THED: Threshold Dilithium from FHE

Jai Hyun Park
CryptoLab Inc.
Lyon, France

Alain Passelègue
CryptoLab Inc.
Lyon, France

Damien Stehlé
CryptoLab Inc.
Lyon, France

Abstract

We describe THED, a threshold version of the Dilithium signature scheme (ML-DSA), whose issued signatures are valid for the genuine Dilithium verification algorithm. The signing protocol has two rounds of communication, one of which that lends itself to preprocessing. The scheme supports arbitrary number of users and threshold parameter.

The construction consists in running Dilithium's signing algorithm under Threshold Fully Homomorphic Encryption (ThFHE), except for the computation of the signing challenge that happens in clear. Due to the type of operations performed, we rely on the CKKS scheme for homomorphic computations. However, a number of challenges remain, for which we develop new tools. In particular, we describe a CKKS-BFV continuum that helps for modular operations in the context of other non-arithmetic operations, a hybrid-format homomorphic comparison when the input is the sum of a bit-decomposed integer and a small integer, and a modulus-thrifty homomorphic comparison of larger non-bit-decomposed integers. Furthermore, to ensure the protocol is communication efficient, we developed a new threshold decryption method for CKKS providing more compact decryption shares.

Our proof-of-concept implementation of the FHE components of the signing protocol runs in under 1.3s on an RTX-5090 GPU, with 23.6KB of communication per party for the NIST level-2 Dilithium variant. Most of the computation can be run in an offline phase without the message to be signed; the online cost then shrinks to under 0.25s. Apart from the two decryption steps, this computation is entirely public and can be delegated to a server with more powerful hardware.

Keywords

Threshold Signature, FHE, CKKS, Dilithium/ML-DSA

1 Introduction

Following its selection by NIST as the primary algorithm for post-quantum signing [AAC+22], Dilithium/ML-DSA [DKL+18, BDK+21] is being rapidly deployed in various environments. As for all signature schemes, the signing key is particularly sensitive and requires dedicated protection. Threshold signatures provide a classical approach to strengthen the availability and security of signature issuance, by distributing the signing key among multiple parties [DF89]: if too few parties participate in the execution of the protocol, then they must be unable to produce a signature that passes verification; if sufficiently many parties collaborate, then they should be able to produce valid signatures. For this reason, threshold versions of Dilithium are actively being looked after (see, e.g., [BP25] and the references below). Given that Dilithium is standardized, a particularly desirable property of a threshold Dilithium protocol would be its verification interchangeability with non-threshold Dilithium: a signature produced by executing the threshold Dilithium protocol must be accepted as valid by Dilithium's verification algorithm.

Dilithium is an optimized version of Lyubashevsky's lattice-based signature scheme [Lyu09, Lyu12]. The latter follows a Fiat-Shamir blueprint like Schnorr's signature and consists of a linear step, a challenge-building step based on a hash function evaluation, another linear step and a rejection test. Many threshold signature schemes have been built by modifying Lyubashevsky's signature, but most of them do not handle Dilithium's optimizations and are not verification interchangeable [CS19, TPCZ23, GKS24, dPKM+24, KRT24, EKT24, CATZ24, BKL+25] (see also [DOTT21, Che23, ADP24] for the restricted n-out-of-n case where all users must collaborate to produce valid signatures, and [VSW21, LSV22, SVL24] for the $n = 2$ subcase). The main difficulties in obtaining a threshold signature scheme that is interchangeable with Dilithium are the signature and verification key compression techniques introduced in [BG14] and [DKL+18].

A few interchangeable schemes have been proposed very recently. Using secure multi-party computation (MPC), Dufka et al. [DKLS25] obtained a solution for $t = n = 2$. Still using MPC, Bienstock et al. [BdCE+26] obtained a solution for arbitrary t and n, but with tens of rounds and MBs of communication per user. A more direct adaptation of ML-DSA was proposed in [BCdP+25], with only three rounds of communication, but it is limited to $t, n \leq 6$.

From a theoretical point of view, it was shown in [BGG+18] that any (deterministic) signature scheme can be turned into a 1-round threshold version thereof. This universal thresholdizer relies on threshold fully homomorphic encryption. Threshold-FHE is a variant of fully homomorphic encryption (FHE) introduced in [AJL+12, BGG+18] where the ability to decrypt is distributed across parties: a 1-round protocol run by the decryptors allows them to open the ciphertext. The threshold signature construction from [BGG+18] provides all users with an FHE encryption of the signing key so that, given a message to be signed, all users can locally compute an FHE encryption of a common signature of the message. To produce a cleartext signature, the parties run the threshold-FHE decryption protocol. This generic construction produces signatures that are interchangeable with the original signature scheme, and could hence be applied to Dilithium. Such an approach has been considered in [ASY22] for a basic version of Lyubashevsky's signature scheme, focusing on removing its use of rejection sampling. Using the construction from [BGG+18] directly on Dilithium is challenging as its signing algorithm involves different types of operations (modular arithmetic, integer arithmetic, bit arithmetic) and FHE can prove cumbersome for general-purpose computations. Further, Dilithium signing involves a while loop, which is cumbersome for the FHE circuit-based model of computation.

Contributions. We describe THED, the first concrete FHE-based threshold signature scheme that is verification interchangeable with Dilithium. We argue that its concrete security (in the selective and passive model) is identical to Dilithium's. The construction relies on the universal thresholdizer from [BGG+18], which we modify and instantiate to achieve practical efficiency in 2 rounds of communication.

For the efficient homomorphic evaluation of Dilithium signing, we exploit both CKKS and BFV schemes by introducing the concept of the CKKS-BFV continuum (Section 4.1). Specifically, we utilize the BFV scheme for the efficient arithmetic over the Dilithium ring $\mathbb{Z}_{8380417}[X]/(X^{256}+1)$ (Section 4.3), and we leverage the CKKS scheme for the fast arithmetic with various precisions (Section 4.4). On the CKKS side, we further introduce multiple methods for high-precision homomorphic comparison: a general one from extending [CKK20] and [CCK+24] (Section 4.2) and another tailored specifically to the Dilithium ring using bit-decomposition [KN24] and binary gates.

Furthermore, we introduce an improved threshold decryption method for CKKS (Section 4.5). Our method is specifically designed for ciphertexts storing ℓ real numbers (or $\ell/2$ complex numbers) within their slots, where ℓ is significantly smaller than the ring degree N. It allows each party to share only ℓ integers for the joint decryption, rather than N integers. Notably, this method is compatible with slot-encoding without requiring a computationally expensive encoding conversion algorithms, such as SlotToCoeff.

Using the implementation of the CKKS FHE scheme [CKKS17] available in the HEaaN library [Cry22], we wrote a proof-of-concept implementation of THED's FHE components for the NIST level-2 instantiation of Dilithium. For a probability of successfully generating a signature that is equal to $p = 0.870$, this costs 1.240s on a commodity GPU (NVIDIA RTX 5090). The communication per party is 23.6KB, assuming up to 128 decryptors in the synchronous setting of [MBH23]. In fact, the first round computation can be preprocessed by making it independent from the message; without taking preprocessing into account, the runtime is decreased to 0.198s. We observe that the computation is public (except for threshold decryption) and can be externalized to a server. If the protocol fails to produce a valid signature, then it can be restarted. Alternatively, the protocol can be executed several times in parallel to produce a signature with probability arbitrarily close to 1.

Comparison. We now provide a comparison to the other existing approaches for threshold Dilithium [DKLS25, BdCE+26, BCdP+25]. Our FHE-based solution has the following advantages.

- It works for every t and n and scales well with them. The main impact is the decryption modulus (appearing at both rounds): it increases pre-decryption computation and communication complexity. It has been significantly reduced in [BFM+25], and, if we assume that the decryptors are aware of who they are, this dependency can be almost entirely removed [MBH23]. Most of the computation is independent of t and n and so is the number of rounds.
- It requires only 2 rounds: this is to be compared to 3 rounds for [BCdP+25] and more than 20 rounds for the MPC-based approaches from [DKLS25, BdCE+26]. Note that these can benefit from trade-offs between communication amount and the number of rounds, but communication is already in the MBs without minimizing the number of rounds. In terms of communication volume, our solution involves tens of KB per party, which is similar to [BCdP+25] when $t \leq 3$ and significantly less otherwise.
- Almost all the computations can be performed publicly: only the threshold-FHE decryption keys and decryption steps are secret. The computations can hence be delegated to powerful devices. It also implies that a much smaller amount of software needs to be protected.

The main drawback of our approach is the high computational cost. However, as already observed, most of the computation is public and can be delegated to powerful machines. The main alternative for arbitrary t and n is the scheme from [BdCE+26], but the high communication and round complexity quickly translates into high runtimes in environments where participants are distant.

Security Model. We focus on passive security for the sake of simplicity, but note that active security can be achieved by adding zero-knowledge proofs of honest threshold-FHE partial decryptions, consistent with public commitments of the partial decryption key shares. This provides robustness with identifiable aborts without any honest majority assumption. The statements to be proved are linear with smallness constraints, which is typical of lattice-based cryptography, and we refer to [BS23, LNP22, NS24, HLSS25] (among others) for potential instantiations.

Independently, it may seem that we rely on a trusted dealer for key generation. Note that we can use the threshold-FHE distributed key generation from [MHPS25] which enables fast CKKS computations, and a homomorphic evaluation of Dilithium's key generation algorithm. This provides an efficient distributed key generation protocol without relying on a trusted dealer.

1.1 Technical Overview

The main challenge in applying [BGG+18] to thresholdize a signature scheme is to find efficient ways to homomorphically implement the various steps of THED's signing algorithm. From that perspective, Dilithium comes with a number of challenges. The step numbers below refer to the Sign algorithm in Figure 5, which recalls the Dilithium signature scheme.

- Dilithium relies on SHAKE for expanding part of the verification key (Step 1), hashing the message (Steps 2 and 4), generating the signing mask (Step 6) and generating the signing challenge (Step 9);
- It involves rejection sampling: if any of the four tests (Steps 13 and 17) is invalid, then the signing process has to be restarted;
- All of these rejection tests rely on comparisons, sometimes on integers for which we have extracted the most and least significant bits (with the HighBits, LowBits and MakeHint functions, at Steps 8, 12 and 16);
- The main data type is integers, but some operations involve modular arithmetic (Step 7), others are over $\mathbb{Z}$ (Step 11), and yet others involve comparisons (see above).

In the following, we explain how we tackle these difficulties, but we first discuss the choice of FHE scheme. As the amount of data to be homomorphically processed is significant, FHE schemes that

provide Single-Instruction-Multiple-Data (SIMD) operations, such as BGV [BGV12], BFV [Bra12, FV12], and CKKS [CKKS17], seem more promising for thresholdizing Dilithium than latency-oriented schemes [DM15, CGGI16]. Among these throughput-oriented schemes, we choose CKKS as it enables relatively efficient homomorphic comparisons [CKK20] based on polynomial approximation over the reals, as opposed to BFV/BGV for which comparison is performed using interpolation over a finite field [IZ21]. In CKKS, the most time-consuming operation is bootstrapping [CHK+18a]: it aims at increasing the ciphertext modulus while preserving the underlying plaintext. This is needed every now and then during a homomorphic computation as multiplications decrease the ciphertext modulus. To obtain an efficient CKKS evaluation, one must strive to minimize the number of bootstraps. Our design is motivated by this objective.

Homomorphically evaluating SHAKE has been little explored in the literature. The state-of-the-art approach [HCLK25] relies on [DM15, CGGI16]. It consumes more than 1 hour in single-thread CPU to evaluate one SHA3 round function. Note that we have several kilobytes of data to hash to obtain the signing challenge and the cost of hashing grows linearly with the amount of data to be hashed. We avoid this high cost in several ways. First, we note that some calls to SHAKE can be done publicly, as they do not involve any secret quantity: the expansion of the uniform component of the verification key (Step 1) and the hashing of the message (Steps 2 and 4). Second, we observe that SHAKE is used as a pseudo-random function (PRF) in the generation of the signing mask (Step 6). As we consider a deterministic signature scheme, the PRF key is part of the signing key and is provided in encrypted form. We replace SHAKE by a PRF that is more FHE-friendly, namely, the Learning-With-Rounding Darkmatter PRF from [BIP+18]. To make it more efficient, we replace the matrix-vector product by a polynomial ring multiplication, where the ring is chosen to be identical to that of CKKS. This leads to signatures that differ from Dilithium's, but does not prevent us from achieving verification interchangeability. Finally, we generate the signing challenge in the clear (Step 9), by decrypting its input with the threshold-FHE distributed decryption algorithm. This is the reason why our scheme has two communication rounds, instead of a single one as would be obtained by directly applying [BGG+18].

The use of rejection sampling implies that the signing algorithm is cumbersome to describe as a circuit. If the probability to pass the tests is p, then the distribution of the number of signing attempts follows a geometric law of parameter p. This is difficult to handle with the circuit-based model of homomorphic computations. This difficulty was circumvented in [ASY22, GKS24, dPKM+24, KRT24, EKT24, CATZ24, BKL+25] by removing rejection sampling altogether: this is supported by a security proof based on the Rényi divergence rather than the statistical distance, but leads to increased parameters. As this requires parameter changes and handles only one of the four Dilithium rejection tests, this approach seems incompatible with verification interchangeability. We instead take a more direct approach: we perform a few signing attempts in parallel, using the high dimension of the CKKS plaintext vectors. On the one hand, we might have several successful iterations: as we only want to output a single signature, we only keep the first successful signature and (homomorphically) discard the other ones. On the other hand, repeating the protocol boosts the success probability, but it remains non-negligibly away from 1 as increasing the number of parallel attempts too much would harm the performance.

For homomorphic comparisons, we rely on [CKK20]. The multiplicative depth it consumes grows fast with the bit-sizes of the integers that have to be compared. Recall that depth consumption is crucial for CKKS: if the multiplicative depth is high, then one needs to bootstrap to re-increase the ciphertext modulus, and bootstrapping is costly compared to the basic homomorphic operations. In our context, the integers to be compared can have up to 20 bits. Our calculations show that such a comparison could require up to 4 bootstraps for typical parameters from [Cry22] (with ring degree $N = 2^{16}$). We simplify some of these tests as follows (see Figure 5 for the description of Dilithium):

- the tests of Step 17 occur on data that is known in the clear to the signers; this can be seen by rewriting $\mathbf{w} - c\mathbf{s}_2$ as $\mathbf{Az} - c\mathbf{t}$ and noting that $\mathbf{t}$ can be given to the signers without impacting security;
- for the remaining test (first half of Step 13), we observe that the encrypted input is the sum of two integers, one of which is known in binary decomposition, thanks to how the signing mask is generated, and the other one is relatively small; for this case, we use a hybrid comparison based on a tree structure for the most significant bits and on [CKK20] for the least significant parts.

It remains to consider the comparison of Step 13. For this one, we use a modulus-thrifty extension of [CKK20], gradually decreasing the precision during the computation: indeed, when the underlying plaintexts become close to 0 or 1, less precision is required.

Homomorphically extracting most/least significant bits for integers of such a bit-size is similarly challenging. For this purpose, we use the iterative bootstrapping technique from [KN24] to transform an integer into a radix decomposition. Extracting the most/least significant bits is then achieved using modular reductions for two different moduli: the first one is easier, as the binary decomposition of Dilithium's modulus is sparse; and the second one is implemented with thrifty comparison.

We now explain how we perform modular multiplications in the context of CKKS ciphertexts. These are required for the PRF evaluation (replacing Step 6, as discussed above) and the construction of the input to the hash function producing the signing challenge (Step 7). For this purpose, we observe that CKKS can be smoothly transformed into BFV and back. We aim at using properties of both, because CKKS is convenient for comparisons and BFV is preferable for modular arithmetic. Assume we have a CKKS ciphertext $(a, b) \in (\mathcal{R}_Q^{\mathsf{FHE}})^2$ with $\mathcal{R}_Q^{\mathsf{FHE}} = \mathbb{Z}_Q[X]/(X^N + 1)$ with N a power of 2 and Q a large integer. Assume further that it decrypts to a vector of bits $\mathbf{m} \in \{0, 1\}^{N/2}$. Concretely, we have:

$$a \cdot s + b = \mathrm{DFT}^{-1}(\Delta \cdot \mathbf{m} + \mathbf{e}) \bmod Q \ ,$$

where s is the secret key, DFT refers to the discrete Fourier transform from $\mathcal{R}^{\mathsf{FHE}} = \mathbb{Z}[X]/(X^N + 1)$ to $\mathbb{C}^{N/2}$ and $\|\mathbf{e}\|_\infty < \Delta/4$. As observed in [DMPS24], by applying the 'cleaning' map $h_1 : x \mapsto 3x^2 - 2x^3$ from [CKK20], we can essentially square Δ while keeping the same message $\mathbf{m}$ and a similar $\|\mathbf{e}\|_\infty$, at the expense of decreasing Q by a factor $\mathrm{poly}(\Delta)$. By repeatedly applying h_1, we quickly increase the gap between $\mathbf{e}$ and $\mathbf{m}$. Then, homomorphically

evaluating the DFT gives a pair $(a', b') \in (\mathcal{R}_{Q'}^{\mathsf{FHE}})^2$ such that

$$a' \cdot s + b' = \Delta' \cdot \mathbf{m} + \mathbf{e}' \bmod Q' \ ,$$

for some $\Delta' \gg \Delta$ and $\|\mathbf{e}'\|_\infty \ll \Delta'$. This is a BFV ciphertext for $\mathbf{m}$ modulo $\lfloor Q'/\Delta' \rceil$. The effectiveness of this approach is quantified by the decrease of Q relative to the growth of Δ. It was recently shown in [CKSS25] that one can almost obtain $Q/Q' \approx \Delta'/\Delta$ (i.e., the modulus decreases almost at the same speed as the scaling factor increases).

In contrast, for the Darkmatter PRF evaluation (replacing Step 6), it is convenient to start in a BFV format and end up in a CKKS format for the subsequent computations. The PRF consists in evaluating

$$x \mapsto \left\lfloor \frac{1}{p} \left(\alpha \cdot x \bmod pq\right) \right\rceil \ ,$$

where p and q are very small integers and $\alpha \in \mathcal{R}_{pq}^{\mathsf{FHE}}$ is the encrypted PRF key (we do not have to take the same ring as the one used for the FHE scheme, but it simplifies the description and the computation). The multiplication modulo pq can be performed in BFV format, and then the rounding can be applied while bootstrapping to a CKKS ciphertext, as described in [BKSS24]. This provides the signing masks in q-ary digits, encrypted in CKKS format. This base-decomposed expression of the signing masks is itself convenient to obtain an efficient homomorphic instantiation of the last rejection test that we mentioned above (first half of Step 13).

The overall computation consumes exactly 10 CKKS bootstraps before the first communication round, and 3 more afterwards. We observe that the computations up to the first communication round can be made independent of the message and considered as an offline computation.

Security Aspects. Creating the signing challenge in the clear decreases the burden on the FHE evaluation of the signing algorithm, but the two-round nature of the threshold signature protocol raises more security questions than if applying the universal thresholdizer [BGG+18] directly. Notably, as several signing attempts are run in parallel (exploiting the SIMD nature of CKKS) when making signing queries, the adversary can see challenges of failed signing attempts, as well as challenges of successful signing attempts that are subsequently discarded as a single signature is issued.

For the security proof, we start from the assumption that deterministic Dilithium is secure when using the Darkmatter PRF for the signing mask (Step 6 of Figure 5) and rely on the recent security analysis from [BdCE+26] (inspired from [DFPS23, PN25]). It proves that Dilithium is secure even if the adversary has oracle access to aborted transcripts, i.e., if it can see commitments of Step 8 for rejected signatures, and both unselected and selected accepted signatures. To achieve this, the authors modified Dilithium by adding a small error term $\mathbf{e}$ at Step 7. This tweak does not affect interchangeability and has no noticeable impact on correctness. Our security proof then follows the blueprint of [ASY22]. Oracle queries to Round 1 are simulated from aborted transcripts, and oracle queries to Round 2 are simulated using a full signature query (including the full Round-1 transcript). As in [ASY22], we rely on the Rényi divergence to minimize the amount of noise in threshold-FHE decryption.

Implementation Aspects. As mentioned earlier, we rely on the HEaaN2 library [Cry22] that implements the CKKS scheme. We use a modified version that incorporates the grafting technique from [CCK+24], since we flexibly use small plaintext precisions to store bits as well as very large plaintext precisions to prepare for noise flooding combined with high bit-size modular arithmetic. To increase the precision of binary plaintexts, we rely on the technique introduced in [CKSS25] that saves modulus consumption. For flooding, we need an upper bound on the magnitude of the noise of the ciphertexts to be decrypted. Finally, to avoid the attack from [CCP+24] based on bootstrapping failures, we use the sparse secret encapsulation from [BTH22] (note that sparse secret encapsulation is supported by the protocol from [MHPS25]).

Other Related Works. The CKKS-BFV continuum that we introduced is related to a series of recent works. Using a scaling factor close to the modulus in CKKS was considered in [BGGJ20, BCKS24, BKSS24] for bootstrapping exact data. It was observed in [KN24, AKP25] that it may be used to perform modular reduction. This was then exploited in [BK25] to perform modular multiplications, by multiplying over $\mathbb{Z}$ using the CKKS encoding of vectors in $\mathbb{C}^{N/2}$, converting to polynomials with a homomorphic DFT and then performing modular reduction. This differs from our approach that performs modular arithmetic by viewing a high scaling factor CKKS ciphertext as a BFV ciphertext.

The CKKS to BFV conversion may seem to contradict [EGM23], which argues that a conversion from CKKS to BGV leads to a fast bootstrapping for BGV. Here we focus on BFV, but a BFV ciphertext can be converted to a BGV ciphertext (and vice-versa) using a multiplication by a scalar. In fact, there is no contradiction: our conversion assumes that the message $\mathbf{m}$ is binary, whereas the conversion-based BGV bootstrapping from [EGM23] requires handling messages $\mathbf{m}$ whose entries are integers larger than the BGV plaintext modulus. We could convert such an integer to bits and apply our method, but the cost of bit extraction is not negligible compared to those of CKKS and BGV bootstrappings. Note that bit extraction has actually been considered in [CKSS25] as an alternative path for CKKS bootstrapping.

The CKKS to BFV conversion, if coupled with the small integers functional bootstrapping from [BKSS24], provides a CKKS-based bootstrapping for BFV ciphertexts for small plaintext moduli. It differs from the CKKS-based BFV bootstrapping from [KSS24] as the latter works for any plaintext modulus but relies on high-precision CKKS bootstrapping [BCC+22, CKSS25].

2 Preliminaries

Due to space constraints, we focus on the main definitions in this section. Additional definitions and preliminaries can be found in Appendix A, in particular definition of Fully Homomorphic Encryption and how to thresholdize it in the context of threshold signatures (using Rényi divergence for noise flooding rather than the statistical distance).

We let $\mathsf{HW}(\mathbf{h})$ denote the Hamming weight of a binary vector $\mathbf{h}$. For $\eta > 0$, we define the 1-dimensional Gaussian function $\rho_\eta : \mathbb{R} \to (0, 1]$ as:

$$\rho_\eta(x) := \frac{1}{\eta} \exp\left(\frac{-\pi |x|^2}{\eta^2}\right) \ .$$

We say that a random variable X over $\mathbb{R}$ follows the Gaussian distribution of standard deviation η and center $c \in \mathbb{R}$, denoted $\mathcal{D}_{\eta,c}$, if its density function is $\rho_X : x \mapsto \rho_\eta(x - c)$. We extend it over polynomials of degree N by letting $\mathcal{D}_{N,\eta,c}$ denote the distribution of degree-N polynomials whose i-th coefficient is sampled from $\mathcal{D}_{\eta,c_i}$, where c_i is the i-th coefficient of c.

2.1 Threshold Signatures

We recall the definition of threshold signatures. Intuitively, a t-out-of-n threshold signature allows one to distribute a signing key among n parties such that any subset of t parties can collaboratively sign any message, while any smaller subset of parties cannot issue valid signatures. We focus on threshold signatures with two rounds of communication in the semi-honest model.

Definition 1. *A 2-round threshold signature scheme is a tuple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{PartSign}_1, \mathsf{Combine}_1, \mathsf{PartSign}_2, \mathsf{Combine}_2, \mathsf{Verify})$ *with the following specifications:*

- $\mathsf{KeyGen}(1^\lambda, 1^n, 1^t)$ *takes a security parameter λ, a number of parties n and a threshold t and outputs public parameters* pp*, a verification key* vk*, and partial signing keys* $(\mathsf{sk}_i)_{i\in[n]}$*;*
- $\mathsf{PartSign}_1(\mathsf{pp}, \mathsf{sk}_i, M)$ *takes the public parameters* pp*, a partial signing key* sk_i *and a message M and outputs a Round-1 contribution* $\mathsf{out}_{1,i}$*;*
- $\mathsf{Combine}_1(\mathsf{pp}, (\mathsf{out}_{1,i_j})_{j\in[t]})$ *is deterministic, takes the public parameters* pp *and t Round-1 contributions* $(\mathsf{out}_{1,i_j})_{j\in[t]}$ *and outputs a Round-2 input message* out_1*;*
- $\mathsf{PartSign}_2(\mathsf{pp}, \mathsf{sk}_i, \mathsf{out}_1, M)$ *takes the public parameters* pp*, a partial signing key* sk_i*, a Round-2 input message* out_1 *and a message M and outputs a Round-2 contribution* $\mathsf{out}_{2,i}$*;*
- $\mathsf{Combine}_2(\mathsf{pp}, (\mathsf{out}_{2,i_j})_{j\in[t]})$ *is deterministic, takes the public parameters* pp *and t Round-2 contributions* $(\mathsf{out}_{2,i_j})_{j\in[t]}$ *and outputs a signature σ;*
- $\mathsf{Verify}(\mathsf{vk}, M, \sigma)$ *takes a verification key* vk*, a message M, and a signature σ and outputs a bit $b \in \{0,1\}$.*

Correctness. *For $\varepsilon > 0$, the scheme is ε-correct if for any M and any sets $\mathcal{S}_1, \mathcal{S}_2 \subseteq [n]$ with $|\mathcal{S}_1| = |\mathcal{S}_2| = t$, we have:*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, M, \sigma) \;\middle|\; \begin{array}{c} (\mathsf{pp}, \mathsf{vk}, (\mathsf{sk}_i)_{i\in[n]}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, 1^t) \\ \mathsf{out}_{1,i} \leftarrow \mathsf{PartSign}_1(\mathsf{pp}, \mathsf{sk}_i, M), \forall i \in \mathcal{S}_1 \\ \mathsf{out}_1 := \mathsf{Combine}_1(\mathsf{pp}, (\mathsf{out}_{1,i})_{i\in\mathcal{S}_1}) \\ \mathsf{out}_{2,i} \leftarrow \mathsf{PartSign}_2(\mathsf{pp}, \mathsf{sk}_i, \mathsf{out}_1, M), \forall i \in \mathcal{S}_2 \\ \sigma := \mathsf{Combine}_2(\mathsf{pp}, (\mathsf{out}_{2,i})_{i\in\mathcal{S}_2}, M) \end{array}\right]$$

is larger or equal to $1 - \varepsilon$, where the probability is taken over the random coins of KeyGen, $\mathsf{PartSign}_1$ *and* $\mathsf{PartSign}_2$.

Compactness. *The scheme achieves compactness if there exists a polynomial p such that, for any M and any sets $\mathcal{S}_1, \mathcal{S}_2 \subseteq [n]$ with $|\mathcal{S}_1|, |\mathcal{S}_2| \geq t$, using the same notation as for correctness, we have that the bit-sizes of* vk *and σ are below $p(\lambda)$.*

Threshold Unforgeability. *The scheme satisfies threshold unforgeability if for any PPT adversary $\mathcal{A}$, the advantage* $\mathsf{Adv}^{\mathsf{thuf}}(\mathcal{A}) := \Pr(\mathsf{Exp}^{\mathsf{thuf}}_{\mathcal{A}}(\lambda, n, t) = 1)$ *is negligible, where experiment* $\mathsf{Exp}^{\mathsf{thuf}}_{\mathcal{A}}$ *is defined in Figure 1.*

We focus on the semi-honest model for our construction. Combining it with non-interactive zero-knowledge proofs (of honest threshold-FHE partial decryption) allows us to achieve malicious security.

$\underline{\mathsf{Exp}^{\mathsf{thuf}}_{\mathcal{A}}(\lambda, n, t)\text{:}}$

$\mathcal{L} \leftarrow \emptyset$; $\mathcal{W} \leftarrow \emptyset$
$\mathcal{S}_{\mathsf{cor}} \leftarrow \mathcal{A}(1^\lambda)$ with $\mathcal{S}_{\mathsf{cor}} \subseteq [n]$ and $|\mathcal{S}_{\mathsf{cor}}| = t - 1$
$(\mathsf{pp}, \mathsf{vk}, (\mathsf{sk}_i)_{i\in[n]}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, 1^t)$
$(\sigma^*, M^*) \leftarrow \mathcal{A}^{\mathsf{OPartSign}_1(\cdot), \mathsf{OPartSign}_2(\cdot)}(\mathsf{pp}, \mathsf{vk}, (\mathsf{sk}_i)_{i\in\mathcal{S}_{\mathsf{cor}}})$
return $\mathsf{Verify}(\mathsf{vk}, M^*, \sigma^*) = 1 \wedge (M^*, \sigma^*) \notin \mathcal{L}$

$\underline{\mathsf{OPartSign}_1(i, M)\text{:}}$

if $i \in \mathcal{S}_{\mathsf{cor}}$ **then return** $\perp$
$\mathsf{out}_{1,i} \leftarrow \mathsf{PartSign}_1(\mathsf{pp}, \mathsf{sk}_i, M)$
for $j \in \mathcal{S}_{\mathsf{cor}}$: $\mathsf{out}_{1,j} \leftarrow \mathsf{PartSign}_1(\mathsf{pp}, \mathsf{sk}_j, M)$
$\mathsf{out}_1 \leftarrow \mathsf{Combine}_1(\mathsf{pp}, (\mathsf{out}_{1,j})_{j\in\mathcal{S}_{\mathsf{cor}}\cup\{i\}})$
$\mathcal{W} \leftarrow \mathcal{W} \cup \{(M, \mathsf{out}_1)\}$
return $\mathsf{out}_{1,i}$

$\underline{\mathsf{OPartSign}_2(i, \mathsf{out}_1, M)\text{:}}$

if $i \in \mathcal{S}_{\mathsf{cor}} \vee (M, \mathsf{out}_1) \notin \mathcal{W}$ **then return** $\perp$
$\mathsf{out}_{2,i} \leftarrow \mathsf{PartSign}_2(\mathsf{pp}, \mathsf{sk}_i, \mathsf{out}_1, M)$
for $j \in \mathcal{S}_{\mathsf{cor}}$: $\mathsf{out}_{2,j} \leftarrow \mathsf{PartSign}_2(\mathsf{pp}, \mathsf{sk}_j, M)$
$\sigma \leftarrow \mathsf{Combine}_2(\mathsf{pp}, (\mathsf{out}_{2,j})_{j\in\mathcal{S}_{\mathsf{cor}}\cup\{i\}})$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{(M, \sigma)\}$
return $\mathsf{out}_{2,i}$

Figure 1: Security experiment $\mathsf{Exp}^{\mathsf{thuf}}_{\mathcal{A}}$.

2.2 CKKS

CKKS [CKKS17] is an FHE scheme based on ring-LWE [SSTX09, LPR10] whose underlying cleartexts are vectors in $\mathbb{R}^{N/2}$, where N is the degree of the ring $\mathcal{R}^{\mathsf{FHE}}$. A ciphertext for a cleartext $\mathbf{m}$ is a pair $(a, b) \in (\mathcal{R}^{\mathsf{FHE}}_Q)^2$ such that

$$a \cdot s + b = \Delta \cdot \mathrm{DFT}^{-1}(\mathbf{m}) + \mathbf{e} \bmod Q \ , \tag{1}$$

where $s \in \mathcal{R}^{\mathsf{FHE}}$ is the secret key, $\mathbf{e}$ has small coefficients, Δ is a scaling factor that drives numerical precision and DFT is the map that sends $p \in \mathcal{R}^{\mathsf{FHE}}$ to $(p(\zeta_i))_{i\in[N/2]}$. Here, the ζ_i's are $N/2$ roots of $X^N + 1$ without any pair of equal or conjugate elements. CKKS natively enables homomorphic component-wise addition, component-wise multiplication, and cyclic rotation of the coordinates by arbitrary amounts.

Multiplication is performed by tensoring the two input ciphertexts, obtaining an element in $(\mathcal{R}^{\mathsf{FHE}}_Q)^3$, then relinearizing to get back to $(\mathcal{R}^{\mathsf{FHE}}_Q)^2$. These two steps lead to a growth of the scaling factor from Δ to Δ^2. The scaling factor is mapped back to Δ by the rescaling step which consists in dividing the ciphertext by Δ and rounding. This decreases the ciphertext modulus by a factor $\approx \Delta$. In contrast, addition and rotation preserve the ciphertext modulus Q.

After several sequential multiplications, the ciphertext modulus may be so small that no further multiplication can be performed. The purpose of CKKS bootstrapping [CHK+18a] (BTS) is to regain modulus while preserving the underlying cleartext, so that computations can be continued. For efficiency, it is important to minimize

the number of bootstraps, as this is significantly more costly than the basic operations.

The default encoding of the cleartext in the ciphertext using DFT^{-1} as in Equation (1) is called slots encoding. An alternative encoding, called coefficients encoding, is as in Equation (1) but without DFT^{-1}. In that case, the cleartext $\mathbf{m}$ belongs to $\mathbb{R}^N$. This encoding is less convenient for homomorphic component-wise multiplication, but allows homomorphic convolution. Going from slots (resp. coefficients) encoding to coefficients (resp. slots) encoding can be achieved by homomorphically evaluating DFT (resp. DFT^{-1}), an operation called S2C (resp. C2S).

In this work, we will mostly encode exact data (in slots and in coeffs). When $\mathbf{m}$ is binary and slots-encoded, then we may perform arbitrary binary gates homomorphically. As the noise $\mathbf{e}$ grows relatively to Δ with binary gate evaluations, the entries in $\mathbf{m}$ become less determined. As shown in [DMPS24], accuracy of the bits can be regained by homomorphically applying the map $h_1 : x \mapsto 3x^2 - 2x^3$. Note that the h_1 map can also be used for pre-flooding cleaning. Furthermore, as shown in [BCKS24], bootstrapping can be significantly improved for binary cleartexts (we refer to this as Bits BTS). Based on [BGGJ20], this bootstrapping was extended to small integers and combined with an arbitrary component-wise look-up table in [BKSS24] (we refer to this as Small integers BTS).

3 The THED Threshold Signature Scheme

In this section, we describe how we combine Dilithium with threshold-FHE in order to obtain an efficient 2-round threshold signature scheme. We first provide a high-level description of our scheme, which is based on homomorphically evaluating Dilithium following the generic approach from [BGG+18], except we add a round of communication to avoid homomorphically evaluating the hash function. Then we detail our construction, in particular explaining the tweaks we make to Dilithium to improve the efficiency of the homomorphic computation (without hurting interchangeability). We also analyze it by adapting the analysis from [ASY22] to our 2-round setting. Most details regarding how we efficiently perform the homomorphic computations are postponed to a dedicated section (Section 4).

3.1 Description

We follow the blueprint from [BGG+18], which thresholdizes a signature scheme by running it homomorphically using the encrypted signing key, and having each signer release a partial decryption of the encrypted signature. The underlying signature must be deterministic (each signer needs to evaluate the same signature/ciphertext), hence our choice of thresholdizing the deterministic variant of Dilithium.

We face two difficulties: (i) the challenge is computed by evaluating a hash function, SHAKE-256, which is particularly difficult to evaluate homomorphically; (ii) when running the signing algorithm homomorphically, one cannot know when the **while** loop should terminate. We circumvent these difficulties as follows: (i) we add one round of interaction, which allows the signers to decrypt the commitment $\mathbf{w}_1$ so that the hash function evaluation can be performed in the clear. Specifically, signers decrypt $\mathbf{w}_1$ after Step 8 of Sign and run Steps 9, 10 in the clear (see Figure 5). (ii) We set a bound L on the number of loop iterations to be run and return the first valid signature computed during all these iterations; if more than one successful iteration occurs, only the first valid signature is revealed.

In addition, and similarly to (i), homomorphically sampling the randomness $\mathbf{y}$ (Step 6) is also challenging, as it requires running SHAKE-256 homomorphically. To circumvent this issue, we adjust the randomness generation by relying on a homomorphic-friendly PRF based on [BIP+18]. As this modification only affects the internal randomness distribution, it preserves interchangeability, and does not affect security, assuming that pseudorandomness of the PRF holds. We provide more details about the design of the PRF in Section 3.2.

Consequently, our (threshold) signing algorithm has the following structure:

(1) Homomorphically sample L commitments $(\mathbf{w}_1^{(i)})_{i\in[L]}$;
(2) Compute in the clear the L associated challenges $(c^{(i)})_{i\in[L]}$;
(3) Homomorphically compute the L corresponding responses $(\mathbf{z}^{(i)})_{i\in[L]}$.

During the homomorphic evaluation, the signers decrypt the outcome of (1) to run (2) in the clear. Then, (3) is performed homomorphically and only its result is decrypted. Furthermore, our homomorphic evaluation only keeps the first valid $\mathbf{z}_i$'s, such that only one valid signature is revealed. Our two rounds of communication thus correspond to the two times the signers communicate their partial decryption shares, at the ends of (1) and (3).

We note that the start of the signing protocol up to the first round of communication (excluded) can be performed offline as the dependency on the message can be removed. This may be preprocessed under the assumption that the signers agree on the input to feed to the PRF. By default, for simplicity, we use the message, but it could be any unique identifier that the signers agree on. That identifier can be the message number, the time or other contextual data. This allows us to modify our construction to a threshold signature with preprocessing. In this setting, as commitments are not bound to the signed message, we emphasize that the threshold decryption of the commitments should only be revealed during the online phase, once the message is known. Decrypting the commitments earlier could enable ROS attacks [BLL+22], even in the semi-honest setting. We further remark that the homomorphic computations (1) and (3) are deterministic and only involve public information, so they can be outsourced to a server; only partial decryptions have to be performed by signers.

3.2 Construction

We now detail our 2-round threshold signature protocol. Its precise description is provided in Figure 2. It is mostly the instantiation of the above high-level idea, up to several tweaks which help improve efficiency. We explain these tweaks and their impact at the end of this section. In addition, we also rely on the modification introduced in [BdCE+26] that adds a small noise to the commitment $\mathbf{w}$, and allows one to prove security of Dilithium given revealed incomplete signatures.

Key Generation. The key generation algorithm samples FHE keys and distributes the decryption key to the n parties. It also samples

signature keys and encrypts the signing key (which includes the PRF key used to deterministically derive randomness) using the FHE scheme. The verification key is exactly that of the underlying signature.

Distributed Sign. To sign a message M, the signers homomorphically evaluate the first part of the signature scheme (Figure 5, Steps 1–8) using the encrypted signing key, and partially decrypt the resulting ciphertext encrypting $(\mathbf{w}_1^{(i)})_{i\in[L]}$. Assuming that t distinct signers participate, their partial decryption shares can be recombined to recover $(\mathbf{w}_1^{(i)})_{i\in[L]}$, and the challenges $(c^{(i)})_{i\in[L]}$ can be computed in the clear (Figure 5, Steps 9 and 10). Finally, at least t users homomorphically evaluate the second part of the signature scheme (Figure 5, Steps 11–19) and partially decrypt the resulting ciphertext. Again, assuming that t distinct signers participate, their partial decryption shares can be recombined: this time, the result is a signature. More precisely, we consider the following signing protocol with 2 rounds of communication.

Round 1. For a message M to be signed, the signers homomorphically evaluate Steps 1–8 of Figure 5, L times in parallel, using the encrypted signing key, and partially decrypt the resulting ciphertext. They broadcast their decryption shares.

Round 2. After receiving the decryption shares, each signer runs $\mathsf{Combine}_1$, which consists in evaluating FinDec on t distinct partial decryptions to recover $(\mathbf{w}_1^{(i)})_{i\in[L]}$. The latter information allows signers to compute (in the clear and locally) the challenges $\tilde{c}^{(i)} := \mathsf{H}(M \parallel \mathbf{w}_1^{(i)})$, therefore avoiding evaluating the hash function homomorphically. The signers then homomorphically run Steps 11–19 of Figure 5 to compute a ciphertext encrypting $(\mathbf{z}^{(i)})_{i\in[L]}$, where only the first valid signature is kept, and partially decrypt it. They broadcast their decryption shares.

Finalize. After receiving the Round-2 decryption shares, the signers recombine t distinct partial shares with FinDec to recover the unique $\mathbf{z}^{(i)} \neq \perp$ (assuming signing is successful), and $(\mathbf{z}^{(i)}, \mathbf{h}^{(i)}, \tilde{c}^{(i)})$ is returned by $\mathsf{Combine}_2$ as the signature of M.

The detailed protocol is described in Figure 2. As can be observed, we slightly modify the steps of the original Dilithium signing algorithm (Figure 5). This is motivated by performance gain and by improved security analysis: we modify the sampling of $\mathbf{y}$ (Figure 5, Step 6) and add noise in the computation of $\mathbf{w}$ (Figure 5, Step 7), we modify the rejection sampling based on $\mathbf{r}_0$ (Figure 5, Steps 12–14), and we postpone the computation of $\mathsf{MakeHint}$ (Figure 5, Step 16). Let us now explain these changes in more detail.

Sampling $\mathbf{y}$*'s and* $\mathbf{e}$*'s.* Using $\mathsf{ExpandMask}$ as a pseudo-random function (PRF) for generating $\mathbf{y}$ (Figure 5, Step 6) is not efficient when computed in the encrypted state. We replace this PRF with a ring variant of the Darkmatter PRF [BIP+18]. For a given PRF key k_{PRF} and $m(X) \in \mathcal{R}_{64}^{\mathsf{FHE}} = \mathbb{Z}_{64}[X]/(X^N+1)$, it outputs

$$\left\lfloor \frac{1}{8}\left(m(X)\cdot k_{\mathsf{PRF}}(X) \bmod 64\right)\right\rfloor \ , \tag{2}$$

which consists of $3N$ random bits, from which we extract $[\![\mathbf{y}^{(k)}]\!]$ and $[\![\mathbf{e}^{(k)}]\!]$, where $\mathbf{e}$ denotes the error added by the modification of Dilithium from [BdCE+26]. For $N = 2^{16}$, this instantiation of the learning-with-rounding Darkmatter PRF achieves far more than 128 bits of security against the best known attacks (this may be checked using the lattice estimator [APS15]). These changes are reflected in Steps 4 and 5 of $\mathsf{PartSign}_1$ in Figure 2. Note that we actually sample an error $\mathbf{e}$ with 3-bit coefficients, which is slightly larger than that from [BdCE+26]. This cannot damage security but could hurt correctness; our experiments do not reveal any visible impact on correctness.

Rejecting $\mathbf{z}$*'s and* $\mathbf{r}_0$*'s.* The original rejection sampling for $\mathbf{z}$ (Figure 5, Steps 12–14) requires a high-precision comparison, which is inefficient when computed in the encrypted state. We do not change this step. We are able to perform this comparison by leveraging a small-precision encrypted comparison and using the binary decomposition of $\mathbf{y}$. For the comparison involving $\mathbf{r}_0$ (Figure 5, Step 13), we instead compute $[\![\mathbf{r}_0']\!] := \mathsf{LowBits}_q([\![\mathbf{w}]\!])$. We emphasize that this does not have any impact, as we have:

$$\begin{aligned} &\|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta \\ &\qquad\qquad \Longleftrightarrow \\ &\qquad\qquad \|\mathsf{LowBits}_q(\mathbf{w}, 2\gamma_2) - c\mathbf{s}_2\|_\infty \geq \gamma_2 - \beta \ . \end{aligned}$$

The second test is actually the test run in the publicly available NIST submission implementation of Dilithium, while the first test is the test described in the specification: both tests have identical responses. These changes are reflected in Step 6 of $\mathsf{PartSign}_2$ in Figure 2. The computation of $[\![\mathbf{r}_0']\!]$ is performed directly after the computation of $[\![\mathbf{w}]\!]$ in Step 7 of $\mathsf{PartSign}_1$.

Rejecting Hints. The rejection sampling concerning $\mathsf{MakeHint}$ (Figure 5, Step 17) could also generate a computation overhead during homomorphic signing. As observed in [BdCE+26], the rejection test based on $\mathbf{h}$ very rarely fails in Dilithium (it rejects $\approx 2\%$ of the signatures that would pass without it). Moreover, this test does not play any role in security, since $\mathbf{h}$ is only there to allow compression of the verification key. We postpone this test to first generate a pre-signature $\mathbf{z}$ that passes all other tests, decrypt $\mathbf{z}$ and then see whether it passes this test (in the clear). Note that the hint vector $\mathbf{h}$ is computable from $\mathbf{z}, \mathbf{w}, \mathbf{t}$ and c via the relation:

$$\mathbf{w} - c\mathbf{s}_2 = \mathbf{A}\mathbf{z} - c\mathbf{t} \ . \tag{3}$$

As a result, the signers do not even have to compute $\mathbf{h}$ homomorphically: they can compute it in the clear after decryption of $\mathbf{z}$ and check if it passes the test or not. In 98% of cases, the hint vector $\mathbf{h}$ passes the test and the signers can return $(\mathbf{z}, \mathbf{h}, \tilde{c})$. In the remaining 2% of cases, they have to restart the protocol. As the signing protocol is deterministic, signing a message M is attempted by first signing $M\| \star \|0$ for a special symbol $\star$. Then, despite determinism, one can attempt to sign M again in case of failure using input $M\| \star \|1$, etc.

Distributed Key Generation. Above and in Figure 2, we describe our key generation algorithm for the threshold signature assuming a trusted dealer. Note however that one can remove such an assumption as soon as we have a distributed key generation (DKG) protocol for the threshold-FHE scheme. Equipped with the latter, one can run the DKG protocol to distribute $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and have each party encrypt randomness to homomorphically generate a seed, which then serves to homomorphically run $\mathsf{Dilithium.KeyGen}$ in order to produce $[\![\mathbf{s}_1]\!]$, $[\![\mathbf{s}_2]\!]$, $[\![k_{\mathsf{PRF}}]\!]$, and vk. A 4-round DKG protocol for CKKS was recently proposed in [MHPS25] (with more

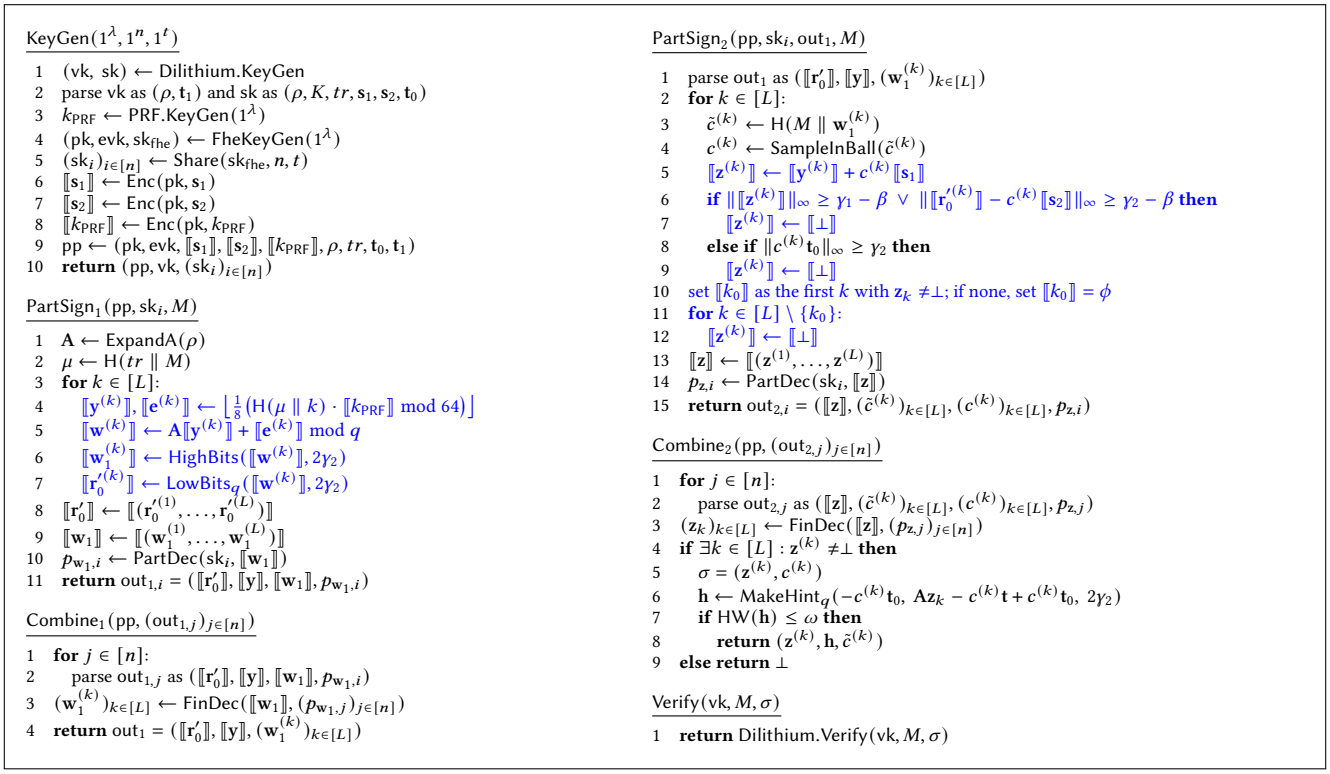


Figure 2: The THED Threshold Dilithium signature protocol. The steps in blue involve homomorphic computations.

costly variants with fewer rounds). By default, recovering vk would require an extra round, but it can be performed during the DKG execution of [MHPS25] without adding a communication round.

3.3 Analysis

Correctness is inherited from the correctness of Dilithium and of the underlying FHE scheme. Compactness holds as the verification key and the signatures returned by our scheme are precisely the same as those of Dilithium, therefore also guaranteeing verification interchangeability. Further, we prove the following security claim.

Theorem 2. *Let q_{sig} denote a bound on the number of signing queries made by an attacker, N denote the ring degree, and B_{eval} denote a bound on the homomorphic computation error of the ciphertexts that are decrypted. Then, assuming the underlying FHE scheme is perfectly correct and* IND-CPA*-secure, assuming the threshold flooding parameter η satisfies $\eta \geq B_{\mathsf{eval}} \cdot \sqrt{10 \cdot N \cdot q_{\mathsf{sig}}}$, and assuming security of Dilithium with the Darkmatter PRF instantiation of Equation* (2)*, the threshold signature scheme described in Figure 2 satisfies threshold unforgeability.*

The proof is adapted from [ASY22] and we detail it in Appendix B. It relies on the security of Dilithium with revealed incomplete signatures shown in [BdCE+26] and of the underlying FHE scheme.

4 Homomorphic Computations

We now explain how to perform the homomorphic computations in the algorithm of Figure 2. We first provide ingredients that may have wider applicability and then explain every step.

4.1 A CKKS-BFV Continuum

Implementing Steps 4 and 5 in PartSign$_1$ (see Figure 2) using the native homomorphic CKKS operations seems costly, because of the modular multiplications, modulo 64 at Step 4 and modulo q at Step 5. For homomorphic modular multiplications, one more often relies on BFV or BGV. However, having a CKKS encryption of the output $[\![\mathbf{y}]\!]$ from Step 4 is preferable to perform the first comparison of Step 6 in PartSign$_2$.

Going from a BFV ciphertext format to a CKKS ciphertext format for small bit-length integers can be achieved using the bootstrapping-based format-switching algorithm from [BGGJ20]. As shown in [BKSS24], this can be combined with a function evaluation and bit extraction, which allows us to have the bits of $\mathbf{y}$ in the CKKS slots of a high-level ciphertext.

We now explain how to transform a CKKS ciphertext for bits into a BFV ciphertext. Let $(a, b) \in (\mathcal{R}_Q^{\mathsf{FHE}})^2$ with

$$a \cdot s + b = \mathrm{DFT}^{-1}(\Delta \cdot \mathbf{m} + \mathbf{e}) \bmod Q \ , \tag{4}$$

where s is the secret key and $\gamma := \|\mathbf{e}\|_\infty / \Delta < 1/4$. As we consider a high-level CKKS ciphertext, the modulus Q is large and the scaling factor Δ is very small compared to Q (concretely, the scaling

factor Δ would typically be of the order of 20-30 bits, compared to several hundreds of bits for Q). Our goal is to transform (a, b) into a pair $(a', b') \in (\mathcal{R}_{Q'}^{\mathsf{FHE}})^2$ with

$$a' \cdot s + b' = \Delta' \cdot \mathbf{m} + \mathbf{e}' \bmod Q' \ , \tag{5}$$

where $\Delta' = \lfloor Q'/t \rceil$ with t being the BFV plaintext modulus and $\|\mathbf{e}'\|$ much smaller than Δ'. The smaller the ratio $\gamma' = \|\mathbf{e}'\|_\infty/\Delta'$ the better, as a small ratio gives more room for performing several BFV computations.

Towards mapping Equation (4) to Equation (5), let us first homomorphically evaluate the map $h_1 : x \mapsto 3x^2 - 2x^3$ to the input ciphertext, used in [DMPS24] to increase the precision of a binary plaintext. More precisely, we have $h_1(b + \varepsilon) = b + O(\varepsilon^2)$ for all $b \in \{0, 1\}$ and all ε such that $|\varepsilon|$ is sufficiently small. Homomorphically apply h_1 on (a, b) without rescaling (i.e., we perform homomorphic multiplication by tensoring and relinearizing only), we obtain $(a_1, b_1) \in (\mathcal{R}_Q^{\mathsf{FHE}})^2$ such that

$$a_1 \cdot s + b_1 = \mathrm{DFT}^{-1}(\Delta^4 \cdot \mathbf{m} + \mathbf{e}_1) \bmod Q \ ,$$

where $\gamma_1 = \|\mathbf{e}_1\|/\Delta^4 \approx \gamma^2$. Note that Δ is raised to the fourth power as evaluating h_1 has multiplicative depth equal to 2. We can now rescale by $\Delta^3\gamma$, to obtain $(a_1', b_1') \in (\mathcal{R}_{Q_1}^{\mathsf{FHE}})^2$ with $Q_1 \approx Q/(\Delta^3\gamma)$ and

$$a_1' \cdot s + b_1' = \mathrm{DFT}^{-1}(\Delta_1 \cdot \mathbf{m} + \mathbf{e}_1') \bmod Q_1 \ ,$$

where $\Delta_1 = \Delta/\gamma$ and $\|\mathbf{e}_1'\|/\Delta_1 \approx \gamma_1$.

When iterating k times, we see that γ_k grows as γ^{2^k} and Δ_k grows as $\approx \Delta/\gamma_k$, while Q_k decreases as $\approx Q\gamma_k^2/\Delta^{3k}$. We note that a refined approach was recently proposed in [MHPS25], with a more thrifty modulus consumption (i.e., a larger Q_k). Overall, the ciphertext modulus decreases while the scaling factor increases, and the error term stays similar. When Δ_k gets close to Q_k, we perform a homomorphic DFT. If tuned correctly, the result has a scaling factor $\Delta' = \lfloor Q'/t \rceil$, where Q' is the current ciphertext modulus and t is the BFV plaintext modulus. The current ciphertext (a', b') hence satisfies Equation (5), where the norm of $\mathbf{e}'$ is very small compared to Δ'.

This transformation allows us to take the vector $[\![\mathbf{y}]\!]$ from Step 5 with coordinates stored in bits in CKKS ciphertexts and transform it into bits of BFV ciphertexts. We can then multiply the entries by powers of 2 to obtain $[\![\mathbf{y}]\!]$ with coordinates stored as integers modulo Dilithium's q. Step 6 then consists of a BFV multiplication between a plaintext matrix and a ciphertext vector.

4.2 Thrifty Comparison

For CKKS-based comparison, the state-of-the-art approach is the iterative method introduced in [CKK20]. It evaluates small degree polynomials iteratively, to gradually increase the precision of the comparison. The small degree polynomials can be selected using the multiple-interval Remez algorithm [LLKN22].

Here, we optimize the modulus consumption of this technique by flexibly changing the scaling factors. This is made convenient thanks to the grafting technique from [CCK+24]. The encrypted input of our comparison is an integer of ≈ 20.5 bits while the output is just a single bit. Note that the role of each small degree polynomial is to gradually decrease the most significant bits. This allows us to slightly decrease the scaling factor across the evaluations of the small-degree polynomials, which overall decreases modulus consumption. For example, during homomorphic comparison for the reduction modulo $2\gamma_2$, we start from a 40.4-bit scaling factor and gradually decrease it to 23.5 bits.

4.3 Computing over the Dilithium Ring

The underlying ring $\mathcal{R}^{\mathsf{FHE}}$ for CKKS and BFV differs from the ring $\mathcal{R}^{\mathsf{SIG}}$ for Dilithium. Concretely, we use $\mathcal{R}^{\mathsf{FHE}} = \mathbb{Z}[X]/(X^N + 1)$ with $N = 2^{16}$ while $\mathcal{R}^{\mathsf{SIG}} = \mathbb{Z}[Y]/(Y^{N'} + 1)$ with $N' = 2^8$. Note that $\mathcal{R}^{\mathsf{SIG}}$ is a subring of $\mathcal{R}^{\mathsf{FHE}}$, in which Y corresponds to X^{256}. In particular, the ring $\mathcal{R}^{\mathsf{FHE}}$ is an $\mathcal{R}^{\mathsf{SIG}}$-module of rank $2^{16}/2^8 = 2^8$.

For the homomorphic computations over $\mathcal{R}^{\mathsf{SIG}}$, we use coefficient-encoded BFV-CKKS. Suppose we are given two elements in $\mathcal{R}^{\mathsf{FHE}}$: $f(X) = \sum_{0\le i<256} m_i(Y) \cdot X^i$ and $f'(X) = \sum_{0\le i<256} m_i'(Y) \cdot X^i$ where m_i and m_i' are elements in $\mathcal{R}^{\mathsf{SIG}}$. Then, their product is

$$f \cdot f' = \sum_{0\le i<256} \left(\sum_{0\le j<256} m_j \cdot m'_{i-j} \right) \cdot X^i \ ,$$

where $m_{-i} = -m_i$ for $1 < i < 256$. As $\mathcal{R}^{\mathsf{FHE}}$ is an $\mathcal{R}^{\mathsf{SIG}}$-module, the product outputs 256 Dilithium ring elements $\{\sum_{0\le j<256} m_j \cdot m'_{i-j}\}_{0\le i<256}$ in parallel. This observation allows us to homomorphically implement Dilithium arithmetic. For example, when the Dilithium dimensions are set to $(k, \ell) = (4, 4)$ and the number of parallel repetitions is set to $L = 8$, for $\mathbf{s}_* = (\mathbf{s}_{*,j})_{0\le j<4} \in (\mathcal{R}^{\mathsf{SIG}})^4$ and $\{c_t\}_{0\le t<8} \in (\mathcal{R}^{\mathsf{SIG}})^8$, the following product over $\mathcal{R}^{\mathsf{FHE}}$ computes $\{c \cdot \mathbf{s}_*\}_{0\le t<8}$:

$$\Big(\sum_{0\le j<4} \mathbf{s}_{*,j}(Y) \cdot X^{8\cdot j} \Big) \cdot \Big(\sum_{0\le t<8} c_t(Y) \cdot X^{32\cdot t} \Big) \ .$$

This allows to compute $c \cdot \mathbf{s}_1$ and $c \cdot \mathbf{s}_2$ during homomorphic signing. The same may be achieved for homomorphically computing $\mathbf{A} \cdot \mathbf{y} \bmod q$. For a given $\mathbf{y}(X) \in \mathcal{R}^{\mathsf{FHE}}$ that stores 3 (random) bits in each of its coefficients and a Dilithium verification key $\mathbf{A}$, one can compute $L = 8$ vectors $\mathbf{w}$ in parallel by using $k = 4$ multiplications over $\mathcal{R}^{\mathsf{FHE}}$, outputting four module-LWE format ciphertexts [BGV12, LS15] with a module of rank 32 of degree 2048. The four module-LWE format ciphertexts can be combined into one BFV-CKKS ciphertext by using format conversion [BCK+23].

4.4 Large Modular Arithmetic

The decomposition steps in Dilithium contain modular arithmetic with moduli q and $2\gamma_2$. In particular, it requires congruent elements precisely in $[0, q)$ and $(-\gamma_2, \gamma_2]$, respectively. We use bit representations to perform those large modular arithmetic operations. To be more precise, from a given input $x = \sum_i b_i 2^i$, we first (approximately) compute $x \bmod k$ as $\sum_i b_i(2^i \bmod k)$, and we adjust the value to be in $[0, k)$ using either a binary circuit or a thrifty comparison.

For the bit decomposition, we adopt the iterative MSB extraction technique from [KN24]. Precisely, we consider 5 iterations of 6-bit Small integer bootstrapping to obtain 28 ciphertexts that decrypt to each bit $\{b_i\}_{0\le i<28}$, i.e., $x = \sum_{0\le i<28} b_i \cdot 2^i$. In our protocol, the inputs for modular arithmetic are $\mathbf{A} \cdot \mathbf{y} + q\mathbf{I}$ where $\mathbf{I}$ is a integral vector. Each element of $\mathbf{I}$ belongs to $[-\frac{\mathtt{hwt}}{2}, \frac{\mathtt{hwt}}{2}]$ where $\mathtt{hwt}$ is the

Hamming weight of the (sparse) secret key (e.g., $\texttt{hwt} = 31$ when using sparse secret encapsulation [BTH22]).

For the reduction modulo q, note that $x_q = \sum_{0 \le i < 28} b_i \cdot (2^i \bmod q)$ is in $[0, 2q)$. Then, we obtain $x \bmod q$ in $[0, q)$ by comparing x_q with q and subtracting q if needed. Since q is $2^{23} - 2^{13} + 1$, one can compare x_q and q using a depth-7 binary circuit that takes the b_i's as inputs. On the other hand, for the reduction modulo $2\gamma_2$, we first compute $\sum_{0 \le i < 28} b_i \cdot (2^i \bmod 2\gamma_2)$ which belongs to $[-4\gamma_2, 4\gamma_2]$. Then, using the thrifty comparison in Section 4.2, we compare it with each multiple of $2\gamma_2$ to obtain the exact $x \bmod 2\gamma_2$ in $(-2\gamma_2, 2\gamma_2]$. The input precision is $\approx \log_2(16 \cdot \gamma_2)$ bits.

4.5 Compact Threshold Decryption

Threshold decryption works for both of the standard FHE encodings: coefficient-encoding and slot-encoding. Coefficient-encoding provides a higher level of granularity for the communication, as it allows sending only the relevant coefficients. On the other hand, slot-encoding requires all the coefficients to be sent regardless of the number of slots it stores. For this reason, coefficient-encoding could be considered preferable for threshold decryption, especially since a primary strength of ThFHE lies in its low communication cost.

However, there are also good reasons for using threshold decryption with slot-encoding. Indeed, almost all homomorphic operations are more compatible with slot-encoding: going to slots would require an encoding conversion (a.k.a., SlotToCoeff). Importantly, to rely on the *noise flooding technique* during threshold decryption, one typically increases the precision of plaintexts, for which the only known approach relies on slot-encoding. Without threshold decryption in slot-encoding, an encoding conversion algorithm (a.k.a., SlotToCoeff) with large precision seems needed, which is computationally expensive and also makes FHE parameter design significantly more difficult.

In this section, we propose a threshold decryption algorithm with small communication for slot-encoded FHE ciphertexts. For a ciphertext that decrypts to a complex (resp. real) vector of dimension $\ell/2$, the proposed PartDec algorithm outputs ℓ (resp. $\ell/2$) integers, instead of N integers with the naive approach.

We start with the following result.

Lemma 3. *Consider a plaintext $m(X) = m_0 + m_1 \cdot X + \cdots + m_{N-1} \cdot X^{N-1} \in \mathcal{R}$, which decodes to a complex vector μ of length $N/2$. Specifically, for each $j = 0, 1, \cdots, \frac{N}{2} - 1$,*

$$m(\zeta_j) = \mu_j \ .$$

Define the plaintext

$$m'(X) = m_0 + m_{\frac{N}{\ell}} \cdot X^{\frac{N}{\ell}} + \cdots + m_{\frac{N}{\ell} \cdot (\ell-1)} \cdot X^{\frac{N}{\ell} \cdot (\ell-1)} \ .$$

Then, for each $j = 0, 1, \cdots, \frac{N}{2} - 1$,

$$m'(\zeta_j) = \mu'_{j \bmod \frac{\ell}{2}} \ ,$$

where $\mu'_\theta = \frac{\ell}{N} \cdot \sum_{k=0}^{\frac{N}{\ell}-1} \mu_{\theta + k \cdot \frac{\ell}{2}}$ for each $0 \le \theta < \ell/2$.

The proof of Lemma 3 is detailed in Appendix C.

Improved Threshold Decryption. Assume that we are given a ciphertext that stores only $\ell/2$ complex numbers out of $N/2$ slots. Typical examples include a ciphertext with replication, which decrypts to a complex vector μ such that

$$\mu_j = \mu_{j + \frac{\ell}{2} \cdot k}$$

for each integer j and k. Another representative case is a zero-padded ciphertext, where the decrypted vector μ contains zero from the $(\ell/2)$-th slots to the end. Let $m(X)$ be the underlying plaintext that encodes μ. Conventionally, to jointly decrypt slot-encoded ciphertext, all decryptors send $a(X) \cdot s_i(X) + e_i(X)$, which consists of N integers.

Based on Lemma 3, we propose the following threshold decryption protocol. The main modifications are marked in blue. For a given ciphertext (b, a),

(1) **Modified partial decryption**: each party computes $m_i(X) = a \cdot s_i + e_i$ where e_i is a flooding noise. Then, it extracts the ℓ coefficients of $m_i(X)$ corresponding to the powers of $X^{N/\ell}$, and it sends their most significant bits to other parties.
(2) **Modified final decryption**: one aggregates the partial decryption results with the ℓ terms of b corresponding to powers of $X^{N/\ell}$.

After the modified joint decryption, each party obtains the *partial sums* of the original message as a result. Precisely, the parties first obtain ℓ terms of $m(X)$, corresponding to the powers of $X^{N/\ell}$. By Lemma 3, it reveals $\ell/2$ partial sums. We note that this is particularly useful when the ciphertext contains $\ell/2$ complex numbers out of $N/2$ slots.

Compared to the original joint decryption protocol, this modified joint decryption reduces the communication cost by a factor of N/ℓ. Note that this approach reduces the number of coefficients to be sent, and it is compatible with the existing compression approach that truncates the least significant bits of each coefficient (see [PS24]).

We can further halve the communication volume for ciphertexts that store real numbers rather than complex numbers. It is known that a plaintext $m(X)$ stores a real (instead of complex) vector if and only if $m(X) = m(X^{-1})$. By combining this with Lemma 3, a sparse message vector of $\ell/2$ real coordinates can be recovered from $\ell/2$ coefficients of $m(X)$. In more details, the parties first jointly recover $m_0, m_{\frac{N}{\ell}}, \cdots, m_{\frac{N}{\ell}(\frac{\ell}{2}-1)}$. Then, they locally compute $m_{\frac{N}{\ell} \cdot \frac{\ell}{2}}, \cdots, m_{\frac{N}{\ell} \cdot (\ell-1)}$ by using the relations

$$\left(X^{\frac{N}{\ell} \cdot t}\right)^{-1} = -X^{\frac{N}{\ell} \cdot (\ell - t)} \ \text{ and } \ m_{\frac{N}{\ell} \cdot t} = -m_{\frac{N}{\ell} \cdot (\ell - t)} \ .$$

This recovers all ℓ terms of $m(X)$ corresponding to the powers of $X^{N/\ell}$, which suffices by Lemma 3.

As a side note, we observe that the approach is applicable to general FHE contexts, not only to ThFHE. In particular, it can be used for reducing the communication cost of sending FHE ciphertexts after homomorphic computation. Assume that we are given an ordinary FHE ciphertext (b, a) that decrypts to a sparse complex vector μ of dimension $\ell/2$. Then, the b-part can be compressed by sending only a subset of coefficients corresponding to the powers of $X^{N/\ell}$. This reduces the communication cost by a factor of up to $2N/(N + \ell)$. If the underlying message is a sparse real vector instead of a complex vector, the cost of sending the b-part can be further halved.

4.6 Overall Protocol

We now describe the precise homomorphic computations performed by THED. The steps not listed below are done in clear.

Homomorphic Ring-Darkmatter (Figure 2, Step 4 of $\mathsf{PartSign}_1$). There are two main steps in the homomorphic evaluation of the Darkmatter PRF: (1) multiplying modulo 64, and (2) extracting the 3 most significant bits. For these, we respectively use the CKKS-BFV continuum (Subsection 4.1) and Small integer functional bootstrapping [BKSS24]. At the end of this step, one obtains 6 CKKS ciphertexts, each of which decrypts to a pseudo-random binary vector of dimension $N/2$. We view this bit-string as the bit representation of $L = 8$ different $(\mathbf{y} + (\gamma_1 - 1) \cdot \mathbf{1})$'s and $(\mathbf{e} + 4 \cdot \mathbf{1})$'s. Note that 6 ciphertexts output more random bits than are required for 8 signing attempts.

Computation of $\mathbf{w}$ (Figure 2, Step 5 of $\mathsf{PartSign}_1$). Computing $\mathbf{w} = \mathbf{A} \cdot \mathbf{y} + \mathbf{e} \bmod q$ consists of two steps: (1) cleaning the random bits $\mathbf{y}$ and $\mathbf{e}$ and (2) computing $\mathbf{A}\cdot\mathbf{y}+\mathbf{e} \bmod q$. The first step increases the precision of the bits to enable the exact binary recombination of $\mathbf{y}$ and $\mathbf{e}$ as an integer, exact multiplication by $\mathbf{A}$ and noise flooding. We achieve it by iteratively applying the h_1 map in the thrifty fashion described in [CKSS25]. The second step can be achieved using the approach introduced in Subsection 4.3.

Note that while the output ciphertext of this procedure stores $\mathbf{A} \cdot \mathbf{y} \bmod q$, it does not ensure that the coordinates are in $[0, q)$. It hence requires post-processing to compute HighBits and LowBits.

HighBits and LowBits (Figure 2, Steps 6 and 7 of $\mathsf{PartSign}_1$). For a given coordinate w of $\mathbf{w} = \mathbf{A} \cdot \mathbf{y}$, except when w is congruent to $-\gamma_2$ modulo q, the equations

$$\begin{cases} \mathsf{LowBits}(w) = [[w + \gamma_2]_q - 1]_{2\gamma_2} - \gamma_2 + 1 \\ \mathsf{HighBits}(w) = \frac{1}{2\gamma_2} \left([w + \gamma_2]_q - \mathsf{LowBits}(w) - \gamma_2\right) \end{cases}$$

hold where $[x]_k$ is the residue of modulo k in $[0, k)$. When w is congruent to $-\gamma_2$ modulo q, the LowBits should be $-\gamma_2$ while the above equation outputs γ_2. However, the LowBits of both $\pm\gamma_2$ are rejected since $|\pm\gamma_2 - c \cdot s_2| \geq \gamma_2 - \beta$. Thus, computing HighBits and LowBits using the above equations leads exactly to the same result as in Dilithium. The large modular arithmetic can be performed using the approach introduced in Subsection 4.4.

Rejection Based on $\|\mathbf{z}\|_\infty$ (Figure 2, Step 6 of $\mathsf{PartSign}_2$). We reject the signature if $\|\mathbf{z}\|_\infty = \|\mathbf{y} + c \cdot \mathbf{s}_1\|_\infty \geq \gamma_1 - \beta$. As we have each bit of $\mathbf{y} + \gamma_1$ after the homomorphic PRF evaluation (Step 5), we can split the comparison into two parts: the high-bits comparison and the low-bits comparison. In particular, as $\|c \cdot \mathbf{s}_1\|_\infty \leq \beta$, we have $\|\mathbf{y} + c \cdot \mathbf{s}_1\|_\infty \geq \gamma_1 - \beta$ if and only if

$$\left(\left\lfloor \frac{\mathbf{y} + \gamma_1 - 1}{256} \right\rfloor = 0 \text{ and } [\mathbf{y} + \gamma_1 - 1]_{256} + c \cdot \mathbf{s}_1 \leq \beta\right) \quad \text{or}$$

$$\left(\left\lfloor \frac{\mathbf{y} + \gamma_1 - 1}{256} \right\rfloor = 1023 \text{ and } [\mathbf{y} + \gamma_1 - 1]_{256} + c \cdot \mathbf{s}_1 \geq 255 - \beta\right),$$

where $[x]_k$ is the residue of x modulo k in $[0, k)$. The high-bits comparison can be performed with an AND-tree, and the low-bits comparison can be performed with the thrifty comparison described in Subsection 4.2. For the low-bits comparison, a precision of $\log_2(4 \cdot 256)$ bits suffices.

Rejection Based on $\|\mathbf{r}'_0\|_\infty$ (Figure 2, Step 6 of $\mathsf{PartSign}_2$). This can also be performed with the thrifty comparison algorithm. It requires a comparison with a precision of $\log_2(8\gamma_2)$ bits.

Disclosure of the Signature (Figure 2, Step 14 of $\mathsf{PartSign}_2$). Before disclosing the signature, we should (homomorphically) select the first non-rejected signature to avoid trivial attacks based on unused but computed signatures. For this purpose, we bootstrap the bits corresponding to the results of the tests using [BCKS24], we then generate the one-hot vector indicating the selected sample and we clean it by iterating the h_1 map. Thanks to this processing, only the first passing $\mathbf{z}$ is kept and all the others are set to $\mathbf{0}$. We do not reuse the encryption of $\mathbf{z}$ computed earlier for the rejection test as it was performed in a low precision that does not allow cleaning: it was computed in a low precision to minimize modulus consumption in the comparison. We hence recompute $c\mathbf{s}_1$ homomorphically with a higher scaling factor (as described in Subsection 4.3), so that after adding the cleaned bits of $\mathbf{y}$, the precision is sufficiently high to enable noise flooding. The result is then ready for distributed decryption.

Threshold Decryption. Following Section 4.5, after the conventional PartDec procedure, each party compresses its decryption share before sending it. Specifically: each party (1) adds the flooding noise, (2) extracts the relevant coefficients, (3) truncates the low bits of the coefficients, and (4) sends the compressed coefficients.

Figure 3 provides an overview of the whole process.

5 Implementation and Experimental Results

We now describe our proof-of-concept implementation of THED. Our experiments are performed using a modified version of the HEaaN2 library [Cry22], on an NVIDIA GeForce RTX 5090 GPU. We did not use RNS-CKKS [CHK+18b] but grafted-CKKS [CCK+24] as it provides increased flexibility for the scaling factors.

5.1 Security Aspects

For CKKS, we use ring degree $N = 2^{16}$ and maximum ciphertext modulus PQ satisfying $\log_2(PQ) = 1533$, with secret key of Hamming weight equal to 256. For bootstrapping, we used the sparse secret encapsulation technique from [BTH22] at modulus $\log_2(PQ) = 116$ with a secret key of Hamming weight 31, to make the bootstrapping failure probability due to the polynomial evaluation in the EvalMod step equal to 0. This makes the attack from [CCP+24] inapplicable. The parameters used achieve 128-bit security against known attacks [APS15].

For the Darkmatter PRF, we rely on the Ring Learning With Rounding assumption, with ring degree $N = 2^{16}$, large modulus $q = 64$ and small modulus $p = 8$. According to our security estimates (based on the lattice estimator [APS15] and the recent analysis from [BHBS26]), these parameters provide well above 128-bit security. We use such parameters as they are sufficient to generate all randomness required with a single Darkmatter evaluation, hence performance would not benefit much from reducing security.

Security also requires sufficient noise flooding. This aspect is discussed in Appendix D.1.

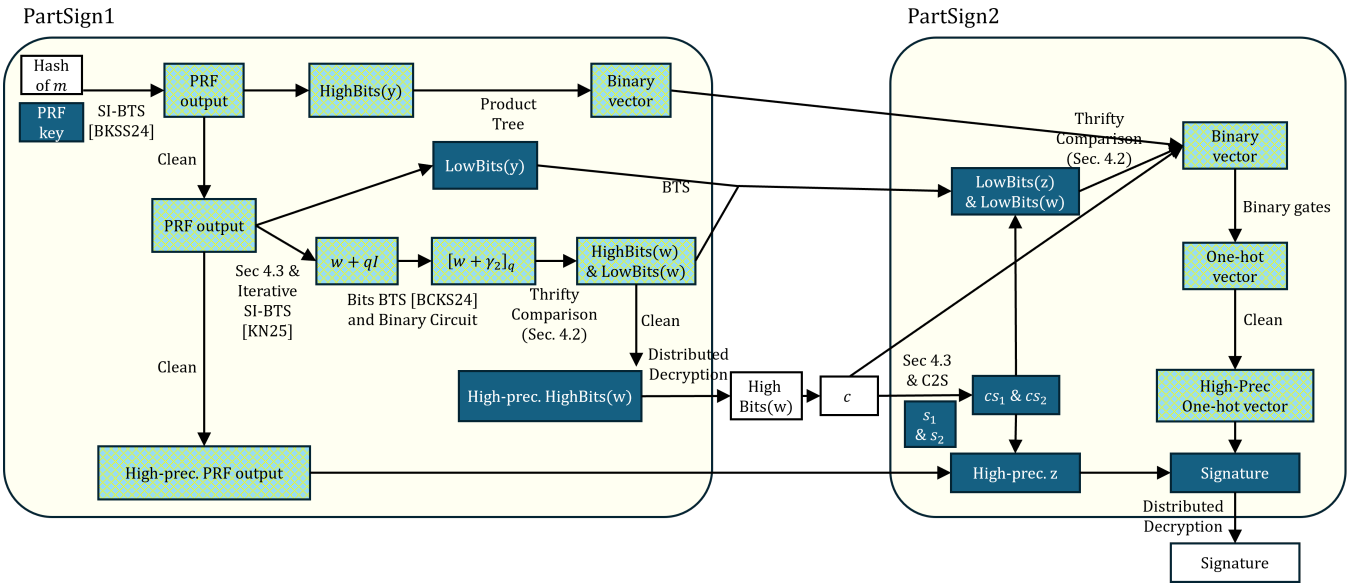


Figure 3: Overview of the signing procedure in threshold Dilithium. Encrypted integer values are in dark boxes, encrypted binary values are in green checkered boxes, and values in the clear are in white boxes.

5.2 Communication Cost

After each communication round, the parties should share their partial decryption shares of HighBits(w) (Round 1) and z (Round 2). As described in Section 4.5, each party sends the most significant bits of the partial decryption shares that store the data with a small gap for FinDec, truncating the least significant bits of their partial decryption shares. The remaining plaintext precision should be sufficient to ensure that the noise does not spill over to the plaintext when the shares are combined.

Considering these optimizations and the synchronous decryption protocol from [MBH23] whose FinDec is a sum, for 128 decryptors, the communication cost per party can be reduced to 19.5KB and 4.10KB for Rounds 1 and 2, respectively. For Round 1, we considered a 19-bit share modulus, and only 2^{13} coefficients need to be sent as each one of the 8 parallel signing attempts consumes 2^{10} coefficients. For Round 2, we considered a 32-bit share modulus, and only 2^{10} coefficients need to be sent as there is a single signing attempt left at this point. Among 100 signing attempts, we experimentally verified that the above parameters are compatible with 60 bits of noise flooding, enabling all parties to recover the signature (if signing succeeds).

5.3 Computation Cost

All signatures pass the reference Dilithium verification code. Among 1000 iterations, 870 iterations succeeded in signing. This matches the theoretical rejection probability of Dilithium. Note that each run of our protocol signs 8 signatures in parallel, and it outputs only the first signature if at least 1 out of the 8 parallel signing processes succeed. Table 1 describes the overview of experimental results. More detailed experimental results are reported in Appendix D.2.

We experimentally observed that in the output ciphertext, there is a gap of 98.77 bits and 79.37 bits between the message and the error in coefficients for each round. As desired, these are strictly larger than our conservative bounds, 90 bits and 75 bits, for the gap for the noise flooding, which we detailed in Appendix D.1.

	Average Latency	Worst Precision	Sign prob.	Comm. per party
$\mathsf{PartSign}_1$	1041.2 ms	98.8 bits	-	19.5 KB
$\mathsf{PartSign}_2$	198.41 ms	79.4 bits	0.870	4.10 KB

Table 1: Overview of the costs. Note that the timings exclude the clear computations such as MakeHint and distributed decryption. The latency is the average latency and the precision is the worst precision in coefficients among 1000 **experiments.**

Reducing the Online Cost. As reported in Table 1, the cost of $\mathsf{PartSign}_2$ is significantly lower than that of $\mathsf{PartSign}_1$. This can be exploited as follows to increase performance. $\mathsf{PartSign}_1$ can be performed independently from the message to be signed. Although it appears to take the message M as input in Figure 2, the seed μ can in fact be chosen independently of M: it only needs to be the same across signers and hence to be agreed upon (it can be set as the output of a random oracle on an input that the signers agreed upon). With this modification, all of $\mathsf{PartSign}_1$ (excluding the threshold decryption, which should occur only once the message M to be signed is revealed) can be executed in an offline phase, and the online phase is reduced to the threshold decryption of $\mathsf{PartSign}_1$ plus $\mathsf{PartSign}_2$.

Acknowledgment

We thank Sascha Roth for pointing out an additional security consideration regarding the preprocessing optimization in an earlier version of this work.

References

[AAC+22] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone. Status report on the third round of the NIST post-quantum cryptography standardization process. Technical report, NIST, 2022.

[ADP24] N. Alkeilani Alkadri, N. Döttling, and S. Pu. Practical lattice-based distributed signatures for a small number of signers. In *ACNS*, 2024.

[AJL+12] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*, 2012.

[AKP25] A. Alexandru, A. Kim, and Y. Polyakov. General functional bootstrapping using CKKS. In *CRYPTO*, 2025.

[APS15] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 2015. Software available at https://github.com/malb/lattice-estimator, git commit# c7def4.

[ASY22] S. Agrawal, D. Stehlé, and A. Yadav. Round-optimal lattice-based threshold signatures, revisited. In *ICALP*, 2022.

[BCC+22] Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim. META-BTS: Bootstrapping precision beyond the limit. In *CCS*, 2022.

[BCdP+25] G. Borin, S. Celi, R. del Pino, T. Espitau, G. Niot, and T. Prest. Threshold signatures reloaded ML-DSA and enhanced Raccoon with identifiable aborts, 2025. IACR eprint 2025/1166.

[BCK+23] Y. Bae, J. H. Cheon, J. Kim, J. H. Park, and D. Stehlé. HERMES: efficient ring packing using MLWE ciphertexts and application to transciphering. In *CRYPTO*, 2023.

[BCKS24] Y. Bae, J. H. Cheon, J. Kim, and D. Stehlé. Bootstrapping bits with CKKS. In *EUROCRYPT*, 2024.

[BdCE+26] A. Bienstock, L. de Castro, D. Escudero, A. Polychroniadou, and A. Takahashi. Quorus: Efficient, scalable threshold ML-DSA signatures: An MPC approach. In *USENIX Security*, 2026.

[BDK+21] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, Seiler G., and Stehlé D. CRYSTALS-Dilithium algorithm specifications and supporting documentation (version 3.1), 2021.

[BFM+25] Z. Brakerski, O. Friedman, A. Marmor, D. Mutzari, Y. Spiizer, and N. Trieu. Threshold FHE with efficient asynchronous decryption, 2025. IACR eprint 2025/712.

[BG14] S. Bai and S. D. Galbraith. An improved compression technique for signatures based on learning with errors. In *CT-RSA*, 2014.

[BGG+18] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, 2018.

[BGGJ20] C. Boura, N. Gama, M. Georgieva, and D. Jetchev. CHIMERA: combining ring-LWE-based fully homomorphic encryption schemes. *J. Math. Cryptol.*, 2020.

[BGV12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[BHBS26] J. Baudrin, R. Heim-Boissier, and F.-X. Standaert. On the concrete hardness of LWR with a power of two modulus, 2026. IACR eprint 2026/250.

[BIP+18] D. Boneh, Y. Ishai, A. Passelègue, A. Sahai, and D. J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In *TCC*, 2018.

[BK25] D. Boneh and J. Kim. Homomorphic encryption for large integers from nested residue number systems. In *CRYPTO*, 2025.

[BKL+25] C. Boschini, D. Kaviani, R. W. F. Lai, G. Malavolta, A. Takahashi, and M. Tibouchi. Ringtail: Practical two-round threshold signatures from learning with errors. In *S&P*, 2025.

[BKSS24] Y. Bae, J. Kim, E. Suvanto, and D. Stehlé. Bootstrapping small integers with CKKS. In *ASIACRYPT*, 2024.

[BLL+22] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. On the (in)security of ROS. In *J. Cryptol.*, 2022.

[BP25] L. T. A. N. Brandão and R. Peralta. NIST first call for multi-party threshold schemes (second public draft). Technical report, NIST, 2025.

[Bra12] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, 2012.

[BS23] W. Beullens and GG. Seiler. LaBRADOR: Compact proofs for R1CS from module-SIS. In *CRYPTO*, 2023.

[BTH22] J.-P. Bossuat, J. Troncoso-Pastoriza, and J.-P. Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In *ACNS*, 2022.

[CATZ24] R. Chairattana-Apirom, S. Tessaro, and C. Zhu. Partially non-interactive two-round lattice-based threshold signatures. In *ASIACRYPT*, 2024.

[CCK+24] J. H. Cheon, H. Choe, M. Kang, J. Kim, S. Kim, J. Mono, and T. Noh. Grafting: Decoupled scale factors and modulus in RNS-CKKS. In *CCS*, 2024.

[CCP+24] J. H. Cheon, H. Choe, A. Passelègue, D. Stehlé, and E. Suvanto. Attacks against the IND-CPAD security of exact FHE schemes. In *CCS*, 2024.

[CGGI16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT*, 2016.

[Che23] Y. Chen. DualMS: Efficient lattice-based two-round multi-signature with trapdoor-free simulation. In *CRYPTO*, 2023.

[CHK+18a] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT*, 2018.

[CHK+18b] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full RNS variant of approximate homomorphic encryption. In *SAC*, 2018.

[CKK20] J. H. Cheon, D. Kim, and D. Kim. Efficient homomorphic comparison methods with optimal complexity. In *ASIACRYPT*, 2020.

[CKKS17] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.

[CKSS25] H. Choe, J. Kim, E. Suvanto, and D. Stehlé. Leveraging discrete CKKS to bootstrap in high precision. In *CCS*, 2025.

[Cry22] CryptoLab. HEaaN library, 2022. https://heaan.it/.

[CS19] D. Cozzo and N. P. Smart. Sharing the LUOV: Threshold post-quantum signatures. In *IMACC*, 2019.

[DF89] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO*, 1989.

[DFPS23] J. Devevey, P. Fallahpour, A. Passelègue, and D. Stehlé. A detailed analysis of Fiat-Shamir with aborts. In *CRYPTO*, 2023.

[DKL+18] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018.

[DKLS25] A. Dufka, S. Kravtsenko, P. Laud, and N. Snetkov. Trilithium: Efficient and universally composable distributed ML-DSA signing. *IACR eprint 2025/675*, 2025.

[DM15] L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT*, 2015.

[DMPS24] N. Drucker, G. Moshkowich, T. Pelleg, and H. Shaul. BLEACH: cleaning errors in discrete computations over CKKS. *J. Cryptol.*, 2024.

[DOTT21] I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In *PKC*, 2021.

[dPKM+24] R. del Pino, S. Katsumata, M. Maller, F. Mouhartem, T. Prest, and M.-J. O. Saarinen. Threshold Raccoon: Practical threshold signatures from standard lattice assumptions. In *EUROCRYPT*, 2024.

[EGM23] K. Eldefrawy, N. Genise, and N. Manohar. On the hardness of scheme-switching between SIMD FHE schemes. In *PQCrypto*, 2023.

[EKT24] T. Espitau, S. Katsumata, and K. Takemure. Two-round threshold signature from algebraic one-more learning with errors. In *CRYPTO*, 2024.

[FV12] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR eprint 2012/144*, 2012.

[GKS24] K. D. Gur, J. Katz, and T. Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In *PQCrypto*, 2024.

[HCLK25] D. Hong, Y. Choi, Y. Lee, and Y.-S. Kim. Overlapped bootstrapping for FHEW/TFHE and its application to SHA3. In *FC*, 2025.

[HLSS25] I. Hwang, H. Lee, J. Seo, and Y. Song. Practical zero-knowledge PIOP for maliciously secure multiparty homomorphic encryption. In *CCS*, 2025.

[IZ21] I. Iliashenko and V. Zucca. Faster homomorphic comparison operations for BGV and BFV. *POPET*, 2021.

[KN24] J. Kim and T. Noh. Modular reduction in CKKS. *IACR Commun. Cryptol.*, 2024.

[KRT24] S. Katsumata, M. Reichle, and K. Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In *CRYPTO*, 2024.

[KSS24] J. Kim, J. Seo, and Y. Song. Simpler and faster BFV bootstrapping for arbitrary plaintext modulus from CKKS. In *CCS*, 2024.

[LLKN22] E. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No. Optimization of homomorphic comparison algorithm on RNS-CKKS scheme. *IEEE Access*, 2022.

[LNP22] V. Lyubashevsky, N. K. Nguyen, and M. Plançon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. In *CRYPTO*, 2022.

[LPR10] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.

[LS15] A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 2015.

[LSS14] A. Langlois, D. Stehlé, and R. Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In *EUROCRYPT*, 2014.

[LSV22] P. Laud, N. Snetkov, and J. Vakarjuk. DiLizium-2.0: Revisiting two-party Crystals-Dilithium. *IACR eprint 2022/644*, 2022.

[Lyu09] V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, 2009.

[Lyu12] V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, 2012.

[MBH23] C. Mouchet, E. Bertrand, and J.-P. Hubaux:. An efficient threshold access-structure for RLWE-based multiparty homomorphic encryption. *J. Cryptol.*, 2023.
[MHPS25] S. Min, G. Hanrot, A. Passelègue, and D. Stehlé. Distributed key generation for efficient threshold-CKKS, 2025. IACR eprint 2025/2057.
[NS24] N. K. Nguyen and G. Seiler. Greyhound: Fast polynomial commitments from lattices. In *CRYPTO*, 2024.
[PN25] R. Del Pino and G. Niot. Finally! A compact lattice-based threshold signature. In *PKC*, 2025.
[PS24] A. Passelègue and D. Stehlé. Low communication threshold fully homomorphic encryption. In *ASIACRYPT*, 2024.
[SSTX09] D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, 2009.
[SVL24] N. Snetkov, J. Vakarjuk, and P. Laud. TOPCOAT: towards practical two-party Crystals-Dilithium. *Discov. Comput.*, 2024.
[TPCZ23] G. Tang, B. Pang, L. Chen, and Z. Zhang. Efficient lattice-based threshold signatures with functional interchangeability. *IEEE Trans. Inf. Forensics Secur.*, 2023.
[VSW21] J. Vakarjuk, N. Snetkov, and J. Willemson. DiLizium: A two-party lattice-based signature scheme. *Entropy*, 2021.

A Additional Definitions

A.1 Rényi divergence

We recall the definition and properties of the Rényi divergence.

Definition 4. *Let $a \in (1, +\infty)$. Let P, Q be two probability distributions with P absolutely continuous with respect to Q. Their Rényi divergence of order a, assuming that it exists, is*

$$\mathsf{R}_a(P\|Q) := \left(\int_{\mathrm{Supp}(P)} \left(\frac{\mathrm{d}P}{\mathrm{d}Q}(x)\right)^{a-1} \mathrm{d}P(x)\right)^{\frac{1}{a-1}} .$$

Their Rényi divergence of infinite order is

$$\mathsf{R}_\infty(P\|Q) := \mathrm{ess\,sup}_{x\in\mathrm{Supp}(P)} \frac{\mathrm{d}P}{\mathrm{d}Q}(x) .$$

Lemma 5. *Let X and Y be two random variables with probability distributions P_X and P_Y such that $\mathrm{Supp}(P_X) \subseteq \mathrm{Supp}(P_Y)$. The following holds for any order $a \in (1, +\infty]$.*

- ***Log. Positivity:*** *$\mathsf{R}_a(P_X\|P_Y) \geq \mathsf{R}_a(P_X\|P_X) = 1$.*
- ***Data Processing Inequality:*** *$\mathsf{R}_a(P_{f(X)}\|P_{f(Y)}) \leq \mathsf{R}_a(P_X\|P_Y)$ for any function f, where $P_{f(Z)}$ denotes the distribution of $f(Z)$ for $Z = X$ or Y.*
- ***Multiplicativity:*** *Let $X = (X_1, X_2)$ and $Y = (Y_1, Y_2)$. Let P_{X_1} and P_{Y_1} denote the probability distributions of X_1 and Y_1. Let $P_{X_2|X_1=x}$ and $P_{Y_2|Y_1=x}$ denote the distributions of X_2 conditioned on $X_1 = x$ as well as Y_2 conditioned on $Y_1 = x$. If X_1 and X_2 are independent and Y_1 and Y_2 are independent, then we have*

  $$\mathsf{R}_a(P_X\|P_Y) = \mathsf{R}_a(P_{X_1}\|P_{Y_1})\mathsf{R}_a(P_{X_2}\|P_{Y_2}) .$$

  Otherwise, we have that $\mathsf{R}_a(P_X\|P_Y)$ is bounded from above by

  $$\mathsf{R}_\infty(P_{X_1}\|P_{Y_1}) \cdot \max_{x\in\mathrm{Supp}(P_{X_1})} \mathsf{R}_a(P_{X_2|X_1=x}\|P_{Y_2|Y_1=x}).$$

- ***Probability Preservation:*** *For any event $E \subseteq \mathrm{Supp}(P_Y)$, we have*

  $$P_Y(E) \geq \frac{P_X(E)^{\frac{a}{a-1}}}{\mathsf{R}_a(P_X\|P_Y)} .$$

Moreover, the Rényi divergence is non-decreasing and continuous as a function of $a \in [1, +\infty]$, as long as it is finite.

We finally recall the following result about the Rényi divergence of two shifted Gaussian distributions with the same standard deviation (adapted from [LSS14, Lemma 4.2]).

Lemma 6. *Let $\eta > 0$ and $c \in \mathbb{R}$. Then for any $a \in (1, \infty)$, we have:*

$$\mathsf{R}_a(\mathcal{D}_{\eta,c}\|\mathcal{D}_\eta) \leq \exp\left(\frac{a\pi c^2}{\eta^2}\right) ,$$

and then, for any $N > 0$, we have (by multiplicativity):

$$\mathsf{R}_a(\mathcal{D}_{N,\eta,e}\|\mathcal{D}_{N,\eta}) \leq \exp\left(\frac{Na\pi c^2}{\eta^2}\right) .$$

A.2 Fully Homomorphic Encryption

We recall the definition of fully homomorphic encryption (FHE).

Definition 7. *An FHE is a tuple of PPT algorithms* (KeyGen, Enc, Dec, Eval) *with the following specifications:*

- $\mathsf{KeyGen}(1^\lambda)$ *takes as input a security parameter and outputs a public key* pk*, an evaluation key* evk*, and a secret key* sk*;*
- $\mathsf{Enc}(\mathsf{pk}; M)$ *takes as inputs a public key* pk *and a plaintext M, and outputs a ciphertext* ct*;*
- $\mathsf{Eval}(\mathsf{evk}, C, (\mathsf{ct}_i)_{i\in[k]})$ *takes as inputs an evaluation key* evk*, a circuit C, and a tuple of ciphertexts $(\mathsf{ct}_i)_{i\in[k]}$ where k is the number of input wires of C, and outputs a ciphertext* ct*;*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ *takes as inputs a secret key* sk *and a ciphertext* ct*, and outputs a plaintext M'.*

Correctness. *The scheme is said to be perfectly correct if for any $(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for any circuit C, for any plaintexts $(M_i)_{i\in[k]}$ where k is the number of input wires of C, the following holds:*

$$\mathsf{Dec}\left(\mathsf{sk}, \mathsf{Eval}(\mathsf{evk}, C, (\mathsf{ct}_i)_{i\in[k]})\right) = C((M_i)_{i\in[k]}) ,$$

where $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, M_i)$ for all $i \in [k]$.

Security. *The scheme is said to be* IND-CPA*-secure if for any PPT adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{indcpa}}(\mathcal{A})$ defined as follows is negligible:*

$$\left|2 \cdot \Pr\left[b = b' \;\middle|\; \begin{array}{c}(\mathsf{pk}, \mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\\ (M_0, M_1) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{evk})\\ b \leftarrow U(\{0,1\})\\ b' \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{evk}, \mathsf{Enc}(\mathsf{pk}, M_b))\end{array}\right] - 1\right| .$$

Our threshold signature construction relies on thresholdizing an FHE scheme by distributing its secret key and splitting decryption into two steps (PartDec, FinDec) following the structure of threshold-FHE [AJL+12, BGG+18]. While we could directly rely on threshold-FHE in a black-box manner, we opt not to, in order to obtain more compact decryption shares by relying on the Rényi divergence in the security analysis.

Thresholdizing an FHE Scheme. We rely on an FHE scheme whose ciphertexts have the form $(a, b) \in (\mathcal{R}_q^{\mathsf{FHE}})^2$ satisfying $as + b = \Delta \cdot M + e$ where $s \in \mathcal{R}^{\mathsf{FHE}}$ is the secret key, M is the underlying plaintext, e is some error satisfying $\|e\|_\infty \leq B_{\mathsf{eval}}$ for some bound B_{eval}, and Δ is a scaling factor. We thresholdize decryption by distributing the secret key s as $(\mathsf{sk}_i)_{i\in[n]} \leftarrow \mathsf{Share}(s, n, t)$, where Share is a t-out-of-n linear secret sharing scheme (see Definition 9).

$\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{A}}(\lambda)$
$\mathcal{L} \leftarrow \emptyset$
$(\mathsf{pp}, \mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{OSign}(\cdot)}(\mathsf{pp}, \mathsf{vk})$
return $\mathsf{Verify}(\mathsf{vk}, M^*, \sigma^*) = 1 \ \wedge \ (M^*, \sigma^*) \notin \mathcal{L}$

$\mathsf{OSign}(M)$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M)$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{(M, \sigma)\}$
return σ

Figure 4: Security experiment $\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{A}}$.

For simplicity, we focus on the n-out-of-n case, though our construction generalizes to the t-out-of-n setting with the usual adjustments (see, among others, [BGG+18, MBH23]). In this setting, we have $\mathsf{sk} = \mathsf{sk}_1 + \ldots + \mathsf{sk}_n$. We then define $\mathsf{PartDec}(\mathsf{sk}_i, (a, b))$ as returning $p_i \leftarrow a \cdot \mathsf{sk}_i + e_i$ where $e_i \leftarrow \mathcal{D}_{\mathsf{flood}}$ is an error, and

$$\mathsf{FinDec}((a, b), (p_i)_{i\in[n]}) = b + \sum_{i\in[n]} p_i = b + as + \sum_{i\in[n]} e_i \ .$$

Note that the flooding noise may overwrite the plaintext, if the plaintext precision (i.e., the ratio between the scaling factor and the noise magnitude) is insufficient. Rather than using a high plaintext precision throughout the computation, when the final plaintext is binary, we compute with sufficient precision to obtain it, and then we prepare flooding by cleaning. The purpose of cleaning is to increase the plaintext precision.

Our security based on the Rényi divergence allows one to set $\mathcal{D}_{\mathsf{flood}} = \mathcal{D}_{N,\eta}$ where the threshold flooding parameter η is set to $\eta \geq B_{\mathsf{eval}} \cdot \sqrt{10 \cdot N \cdot q_{\mathsf{sig}}}$, with q_{sig} being the number of signing queries made by the attacker and N being the ring degree. In contrast, relying on black-box threshold-FHE would lead to $\eta = \Omega(B_{\mathsf{eval}} \cdot 2^{\lambda/2})$ and therefore to degraded performance. This was already observed in [ASY22].

A.3 Digital Signatures

We recall the definition of digital signatures.

Definition 8. *A signature scheme is a tuple of PPT algorithms* (KeyGen, Sign, Verify) *with the following specifications:*

- $\mathsf{KeyGen}(1^\lambda)$ *takes as inputs a security parameter* λ *and outputs public parameters* pp*, a verification key* vk*, and a signing key* sk*;*
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M)$ *takes as inputs the public parameters* pp*, a signing key* sk*, and a message* M*, and outputs a signature* σ*.*
- $\mathsf{Verify}(\mathsf{vk}, M, \sigma)$ *takes as inputs a verification key* vk*, a message* M*, and a signature* σ *and outputs a bit* $b \in \{0, 1\}$*.*

Correctness. *For* $\varepsilon > 0$*, the scheme is* ε*-correct if for any* M*, the following probability is* $\geq 1 - \mathsf{negl}(\lambda)$*:*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, M, \sigma) = 1 \,\middle|\, \begin{matrix}(\mathsf{pp}, \mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M)\end{matrix}\right],$$

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

Strong Unforgeability. *The scheme satisfies strong unforgeability if for any PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{uf}}(\mathcal{A}) := \Pr(\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{A}}(\lambda) = 1)$ *is negligible, where experiment* $\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{A}}$ *is defined in Figure 4.*

A.4 Threshold Linear Secret Sharing Schemes

We recall the definition of (threshold) linear secret sharing schemes (LSSS) over a ring $\mathcal{R}$.

Definition 9. *A linear secret sharing scheme* LSSS *consists of PPT algorithms* (Share, Combine) *such that:*

- $\mathsf{Share}(s, n, t)$ *takes as input a number* n *of parties* $(P_i)_{i\in[n]}$*, a threshold* $t \leq n$*, and a secret* $s \in \mathcal{R}$*, and it outputs* $(\mathsf{sh}_i)_{i\in[n]}$*;*
- $\mathsf{Combine}(\{\mathsf{sh}_i\}_{i\in\mathcal{S}})$ *takes as input a set of shares* $\{\mathsf{sh}_i\}_{i\in\mathcal{S}}$ *and outputs a value* $s' \in \mathcal{R}$*.*

For any $n \geq t > 0$*, and any* $\mathcal{S} \subseteq [n]$*, we say that* $\mathcal{S}$ *is valid if* $|\mathcal{S}| = t$*, and otherwise we say it is invalid. Then, we further require that the following properties hold.*

Correctness. For any $s \in \mathcal{R}$ *and* $(\mathsf{sh}_i)_{i\in[n]} \leftarrow \mathsf{Share}(s, n, t)$*, we have* $\mathsf{Combine}(\{\mathsf{sh}_i\}_{i\in\mathcal{S}}) = s$*, for any valid set* $\mathcal{S}$*.*

Linearity. We say it is linear if Combine *is a linear function.*

Privacy. For any $s, s' \in \mathcal{R}$ *and any invalid* $\mathcal{S}$*, we have that the distributions:*

$$\{(\mathsf{sh}_i)_{i\in\mathcal{S}}\} \ \text{ and } \ \{(\mathsf{sh}'_i)_{i\in\mathcal{S}}\}$$

are identical, where $(\mathsf{sh}_i)_{i\in[n]} \leftarrow \mathsf{Share}(s, n, t)$ *and* $(\mathsf{sh}'_i)_{i\in[n]} \leftarrow \mathsf{Share}(s', n, t)$*.*

A.5 Dilithium

We recall, in Figure 5, the precise description of (the deterministic version of) Dilithium [BDK+21]. We use the same notations and elementary functions and refer the reader to the Dilithium documentation [BDK+21] for their definitions. In the signing algorithm Sign, we refer to $\mathbf{w}_1$ (Step 8) as the commitment, to c (Step 10) as the challenge, and to $\mathbf{z}$ (Step 11) as the response.

B Proof of Theorem 2

First, let us argue that the fact we tweak the signature by performing the test on the Hamming weight of the hint vector $\mathbf{h}$ only after decryption (Fig. 2, Steps 6–8) does not hurt security. This tweak implies that we are slightly under-rejecting (about 2% of produced signatures do not pass this test) and that we reveal invalid signatures to the attacker that would be rejected in Dilithium. This is actually not an issue: the hint does not play any role in the security of Dilithium, it is provided because $\mathbf{t}_0$ is not known to the verifier in Dilithium. We can therefore rely on the unforgeability of the Dilithium without verification key compression to argue about the security of our construction. In this variant, the vector $\mathbf{t}$ is public and there is no hint $\mathbf{h}$. Yet, the verification key can be compressed, and the hint $\mathbf{h}$ can be computed from the signature using Equation (3). Any attacker against our threshold scheme would therefore lead to an attacker against unforgeability of the variant of Dilithium without verification key compression. As a consequence, we argue about the security of our scheme by relying on the unforgeability of the deterministic variant of Dilithium without verification key compression.

For simplicity, we focus on the case $t = n$, so that the FHE decryption key s satisfies $s = \mathsf{sk}_1 + \cdots + \mathsf{sk}_n$. The proof extends beyond this setting with the usual technical adjustments (see, e.g., [BGG+18, MBH23]). The threshold signature construction has the following structure.

Dilithium.KeyGen

$\zeta \leftarrow U(\{0,1\}^{256})$
$(\rho, \rho', K) \leftarrow \mathsf{H}(\zeta)$
$\mathbf{A} \leftarrow \mathsf{ExpandA}(\rho) \in (\mathcal{R}_q^{\mathrm{SIG}})^{k \times \ell}$
$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \mathsf{ExpandS}(\rho') \in S_\eta^\ell \times S_\eta^k$
$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \bmod q$
$(\mathbf{t}_0, \mathbf{t}_1) \leftarrow \mathsf{Power2Round}_q(\mathbf{t}, d)$
$tr \leftarrow (\rho \parallel \mathbf{t}_1) \in \{0,1\}^{384}$
return $\mathsf{vk} = (\rho, \mathbf{t}_1)$, $\mathsf{sk} = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

$\mathsf{Sign}(\mathsf{sk}, M)$

$\mathbf{A} \leftarrow \mathsf{ExpandA}(\rho)$
$\mu \leftarrow \mathsf{H}(tr \parallel M)$
$\kappa \leftarrow 0, \quad (\mathbf{z}, \mathbf{h}) \leftarrow \perp$
$\rho' \leftarrow \mathsf{H}(K \parallel \mu)$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \leftarrow \mathsf{ExpandMask}(\rho', \kappa) \in S_{\gamma_1 - 1}^\ell$
  $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y} \bmod q$
  $\mathbf{w}_1 \leftarrow \mathsf{HighBits}(\mathbf{w}, 2\gamma_2)$
  $\tilde{c} \leftarrow \mathsf{H}(\mu \parallel \mathbf{w}_1) \in \{0,1\}^{256}$
  $c \leftarrow \mathsf{SampleInBall}(\tilde{c}) \in B_\tau$
  $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_1$
  $\mathbf{r}_0 \leftarrow \mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta \ \vee \ \|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ **then**
    $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  else
    $\mathbf{h} \leftarrow \mathsf{MakeHint}_q(-c\mathbf{t}_0, \ \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, \ 2\gamma_2)$
    if $\|c\mathbf{t}_0\|_\infty \geq \gamma_2 \ \vee \ \mathsf{HW}(\mathbf{h}) > \omega$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  $\kappa \leftarrow \kappa + \ell$
return $\sigma = (\mathbf{z}, \mathbf{h}, \tilde{c})$

$\mathsf{Verify}(\mathsf{vk}, M, \sigma = (\mathbf{z}, \mathbf{h}, \tilde{c}))$

$\mathbf{A} \leftarrow \mathsf{ExpandA}(\rho)$
$\mu \leftarrow \mathsf{H}(\mathsf{H}(\rho \parallel \mathbf{t}_1) \parallel M)$
$c \leftarrow \mathsf{SampleInBall}(\tilde{c})$
$\mathbf{w}_1' \leftarrow \mathsf{UseHint}_q(\mathbf{h}, \ \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, \ 2\gamma_2)$
return $\left(\|\mathbf{z}\|_\infty < \gamma_1 - \beta\right) \wedge \left(\tilde{c} = \mathsf{H}(\mu \parallel \mathbf{w}_1')\right) \wedge \left(\mathsf{HW}(\mathbf{h}) \leq \omega\right)$

Figure 5: Deterministic Dilithium.

- $\mathsf{PartSign}_1(\mathsf{pp}, \mathsf{sk}_i, M)$ homomorphically evaluates Sign_1. At the end of this first step, one obtains a ciphertext (a, b) encrypting commitments $\mathbf{w}_1 := (\mathbf{w}_1^{(1)}, \ldots, \mathbf{w}_1^{(L)})$: it satisfies $as + b = \Delta \cdot \mathbf{w}_1 + e_{\mathsf{eval}}$ for some small computation error e_{eval} satisfying $\|e_{\mathsf{eval}}\|_\infty \leq B_{\mathsf{eval}}$. Then, it returns a partial decryption $\mathsf{out}_{1,i}$ of (a, b) for the key share sk_i, defined as $\mathsf{out}_{1,i} := a \cdot \mathsf{sk}_i + e_{1,i}$, where $e_{1,i} \leftarrow \mathcal{D}_{N,\eta}$ is a Gaussian noise.
- Given all partial decryptions $(\mathsf{out}_{1,1}, \ldots, \mathsf{out}_{1,n})$ of (a, b), it is possible to recover $\mathbf{w}_1$ by combining the shares. We have:
$$\Delta \cdot \mathbf{w}_1 + e = b + \sum_{i \in [n]} \mathsf{out}_{1,i} \ ,$$
where $e = e_{\mathsf{eval}} + \sum_{i \in [n]} e_{1,i}$ is the decryption error term.
- Similarly, $\mathsf{PartSign}_2(\mathsf{pp}, \mathsf{sk}_i, \mathsf{out}_1, M)$ homomorphically evaluates Sign_2, in order to obtain a ciphertext (a', b') encrypting the responses $\mathbf{z} := (\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)})$, i.e., satisfying $a's + b' = \Delta \cdot \mathbf{z} + e'_{\mathsf{eval}}$ for some small computation error e'_{eval} satisfying $\|e'_{\mathsf{eval}}\|_\infty \leq B_{\mathsf{eval}}$. It returns the partial decryption $\mathsf{out}_{2,i}$ of (a', b') using the key share sk_i, defined as $\mathsf{out}_{2,i} := a' \cdot \mathsf{sk}_i + e_{2,i}$. We have:
$$\Delta \cdot \mathbf{z} + e' = b' + \sum_{i \in [n]} \mathsf{out}_{2,i} \ ,$$
where $e' = e'_{\mathsf{eval}} + \sum_{i \in [n]} e_{2,i}$.

We now provide a security analysis for this threshold signature scheme.

Let $\mathcal{A}$ denote an adversary against the unforgeability of our threshold signature scheme described in Figure 2. Let $\mathcal{S}_{\mathsf{cor}}$ denote the set of $n-1$ parties corrupted by $\mathcal{A}$. For the sake of simplicity, we assume that $\mathcal{A}$ only makes queries to $\mathsf{OPartSign}_1$ and $\mathsf{OPartSign}_2$ for the index $i \notin \mathcal{S}_{\mathsf{cor}}$. This is without loss of generality since it can answer by itself any queries for $j \in \mathcal{S}_{\mathsf{cor}}$, as it knows the corresponding secret key share sk_j. Our goal is to prove unforgeability based on the security (and correctness) of the FHE scheme, as well as the security of Dilithium. We rely on the analysis of Dilithium from [BdCE+26] which proves that it secure given aborted transcripts. That is, the unforgeability of Dilithium holds even when the attacker is given oracle access to OSign_1, which can be queried with a message M and returns L aborted transcripts, i.e., commitments $\mathbf{w}_1^{(1)}, \ldots, \mathbf{w}_1^{(L)}$. The attacker also has access to the standard signing oracle OSign, which returns a valid signature. As we are relying on the deterministic variant of Dilithium without verification key compression, a valid Dilithium signature is a pair $(c, \mathbf{z})$, which is the first signature that passes rejection checks. It is possible to retrieve the corresponding commitment $\mathbf{w}_1$ following the verification algorithm, and therefore to know the index $i \in [L]$ which corresponds to the first successful signature by matching $\mathbf{w}_1$ with the aborted transcript (there is a match as the scheme is deterministic).

We now proceed via a sequence of hybrid games, defined as follows.

$\mathbf{Hyb}_0$: This is the threshold unforgeability game, described in Figure 1.

$\mathbf{Hyb}_1$: In this hybrid game, we modify how the challenger replies to signing queries $\mathsf{OPartSign}_1$ and $\mathsf{OPartSign}_2$.

When $\mathcal{A}$ makes a query $\mathsf{OPartSign}_1(i, M)$, the challenger does the following. It first runs (in the clear) the Dilithium signing protocol L times, to recover $((\mathbf{w}_1^{(1)}, c^{(1)}, \mathbf{z}^{(1)}), \ldots, (\mathbf{w}_1^{(L)}, c^{(L)}, \mathbf{z}^{(L)}))$. It then homomorphically computes the ciphertext (a, b) as follows. Recall that the expected output of $\mathsf{PartSign}_1(\mathsf{pp}, \mathsf{sk}_i, M)$ is a partial decryption $\mathsf{out}_{1,i}$ by party P_i of (a, b). As we assume that $i \notin \mathcal{S}_{\mathsf{cor}}$, this implies that the output of $\mathsf{OPartSign}_1(i, M)$ is sufficient for the adversary to recover $\mathbf{w}_1 := (\mathbf{w}_1^{(1)}, \ldots, \mathbf{w}_1^{(L)})$.

Recall that (a, b) satisfies $b + as = \Delta \cdot \mathbf{w}_1 + e_{\mathsf{eval}}$, where e_{eval} is the underlying noise term, and s is the FHE secret key, and we have:
$$s = \mathsf{sk}_i + \sum_{j \in \mathcal{S}_{\mathsf{cor}}} \mathsf{sk}_j \ .$$
Then, in $\mathbf{Hyb}_1$, the challenger replies to a query $\mathsf{OPartSign}_1(i, M)$ as follows. It computes (a, b) as above and samples $e_{\mathsf{fl}} \leftarrow \mathcal{D}_{N,\eta}$. Then, it samples $\mathsf{out}_{1,i}$ as:
$$\mathsf{out}_{1,i} \leftarrow \Delta \cdot \mathbf{w}_1 - \sum_{j \in \mathcal{S}_{\mathsf{cor}}} a \cdot \mathsf{sk}_j - b + e_{\mathsf{fl}} \ . \tag{6}$$

Similarly, when $\mathcal{A}$ makes a query $\mathsf{OPartSign}_2(i, \mathsf{out}_1, M)$, the challenger first runs (in the clear) the (deterministic) Dilithium signing protocol, to recover $((\mathbf{w}_1^{(1)}, c^{(1)}, \mathbf{z}^{(1)}), \ldots, (\mathbf{w}_1^{(L)}, c^{(L)}, \mathbf{z}^{(L)}))$. Then, it computes the ciphertext (a', b') to be partially decrypted during this phase, which satisfies $b' + a's = \Delta \cdot \mathbf{z} + e'_{\mathsf{eval}}$. Finally, it

samples $e'_{\mathsf{fl}} \leftarrow \mathcal{D}_{N,\eta}$ and returns:

$$\mathsf{out}_{2,i} \leftarrow \Delta \cdot \mathbf{z} - \sum_{j \in \mathcal{S}_{\mathsf{cor}}} a' \mathsf{sk}_j - b' + e'_{\mathsf{fl}} \ . \tag{7}$$

Lemma 10. *Assuming perfect correctness of the threshold-FHE scheme, if there is an adversary which breaks unforgeability in* $\mathbf{Hyb}_0$ *with success probability* ε*, then its success probability in* $\mathbf{Hyb}_1$ *is at least* $\varepsilon^2/2$.

Proof. The only difference between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ is that, since we simulate the partial decryption queries based on the result of the decryption, we are missing the computation error e_{eval} that results from the homomorphic evaluation of the first (or second) stage of the underlying signature scheme in the partial decryption share. Concretely, in $\mathbf{Hyb}_1$, we set $\mathsf{out}_{1,i} \leftarrow \Delta \cdot \mathbf{w} - \sum_{j \in \mathcal{S}_{\mathsf{cor}}} a \cdot \mathsf{sk}_j - b + e_{\mathsf{fl}}$ while in $\mathbf{Hyb}_0$, we have $\mathsf{out}_{1,i} = \Delta \cdot \mathbf{w} - \sum_{j \in \mathcal{S}_{\mathsf{cor}}} a \cdot \mathsf{sk}_j - b + e_{\mathsf{eval}} + e_{\mathsf{fl}}$.

For simplicity, we ignore the public parameters, the verification key and the corrupted partial secret keys $\{\mathsf{sk}_j\}_{j \in \mathcal{S}_{\mathsf{cor}}}$ in the analysis. This is possible as they are fixed before the adversary can make any query. However, the messages it queries to its oracles as well as the responses it gets are not fixed and can adaptively depend on the prior queries. Let $\mathcal{D}_b$ denote their distribution in $\mathbf{Hyb}_b$, for $b \in \{0,1\}$.

Let Q_j denote the random variable corresponding to the input of the j-th query (which could be a query to $\mathsf{OPartSign}_1$ or $\mathsf{OPartSign}_2$), and let $\mathcal{Q}_{b,j}$ denote its distribution in $\mathbf{Hyb}_b$, for $b \in \{0,1\}$.

Similarly, we let E_j denote the random variable corresponding to the error term underlying the j-th ciphertext computed and partially decrypted in the j-th query (which could be a query to $\mathsf{OPartSign}_1$ or $\mathsf{OPartSign}_2$), and let $\mathcal{E}_{b,j}$ denote its distribution in $\mathbf{Hyb}_b$, for $b \in \{0,1\}$. By definition of $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$, we have $\mathcal{E}_{1,j} = \mathcal{D}_{N,\eta}$ while $\mathcal{E}_{0,j} = \mathcal{D}_{N,\eta,e_{\mathsf{eval},j}}$ where $e_{\mathsf{eval},j}$ is the computation error in the ciphertext that is being partially decrypted in the j-th query.

Let q_{sig} denote the total number of queries made by the adversary. Then, we have $\mathcal{D}_b = (\mathcal{E}_{b,q_{\mathsf{sig}}}, \mathcal{Q}_{b,q_{\mathsf{sig}}}, \ldots, \mathcal{E}_{b,1}, \mathcal{Q}_{b,1})$, for $b \in \{0,1\}$, and we aim to measure $\mathsf{R}_a(\mathcal{D}_0 \| \mathcal{D}_1)$.

By the multiplicativity property of the Rényi divergence (Lemma 5), we have:

$$\begin{aligned}
\mathsf{R}_a(\mathcal{D}_0 \| \mathcal{D}_1) &\le \max_{x \in X} \mathsf{R}_a(\mathcal{E}_{0,q_{\mathsf{sig}}} | X = x \| \mathcal{E}_{1,q_{\mathsf{sig}}} | X = x) \\
&\quad \cdot \mathsf{R}_a(\mathcal{Q}_{0,q_{\mathsf{sig}}}, \ldots, \mathcal{E}_{0,1}, \mathcal{Q}_{0,1} \| \mathcal{Q}_{1,q_{\mathsf{sig}}}, \ldots, \mathcal{E}_{1,1}, \mathcal{Q}_{1,1}) \\
&\le \exp\left(\frac{Na\pi B_{\mathsf{eval}}^2}{\eta^2}\right) \\
&\quad \cdot \mathsf{R}_a(\mathcal{Q}_{0,q_{\mathsf{sig}}}, \ldots, \mathcal{E}_{0,1}, \mathcal{Q}_{0,1} \| \mathcal{Q}_{1,q_{\mathsf{sig}}}, \ldots, \mathcal{E}_{1,1}, \mathcal{Q}_{1,1}) \ ,
\end{aligned}$$

where the last inequality follows from Lemma 6 and $\|e_{\mathsf{eval}}\|_\infty \le B_{\mathsf{eval}}$. Indeed, we have:

$$\begin{aligned}
\mathsf{R}_a(\mathcal{E}_{0,q_{\mathsf{sig}}} \| \mathcal{E}_{1,q_{\mathsf{sig}}}) &= \mathsf{R}_a(\mathcal{D}_{N,\eta,e_{\mathsf{eval},q_{\mathsf{sig}}}} \parallel \mathcal{D}_{N,\eta}) \\
&\le \exp\left(\frac{Na\pi e_{\mathsf{eval},q_{\mathsf{sig}}}^2}{\eta^2}\right) \\
&\le \exp\left(\frac{Na\pi B_{\mathsf{eval}}^2}{\eta^2}\right) \ .
\end{aligned}$$

Now, since $Q_{q_{\mathsf{sig}}}$ is the adaptive query made by the adversary given all prior information, it is a function of $E_{q_{\mathsf{sig}}-1}, Q_{q_{\mathsf{sig}}-1}, \ldots, E_1, Q_1$, and the data processing inequality (Lemma 5) gives us:

$$\begin{aligned}
&\mathsf{R}_a(\mathcal{Q}_{0,q_{\mathsf{sig}}}, \mathcal{E}_{0,q_{\mathsf{sig}}-1}, \ldots, \mathcal{E}_{0,1}, \mathcal{Q}_{0,1} \\
&\qquad \| \mathcal{Q}_{1,q_{\mathsf{sig}}}, \mathcal{E}_{1,q_{\mathsf{sig}}-1}, \ldots, \mathcal{E}_{1,1}, \mathcal{Q}_{1,1}) \\
&\le \mathsf{R}_a(\mathcal{E}_{0,q_{\mathsf{sig}}-1}, \ldots, \mathcal{E}_{0,1}, \mathcal{Q}_{0,1} \| \mathcal{E}_{1,q_{\mathsf{sig}}}, \ldots, \mathcal{E}_{1,1}, \mathcal{Q}_{1,1}) \ .
\end{aligned}$$

Using a recursion, we therefore obtain:

$$\mathsf{R}_a(\mathcal{D}_0 \| \mathcal{D}_1) \le \exp\left(\frac{Na\pi B_{\mathsf{eval}}^2 q_{\mathsf{sig}}}{\eta^2}\right) \ . \tag{8}$$

Setting $\eta = B_{\mathsf{eval}}\sqrt{10 \cdot N \cdot q_{\mathsf{sig}}}$, we get $\mathsf{R}_a(\mathcal{D}_0 \| \mathcal{D}_1) \le \exp(a\pi/10)$, and by the probability preservation property of the Rényi divergence, we obtain the claim by setting $a = 2$. We have

$$\Pr[\mathcal{A} \text{ wins } \mathbf{Hyb}_1] \ge \Pr[\mathcal{A} \text{ wins } \mathbf{Hyb}_0]^2 \cdot \exp\left(-\frac{2\pi}{10}\right) \ ,$$

which leads to our claim as $\exp(-2\pi/10) \ge 1/2$. □

$\mathbf{Hyb}_2$*:* This game is defined as $\mathbf{Hyb}_1$ except that the shares of the FHE secret key are now replaced by shares of 0, i.e., we set $\{\mathsf{sk}_i\}_{i \in [n]} \leftarrow \mathsf{Share}(0)$. Due to the privacy of the secret sharing scheme (Definition 9), the two games are perfectly indistinguishable.

$\mathbf{Hyb}_3$*:* In this game, we replace the encrypted signing key (including the PRF key) by an encryption of 0's. Note that we can make this changes as the challenger now simulates partial decryptions from Dilithium signatures that it computed in the clear. Thanks to IND-CPA security of the FHE scheme, games $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ are computationally indistinguishable.

We can now complete the proof of security of our signature by proving the following lemma.

Lemma 11. *If there is an adversary* $\mathcal{A}$ *breaking unforgeability in* $\mathbf{Hyb}_3$ *with success probability* ε*, then there exists an attacker with oracle access to aborted transcripts against Dilithium that breaks unforgeability with success probability* ε *and runtime essentially the same as* $\mathcal{A}$*'s.*

Proof. Let $\mathcal{A}$ denote an attacker that succeeds in $\mathbf{Hyb}_3$. We construct an attacker $\mathcal{B}$ against the unforgeability of Dilithium with oracle access to aborted transcripts as follows. Attacker $\mathcal{B}$ receives the signature verification key vk from its challenger. It generates a pair of FHE public and secret keys and computes encryptions of 0 as for the encrypted signing key and for the encrypted PRF key. It also generates $\{\mathsf{sk}_i\}_{i \in [n]} \leftarrow \mathsf{Share}(0)$. Then $\mathcal{B}$ runs $\mathcal{A}$. When the latter sends its target set of corruptions $\mathcal{S}_{\mathsf{cor}}$, $\mathcal{B}$ forwards the public parameters to $\mathcal{A}$ as well as the partial signing keys sk_i for $j \in \mathcal{S}_{\mathsf{cor}}$. Then, when $\mathcal{A}$ makes a query to $\mathsf{OPartSign}_1$ (resp. $\mathsf{OPartSign}_2$) with some message M for some $i \notin \mathcal{S}_{\mathsf{cor}}$, attacker $\mathcal{B}$ makes a signing query M to its oracle for aborted transcripts OSign_1 (resp. OSign). It obtains $(\mathbf{w}_1^{(1)}, \ldots, \mathbf{w}_1^{(L)})$ as a response (resp. $(c, \mathbf{z})$), and replies to $\mathcal{A}$ by producing the requested partial decryption using the simulation described in Equation (6) (resp. Equation (7), by first recovering the index $i \in [L]$ corresponding to the response $\mathbf{z}$ obtained by comparing the commitment $\mathbf{w}_1$ induced by it with the

aborted commitments obtained from OSign_1 for the same message). When $\mathcal{A}$ halts with some forgery (M^*, σ^*), so does $\mathcal{B}$. □

This completes the proof of Theorem 2. □

Concrete Choice for the Flooding Parameter η. The above proof sets $\eta = B_{\mathsf{eval}}\sqrt{10 \cdot N \cdot q_{\mathsf{sig}}}$, which is fine for the proof to go through in the theoretical sense. We now explain how to set η concretely for practical deployment, by going back to Equation (8) from the proof of Lemma 10.

Define $\varepsilon_b = \Pr[\mathcal{A} \text{ wins } \mathbf{Hyb}_b]$, for $b \in \{0,1\}$. By the probability preservation property of the Rényi divergence, we have:

$$\varepsilon_0^{\frac{a}{a-1}} \leq \varepsilon_1 \cdot \mathrm{R}_a(\mathcal{D}_0 \| \mathcal{D}_1) \ ,$$

and then, from Equation (8), we obtain:

$$\varepsilon_0^{\frac{a}{a-1}} \leq \varepsilon_1 \cdot \exp\left(\frac{Na\pi B_{\mathsf{eval}}^2 q_{\mathsf{sig}}}{\eta^2}\right) \ .$$

We raise the above inequality to the power $\frac{a-1}{a}$ and define η as $\eta = B_{\mathsf{eval}}\sqrt{N\pi q_{\mathsf{sig}} \cdot \ln(2) \cdot x}$ for $x > 0$, so that we have:

$$\varepsilon_0 \leq \varepsilon_1^{\frac{a-1}{a}} \cdot 2^{\frac{(a-1)}{x}} \ .$$

Our objective is to have $T_0/\varepsilon_0 \geq 2^{128}$, under the assumption that T_1/ε_1 is a little higher than 2^{128}, where T_b is the adversary's run-time in $\mathbf{Hyb}_b$ for $b \in \{0,1\}$. Note that $T_0 = T_1$ and that only ε_b changes in the transition between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$. Let us define $E_b = -\log_2 \varepsilon_b$ for $b \in \{0,1\}$. Then, passing the above inequality through $\log_2$, we want:

$$E_1 \cdot \frac{a-1}{a} - \frac{a-1}{x} \leq E_0 \ ,$$

which is an equality when $x = \frac{a(a-1)}{E_1(a-1) - E_0 a}$. In order to minimize η, our goal is to minimize x, for $a > 1$, given E_1. This is done by setting $a = \sqrt{E_1}/(\sqrt{E_1} - \sqrt{E_0})$, which leads to $x = 1/(\sqrt{E_1} - \sqrt{E_0})^2$. We obtain:

$$\log_2 \eta = \log_2(B_{\mathsf{eval}}\sqrt{N\pi q_{\mathsf{sig}} \cdot \ln(2)}) - \log_2(\sqrt{E_1} - \sqrt{E_0}) \ .$$

With $N = 2^{16}$ and $q_{\mathsf{sig}} = 2^{64}$, since $\log_2(\sqrt{\pi \ln(2)}) \approx 0.56$, we then have:

$$\log_2 \eta \approx \log_2 B_{\mathsf{eval}} + 40.56 - \log_2(\sqrt{E_1} - \sqrt{E_0}) \ .$$

Assuming that $E_1 \geq E_0 + 1$, so that at most one bit of security is lost between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$, it can be checked that $-\log_2(\sqrt{E_1} - \sqrt{E_0}) \leq 5$ for all $E_1 \leq 256$. Hence, to obtain a security loss bounded by 1 bit (for any security level below 256 bits), it suffices to set $\eta = 2^{46} \cdot B_{\mathsf{eval}}$. Finally, note that a Gaussian polynomial e with $N = 2^{16}$ coordinates sampled according to $\mathcal{D}_\eta$ satisfies $\|e\|_\infty \leq 6\eta$ with probability at least $1 - 2^{-128}$, implying that it suffices to flood with a Gaussian error that is at most 49-bit larger than B_{eval}.

C Proof of Lemma 3

Since the order of 5 modulo $2N$ is $N/2$, we have $(5^{k \cdot \frac{\ell}{2}})^{N/\ell} \equiv 1 \bmod 2N$ for each integer k. Moreover, the set $\{5^{k \cdot \frac{\ell}{2}} \bmod 2N : k \in [0, N/\ell) \cap \mathbb{Z}\}$ is the set of roots of $X^{\frac{N}{\ell}} - 1 = 0$ modulo $2N$, which is also $\{1 + 2\ell \cdot k : k \in [0, N/\ell) \cap \mathbb{Z}\}$.

Therefore, for each $j = 0, 1, \cdots, \ell/2 - 1$, we have:

$$\begin{aligned}
\frac{\ell}{N} \cdot \sum_{k=0}^{\frac{N}{\ell}-1} m(\zeta_{j+k \cdot \frac{\ell}{2}}) &= \frac{\ell}{N} \cdot \sum_{k=0}^{\frac{N}{\ell}-1} \sum_{t=0}^{N-1} m_t \cdot \zeta_{j+k \cdot \frac{\ell}{2}}^t \\
&= \frac{\ell}{N} \cdot \sum_{t=0}^{N-1} m_t \cdot \left(\sum_{k=0}^{\frac{N}{\ell}-1} \zeta^{t \cdot 5^j \cdot 5^{k \cdot \frac{\ell}{2}}} \right) \\
&= \frac{\ell}{N} \cdot \sum_{t=0}^{N-1} m_t \cdot \left(\sum_{k=0}^{\frac{N}{\ell}-1} \zeta^{t \cdot 5^j \cdot (1 + 2\ell \cdot k)} \right) \\
&= \frac{\ell}{N} \cdot \sum_{t=0}^{N-1} m_t \cdot \zeta^{t \cdot 5^j} \cdot \left(\sum_{k=0}^{\frac{N}{\ell}-1} \zeta^{t \cdot 5^j \cdot 2\ell \cdot k} \right) \ . \qquad (9)
\end{aligned}$$

If t is a multiple of N/ℓ, then $\zeta^{t \cdot 2\ell \cdot 5^j} = 1$; hence, we have

$$\sum_{k=0}^{\frac{N}{\ell}-1} \zeta^{t \cdot 5^j \cdot 2\ell \cdot k} = \frac{N}{\ell} \ .$$

If t is *not* a multiple of N/ℓ, then $\zeta^{t \cdot 2\ell \cdot 5^j} \neq 1$, because 5^j is odd; hence, we have

$$\sum_{k=0}^{\frac{N}{\ell}-1} \zeta^{t \cdot 5^j \cdot 2\ell \cdot k} = \frac{1 - \zeta^{t \cdot 5^j \cdot 2\ell \cdot \frac{N}{\ell}}}{1 - \zeta^{t \cdot 5^j \cdot 2\ell}} = 0 \ .$$

Going back to Equation (9), we obtain, for each $j = 0, 1, \cdots, N/2 - 1$:

$$\frac{\ell}{N} \cdot \sum_{k=0}^{\frac{N}{\ell}-1} m(\zeta_{j+k \cdot \frac{\ell}{2}}) = \sum_{t=0}^{\ell-1} m_{t \cdot \frac{N}{\ell}} \cdot \zeta^{t \cdot \frac{N}{\ell} \cdot 5^j} = m'(\zeta_j) \ .$$

This proves the theorem. □

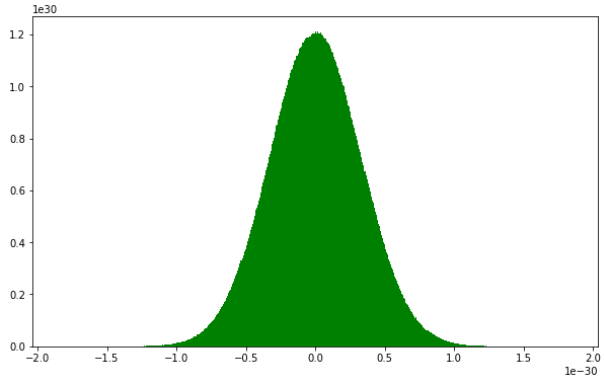
D Additional Implementation Aspects

D.1 Error Analysis and Noise Flooding

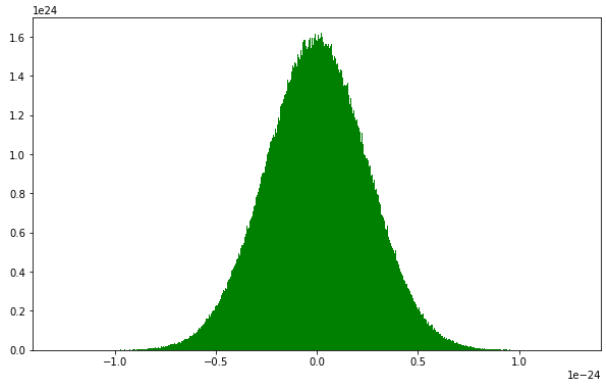
We analyzed the *error* in ciphertexts by measuring the deviation of the errors in the coefficients. To be more precise, we first measured the decrypted slot values from their closest integers and decoded them. Then, we extracted the errors in the coefficients that will be used during threshold decryption (multiples of 4 from 0 to $N/2 - 1$ for Round 1 and all multiples of 64 for Round 2). This deviation represents the noise in coefficients normalized by the scale factor Δ. This measurement was then used to set the magnitude of noise flooding.

After Round 2, the output ciphertext contains a valid signature if the signing succeeds, while the ciphertext contains a zero vector if the signing fails. We analyzed each case separately. Note that if the ciphertext decrypts to a zero vector, each error in the coefficient is an integer divided by the scale factor. Hence, the errors are discrete as we verified in the experiments.

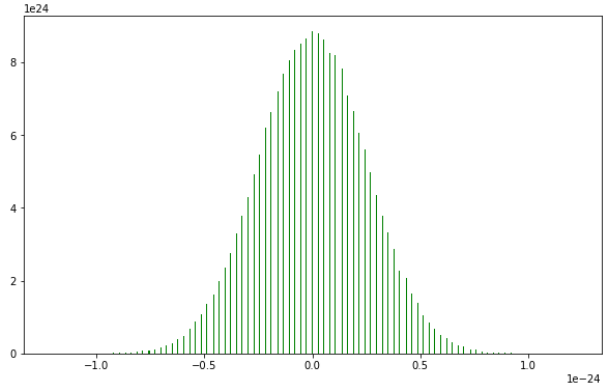
Figure 6 illustrates the error distribution before threshold decryption during $\mathsf{PartSign}_1$, $\mathsf{PartSign}_2$ with a signature, and $\mathsf{PartSign}_2$ without signatures (all samples rejected). All error distributions follow either a Gaussian or a rounded Gaussian distribution; the p-values of the K-S test are 0.79, 0.92, and 0.68, respectively. Note



(a) $\mathsf{PartSign}_1$.



(b) $\mathsf{PartSign}_2$ **with a non-rejected signature.**



(c) $\mathsf{PartSign}_2$ **with all rejected signatures.**

Figure 6: Error distribution before threshold decryption for each round. Figure 6a comprises 65 536 000 **error values obtained with** 1000 **iterations. Figures 6b and 6c comprise** 57 933 824 **and** 7 602 176 **values obtained from** 884 **and** 116 **samples, respectively.**

that the kurtoses of the pre-decryption Round-1 and Round-2 error distributions are 3.00.

The worst errors in coefficients among 1000 iterations were $2^{-98.77}$, $2^{-79.37}$, and $2^{-79.44}$, and the standard deviations σ are $2^{-101.26}$, $2^{-81.72}$, and $2^{-81.73}$, respectively.

We added 60 bits of flooding noise on top of conservative error bounds: 2^{-90} and 2^{-75} in Round-1 and Round-2, respectively. These bounds correspond to 2447.49σ and 105.64σ (105.77σ for the rejected ciphertexts). Assuming that the error distributions are Gaussian, the probability of an error occurring in the tails is much smaller than 2^{-8000}. This ensures that neither ciphertext decrypted during signing process contains a slot with outlying error, with probability $> 1 - 2^{-128}$. According to the theoretical analysis of Section 3.3 and the concrete discussion at the end of Appendix B, this allows us to support $q_{\text{sig}} = 2^{64}$ signature issuances.

D.2 Detailed Experimental Results

In Tables 2 and 3, we list the main steps performed homomorphically. For each one, we provide the input and output ciphertext modulus and scaling factor, as well as the time consumed by that step.

	Input ($\log q$, $\log \Delta$)	Output ($\log q$, $\log \Delta$)	Latency
Small integers BTS	(38, 32.0)	(821, 38.0)	71.4 ms
Bits extraction	(813, 30.0)	(663, 31.0)	37.2 ms
Clean random bits	(654, 22.0)	(613, 25.0)	-
-		(569, 31.1)	-
-		(519, 43.4)	-
-		(456, 67.2)	-
-		(366, 111.6)	25.3 ms
Product tree on MSB	(663, 31.0)	(508, 31.5)	14.9 ms
Compute LSB	(456, 67.2)	(344, 45.9)	3.1 ms
Compute $\mathbf{y}$	(366, 111.6)	(241,120.5)	1.1 ms
Compute C2S & $\mathbf{w}$	(266, 66.0)	(68, 45.0)	11.2 ms
Bit extraction for $\mathbf{w}$	(68, 45.0)	(663, 31.0)	537.0 ms
Reduction modulo q	(663, 31.0)	(499, 22.9)	28.8 ms
Bits BTS	(499, 22.9)	(516, 40.4)	44.2 ms
Reduction modulo $2\gamma_2$	(516, 40.4)	(414, 23.5)	106.0 ms
Clean & Combine bits	(413, 22.5)	(121, 110.8)	119.3 ms
BTS LowBits & LSB	(413, 22.5)	(661, 59.0)	41.6 ms
Threshold Decrypt	(121,113.8)	-	-
Total	-	-	1041.2 ms

Table 2: Homomorphic computations for $\mathsf{PartSign}_1$. For all steps, we provide the ciphertext modulus q and the scaling factor Δ at the beginning and end of the step, in bits. 'MSB' and 'LSB' refer to most and least significant bits of $\mathbf{y}$, respectively.

During the cleaning step, the scale factor Δ and the error term are amplified at each iteration of the map $h_1 : x \mapsto 3x^2 - 2x^3$. The CKKS rescaling operation is used to suppress the error. Notably, we set the rescale amounts conservatively large, ensuring that the dominant error component after rescaling comes from rescale procedure itself, rather than from the error accumulated before the rescaling. Specifically, we rescale by more than $\log\left(14 \cdot \sqrt{N/24} \cdot hwt\right)$ bits after each h_1 evaluation. The hwt term addresses the CKKS encoding.

	Input ($\log q$, $\log \Delta$)	Output ($\log q$, $\log \Delta$)	Latency
$c \cdot \mathbf{s}_1$, $c \cdot \mathbf{s}_2$ & C2S	(1287,119.0)	(692,119.0)	12.8 ms
Thrifty comparison	(530, 42.0)	(414, 23.9)	107.4 ms
Binary gates 1	(414, 23.9)	(248, 24.0)	4.7 ms
Bits BTS	(248, 24.0)	(861, 33.0)	42.1 ms
Binary gates 2	(857, 29.0)	(542, 28.5)	21.9 ms
Clean bit	(542, 28.5)	(241,120.5)	4.6 ms
Mask	(241,120.5)	(103, 84.9)	5.0 ms
Threshold Decrypt	(103, 84.9)	-	-
Total	-	-	198.4 ms

Table 3: Homomorphic computations for $\mathsf{PartSign}_2$. For all steps, we provide the ciphertext modulus q and the scaling factor Δ at the beginning and end of the step, in bits.