honest}} = \varnothing$ then
      $\mathbf{s}_{\mathsf{honest}} \leftarrow \mathcal{D}_{\mathbb{Z}^{2n}, \sqrt{|\mathsf{honest}|}\sigma'_{\mathsf{sk}}} + Y$
                    ▷ Where $Y$ is a random variable independent of the Gaussian
    $[\![b]\!]_i = [\![b]\!]_i - [1\ a] \cdot \left(\mathbf{s}_{\mathsf{honest}} - \sum_{j\in\mathsf{honest}} \sum_{k\in S} \lambda_k^i [\![\mathbf{s}_j]\!]_k\right) \cdot (\lambda_i^i)^{-1}$
    Store $[\![\mathbf{s}]\!]_i$ in state$_i$
    return contrib$_3[i]$ := (state$_i$.session.salt, $[\![b]\!]_i$)
```

Fig. 36: The seventh hybrid of the security proof of the unforgeability of Pelican with DKG. Difference with the previous hybrid are highlighted.