Dilithium-Based Verifiable Timed Signature Scheme

Erkan Uslu[1,2][0000−0001−8033−8922] and Oğuz Yayla[1][0000−0001−8945−2780]

[1] Middle East Technical University, Ankara, TURKEY
{erkan.uslu,oguz}@metu.edu.tr
[2] ASELSAN Inc. Ankara, TURKEY
erkanuslu@aselsan.com

Abstract. Verifiable Timed Signatures (VTS) are cryptographic constructs that enable obtaining a signature at a specific time in the future and provide evidence that the signature is legitimate. This framework particularly finds utility in applications such as payment channel networks, multiparty signing operations, or multiparty computation, especially within blockchain architectures. Currently, VTS schemes are based on signature algorithms such as BLS signature, Schnorr signature, and ECDSA. These signature algorithms are considered insecure against quantum attacks due to the effect of Shor's Algorithm on the discrete logarithm problem. We present a new VTS scheme called VT-Dilithium based on CRYSTALS-Dilithium Digital Signature Algorithm that has been selected as NIST's quantum-resistant digital signature standard and is considered secure against both classical and quantum attacks. Integrating Dilithium into the VTS scheme is more challenging problem due to its complex mathematical operations (i.e. polynomial multiplications, rounding operations) and large module parameters such as polynomials, polynomial vectors, and matrices. This work aims to provide a comprehensive exposition of the VT-Dilithium scheme.

Keywords: Verifiable Timed Signatures · CRYSTALS-Dilithium · Threshold Secret Sharing · Post-Quantum Cryptography.

1 Introduction

Verifiable Timed Signatures (VTS) [16] represent a cryptographic protocol applied to digital signature algorithms. Essentially, a committer may wish for a digital signature to become obtainable after a specific period of time. By embedding this signature into a time-lock puzzle, she ensures that the signature can only be acquired after a pre-determined duration time has elapsed. Regarding verifiability, a prover seeking to obtain the signature can verify whether it is valid before solving the time-lock puzzle [12]. This prevents irreversible efforts to reveal an invalid signature.

Current VTS schemes are based on BLS Signature [4], Schnorr Signature [13] and Elliptic Curve Digital Signature Algorithm (ECDSA) [7] relying on discrete

logarithm problem named as VT-BLS, VT-Schnorr, and VT-ECDSA respectively. It is assumed that with the emergence of quantum computers possessing sufficient computational power, the discrete logarithm problem could be solved by Shor's Algorithm [15]. Consequently, digital signature algorithms based on the discrete logarithm problem will become insecure.

CRYSTALS-Dilithium [10] is a quantum-resistant digital signature algorithm relying on the lattice-based (module) learning with errors problem. The algorithm is resilient to both classical and quantum attacks. It is built upon the Fiat-Shamir with Aborts [9] method. Due to its parametric structure, performance, parameter sizes and comprehensibility, it was selected as the standard algorithm for quantum-resistant signature in July 2022 by National Institute of Standards and Technology (NIST). CRYSTALS-Dilithium is referred as ML-DSA (Module Lattice - Digital Signature Algorithm) in [1].

The main components of a VTS system are Non-Interactive Zero-Knowledge (NIZK) proofs, Time-Lock Puzzles, Range Proofs, Threshold Secret Sharing (TSS) algorithm, and Digital Signature algorithms. Different signature schemes can be integrated into a VTS scheme while adhering to these structures. In order to integrate these signature algorithms into a VTS system, their secret keys, public keys, and signatures need to be split into a certain number of shares using the Threshold Secret Sharing (TSS) algorithm, and should be reconstructable from these shares. The process for dividing them as follows:

1. The secret key of the main signature is divided into shares according to the TSS method. These shares are generated according to the secret key constraints of the respective signature algorithm. The term 'shared' in shared secret keys comes from the concept of shares in the TSS algorithm. In VTS schemes, shared secret keys always remain secret.
2. Shared public keys are produced from the shared secret keys. This process follows the same logic as generating public keys from secret keys in the algorithm's key generation procedures. The main public key should be obtainable from a threshold number of shared public keys.
3. Similarly, shared signatures are produced from the shared secret keys according to the signing operation of the algorithm. The main signature should be obtainable from a threshold number of shared signatures.
4. The obtained shared signatures should be verifiable against the shared public keys at the same index.

This approach ensures that the VTS system maintains security claims while incorporating various digital signature schemes.

In this work, we introduce the VT-Dilithium scheme, a verifiable timed signature mechanism based on the CRYSTALS-Dilithium Digital Signature Algorithm. Dilithium algorithm involves significantly more complex operations for key generation, signing, and signature verification compared to BLS, Schnorr and ECDSA signature schemes. In particular, operations within the Dilithium involve computations such as the multiplication of vectors and matrices containing polynomials of degree 255. These parameters are uniformly generated using

SHAKE functions [5]. Operations aimed at reducing the size of the public key [2] necessitate the definition of specific rounding or hint functions within the algorithm, rendering it more complex. Additionally, ensuring the rejection conditions in the signing operation requires certain parameters to fall within specific ranges, leading to the possibility of repeating the signing process if these conditions are not met. When compared to signature algorithms based on discrete logarithms or integer factorization, the complex structures of Dilithium make its integration into signing protocols more challenging.

The outline of the work is as follows. In Section 2, we provide the preliminaries. Here, we presented the necessary cryptographic structures and definitions for defining VT-Dilithium. In Section 3, we define the Dilithium - Based Verifiable Timed Signature scheme and its sub-algorithms required for its operation. In Section 4, we extensively detailed the correctness of the sub-algorithms defined for VT-Dilithium. Finally, in Section 5, we conclude our work and discuss potential future research directions.

2 Preliminaries

Ring operations in Dilithium are over $R = \mathbb{Z}[X]/(X^N+1)$ and $R_q = \mathbb{Z}_q[X]/(X^N+1)$ where $q = 8380417$ and $N = 256$. Regular letters represent only a polynomial in R and R_q, bold lower-case letters correspond to column vectors and bold upper-case letters are matrices. $[\![\text{condition}]\!]$ returns 1 if the condition is true, otherwise returns 0. For modular reduction $x' \coloneqq x \bmod^{\pm} a$ implies $x' \in (\frac{-a}{2}, \frac{a}{2}]$ (if a is odd then $x' \in [\frac{-a-1}{2}, \frac{a-1}{2}]$). In addition, $x' \coloneqq x \bmod^{+} a$ implies $x' \in [0, a)$. Lastly, $\|\mathbf{w}\|_\infty$ implies the maximum coefficient of all of polynomials in $\mathbf{w}$. We represent the set containing integers $\{1, \ldots, n\}$ as $[n]$. The other parameters used in our work are provided in Table 1. In this paper, we utilize the supportive algorithms of Dilithium, namely $\mathsf{Power2Round}_q$, $\mathsf{SampleInBall}(\rho)$, $\mathsf{Decompose}_q$, $\mathsf{HighBits}_q$, $\mathsf{LowBits}_q$, $\mathsf{MakeHint}_q$ and $\mathsf{UseHint}_q$, in the same manner as described in [10, Figure 3]

2.1 Cryptographic Primitives

VTS schemes incorporate multiple cryptographic primitives, and we give the main ones in this section.

Digital Signatures Digital signatures [8] are mathematical methods employed to authenticate and validate the integrity of messages or documents. They encompass the generation of a distinct signature that is attached to the message by the sender. A digital signature is produced using the secret key and can solely be validated using the corresponding public key. Digital signatures serve to confirm that the message remains unaltered during transmission and indeed originates from the purported sender.

Table 1: List of Parameters

R	Polynomial Ring over $\mathbb{Z}[X]/(X^N+1)$
N	Degree of Polynomials
q	Prime Modulus of Dilithium
c	Challenge Polynomial of Dilithium
ω	max # of 1's in $\mathbf{h}_i$ vectors
w_i	i'th polynomial of $\mathbf{w}$
$w[i]$	i'th integer coefficient of w
$\mathbf{w}^{(i)}$	i'th vector share of $\mathbf{w}$
$w_j^{(i)}$	i'th polynomial share of $w_j \in \mathbf{w}$
$w_j^{(i)}[k]$	i'th integer coefficient share of $w_j[k] \in w_j$
pp	Public Parameter
$\mathbf{T}$	A Specific Time For Time-Lock Puzzles
λ	Security Parameter
$\mathbf{y}$	Final mask value that used for obtaining the σ
pk	Public Key of Main Signature
sk	Secret Key of Main Signature
σ	Main Signature
pk_i	Shared Public Key of σ_i
sk_i	Shared Secret Key of σ_i
σ_i	i'th Shared Signature of σ
crs	Common Reference String
$\ell_i(\cdot)$	i'th Lagrange interpolation basis

Time-lock Puzzles (TLP) TLP [12] are cryptographic technique that involves securing data or messages in a way that they cannot be accessed until a certain period of time has passed. These puzzles utilize computational challenges, such as factoring large numbers or solving complex mathematical problems, to create a delay mechanism. Once the specified time has elapsed, the puzzle can be efficiently solved, granting access to the hidden information. Time-lock puzzles have diverse applications ranging from secure communications to digital asset management, where delaying access to sensitive information is crucial for security and privacy. In VT-BLS, VT-Schnorr, and VT-ECDSA schemes, a linearly homomorphic time-lock puzzle method [11] is used. This is because the homomorphic properties of the time-lock puzzle can be leveraged for performance improvements in the system. When defining VT-Dilithium, we will provide a general definition of TLP.

Definition 1. *The following functions of Time-Lock Puzzles are used in our VT-Dilithium construction:*

- $pp \coloneqq \mathsf{TLP.PuzzleSetup}(1^\lambda, \mathbf{T})$ *Takes security parameter* λ *and desired time* $\boldsymbol{T}$ *as inputs and generate the public parameters pp.*
- $Z_i \coloneqq \mathsf{TLP.PuzzleGen}(pp, \sigma_i; r_i)$ *Takes public parameters pp, shared signatures* σ_i *and random coins* r_i *and outputs corresponding puzzles* Z_i*.*

– $\sigma_i \coloneqq \mathsf{TLP.PuzzleSolve}(pp, Z_i)$ *This sub-algorithm is to solve puzzle* Z_i *and obtain corresponding shared signature* σ*.*

Verifiable Timed Signatures (VTS) VTS [16] are concept where a sender creates a commitment to a signature that remains hidden until a specified time **T** elapses. This commitment is accompanied by proof that confirms the validity of the enclosed signature relative to the correct public key. Any person can verify this proof to ensure the commitment's validity. Neither the commitment nor the proof must not reveal any information about the signature to any potential adversary with limited computational capacity, as denoted by **T**. This property is called as privacy. Furthermore, the algorithm should satisfy the soundness property so that the adversaries should be unable to generate valid proof for a commitment lacking a valid signature corresponding to a public key.

More formally, let σ is a digital signature. In VTS schemes, σ is committed to in commitment C. $\mathsf{ForceOp}$ algorithm in VTS, is to obtain σ after time **T** if Vrfy algorithm outputs 1. To be more clear, we give a formal definition of Verifiable Timed Signatures.

Definition 2. *The concept of Verifiable Timed Signatures (VTS) consists of the following functions.*

– $(C, \pi) \coloneqq \mathsf{Commit}(\sigma, \mathbf{T})$: *The commitment algorithm takes a digital signature* σ *and time* $\boldsymbol{T}$ *and outputs the commitment* C *and proof* π*.*
– $0/1 \coloneqq \mathsf{Vrfy}(pk, M, C, \pi)$: *Verify checks whether* C *has a legitimate signature in terms of public key of digital signature algorithm - pk, message -* M *and proof -* π *and then, algorithm outputs 1 otherwise 0.*
– $(\sigma, r) \coloneqq \mathsf{Open}(C)$: *This sub-algorithm outputs signature and randomness - r, when the committee wants to open signature without any kind of effort.*
– $\sigma \coloneqq \mathsf{ForceOp}(C)$: *Force open phase takes the commitment* C *and outputs* σ *after time* $\boldsymbol{T}$*.*

Non-interactive Zero-Knowledge Proofs (NIZKPs). NIZKPs [3] represent a powerful cryptographic tool with wide-ranging applications in ensuring data privacy and security. This protocol enables a prover to convince a verifier of the validity of a statement without any direct interaction. This property is particularly valuable in scenarios where direct communication between parties is impractical or impossible. NIZKP's are employed in VTS protocols to generate legitimate evidence of the validity of signatures embedded within puzzles.

Definition 3. *A NIZKP is composed of the following functions.*

– $crs \coloneqq \mathsf{ZKsetup}(1^\lambda)$: *Takes security parameter as an input, outputs common reference string* crs*.*
– $\pi \coloneqq \mathsf{ZKprove}(crs, x, w)$: *Takes* crs*, statement* x *and witness* w*, outputs the proof* π*.*
– $0/1 \coloneqq \mathsf{ZKverify}(crs, x, \pi)$: *Outputs 1 if the statement* x *is verified with the proof* π*, otherwise 0.*

Threshold Secret Sharing (TSS) TSS [14] is a cryptographic technique designed to distribute a secret among multiple parties in such a way that it can be only reconstructed when a predetermined threshold of participants collaborate. This method ensures that no single entity holds complete access to the secret, enhancing security against potential breaches or unauthorized access. In threshold secret sharing, the secret is divided into shares, each distributed to different participants. The original secret can be recovered only when a sufficient number of shares are combined.

Definition 4. *Threshold secret sharing consists of share and reconstruct algorithms for threshold t out of n shares defined as:*

- $(s^{(1)}, \ldots, s^{(n)}) \coloneqq \mathsf{TSS.IntShare}(s)$: *Takes integer s as an input and outputs integer shares of s.*
- $s \coloneqq \mathsf{TSS.IntReconstruct}(s^{(1)}, \ldots, s^{(t)})$: *Takes any t out of n integer shares as an input and outputs secret integer s.*

A secret can be embedded into a polynomial in constant term. Let $f(x) = a_0 + a_1 X + \cdots + a_n X^{n-1}$. In this construction a_0 is secret. After that, for all $i \in [n]$ shared points can be calculated as $(i, f(i)) = (x_i, y_i)$. The secret a_0 can be obtained from any threshold t points using Lagrange interpolation

$$a_0 = f(0) = \sum_{j=0}^{k-1} y_j \ell_j(0) \text{ where } \ell_j(x) = \prod_{0 \le m \le k, m \ne j} \frac{x - x_m}{x_j - x_m}.$$

In the VT-Dilithium scheme, we will use n-out-of-n TSS. This means that in our scheme, all n shares are required to reconstruct the secret. The reasons for using n-out-of-n in VT-Dilithium will be explained in detail in Section 3.2.

Range Proofs Range Proofs are a method introduced in [16] to efficiently demonstrate that the solution to a TLP falls within a specified range. Unlike previous methods, this protocol can verify multiple TLPs at once, and the size of the proof remains consistent regardless of the size of the interval. This protocol is broadly applicable to all time-lock puzzles or ciphertexts that exhibit properties of plaintext and randomness homomorphism. It may be of interest independently for various applications.

2.2 CRYSTALS-Dilithium

CRYSTALS-Dilithium [10] is a digital signature algorithm selected as a standard in NIST Post-Quantum Standardization process. It relies on lattice based - module learning with errors problem and utilizes the Fiat-Shamir with Aborts method. Dilithium is believed to be safe from opponents who has large-scale quantum computers. Specifically, it is thought that Dilithium is highly unforgeable, meaning that the system can be used to identify unauthorized changes to data and verify the identity of the signatory.

In this work, we use the notations of the CRYSTALS-Dilithium proposal [10]. Therefore, we specify the algorithms and notations related to Dilithium in this paper in the same manner.

Three different parameter sets are defined for the Dilithium. These parameter sets are determined according to the security levels defined by NIST. The parameter sets and the security levels they provide are given in [10].

3 Dilithium - Based Verifiable Timed Signature Scheme

In this section, we define our VTS scheme, VT-Dilithium, based on Dilithium. The general structure of VT-Dilithium is outlined in Figure 1. To define VT-Dilithium, our first step is to demonstrate that the polynomials and vectors used in the key generation, signing, and signature verification algorithms in Dilithium can also be defined in TSS. Once this is established, the next step is to split the secret key parts (i.e. $\mathbf{s}_1$ and $\mathbf{s}_2$) of the main signature (i.e. σ) for VT-Dilithium into shares, and generate corresponding public keys and signatures for these parts. We explain how these steps are carried out in the subsequent parts of this section.

3.1 Threshold Secret Sharing for Dilithium's Parameters

In BLS, Schnorr, and ECDSA-based VTS schemes, the secret keys of the algorithms are integer values in $\mathbb{Z}_p$. It is quite straightforward to divide these values into shares using the TSS algorithm and ensure that these shares also lie in $\mathbb{Z}_p$. However, in Dilithium, most of parameters including parts of secret key, public key or signature are polynomials and vectors rather than integers. Therefore, to divide the parameters of Dilithium into shares using the TSS algorithm, we first need to demonstrate how a polynomial and a vector can be divided into shares and subsequently reconstructed. Throughout the entire paper, we consistently use an n-out-of-n TSS scheme. This means that all n shares generated must be used in the reconstruction process.

In our scheme, we define Algorithm 1, which takes a polynomial that is used in Dilithium as an input and outputs shares of this polynomial, and Algorithm 2, which is used to obtain the secret polynomial from its secret shares.

Algorithm 1 $\mathsf{TSS.PolyShare}(z)$

INPUT: Polynomial z where,
$z \coloneqq (z^{(i)}[0], \ldots, z^{(i)}[255]) = z[0] + z[1]X + \cdots + z[255]X^{255}$
OUTPUT: Polynomial shares $(z^{(1)}, \ldots, z^{(n)})$ of z
i.e. $\forall i \in [n]$, $z^{(i)} = z^{(i)}[0] + z^{(i)}[1]X + \cdots + z^{(i)}[255]X^{255}$

1: **for** $i = 0$ to 255 **do** ▷ Obtain shares of coef's of z
2: $\quad (z^{(1)}[i], \ldots, z^{(n)}[i]) \coloneqq \mathsf{TSS.IntShare}(z[i])$ ▷ Shares and coef's are in same set
3: **for** $i = 1$ to n **do** ▷ Construct polynomial shares
4: $\quad z^{(i)} \coloneqq (z^{(i)}[0], \ldots, z^{(i)}[255])$ ▷ $z^{(i)} = z^{(i)}[0] + z^{(i)}[1]X + \cdots + z^{(i)}[255]X^{255}$
5: **return** $(z^{(1)}, \ldots, z^{(n)})$

Algorithm 2 TSS.PolyReconstruct$((z^{(1)}, \ldots, z^{(n)}))$

INPUT: Polynomial shares $(z^{(1)}, \ldots, z^{(n)})$
OUTPUT: Polynomial $z = z[0] + z[1]X + \cdots + z[255]X^{255}$

1: **for** $i = 0$ to 255 **do**
2: $\quad z[i] \coloneqq$ TSS.IntReconstruct$((z^{(1)}[i], \ldots, z^{(n)}[i]))$
3: **return** z

Furthermore, we define Algorithm 3 and Algorithm 4 to generate secret shares of a vector that is used in Dilithium operations and to reverse the process, respectively.

Algorithm 3 TSS.VectorShare$(\mathbf{z})$

INPUT: Polynomial vector $\mathbf{z} = (z_1, \ldots, z_\ell)$
OUTPUT: Vector Shares $(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)})$ of $\mathbf{z}$
where $\mathbf{z} = \sum_{i \in [n]} \mathbf{z}^{(i)} \cdot \ell_i(0)$

1: **for** $i = 1$ to ℓ **do** ▷ Obtain shares of polynomials of $\mathbf{z}$
2: $\quad (z_i^{(1)}, \ldots, z_i^{(n)}) \coloneqq$ TSS.PolyShare(z_i)
3: **for** $i = 1$ to n **do**
4: $\quad \mathbf{z}^{(i)} \coloneqq (z_1^{(i)}, \ldots, z_\ell^{(i)})$ ▷ Construct vector shares
5: **return** $(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)})$

Algorithm 4 TSS.VectorReconstruct$((\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}))$

INPUT: Vector Shares $(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)})$ of $\mathbf{z}$
where $\mathbf{z} = \sum_{i \in [n]} \mathbf{z}^{(i)} \cdot \ell_i(0)$
OUTPUT: Polynomial Vector $\mathbf{z} = (z_1, \ldots, z_\ell)$

1: **for** $i = 1$ to ℓ **do**
2: $\quad z_i \coloneqq$ TSS.PolyReconstruct$((z_i^{(1)}, \ldots, z_i^{(n)}))$
3: **return** $\mathbf{z} = (z_1, \ldots, z_\ell)$

3.2 Supportive Algorithms for VT-Dilithium

In this section, we define auxiliary algorithms for VT-Dilithium. We also show how to construct n-out-of-n Threshold Secret Sharing on Dilithium and produce VT-Dilithium's keys and signatures. We also give an algorithm for verification of VT-Dilithium scheme which can verify each shared signature.

Secret Key. While generating the main Dilithium signature σ for the message M, the main secret key $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$ is used. In VT-Dilithium al-

gorithm, it is sufficient to use only $\mathbf{s}_1$ and $\mathbf{s}_2$ vectors of the main secret key to obtain shared secret keys in a TSS method.

Algorithm 5 VT-Dilithium.SkShares$(\mathbf{s}_1, \mathbf{s}_2)$

INPUT: Secret Key Vectors $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^\ell \times S_\eta^k$
OUTPUT: sk.Shares $= ((\mathbf{s}_1^{(1)}, \mathbf{s}_2^{(1)}), \ldots, (\mathbf{s}_1^{(n)}, \mathbf{s}_2^{(n)}))$ where $\mathbf{s}_1$, $\mathbf{s}_2$ can be reconstructed by sk.Shares and for $i \in [n]$ $(\mathbf{s}_1^{(i)}, \mathbf{s}_2^{(i)}) \in S_\eta^\ell \times S_\eta^k$

1: $(\mathbf{s}_1^{(1)}, \ldots, \mathbf{s}_1^{(n)}) \in S_\eta^\ell \coloneqq$ TSS.VectorShare$(\mathbf{s}_1)$
2: $(\mathbf{s}_2^{(1)}, \ldots, \mathbf{s}_2^{(n)}) \in S_\eta^k \coloneqq$ TSS.VectorShare$(\mathbf{s}_2)$
3: **return** sk.Shares $= ((\mathbf{s}_1^{(1)}, \mathbf{s}_2^{(1)}), \ldots, (\mathbf{s}_1^{(n)}, \mathbf{s}_2^{(n)}))$

The main reason for using an n-out-of-n Shamir secret sharing scheme is that valid signatures also have to be produced by the sk.Shares. Thus, all of the coefficients of polynomials in sk.Shares have to be in the range $[-\eta, \eta]$, which causes interpolating secret polynomials to yield results within this interval. We can address this situation by employing n-out-of-n Threshold Secret Sharing. If we had attempted to handle it with a specific t-out-of-n threshold, it would have been quite challenging for some points in the Shamir secret polynomial to fall within the interval. Therefore, the entire system utilizes an n-out-of-n Threshold Secret Sharing scheme.

Public Key. Let $pk = (\rho, \mathbf{t}_1)$ be the public key of Dilithium. Algorithm 6 returns the shares of pk, and Algorithm 7 reconstructs pk from pk.Shares.

Algorithm 6 VT-Dilithium.PkShares$(\rho,$ sk.Shares$)$

INPUT: First tuple of pk, ρ
INPUT: sk.Shares $= ((\mathbf{s}_1^{(1)}, \mathbf{s}_2^{(1)}), \ldots, (\mathbf{s}_1^{(n)}, \mathbf{s}_2^{(n)}))$
OUTPUT: pk.Shares $= (pk_1 = (\rho, \mathbf{t}^{(1)}), ..., pk_n = (\rho, \mathbf{t}^{(n)}))$ where $\mathbf{t} = \sum_{i \in [n]} \mathbf{t}^{(i)} \cdot \ell_i(0)$

1: $\mathbf{A} \in R_q^{k \times \ell} \coloneqq$ ExpandA(ρ)
2: $(\mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(n)}) \coloneqq (\mathbf{A}\mathbf{s}_1^{(1)} + \mathbf{s}_2^{(1)}, \ldots, \mathbf{A}\mathbf{s}_1^{(n)} + \mathbf{s}_2^{(n)})$
3: **return** pk.Shares $= ((\rho, \mathbf{t}^{(1)}), \ldots, (\rho, \mathbf{t}^{(n)}))$

As noted in Algorithm 6, after obtaining the shared $\mathbf{t}$ values, we did not run the Power2Round$_q$ function for each of them. We could assume that the shared $\mathbf{t}^{(i)}$ values would yield the shared $\mathbf{t}_1^{(i)}$ values after passing through the Power2Round$_q$ function. However, the TSS relation between the shared $\mathbf{t}^{(i)}$ values is disrupted after the rounding operation performed by the Power2Round$_q$ function. Therefore, it is not possible to reconstruct the main public key component $\mathbf{t}_1$ from the $\mathbf{t}_1^{(i)}$ values. Instead, we placed all the shared $\mathbf{t}$ values into the shared public keys and performed the rounding during the reconstruction

Algorithm 7 VT-Dilithium.PkReconstruct(pk.Shares)

INPUT: $\mathsf{pk.Shares} = ((\rho, \mathbf{t}^{(1)}), \ldots, (\rho, \mathbf{t}^{(n)}))$
OUTPUT: $pk = (\rho, \mathbf{t}_1)$
1: $\mathbf{t} \coloneqq \mathsf{TSS.VectorReconstruct}(\mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(n)})$
2: $(\mathbf{t}_1, \mathbf{t}_0) \coloneqq \mathsf{Power2Round}_q(\mathbf{t}, d)$
3: **return** $pk = (\rho, \mathbf{t}_1)$

process. The drawback of this approach is that the size of each shared public key increased compared to main public key.

When the shared $\mathbf{t}^{(i)}$ values are included in the shared public keys, the other output of the $\mathsf{Power2Round}_q$ function, $\mathbf{t}_0^{(i)}$ values, are also naturally obtained. Although the main secret key in Dilithium includes the $\mathbf{t}_0$ parameter, the designers have indicated that $\mathbf{t}_0$ is not actually a secret parameter and can be considered a public parameter [10]. The only reason for its inclusion in the secret key is to speed up the signing process. Therefore, there is no security issue with having the shared $\mathbf{t}_0^{(i)}$ (in $\mathbf{t}^{(i)}$) values included in the shared public keys in VT-Dilithium.

Signature. Let σ be the Dilithium signature where $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$. Algorithm 8 outputs the shares of σ in order to obtain verification and force open phase of VT-Dilithium algorithm. Algorithm 9 reconstructs σ from sign.Shares.

Algorithm 8 VT-Dilithium.SignShares($\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$, pk.Shares, $\mathbf{y}$, sk.Shares, M)

INPUT: Main Signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$
INPUT: $\mathsf{pk.Shares} = ((\rho, \mathbf{t}^{(1)}), \ldots, (\rho, \mathbf{t}^{(n)}))$
INPUT: Final Mask Value $\mathbf{y} \in \tilde{S}_{\gamma_1}^{\ell}$ of σ.
INPUT: $\mathsf{sk.Shares} = ((\mathbf{s}_1^{(1)}, \mathbf{s}_2^{(1)}), \ldots, (\mathbf{s}_1^{(n)}, \mathbf{s}_2^{(n)}))$
INPUT: Message M
OUTPUT: $\mathsf{sign.Shares} = (\sigma_1 = (\tilde{c}, \tilde{c}_1, \mathbf{z}^{(1)}, \mathbf{h}_1), \ldots, \sigma_n = (\tilde{c}, \tilde{c}_n, \mathbf{z}^{(n)}, \mathbf{h}_n))$
1: $\mathbf{A} \in R_q^{k \times \ell} \coloneqq \mathsf{ExpandA}(\rho)$
2: $\forall i \in [n], (\mathbf{t}_1^{(i)}, \mathbf{t}_0^{(i)}) \coloneqq \mathsf{Power2Round}_q(\mathbf{t}^{(i)}, d)$
3: $\forall i \in [n], \mu_i \in \{0,1\}^{512} \coloneqq \mathrm{H}(\mathrm{H}(\rho \| \mathbf{t}_1^{(i)}) \| M)$
4: **while** *bool* $= \perp$ **do**
5: $\quad (\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n)}) \in \tilde{S}_{\gamma_1}^{\ell} \coloneqq \mathsf{TSS.VectorShare}(\mathbf{y})$
6: $\quad c \in B_\tau \coloneqq \mathsf{SampleInBall}(\tilde{c})$
7: $\quad (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}) \coloneqq (\mathbf{A}\mathbf{y}^{(1)}, \ldots, \mathbf{A}\mathbf{y}^{(n)})$
8: $\quad \forall i \in [n], \tilde{c}_i \coloneqq \mathrm{H}(\mu_i \| \mathsf{HighBits}_q(\mathbf{w}^{(i)}, 2\gamma_2))$
9: $\quad (\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}) \coloneqq (\mathbf{y}^{(1)} + c\mathbf{s}_1^{(1)}, \ldots, \mathbf{y}^{(n)} + c\mathbf{s}_1^{(n)})$
10: $\quad \forall i \in [n], \mathbf{h}_i \coloneqq \mathsf{MakeHint}_q(-c\mathbf{t}_0^{(i)}, \mathbf{w}^{(i)} - c\mathbf{s}_2^{(i)} + c\mathbf{t}_0^{(i)}, 2\gamma_2)$
11: $\quad$ **if** $\|\mathbf{z}^{(i)}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{w}^{(i)} - c\mathbf{s}_2^{(i)}\|_\infty \geq \gamma_2 - \beta$ or $\|c\mathbf{t}_0^{(i)}\|_\infty \geq \gamma_2$ or # of 1's in $\mathbf{h}_i$ is greater than ω **then** *bool* $\coloneqq \perp$
12: **return** $\mathsf{sign.Shares} = (\sigma_1, \ldots, \sigma_n)$

Algorithm 9 VT-Dilithium.SignReconstruct(sign.Shares, $\mathbf{h}$)

INPUT: sign.Shares $= (\sigma_1 = (\tilde{c}, \tilde{c}_1, \mathbf{z}^{(1)}, \mathbf{h}_1), \ldots, \sigma_n = (\tilde{c}, \tilde{c}_n, \mathbf{z}^{(n)}, \mathbf{h}_n))$
OUTPUT: $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$
1: $\mathbf{z} \coloneqq$ TSS.VectorReconstruct$(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)})$
2: **return** $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

The VT-Dilithium.SignShares algorithm takes as input the mask value $\mathbf{y}$ used in the signature operation of the main Dilithium signature (produced at the end of the *while loop*). The reason for usage of this parameter is that the $\mathbf{y}$ is the only one that can be used to bind all shared signatures to the main signature according to the TSS manner. The operations performed on $\mathbf{y}$ allow for the derivation of the shared signature parameters $\mathbf{z}^{(i)}$, and with the VT-Dilithium.SignReconstruct algorithm, the $\mathbf{z}$ vector of the main signature can be obtained.

In the VT-Dilithium.SignShares algorithm, similar to the Dilithium, there is an abort operation derived from the Fiat-Shamir with Aborts method. For the shared signatures to be obtained as a result of the *while condition*, the conditions in the if statement must be satisfied. If these conditions are not met, the *while loop* restarts. Since the TSS.VectorShare($\mathbf{y}$) operation generates fresh random $\mathbf{y}^{(i)}$ values, other parameters also change. This process continues until the *while condition* is satisfied. This process continues until the *while condition* is satisfied. The completion of the *while condition* depends on the number of iterations determined by the parameters defined in Dilithium's signature process and the n threshold in TSS. Therefore, the signing time should be considered when selecting the value of n.

Verification In the verification step of VT-Dilithium, some of the chosen shared signatures i.e. σ_i where $i \in I$ should be verified by their corresponding shared public keys i.e. pk_i. Algorithm 10 shows how we construct the verification process for VT-Dilithium. It differs slightly from the original Dilithium's verification method.

The main difference of VT-Dilithium.VerifySign from verification function of Dilithium is that it takes the entire $\mathbf{t}$ as the part of public key input instead of just $\mathbf{t}_1$. Therefore, the algorithm requires running the Power2Round$_q$ function once. Another difference is that the shared signature includes the $\tilde{c}$ from the main signature and the $\tilde{c}_i$ parameters associated with the shared signature are taken as input. However, using these parameters together does not require an additional function call.

3.3 VT-Dilithium Scheme

In this section, we present the construction of the VT-Dilithium in Figure 1.

The VT-Dilithium is assumed to be secure if the privacy and soundness properties are satisfied. We propose the following theorems to show that privacy and soundness properties of VT-Dilithium is satisfied. We utilize the definitions

Algorithm 10 VT-Dilithium.VerifySign$(pk_i = (\rho, \mathbf{t}^{(i)}), M, \sigma_i = (\tilde{c}, \tilde{c}_i, \mathbf{z}^{(i)}, \mathbf{h}_i))$

INPUT: i'th shared public key $pk_i = (\rho, \mathbf{t}^{(i)})$
INPUT: Message M
INPUT: i'th shared signature $\sigma_i = (\tilde{c}, \tilde{c}_i, \mathbf{z}^{(i)}, \mathbf{h}_i)$
OUTPUT: Accept or Reject

1: $\mathbf{A} \in R_q^{k \times \ell} \coloneqq \mathsf{ExpandA}(\rho)$
2: $(\mathbf{t}_1^{(i)}, \mathbf{t}_0^{(i)}) \coloneqq \mathsf{Power2Round}_q(\mathbf{t}^{(i)}, d)$
3: $\mu_i \in \{0,1\}^{512} \coloneqq \mathrm{H}(\mathrm{H}(\rho \| \mathbf{t}_1^{(i)}) \| M)$
4: $c \in B_\tau \coloneqq \mathsf{SampleInBall}(\tilde{c})$
5: $\mathsf{HighBits}_q(\mathbf{w}^{(i)'}, 2\gamma_2) \coloneqq \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{A}\mathbf{z}^{(i)} - c\mathbf{t}_1^{(i)} \cdot 2^d, 2\gamma_2)$
6: **return** $[\![\|\mathbf{z}^{(i)}\|_\infty < \gamma_1 - \beta]\!]$ and $[\![\tilde{c}_i = \mathrm{H}(\mu_i \| \mathsf{HighBits}_q(\mathbf{w}^{(i)'}, 2\gamma_2))]\!]$ and $[\![\#$ of 1's in $\mathbf{h}_i$ is $\leq \omega]\!]$

and assumptions for Theorem 1 from [16, Definition 3], and for Theorem 2 from [16, Definition 2]. See Appendix A for their proofs.

Theorem 1 (Privacy). *Assuming that our NIZK and TLP systems rely on the assumptions provided in [16]. VT-Dilithium satisfies privacy as described in [16, Definition 3] in the random oracle model.*

Theorem 2 (Soundness). *Assuming that our NIZK and TLP systems rely on the assumptions provided in [16]. VT-Dilithium satisfies soundness as described in [16, Definition 2] in the random oracle model.*

4 Proof of Concept

In this section, we will outline the correctness of the algorithms necessary for defining the VT-Dilithium scheme.

4.1 Correctness of VT-Dilithium

Correctness of Algorithm 5. Let $(\mathbf{s}_1, \mathbf{s}_2)$ is the part of secret key of Dilithium where $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^\ell \times S_\eta^k$. Sample for all $i \in [n]$, $(\mathbf{s}_1^{(i)}, \mathbf{s}_2^{(i)}) \in S_\eta^\ell \times S_\eta^k$ where,

$$\begin{aligned}(\mathbf{s}_1, \mathbf{s}_2) &= (\mathbf{s}_1^{(1)} \cdot \ell_1(0), \mathbf{s}_2^{(1)} \cdot \ell_1(0)) + \cdots + (\mathbf{s}_1^{(n)} \cdot \ell_n(0), \mathbf{s}_2^{(n)} \cdot \ell_n(0)) \\ &= (\sum_{\forall i \in [n]} \mathbf{s}_1^{(i)} \cdot \ell_i(0), \sum_{\forall i \in [n]} \mathbf{s}_2^{(i)} \cdot \ell_i(0))\end{aligned}$$

and $\ell_i(\cdot)$ is the i-th Lagrange interpolation basis.

Algorithm 11 VT-Dilithium.Setup$(\lambda, \mathbf{T})$

INPUT: Time $\mathbf{T}$
INPUT: Security Parameter λ
OUTPUT: Common Reference String

1: $crs_{range} \coloneqq \mathsf{ZKsetup}(1^\lambda)$
2: $pp \coloneqq \mathsf{TLP.PSetup}(1^\lambda, \mathbf{T})$
3: **return** $crs \coloneqq (crs_{range}, pp)$

Algorithm 12 VT-Dilithium.CommitAndProof(crs, σ)

INPUT: $crs \coloneqq (crs_{range}, pp)$
INPUT: Main Signature σ
OUTPUT: Commitment C and Range Proof π

1: $\mathsf{sk.Shares} \coloneqq \mathsf{VT\text{-}Dilithium.SkShares}(\mathbf{s}_1, \mathbf{s}_2)$
2: $\mathsf{pk.Shares} \coloneqq \mathsf{VT\text{-}Dilithium.PkShares}(pk, \mathsf{sk.Shares})$
3: $\mathsf{sign.Shares} \coloneqq \mathsf{VT\text{-}Dilithium.SignShares}(\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}), \mathsf{pk.Shares}, \mathbf{y}, \mathsf{sk.Shares}, M)$
4: $\forall i \in [n]$, $r_i \leftarrow \{0,1\}^\lambda$ ▷ Random Sampling
5: $\forall i \in [n]$, $Z_i \coloneqq \mathsf{TLP.PGen}(pp, \sigma_i; r_i)$
6: $\forall i \in [n]$, $\pi_{range,i} \coloneqq \mathsf{ZKprove}(crs_{range}, (Z_i, 0, 2^\lambda, \mathbf{T}), (\sigma_i, r_i))$
7: $I \coloneqq H'(pk, (pk_1, Z_1, \pi_{range,1}), ..., (pk_n, Z_n, \pi_{range,n}))$
8: **return** $C \coloneqq (Z_1, ..., Z_n, \mathbf{T})$ and $\pi \coloneqq (\{pk_i, \pi_{range,i}\}_{i\in[n]}, I, \{\sigma_i, r_i\}_{i\in I})$

Algorithm 13 VT-Dilithium.Vrfy(crs, pk, M, C, π)

INPUT: Commitment $C \coloneqq (Z_1, ..., Z_n, \mathbf{T})$
INPUT: $\pi \coloneqq (\{pk_i, \pi_{range,i}\}_{i\in[n]}, I, \{\sigma_i, r_i\}_{i\in I})$
INPUT: Common Reference String $crs \coloneqq (crs_{range}, pp)$
INPUT: Message M, Main Public Key pk
OUTPUT: Accept or Reject

1: **if** $\mathsf{VT\text{-}Dilithium.PkReconstruct}(\mathsf{pk.Shares}) \neq pk$ where $\exists j \notin I$ such that $pk_j \notin \mathsf{pk.Shares}$ **then return** Reject
2: **else if** $\mathsf{ZKverify}(crs_{range}, (Z_i, 0, 2^\lambda, \mathbf{T}), \pi_{range,i}) \neq 1$ such that $\exists i \in [n]$ **then return** Reject
3: **else if** $\exists i \in I$ such that $Z_i \neq \mathsf{TLP.PGen}(pp, \sigma_i; r_i)$ or $\mathsf{VT\text{-}Dilithium.VerifySign}(pk_i, M, \sigma_i) \neq 1$ **then return** Reject
4: **else if** $I \neq H'(pk, (pk_1, Z_1, \pi_{range,1}), ..., (pk_n, Z_n, \pi_{range,n}))$ **then return** Reject
5: **else return** Accept

Algorithm 14 VT-Dilithium.ForceOpen(C)

INPUT: Commitment $C \coloneqq (Z_1, ..., Z_n, \mathbf{T})$
OUTPUT: Main Signature σ

1: $\forall i \in [n]$, $\sigma_i \in \mathsf{sign.Shares} \coloneqq \mathsf{TLP.PSolve}(pp, Z_i)$
2: **return** $\sigma \coloneqq \mathsf{VT\text{-}Dilithium.SignReconstruct}(\mathsf{sign.Shares}, \tilde{c}, \mathbf{h})$

Fig. 1: VT-Dilithium Scheme

Correctness of Algorithm 6 and 7. Algorithm 6 and Algorithm 7 show that pk can be separated and obtained in terms of Threshold Secret Sharing manner respectively. Correctness of these algorithms is given below. Let public Key $pk = (\rho, \mathbf{t}_1)$, $\mathbf{t}$ can be defined as

$$\begin{aligned}\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 &= \mathbf{A}\sum_{\forall i\in[n]} \mathbf{s}_1^{(i)} \cdot \ell_i(0) + \sum_{\forall i\in[n]} \mathbf{s}_2^{(i)} \cdot \ell_i(0) \\ &= (\mathbf{A}\mathbf{s}_1^{(1)} + \mathbf{s}_2^{(1)}) \cdot \ell_1(0) + \cdots + (\mathbf{A}\mathbf{s}_1^{(n)} + \mathbf{s}_2^{(n)}) \cdot \ell_n(0),\end{aligned}$$

where $\mathbf{A}$ is the matrix of the Dilithium. Then $\mathbf{t}$ can be written as

$$\mathbf{t} = \mathbf{t}^{(1)} \cdot \ell_1(0) + \cdots + \mathbf{t}^{(n)} \cdot \ell_n(0) = \sum_{\forall i\in[n]} \mathbf{t}^{(i)} \cdot \ell_i(0).$$

After this calculation, $\mathbf{t}_1$ can be reconstructed by

$$\mathsf{Power2Round}_q(\sum_{\forall i\in[n]} \mathbf{t}^{(i)} \cdot \ell_i(0), d) = (\mathbf{t}_1, \mathbf{t}_0).$$

For the verification part of VT-Dilithium, there is no need to separate the parameter ρ in pk.

Correctness of Algorithm 8 and 9. Algorithm 8 and Algorithm 9 shows that σ can be separated and obtained in terms of Threshold Secret Sharing manner respectively. Correctness of these algorithms as follows: Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ is the Dilithium signature. σ can be seperated to shares as follows. Let $\mathbf{y} \in \tilde{S}_{\gamma_1}^{\ell}$ is the mask of σ. $\mathbf{y}$ can be defined as follows:

$$\mathbf{y} = \sum_{i\in[n]} \mathbf{y}^{(i)} \cdot \ell_i(0)$$

When we multiply both sides of the equation by the matrix $\mathbf{A}$, we obtain the vector $\mathbf{w}$ as shown below:

$$\begin{aligned}\mathbf{w} = \mathbf{A}\mathbf{y} &= \mathbf{A}\sum_{\forall i\in[n]} \mathbf{y}^{(i)} \cdot \ell_i(0) = \sum_{\forall i\in[n]} \mathbf{A}\mathbf{y}^{(i)} \cdot \ell_i(0) \\ &= \sum_{\forall i\in[n]} \mathbf{w}^{(i)} \cdot \ell_i(0), \text{where } \mathbf{A} \in R^{k\times\ell}.\end{aligned}$$

Shares of $\mathbf{z}$ can be defined as follows:

$$\mathbf{z} = \mathbf{y} + c\mathbf{s}_1 = \sum_{\forall i\in[n]} \mathbf{y}^{(i)} \cdot \ell_i(0) + c\sum_{\forall i\in[n]} \mathbf{s}_1^{(i)} \cdot \ell_i(0) = \sum_{\forall i\in[n]} \mathbf{z}^{(i)} \cdot \ell_i(0).$$

The shares of the hint vector $\mathbf{h}_i$ for all $i \in [n]$ can be determined as follows:

$$\mathbf{h}_i \coloneqq \mathsf{MakeHint}_q(-c\mathbf{t}_0^{(i)}, \mathbf{w}^{(i)} - c\mathbf{s}_2^{(i)} + c\mathbf{t}_0^{(i)}, 2\gamma_2).$$

Consequently, the signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ can be verifiable and extractable from shares for all $i \in [n]$

$$\sigma_i = (\tilde{c}, \tilde{c}_i, \mathbf{z}^{(i)}, \mathbf{h}_i),$$

where $\|\mathbf{z}^{(i)}\|_\infty < \gamma_1 - \beta$ and $\|\mathsf{LowBits}_q(\mathbf{w}^{(i)} - c\mathbf{s}_2^{(i)}, 2\gamma_2)\|_\infty < \gamma_2 - \beta$ and $\|c\mathbf{t}_0^{(i)}\|_\infty < \gamma_2$ and # of 1's in $\mathbf{h}$ is less than or equals to ω.

Since signature verification will fail without $\mathbf{z}$, it is sufficient to only share $\mathbf{z}$ for reconstructing the signature. Thus, verifier can't obtain σ only from $\tilde{c}$ and $\mathbf{h}$. In addition, for all $i \in [n]$ parameters $\tilde{c}_i, \mathbf{h}_i$ will be used for verification of shared signatures σ_i.

Correctness of Algorithm 10. In order to show correctness of Algorithm 10, we utilize Lemma 1 and Lemma 2 which are given in [10].

For the proof of verification, we have to show that

$$[\![\mathsf{HighBits}_q(\mathbf{w}^{(i)'}, 2\gamma_2) = \mathsf{HighBits}_q(\mathbf{w}^{(i)}, 2\gamma_2)]\!]$$

in order to obtain $\tilde{c}_i$ both VT-Dilithium.SignShares and VT-Dilithium.VerifySign.

By using $\mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{A}\mathbf{z}^{(i)} - c\mathbf{t}_1^{(i)} \cdot 2^d, 2\gamma_2)$ from Algorithm 10, we can show following equality:

$$\begin{aligned}
&\mathsf{HighBits}_q(\mathbf{w}^{(i)'}, 2\gamma_2) = \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{A}\mathbf{z}^{(i)} - c\mathbf{t}_1^{(i)} \cdot 2^d, 2\gamma_2) \\
&= \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{A} \cdot (\mathbf{y}^{(i)} + c\mathbf{s}_1^{(i)}) - c \cdot (\mathbf{t}^{(i)} - \mathbf{t}_0^{(i)}), 2\gamma_2) \\
&\quad \text{where } \mathbf{z}^{(i)} = \mathbf{y}^{(i)} + c\mathbf{s}_1^{(i)} \text{ and } \mathbf{t}_1^{(i)} \cdot 2^d = \mathbf{t}^{(i)} - \mathbf{t}_0^{(i)} \\
&= \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{A}\mathbf{y}^{(i)} + \mathbf{A}c\mathbf{s}_1^{(i)} - c\mathbf{t}^{(i)} + c\mathbf{t}_0^{(i)}, 2\gamma_2) \\
&= \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{w}^{(i)} + \mathbf{A}c\mathbf{s}_1^{(i)} - c\mathbf{t}^{(i)} + c\mathbf{t}_0^{(i)}, 2\gamma_2) \text{ where } \mathbf{A}\mathbf{y}^{(i)} = \mathbf{w}^{(i)} \\
&= \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{w}^{(i)} + \mathbf{A}c\mathbf{s}_1^{(i)} - c \cdot (\mathbf{A}\mathbf{s}_1^{(i)} + \mathbf{s}_2^{(i)}) + c\mathbf{t}_0^{(i)}, 2\gamma_2) \\
&\quad \text{where } \mathbf{t}^{(i)} = \mathbf{A}\mathbf{s}_1^{(i)} + \mathbf{s}_2^{(i)} \\
&= \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{w}^{(i)} + \mathbf{A}c\mathbf{s}_1^{(i)} - c\mathbf{A}\mathbf{s}_1^{(i)} - c\mathbf{s}_2^{(i)} + c\mathbf{t}_0^{(i)}, 2\gamma_2) \\
&= \mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{w}^{(i)} - c\mathbf{s}_2^{(i)} + c\mathbf{t}_0^{(i)}, 2\gamma_2)
\end{aligned}$$

A valid signature satisfies these inequalities, $\|c\mathbf{t}_0^{(i)}\|_\infty < \gamma_2$ and $\|\mathbf{z}^{(i)}\|_\infty < \gamma_1 - \beta$. Thus, from Lemma 1 and Lemma 2 in [10], we have

$$\mathsf{UseHint}_q(\mathbf{h}_i, \mathbf{w}^{(i)} - c\mathbf{s}_2^{(i)} + c\mathbf{t}_0^{(i)}, 2\gamma_2) = \mathsf{HighBits}_q(\mathbf{w}^{(i)}, 2\gamma_2).$$

4.2 Threshold Secret Sharing over Dilithium Parameters

We are utilizing Threshold Secret Sharing over Dilithium's column vectors in VT-Dilithium scheme. For instance, secret column vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ in Algorithm 5, public key parameter $\mathbf{t}$ in Algorithm 6 and parameters of signature process $\mathbf{y}, \mathbf{w}$

and $\mathbf{z}$ in Algorithm 8 can be separated to their shares. We introduce supporting algorithms, Algorithm 1, 2, 3 and 4, for splitting Dilithium vectors or polynomials into their shares and reconstructing them again. We demonstrate how this can be achieved through an example.

Let $\mathbf{z} \in R_q^l$ be a Dilithium column vector. Following equation is satisfied with finding appropriate shares

$$\mathbf{z} = \sum_{i \in [n]} \mathbf{z}^{(i)} \cdot \ell_i(0).$$

Let $z_1, \ldots, z_\ell$ are polynomials in $\mathbf{z} \in R_q^\ell$. We can obtain the shares of vectors with Algorithm 3 ,i.e., $(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)}) \coloneqq \mathsf{TSS.VectorShare}(\mathbf{z})$.

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_\ell \end{bmatrix} = \begin{bmatrix} z_1^{(1)} \cdot \ell_1(0) + \cdots + z_1^{(n)} \cdot \ell_n(0) \\ z_2^{(1)} \cdot \ell_1(0) + \cdots + z_2^{(n)} \cdot \ell_n(0) \\ \vdots \\ z_\ell^{(1)} \cdot \ell_1(0) + \cdots + z_\ell^{(n)} \cdot \ell_n(0) \end{bmatrix} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ \vdots \\ z_\ell^{(1)} \end{bmatrix} \cdot \ell_1(0) + \cdots + \begin{bmatrix} z_1^{(n)} \\ z_2^{(n)} \\ \vdots \\ z_\ell^{(n)} \end{bmatrix} \cdot \ell_n(0)$$

$$= \mathbf{z}^{(1)} \cdot \ell_1(0) + \cdots + \mathbf{z}^{(n)} \cdot \ell_n(0)$$

Contrary, the equality in the reverse direction, i.e., $\mathbf{z} \coloneqq \mathsf{TSS.VectorReconstruct}((\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(n)})$, is also valid. To illustrate that the equality holds, it is necessary to show that the polynomials in $\mathbf{z}$ can also be separated to sharing polynomials, i.e., Algorithm 1 is also correct.

Let $(z_i^{(1)}, \ldots, z_i^{(n)}) \coloneqq \mathsf{TSS.PolyShare}(z_i)$ where $\forall i \in [\ell], z_i \in \mathbf{z}$ i.e. $z_i = z_i^{(1)} \cdot \ell_1(0) + \cdots + z_i^{(n)} \cdot \ell_n(0)$.

$$z_i[0] + \cdots + z_i[255]X^{255} = (z_i^{(1)}[0] + \cdots + z_i^{(1)}[255]X^{255}) \cdot \ell_1(0) + \cdots + (z_i^{(n)}[0] + \cdots + z_i^{(n)}[255]X^{255}) \cdot \ell_n(0)$$

This implies, all of polynomials can be divided into their shares coefficient-wise only.

$$
\begin{aligned}
&X^0 \to z_i[0] = z_i^{(1)}[0] \cdot \ell_1(0) + \cdots + z_i^{(n)}[0] \cdot \ell_n(0) \\
&\text{where } (z_i^{(1)}[0], \ldots, z_i^{(n)}[0]) = \mathsf{TSS.share}(z_i[0]) \\
&X^1 \to z_i[1] = z_i^{(1)}[1] \cdot \ell_1(0) + \cdots + z_i^{(n)}[1] \cdot \ell_n(0) \\
&\text{where } (z_i^{(1)}[1], \ldots, z_i^{(n)}[1]) = \mathsf{TSS.share}(z_i[1]) \\
&\qquad\qquad \vdots \\
&X^{255} \to z_i[255] = z_i^{(1)}[255] \cdot \ell_1(0) + \cdots + z_i^{(n)}[255] \cdot \ell_n(0) \\
&\text{where } (z_i^{(1)}[255], \ldots, z_i^{(n)}[255]) = \mathsf{TSS.IntShare}(z_i[255])
\end{aligned}
$$

5 Conclusion And Future Work

In this work, we introduced VT-Dilithium which is a Verifiable Timed Signature scheme based on CRYSTALS-Dilithium Digital Signature Algorithm. We successfully integrated the Dilithium into a VTS scheme by ensuring that each parameter, including the secret key, public key, and signature, can be divided into shared components and reconstructed using Shamir's Threshold Secret Sharing algorithm. Furthermore, we demonstrated that for any given index, the shared secret key can generate the corresponding shared public key and shared signature at the same index. Despite the complexities posed by Dilithium's intricate mathematical operations, we were able to overcome these challenges and achieve the desired functionality. In future work, we aim to make VT-Dilithium more efficient by optimizing the system's performance. Moreover, we plan to increase the flexibility of our system by transitioning from the current n-out-of-n TSS method to a more versatile t-out-of-n scheme, where t can be less than n. This will allow for more adaptable and versatile implementations. Additionally, we aim to provide quantum secure NIZK and TLP on VT-Dilithium to obtain fully quantum secure VTS scheme.

References

1. Module-lattice-based digital signature standard. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD (2023). https://doi.org/10.6028/NIST.FIPS.204.ipd
2. Bai, S., Galbraith, S.D.: An improved compression technique for signatures based on learning with errors. In: Topics in Cryptology–CT-RSA 2014: The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings. pp. 28–47. Springer (2014)
3. Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM Journal on Computing **20**(6), 1084–1118 (1991)
4. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)
5. Dworkin, M.J.: Sha-3 standard: Permutation-based hash and extendable-output functions (2015)
6. Fischlin, M., Mittelbach, A.: An overview of the hybrid argument. Cryptology ePrint Archive (2021)
7. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**, 36–63 (2001)
8. Katz, J.: Digital signatures, vol. 1. Springer (2010)
9. Lyubashevsky, V.: Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 598–616. Springer (2009)
10. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: Crystals-dilithium. Algorithm Specifications and Supporting Documentation (Version 3.1) (2021)
11. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Annual International Cryptology Conference. pp. 620–649. Springer (2019)
12. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
13. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**, 161–174 (1991)
14. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
15. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)
16. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable timed signatures made practical. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1733–1750 (2020)

A Proof of Theorems

We give the formal proofs of VT-Dilithium Scheme.

A.1 Proof of Theorem 1

Proof. Let A is an adversary with a depth bounded by T^{ϵ}, where $0 \leq \epsilon < 1$, and T is the pre-defined time for the time-lock puzzles. We utilize the hybrid argument approach [6] to create a sequence of hybrids akin to the privacy proof in VTS Schemes [16].

Hybrid $\mathcal{H}_0$: This is the original execution.

Hybrid $\mathcal{H}_1$: This hybrid is essentially the same as the previous one, except that the random oracle is emulated using lazy sampling. Additionally, a random set I, where $|I| = n - 1$, is pre-sampled, and the output of the random oracle for the cut-and-choose instance is set to I^*.

Hybrid $\mathcal{H}_2$: A simulated crs_{range} is sampled and it is computationally indistinguishable because of the property of by $\mathsf{ZKsetup}$ function.

Hybrid $\mathcal{H}_3 \dots \mathcal{H}_{3+n}$: $\forall i \in [n]$, $\pi_{range,i}$ is calculated by underlying NIZK proof. In the Zero-Knowledge setup, the difference between all hybrids is negligible.

Hybrid $\mathcal{H}_{3+n+1} \dots \mathcal{H}_{3+2n-1}$: $\forall i \in [n-1]$, i'th puzzle in the complement of set I^* is calculated by $\mathsf{TLP.PGen}(pp, 0^{\lambda}; r_i)$ function. Because the distinguisher is limited in depth, the inability to distinguish follows from invoking the security of TLP.

Hybrid $\mathcal{H}_{3+2n}$: In this hybrid, let prover samples uniform shared secret keys

$$\mathsf{sk.Shares} \coloneqq \mathsf{VT\text{-}Dilithium.SkShares}(\mathbf{s}_1, \mathbf{s}_2)$$

and obtain shared public keys via

$$\mathsf{pk.Shares} \coloneqq \mathsf{VT\text{-}Dilithium.PkShares}(pk, \mathsf{sk.Shares}).$$

We can observe that for all $i \in I^*$ pk can be obtained only we know the rest public key part where $i \notin I^*$ via

$$pk \coloneqq \mathsf{VT\text{-}Dilithium.PkReconstruct}(\mathsf{pk.Shares}).$$

Simulator $\mathcal{S}$: The simulator is defined to match the characteristics of the previous hybrid. It's important to note that the proof computation by the simulator does not rely on any information regarding the witness. With this, we conclude our proof. □

A.2 Proof of Theorem 2

Proof. We examine the protocol in its interactive form, and the integrity of the non-interactive protocol is derived from the Fiat-Shamir transformation for protocols with constant rounds. Suppose A is an adversary capable of efficiently compromising the integrity of the protocol. This implies that, an adversary produces the puzzles $(Z_1, \ldots, Z_n)$ where for every $Z_i \notin I$ and $\mathsf{TLP.PSolve}(pp, Z_i) = \tilde{\sigma}_i$. These signatures $\tilde{\sigma}_i$ should not be verified i.e.

$$\mathsf{VT\text{-}Dilithium.Verify}(pk_i, M, \tilde{\sigma}_i) \neq 1.$$

Suppose the opposite were true, then we could obtain a legitimate signature on message m by interpolating $\tilde{\sigma}_i$ with $\{\sigma_i\}_{i \in I}$, which satisfy the given relation as defined by the verification algorithm. Additionally, note that all puzzles $(Z_1, \ldots, Z_n)$ are properly structured, meaning the solving algorithm consistently produces a well-defined value, except for negligible probabilities, as guaranteed by the soundness of the range NIZK. This implies that, given $(Z_1, \ldots, Z_n)$, we can efficiently recover some set I' by solving the puzzles and verifying which signatures satisfy the specified relation. For the verifier to approve $I' = I$ indicating that the prover accurately guesses a random n-bit string uniformly selected from the set of strings with precisely $\frac{n}{2}$-many 0's where this probability is $\frac{((n/2)!)^2}{n!}$. In the non-interactive version of our protocol, this assertion holds valid regardless of the quantity of simulated proofs, provided that the NIZK exhibits simulation-soundness. Thus, initiating with a simulation-sound NIZK ensures the simulation-soundness of our scheme too. □