UOV-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 primitives that enable the creation of a signature that can only be retrieved after a specific time delay, while also providing verifiable evidence of its existence. This framework is particularly useful in blockchain applications. Current VTS schemes rely on signature algorithms such as BLS, Schnorr, and ECDSA, which are vulnerable to quantum attacks due to the vulnerability of the discrete logarithm problem to Shor's Algorithm. We introduce VT-UOV, a novel VTS scheme based on the Salt-Unbalanced Oil and Vinegar (Salt-UOV) Digital Signature Algorithm. As a multivariate polynomial-based cryptographic primitive, Salt-UOV provides strong security against both classical and quantum adversaries. Adapting Salt-UOV into the VTS framework requires addressing challenges such as complex parameters instead of a integer, the computational complexity of solving multivariate equations, and the integration of Time-Lock Puzzles (TLPs) for enforcing delayed signature generation. Our experimental results show that VT-UOV exhibits a unique performance profile among existing VTS constructions. This paper offers a detailed exploration of the VT-UOV scheme and its overall security and performance properties.

Keywords: Verifiable Timed Signatures, UOV Signature Scheme, Threshold Secret Sharing, Post-Quantum Cryptography

1 Introduction

Verifiable Timed Signatures (VTS) [13] ensure that a digital signature remains inaccessible until a predefined time elapses. By embedding the signature in a time-lock puzzle, only after solving the puzzle can the signature be retrieved. Additionally, verifiability allows checking signature validity beforehand [8], preventing unnecessary computational effort.

Existing VTS schemes rely on BLS [3], Schnorr [10], and ECDSA [5], which are based on the discrete logarithm problem. However, Shor's Algorithm [12] renders these schemes insecure against quantum attacks.

The Unbalanced Oil and Vinegar (UOV) signature scheme [7] is a post-quantum algorithm based on the hardness of solving multivariate quadratic

equations over finite fields. It offers short signatures, fast verification, and strong security, making it a candidate in NIST's post-quantum standardization [1].

The current UOV design in NIST's process follows the Salt-UOV structure [9], ensuring security guarantees. In this work, we introduce VT-UOV, a VTS scheme integrating Salt-UOV. Unlike discrete logarithm-based schemes, UOV employs multivariate quadratic equations, leading to different computational requirements for key generation, signing, and verification.

Integrating UOV into a VTS system is more challenging due to post-quantum cryptographic complexities. UOV's secret keys, public keys, and signatures involve large structures, complicating their use in VTS. Efficient key and signature sharing, along with threshold verification, requires additional techniques.

A VTS system comprises Non-Interactive Zero-Knowledge (NIZK) proofs, Time-Lock Puzzles (TLPs), Range Proofs, Threshold Secret Sharing (TSS), and Digital Signature algorithms. To integrate a signature scheme, its keys and signatures must be shared via TSS, ensuring reconstructability. The process involves:

1. Secret key sharing via TSS, ensuring compliance with the scheme's constraints.
2. Deriving shared public keys, maintaining reconstructability.
3. Generating shared signatures from shared secret keys.
4. Verifying shared signatures against shared public keys.

This preserves VTS security while supporting different signature schemes. The paper is structured as follows: Section 2 outlines cryptographic structures for VT-UOV. Section 3 details VT-UOV, its sub-algorithms, correctness, and performance analysis. Section 5 concludes with future directions.

2 PRELIMINARIES

The list of parameters used in our work are provided in Table 1.

2.1 Cryptographic Primitives

VTS schemes are structures formed by the combination of multiple cryptographic components. Therefore, in order to define a VTS scheme, certain cryptographic structures must be determined. In this section, the cryptographic primitives of VTS schemes are presented.

Digital Signatures Digital signatures [6] are mathematical constructs designed to verify the authenticity and integrity of messages or documents. They involve creating a unique signature that is appended to the message by the sender. Using a secret key, a digital signature is generated, and it can only be verified with the corresponding public key. Digital signatures ensure that the message has not been altered during transit and originates from the claimed sender.

Table 1: List of Parameters

$\mathbb{F}_q$	Finite Field order q
$\mathbb{F}_q^n$	n-dimensional vector space over $\mathbb{F}_q$
t	Threshold Value of TSS
n	Whole Shares of TSS
$[n]$	Set of Containing Integers $\{1, \ldots, n\}$
o	Number of Oil Values
v	Number of Vinegar Values
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 $w_j \in \mathbf{w}$
pp	Public Parameter
$\mathbf{T}$	A Specific Time For Time-Lock Puzzles
λ	Security Parameter
pk	Public Key of Main Signature
sk	Secret Key of Main Signature
σ	Main Signature
$pk_i = \mathcal{P}^{(i)}$	Shared Public Key of σ_i
$(\mathcal{F}^{(i)}, \mathcal{T})$	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
$\mathcal{T}$	Linear transformation in UOV
$\mathcal{F}$	UOV polynomials defining the secret key

Time-lock Puzzles (TLP) TLPs [8] are cryptographic constructs that secure data or messages such that access is only possible after a specific time has elapsed. These puzzles rely on computational challenges, like factoring large integers or solving complex mathematical problems, to implement the delay mechanism. Once the set time has passed, the puzzle can be efficiently solved, revealing the protected information. For VT-UOV, we define a general description of time-lock puzzles.

Definition 1. *The following TLP functions are incorporated into the VT-UOV construction:*

- $pp \coloneqq \mathsf{TLP.PuzzleSetup}(1^\lambda, \mathbf{T})$*: Generates public parameters pp based on security parameter λ and desired time $\boldsymbol{T}$.*
- $Z_i \coloneqq \mathsf{TLP.PuzzleGen}(pp, \sigma_i; r_i)$*: Produces puzzles Z_i using public parameters pp, shared signatures σ_i, and random coins r_i.*
- $\sigma_i \coloneqq \mathsf{TLP.PuzzleSolve}(pp, Z_i)$*: Solves the puzzle Z_i to obtain the shared signature σ_i.*

Verifiable Timed Signatures (VTS) Verifiable timed signatures [13] allow a sender to commit to a signature that remains hidden until a designated time

T has elapsed. This commitment includes proof validating the signature against the public key, which can be verified independently. Privacy ensures that no signature information leaks before **T**, while soundness prevents adversaries from forging valid proofs for invalid commitments.

VTS employs the cut-and-choose protocol to verify a signature within a time-lock puzzle before solving it, avoiding computational waste on invalid signatures. The signature is split into shares, each inside a time-lock puzzle. The verifier randomly selects some shares for immediate disclosure and verification. If these pass, the signature is considered valid. Time-lock puzzles enable mathematical operations ensuring correctness of the remaining shares. This approach reduces overhead, prevents manipulation, and enhances efficiency in VTS systems.

Below, formal definition of Verifiable Timed Signatures is given.

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 [2] are cryptographic tools that allow a prover to demonstrate the validity of a statement without requiring direct interaction with the verifier. This is especially beneficial in scenarios where direct communication is unfeasible. NIZKPs are used in VTS schemes to generate valid proofs for signatures embedded in puzzles.

Definition 3. *A NIZKP comprises the following algorithms:*

- $crs \coloneqq \mathsf{ZKsetup}(1^\lambda)$*: Produces a common reference string crs from the security parameter λ.*
- $\pi \coloneqq \mathsf{ZKprove}(crs, x, w)$*: Generates proof π for statement x using witness w.*
- $0/1 \coloneqq \mathsf{ZKverify}(crs, x, \pi)$*: Verifies proof π for statement x and returns 1 if valid, otherwise 0.*

Threshold Secret Sharing (TSS) TSS [11] securely distributes a secret among multiple participants such that it can only be reconstructed when a threshold number of shares are combined. This ensures no individual has full access to the secret, enhancing protection against unauthorized access.

Definition 4. *TSS involves the following algorithms:*

- $(s^{(1)}, \ldots, s^{(n)}) \coloneqq \mathsf{TSS.IntShare}(s)$*: Splits a secret* s *into* n *integer shares.*
- $s \coloneqq \mathsf{TSS.IntRecon}(s^{(1)}, \ldots, s^{(t)})$*: Combines* t *shares to reconstruct the secret* s*.*

The secret is embedded in the constant term of a polynomial $f(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$, where a_0 is the secret. Shares are points $(i, f(i)) = (x_i, y_i)$ for $i \in [n]$. Using Lagrange interpolation, the secret a_0 is reconstructed as:

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

In the VT-UOV scheme, we utilize a t-out-of-n TSS approach. This means that in our scheme, any t shares (where $t \le n$) are sufficient to reconstruct the secret.

Salt-UOV Scheme and Its Security Argument The Salt-UOV[9] scheme is a variant of the submitted UOV scheme to NIST standardization process introduced earlier. The primary difference lies in the signing algorithm, where Salt-UOV incorporates a random "salt" value to enhance security. During the signing process, a random vinegar vector $v \in \mathbb{F}_q^v$ is selected, followed by uniformly and independently chosen multiple salt values. The system of linear equations $\mathcal{F}\left(\begin{bmatrix} v \\ u \end{bmatrix}\right) = \text{Hash}(\mu \| \text{salt})$ is solved until a valid solution is found. The full details of the scheme are provided in Figure 1.

3 UOV-BASED VERIFIABLE TIMED SIGNATURE SCHEME

In this chapter, we define our VTS scheme, VT-UOV, based on the Unbalanced Oil and Vinegar (UOV) signature scheme. The general structure of VT-UOV is presented in Figure 2. To construct VT-UOV, the first step is to demonstrate that the polynomials used in the key generation, signing, and verification processes within UOV can also be adapted to Threshold Secret Sharing (TSS). Once this is established, we proceed by splitting the secret key components (i.e., the vinegar and oil variables) of the main signature σ into shares and generating the corresponding shared public keys and signatures. The details of these steps are elaborated in the following sections.

3.1 Threshold Secret Sharing for Salt-UOV's Parameters

To divide the parameters of VT-UOV into shares using the TSS algorithm, it is essential first to define a method for splitting an array or a vector into shares and then reconstructing them accurately. Throughout this paper, we employ a t-out-of-n TSS scheme where $t \le n$, meaning that at least t of the generated

Fig. 1: Salt-UOV Signature Algorithm

Salt-UOV Signature Processes

KeyGen(params = $(o+v, o, q, \mathsf{salt_len})$)

Choose OV-polynomials $f_1(x_1, \ldots, x_{o+v}), \ldots, f_o(x_1, \ldots, x_{o+v})$ uniformly at random
$\mathcal{F} := (f_1, \ldots, f_o)$
Choose an invertible linear transformation $\mathcal{T} : \mathbb{F}_q^{o+v} \to \mathbb{F}_q^{o+v}$ uniformly at random
$\mathcal{P} := \mathcal{F} \circ \mathcal{T}$
pk := $\mathcal{P}$
sk := $(\mathcal{F}, \mathcal{T})$
return (pk, sk)

Sign(params, sk = $(\mathcal{F}, \mathcal{T}), \mu \in \{0,1\}^*$)

$v \leftarrow \mathbb{F}_q^v$
Repeat
  salt $\leftarrow \{0,1\}^{\mathrm{salt_len}}$
  $t \leftarrow \mathrm{Hash}(\mu \| \mathrm{salt})$
  $\Delta_k := \left\{ \begin{bmatrix} v \\ u \end{bmatrix} \in \mathbb{F}_q^{o+v} \;\middle|\; \mathcal{F}\left(\begin{bmatrix} v \\ u \end{bmatrix} \right) = k \right\}$
until $w \leftarrow \Delta_k$
$s := \mathcal{T}^{-1}(w)$
$\sigma := (s, \mathrm{salt})$
return σ

Verify(params, pk = $\mathcal{P}, (\mu, \sigma = (s, \mathsf{salt}))$)

$k \leftarrow \mathrm{Hash}(\mu \| \mathrm{salt})$
$k' := \mathcal{P}(s)$
return $(k == k')$

shares are required to successfully reconstruct the original value. This ensures both flexibility and security in the secret-sharing process.

In the proposed scheme, we define Algorithm 1, which takes a array used in VT-UOV as input and outputs its shares. Algorithm 2 is then used to reconstruct the original array from its shares.

Additionally, we define Algorithm 3 and Algorithm 4 to generate secret shares of a vector used in VT-UOV operations and to reconstruct the vector from its shares.

Algorithm 1 VT-UOV.TSS.ArrayShare

INPUT: Coefficients $f[i]$ of f for all $i \in [o \cdot (o+v)]$ where $f \in \mathbb{F}_q^{o \cdot (o+v)}$
OUTPUT: Shares $(f^{(1)}, \ldots, f^{(n)})$ of f where
$f^{(i)} \in \mathbb{F}_q^{o \cdot (o+v)}$

for $i = 0$ to $o \cdot (o+v)$ **do**
    $(f^{(1)}[i], \ldots, f^{(n)}[i]) \leftarrow \mathsf{TSS.IntShare}(f[i])$ ▷ Shared coefficients of f
for $i = 1$ to n **do**
    $f^{(i)} \leftarrow (f^{(i)}[0], \ldots, f^{(i)}[o \cdot (o+v) - 1])$ ▷ Construct array shares
return $(f^{(1)}, \ldots, f^{(n)})$

Algorithm 2 VT-UOV.TSS.ArrayRecon

INPUT: Chosen t-threshold arrays from $(f^{(1)}, \ldots, f^{(n)})$
OUTPUT: $f \in \mathbb{F}_q^{o \cdot (o+v)}$

for $i = 0$ to $o \cdot (o+v)$ **do**
    $f[i] \leftarrow \mathsf{TSS.IntRecon}(f^{(1)}[i], \ldots, f^{(t)}[i])$
return f

3.2 Supportive Algorithms for VT-UOV

In this section, we introduce the supporting algorithms for the VT-UOV scheme. These algorithms explain how to set up a t-out-of-n Threshold Secret Sharing mechanism, where $t \leq n$, to securely divide and reconstruct the secret components of the scheme. We also describe the process for generating keys and signatures in VT-UOV, ensuring compatibility with the Verifiable Timed Signature (VTS) framework. Additionally, we provide an algorithm for verifying VT-UOV signatures, which allows for the validation of each shared signature and ensures the correctness of the overall reconstructed signature. These algorithms form the foundation for the secure and efficient operation of VT-UOV.

Key Generation of VT-UOV To define the VT-UOV scheme, it is essential to first split the secret key values into their respective shares using the Threshold Secret Sharing (TSS) method. The main secret keys will be divided into shared secret keys, which will then be used for cut-and-choose zero-knowledge verification. These shared secret keys are subsequently utilized to generate shared public keys and shared signatures. It is important to note that the shared secret keys are not the parameters sent to the verifier. Instead, only the shared public keys and shared signatures are intended for sharing and external verification. This structure ensures that the VT-UOV scheme maintains both security and verifiability during its operations. The key generation process of the VT-UOV scheme is defined in Algorithm 5.

The pk.Shares parameters generated by Algorithm 5 must allow the reconstruction of the main public key pk in a TSS manner using at least the threshold value $t \leq n$. This reconstruction process is detailed in Algorithm 6.

Algorithm 3 VT-UOV.TSS.VectorShare

INPUT: Vector of Arrays $\mathbf{f} = (f_1, \ldots, f_\ell)$
OUTPUT: Vector shares $(\mathbf{f}^{(1)}, \ldots, \mathbf{f}^{(n)})$ of $\mathbf{f}$

1: **for** $i = 1$ to ℓ **do**
2: $(f_i^{(1)}, \ldots, f_i^{(n)}) \leftarrow$ VT-UOV.TSS.ArrayShare(f_i) ▷ Share each array in the vector
3: **for** $i = 1$ to n **do**
4: $\mathbf{f}^{(i)} \leftarrow (f_1^{(i)}, \ldots, f_\ell^{(i)})$ ▷ Construct vector shares
5: **return** $(\mathbf{f}^{(1)}, \ldots, \mathbf{f}^{(n)})$

Algorithm 4 VT-UOV.TSS.VectorRecon

INPUT: Chosen t-threshold vector shares from $(\mathbf{f}^{(1)}, \ldots, \mathbf{f}^{(n)})$
OUTPUT: Vector $\mathbf{f} = (f_1, \ldots, f_\ell)$.

1: **for** $i = 1$ to ℓ **do**
2: $f_i \leftarrow$ TSS.PolyRecon$(f_i^{(1)}, \ldots, f_i^{(t)})$
3: **return** $\mathbf{f} = (f_1, \ldots, f_\ell)$

Signature Process of VT-UOV Algorithm 7 takes the message μ, sk.Shares, and the main signature σ as input and outputs the shared signatures sign.Shares. In this process, the salt value from the main signature is divided into shares using the TSS method, ensuring that each shared signature is generated using the corresponding shared secret key and the shared salt value at the same index. The resulting s values, along with the shared salt values, form the shared signatures.

The values embedded into the TLP are the shared salt values. A verifier attempting to validate a shared signature against the corresponding shared public key at the same index cannot obtain the signature without first extracting the salt value from the TLP. Therefore, embedding the salt into the TLP ensures the proper functioning of the structure and guarantees the integrity of the VT-UOV signing mechanism.

Algorithm 8 is used in the reconstruction phase of the main signature in the VT-UOV scheme. In this process, knowing the s value of the main signature alone is not sufficient, as the salt value remains unknown. In our scheme, the main signature's salt value is reconstructed from the shared salt values, which are embedded within the TLP.

Therefore, in the force open phase, only the shared salt values need to be extracted from the TLP exactly at time $\mathbf{T}$ to recover the main signature's salt value. If at least the threshold number of shared salt values is obtained, the main salt value can be reconstructed, ultimately allowing access to the main signature.

Verification Process of VT-UOV The VT-UOV.SignVerify algorithm is used to verify a shared signature against the corresponding shared public key at the same index along with the message.

Algorithm 5 VT-UOV.KeyGen

INPUT: Main Secret Key sk $= (\mathcal{F} \coloneqq \mathbf{f} = (f_1, \ldots, f_{o+v}), \mathcal{T})$ OUTPUT: Shared Public Keys pk.Shares, Shared Secret Keys sk.Shares

1: $(\mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(n)}) \coloneqq \mathsf{VT\text{-}UOV.TSS.VectorShare}(\mathcal{F})$
2: **for** $i = 1$ to n **do**
3: $\quad pk_i = \mathcal{P}^{(i)} \leftarrow \mathcal{F}^{(i)} \circ \mathcal{T}$
4: pk.Shares $\leftarrow (\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(n)})$
5: sk.Shares $\leftarrow ((\mathcal{F}^{(1)}, \mathcal{T}), \ldots, (\mathcal{F}^{(n)}, \mathcal{T}))$
6: **return** (pk.Shares, sk.Shares)

Algorithm 6 VT-UOV.PkRecon

INPUT: A subset of pk.Shares, $(\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(t)})$ where $t \leq n$ OUTPUT: Main Public Key pk

1: $pk = \mathcal{P} \leftarrow \mathsf{VT\text{-}UOV.TSS.VectorRecon}(\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(t)})$
2: **return** pk

Algorithm 9 VT-UOV.SignVerify Algorithm

INPUT: Message μ, i'th Shared Public Key $\mathcal{P}^{(i)}$ INPUT: i'th Shared Signature $\sigma_i = (s_i, \text{salt}^{(i)})$
OUTPUT: Boolean value indicating signature validity

1: $k_i \leftarrow \text{Hash}(\mu \| \text{salt}^{(i)})$
2: $k_i' \leftarrow \mathcal{P}^{(i)}(s_i)$
3: **return** $(k_i == k_i')$

3.3 VT-UOV Scheme

In this section, we present the construction of VT-UOV, as illustrated in Figure 2.

3.4 Correctness of VT-UOV

In this section, we will outline the correctness of the algorithms required for the construction of the VT-UOV scheme. The correctness of these sub-algorithms is critical to ensure the overall security, functionality, and reliability of the VT-UOV system. Each algorithm plays a vital role in different stages of the scheme, from key generation to signing, verification, and reconstruction. Demonstrating their correctness not only validates the integrity of individual components but also ensures that the VT-UOV scheme as a whole operates as intended.

Correcntess of Algorithm 5 and Algorithm 6 Let $sk = (\mathcal{F}, \mathcal{T})$ is the secret key of salt-UOV where $\mathcal{F}$ consist of the coefficient arrays and $\mathcal{T}$ is assumed to be linear transformation matrix. For all $i \in [n]$, $\mathcal{F}^{(i)}$ where,

Fig. 2: VT-UOV Scheme

Algorithm 10 VT-UOV.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 11 VT-UOV.CommitAndProof(crs, σ)

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

1: $(\mathsf{sk.Shares}, \mathsf{pk.Shares}) \leftarrow \mathsf{VT\text{-}UOV.KeyGen}(\mathcal{F})$
2: $\mathsf{sign.Shares} \leftarrow \mathsf{VT\text{-}UOV.SignShares}(\sigma = (s, \text{salt}), \mathsf{sk.Shares}, \text{Message } \mu)$
3: $\forall i \in [n]$, $r_i \leftarrow \{0,1\}^\lambda$ ▷ Random Sampling
4: $\forall i \in [n]$, $Z_i \coloneqq \mathsf{TLP.PGen}(pp, \sigma_i; r_i)$
5: $\forall i \in [n]$, $\pi_{range,i} \coloneqq \mathsf{ZKprove}(crs_{range}, (Z_i, 0, 2^\lambda, \mathbf{T}), (\sigma_i, r_i))$
6: $I \coloneqq H'(pk, (pk_1, Z_1, \pi_{range,1}), ..., (pk_n, Z_n, \pi_{range,n}))$ where $H' : \{0,1\}^* \to I \subset [n]$ of size $|I| = t - 1$
7: **return** $\pi \coloneqq (\{pk_i, \pi_{range,i}\}_{i\in[n]}, I, \{\sigma_i, r_i\}_{i\in I})$ and $C \coloneqq (Z_1, ..., Z_n, \mathbf{T})$

Algorithm 12 VT-UOV.Verify(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** $\mathsf{VT\text{-}UOV.PkRecon}(\mathsf{pk.Shares}) \neq pk$ 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 $\mathsf{VT\text{-}UOV.SignVerify}(pk_i, M, \sigma_i) \neq 1$ **then return** Reject
4: **else if** $I \neq H'(pk, (pk_1, Z_1, \pi_{range,1}), \ldots, (pk_n, Z_n, \pi_{range,n}))$ **then return** Reject
5: **else return** Accept

Algorithm 13 VT-UOV.ForceOpen(C)

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

1: $\forall i \in [t]$, $\sigma_i \in \mathsf{sign.Shares} \coloneqq \mathsf{TLP.PSolve}(pp, Z_i)$
2: **return** $\sigma \coloneqq \mathsf{VT\text{-}UOV.SignRecon}(\mathsf{sign.Shares}, s)$

Algorithm 7 VT-UOV.Sign

INPUT: sk.Shares, Message μ, Main Signature $\sigma := (s, \text{salt})$ OUTPUT: sign.Shares

$v \leftarrow \mathbb{F}_q^v$
repeat
  $(\text{salt}^{(1)}, \ldots, \text{salt}^{(n)}) \leftarrow \mathsf{TSS.IntShare}(\text{salt})$
  for $i = 1$ to n **do**
    $k_i \leftarrow \text{Hash}(\mu \| \text{salt}^{(i)})$
    $\Delta_{t_i} \leftarrow \begin{bmatrix} v \\ u \end{bmatrix} \in \mathbb{F}_q^{o+v}$
    where $\mathcal{F}^{(i)}\left(\begin{bmatrix} v \\ u \end{bmatrix}\right) = k_i$
until $w_i \leftarrow \Delta_{k_i} \mid \forall i \in [n]$
for $i = 1$ to n **do**
  $s_i \leftarrow \mathcal{T}^{-1}(w_i)$
  $\sigma_i \leftarrow (s_i, \text{salt}^{(i)})$
sign.Shares $\coloneqq (\sigma_1, \ldots, \sigma_n)$
return sign.Shares

Algorithm 8 VT-UOV.SignRecon

INPUT: A subset of sign.Shares, $(\sigma_1, \ldots, \sigma_t)$ where $t \leq n$, First tuple of σ, s
OUTPUT: Main Signature σ

salt $\leftarrow \mathsf{TSS.IntRecon}(\text{salt}^{(1)}, \ldots, \text{salt}^{(t)})$
$\sigma \leftarrow (s, \text{salt})$
return σ

$$\mathcal{F} = \mathcal{F}^{(1)}.\ell_1(0) + ... + \mathcal{F}^{(n)}.\ell_n(0) = \sum_{\forall i \in [n]} \mathcal{F}^{(i)}.\ell_i(0)$$

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

After demonstrating that the value $\mathcal{F}$ can be split using the TSS method, we will show how the shared public keys $\mathcal{F}^{(i)}$ are obtained and how these values are used to reconstruct the main public key $\mathcal{P}$. Here, it is sufficient to demonstrate that when the convolution operation between $\mathcal{F}$ and $\mathcal{T}$ is applied to the shared values $\mathcal{F}^{(i)}$, the equality holds due to the properties of convolution.

$$\begin{aligned}
\mathcal{F} \circ \mathcal{T} &= (\mathcal{F}^{(1)}.\ell_1(0) + ... + \mathcal{F}^{(n)}.\ell_n(0)) \circ \mathcal{T} \\
&= (\mathcal{F}^{(1)}.\ell_1(0)) \circ \mathcal{T} + ... + (\mathcal{F}^{(n)}.\ell_n(0)) \circ \mathcal{T} \\
&= (\mathcal{F}^{(1)} \circ \mathcal{T}).\ell_1(0) + ... + (\mathcal{F}^{(n)} \circ \mathcal{T}).\ell_n(0) \\
&= \mathcal{P}^{(1)}.\ell_1(0) + ... + \mathcal{P}^{(n)}.\ell_n(0) \\
&= \sum_{\forall i \in [n]} \mathcal{P}^{(i)}.\ell_i(0) = \mathcal{P}
\end{aligned}$$

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

The correctness of the equation follows directly from key properties of convolution, ensuring that the threshold secret sharing (TSS) reconstruction correctly retrieves the main public key $\mathcal{P}$. First, the linearity (distributivity over summation) of convolution guarantees that the convolution of a sum is equal to the sum of convolutions, allowing the shared values $\mathcal{F}^{(i)}$ to be processed independently. Next, the associativity property enables reordering of scalar multiplication and convolution, ensuring that applying Lagrange coefficients $\ell_i(0)$ before or after convolution yields the same result. Additionally, the commutativity of convolution allows the operation order to be switched, meaning that convolving $\mathcal{F}^{(i)}$ with $\mathcal{T}$ produces the same result as convolving $\mathcal{T}$ with $\mathcal{F}^{(i)}$, preserving consistency across all shared components. Finally, the summation property of TSS reconstruction ensures that the individual convolved shares $\mathcal{P}^{(i)}$ sum correctly to reconstruct the original public key $\mathcal{P}$. Together, these properties establish the correctness of the equation and validate that the VT-UOV scheme maintains structural integrity while allowing secure reconstruction of the main public key.

Correcntess of Algorithm 8 The VT-UOV.SignRecon algorithm ensures the correct reconstruction of the main signature $\sigma = (s, \text{salt})$ by leveraging the threshold properties of the TSS method. During the VT-UOV.Sign process, the salt value salt is split into shares using the TSS sharing function, ensuring that each $\sigma_i = (s_i, \text{salt}^{(i)})$ contains a share of the salt. In VT-UOV.SignRecon, the main salt value is reconstructed using at least t valid shares:

$$\text{salt} \leftarrow \mathsf{TSS.IntRecon}(\text{salt}^{(1)}, \ldots, \text{salt}^{(t)})$$

where

$$\text{salt} = \sum_{\forall i \in [t]} \text{salt}^{(i)}.\ell_i(0)$$

Since TSS is a linear threshold scheme, if at least t correct shares are provided, the original salt value is uniquely reconstructed. The signature is then reconstructed as:

$$\sigma = (s, \text{salt})$$

Correcntess of Algorithm 7 and Algorithm 9 To ensure the correctness of the VT-UOV.SignVerify algorithm, we must show that a shared signature $\sigma_i = (s_i, \text{salt}^{(i)})$ generated using the shared secret key $\mathcal{F}^{(i)}$ can be correctly verified using the corresponding shared public key $\mathcal{P}^{(i)}$.

During signing, the hash value t_i is computed as:

$$k_i = \text{Hash}(\mu | \text{salt}^{(i)})$$

and the value s_i is obtained as:

$$s_i = (\mathcal{F}^{(i)-1} \circ \mathcal{T}^{-1})(k_i)$$

Since the shared public key is:

$$\mathcal{P}^{(i)} = \mathcal{F}^{(i)} \circ \mathcal{T},$$

the verification step computes:

$$k_i' = (\mathcal{F}^{(i)} \circ \mathcal{T})(s_i).$$

By substituting s_i and applying the identity transformation property

$$k_i' = k_i.$$

Thus, the VT-UOV.SignVerify algorithm correctly validates the signature by ensuring that the shared public key $\mathcal{P}^{(i)}$ reconstructs the same hash value used in signing, thereby verifying the integrity of the signed message.

3.5 Security Analysis of VT-UOV

The VT-UOV scheme is constructed by importing the UOV digital signature algorithm into the VTS framework defined in [13] for VT-BLS, VT-Schnorr, and VT-ECDSA. This implies that the security proofs established for these schemes remain valid even when the underlying signature scheme is replaced. Therefore, since VT-UOV references the security proof of a VTS scheme in [13], we assume that it also satisfies the privacy and soundness properties demonstrated in the security proof.

4 Performance Results

In this section, we present a performance comparison between the VT-BLS[13] algorithm and the quantum-resistant signature schemes used in the VT-Dilithium[**?**] and VT-UOV algorithms.

4.1 Performance Analysis of VT-UOV

In order to conduct a performance analysis of the VT-UOV scheme, it is sufficient to examine the computational complexity of the supportive algorithms belonging to VT-UOV and the complexity of the VT-UOV.TSS functions used within these algorithms.

Complexities of Threshold Secret Sharing Methods on VT-UOV To analyze the computational complexity of the threshold secret sharing method in VT-UOV, the fundamental functions TSS.IntShare and TSS.IntRecon will be taken as the basis. These functions constitute the smallest building blocks of the VT-UOV.TSS.ArrayShare, VT-UOV.TSS.VectorShare, VT-UOV.TSS.ArrayRecon, and VT-UOV.TSS.VectorRecon functions. Therefore, the number of times TSS.IntShare or TSS.IntRecon is used by each of these functions will be determined. Afterwards, by evaluating the implementation results of TSS.IntShare and TSS.IntRecon, the computational complexity of the remaining VT-UOV.TSS functions can be obtained.

Table 2: Computational Complexities of VT-UOV.TSS Functions

Function	**Complexity Description**
TSS.IntShare	Single integer sharing operation
TSS.IntRecon	Single integer reconstruction operation
VT-UOV.TSS.ArrayShare	$o + v$ TSS.IntShare operations
VT-UOV.TSS.ArrayRecon	$o + v$ TSS.IntRecon operations
VT-UOV.TSS.VectorShare	$o \cdot (o + v)$ TSS.IntShare operations
VT-UOV.TSS.VectorRecon	$o \cdot (o + v)$ TSS.IntRecon operations

Complexity of the VT-UOV.TSS.ArrayShare and VT-UOV.TSS.ArrayRecon In the VT-UOV.TSS.ArrayShare function, all integer values that are used in UOV are divided into integer shares using the TSS.IntShare function. Since the polynomials in UOV requires a total of $o + v$ coefficients, each of these values must be shared individually via TSS.IntShare. The other operations in the function are negligible in terms of computational cost. Therefore, the total number of TSS.IntShare operations performed in the VT-UOV.TSS.ArrayShare function is exactly $o + v$.

In the VT-UOV.TSS.ArrayRecon algorithm, the original integers are reconstructed from the shared integers that belong to an array. This reconstruction is carried out using the TSS.IntRecon function. Since the array contains $o \cdot (o + v)$ shared integers, the TSS.IntRecon function is executed exactly $o \cdot (o + v)$ times within this function.

Complexity of the VT-UOV.TSS.VectorShare and VT-UOV.TSS.VectorRecon In the VT-UOV.TSS.VectorShare function, the VT-UOV.TSS.ArrayShare function is executed once for each array contained in the input vector. Since the other operations in this function are negligible in terms of computational cost, the overall complexity is determined solely by the number of arrays. Therefore, the number of VT-UOV.TSS.ArrayShare executions equals the number of arrays in the vector.

In the VT-UOV.TSS.VectorRecon function, the original arrays are reconstructed from their corresponding shared arrays. Consequently, the VT-UOV.TSS.ArrayRecon function is called once for each array. As the other operations are negligible, the total number of VT-UOV.TSS.ArrayRecon executions is equal to the number of arrays in the shared vector.

Table 3: Computational Complexities of VT-UOV Algorithms

Function	**Complexity Description**
VT-UOV.KeyGen	$1\times$ VT-UOV.TSS.VectorShare + $n\cdot$ UOV.Keygen
VT-UOV.PkRecon	$1\times$ VT-UOV.TSS.VectorRecon
VT-UOV.Sign	$1\times$ TSS.IntShare + $n\times$ UOV.Sign
VT-UOV.SignRecon	$1\times$ TSS.IntRecon
VT-UOV.SignVerify	$1\times$ UOV.Verify

Complexities of Supportive Algorithms of VT-UOV

VT-UOV.Keygen In the VT-UOV.KeyGen, a single VT-UOV.TSS.VectorShare function is executed to share the master secret key $\mathcal{F}$ among n participants, resulting in n shared secret keys denoted by $\mathcal{F}^{(i)}$ for $i \in [n]$. Therefore, this operation has a direct impact on the overall complexity of the key generation phase. Each of the resulting shared keys then undergoes a convolution with the public trapdoor $\mathcal{T}$, i.e., $\mathcal{F}^{(i)} \circ \mathcal{T}$ for all $i \in [n]$. As a result, the VT-UOV.KeyGen operation includes the complexity of one VT-UOV.TSS.VectorShare function and n convolution operations.

VT-UOV.PkRecon In the VT-UOV.PkRecon function, the only contributing factor to computational complexity is a single VT-UOV.TSS.VectorRecon$(\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(t)})$ operation.

VT-UOV.Sign In the VT-UOV.Sign operation, the repeat process continues until a suitable w_i is found that satisfies the secret key $\mathcal{F}$. In each iteration of this procedure, one TSS.IntShare function is executed along with n secret key solving operations. After the repeat process successfully terminates, n linear inverse operations are performed. Assuming that the repeat process completes after m iterations, the overall computational complexity of the VT-UOV.Sign operation includes m executions of TSS.IntShare, $m \cdot n$ secret key solving operations, and n linear inverse operations.

VT-UOV.SignRecon In the VT-UOV.SignRecon function, the main salt value is recovered, enabling the reconstruction of the original signature. This main

salt value can be obtained using a single TSS.IntRecon operation. Therefore, the total computational complexity of the VT-UOV.SignRecon function consists of only one TSS.IntRecon execution.

VT-UOV.SignVerify The VT-UOV.SignVerify function performs the same operations as the verification function of the underlying Salt-UOV scheme. Therefore, the computational complexity of VT-UOV.SignVerify is equivalent to that of the Salt-UOV verification function.

Signature Algorithms In our study, the performance evaluation is conducted based on the auxiliary algorithms and TSS structures employed within the VT schemes. Table 4 provides the cycle counts of the signature schemes used in the VT schemes.

Table 4: Clock Cycle Counts for Signature Schemes with 128-bit Security

Scheme	Operation	Average (cycles)
Dilithium	Keygen	480,940
	Sign	1,797,112
	Verify	502,804
UOV	Keygen	93,905,146
	Sign	1,683,266
	Verify	231,285
BLS	Keygen	335,206
	Sign	264,646
	Verify	4,534,321

Threshold Secret Sharing Operations In the TSS method, the VT-UOV.TSS.ArrayShare and VT-UOV.TSS.VectorShare functions are internally expressed in terms of TSS.IntShare, while the VT-UOV.TSS.ArrayRecon and VT-UOV.TSS.VectorRecon functions are based on TSS.IntRecon. Therefore, to estimate the total cycle counts of these higher-level functions, it is sufficient to analyze the performance of the TSS.IntShare and TSS.IntRecon primitives individually.

Based on our C implementation of these primitives, we have determined that the size of the prime number used for threshold operations does not significantly affect the cycle count performance of either TSS.IntShare or TSS.IntRecon. Table 5 presents the average cycle counts of these functions for various threshold values t with a fixed number of total participants $n = 20$.

It is observed that the performance of the TSS.IntShare function is influenced by both the total number of shares n and the threshold value t, as the sharing process involves operations dependent on these parameters. Similarly, the complexity of the TSS.IntRecon function increases proportionally with both t and n, since the number of shares used in the reconstruction and the underlying polynomial evaluations are directly affected by these values.

Table 5: Clock Cycle Performance of Shamir Secret Sharing at Various Thresholds (out of 20)

Threshold (t)	Function	Average (cycles)
10	TSS.IntShare	7,531
	TSS.IntRecon	11,165
15	TSS.IntShare	13,948
	TSS.IntRecon	23,517
19	TSS.IntShare	17,981
	TSS.IntRecon	30,375
20	TSS.IntShare	19,092
	TSS.IntRecon	35,223

4.2 Performance Comparison of VT-Schemes

The performance comparison was conducted for threshold values $t = 10$, 15, 19, and 20 in Table 6, 7, 8 and 9 respectively. Due to configuration limitations in the VT-Dilithium algorithm, performance evaluation could not be performed for threshold values $t = 10$ and $t = 15$. Therefore, for these two thresholds, only VT-UOV and VT-BLS algorithms were compared. For the threshold configurations with $t = 19$ and $t = 20$, all three algorithms—VT-Dilithium, VT-UOV, and VT-BLS—were evaluated and compared.

Table 6: Clock Cycle Results of VTS Algorithms $t = 10$

Scheme	Function	Average (cycles)
VT-UOV	KeyGen	1,898,876,185
	PkRecon	53,473,400
	Sign	33,662,760
	SignRecon	11,140
	SignVerify	232,944
VT-BLS	KeyGen	6,801,605
	PkRecon	14,369
	Sign	5,263,265
	SignRecon	12,736
	SignVerify	80,876,040

Our observations show that as the threshold value increases, the performance of functions involving TSS.IntRecon is significantly impacted. For example, in the VT-UOV.PkRecon function, 4928 invocations of TSS.IntShare are required to reconstruct the main public key. This number is 1024 for VT-Dilithium and only 1 for VT-BLS. Consequently, VT-UOV is the slowest among the three in public key reconstruction across all threshold settings. The same trend is observed in signature reconstruction functions. However, only a single call to TSS.IntRecon is sufficient to reconstruct a VT-UOV signature, just as it is for VT-BLS. In

Table 7: Clock Cycle Results of VTS Algorithms $t = 15$

Scheme	Function	Average (cycles)
VT-UOV	KeyGen	1,917,797,986
	PkRecon	107,383,691
	Sign	33,625,638
	SignRecon	22,207
	SignVerify	224,871
VT-BLS	KeyGen	6,781,443
	PkRecon	23,339
	Sign	6,063,263
	SignRecon	24,705
	SignVerify	82,836,120

Table 8: Clock Cycle Results of VTS Algorithms $t = 19$

Scheme	Function	Average (cycles)
VT-Dilithium	KeyGen	24,470,200
	PkRecon	32,556,428
	Sign	405,275,087
	SignRecon	31,511,108
	SignVerify	9,697,558
VT-UOV	KeyGen	1,872,352,235
	PkRecon	146,198,969
	Sign	32,335,166
	SignRecon	29,222
	SignVerify	225,976
VT-BLS	KeyGen	6,663,251
	PkRecon	23,622
	Sign	6,201,945
	SignRecon	26,389
	SignVerify	80,645,845

contrast, VT-Dilithium requires 1024 such operations. Therefore, VT-UOV is more efficient than VT-Dilithium in the signature reconstruction phase.

In the key generation phase, VT-UOV performs significantly slower than both VT-Dilithium and VT-BLS. This inefficiency is primarily due to the inherently expensive KeyGen process of the UOV algorithm, which is repeated n times in the VT-UOV construction. Although the VT-UOV.TSS.VectorShare function also contributes to the total cost, its impact is comparatively smaller. Based on our measurements, the VT-UOV.KeyGen function is approximately 81 times slower than that of VT-Dilithium and 309 times slower than that of VT-BLS.

In terms of signature generation, VT-BLS provides the most efficient performance. As seen in Table 4, the native Dilithium and UOV schemes exhibit comparable signing performance. However, in the VT setting, the VT-Dilithium.Sign function is approximately 13 times slower than VT-UOV.Sign. The primary rea-

Table 9: Clock Cycle Results of VTS Algorithms $t = 20$

Scheme	Function	Average (cycles)
VT-Dilithium	KeyGen	24,419,468
	PkRecon	36,275,341
	Sign	423,220,962
	SignRecon	35,304,808
	SignVerify	9,439,661
VT-UOV	KeyGen	1,998,863,856
	PkRecon	167,699,955
	Sign	32,642,961
	SignRecon	33,844
	SignVerify	222,274
VT-BLS	KeyGen	6,451,491
	PkRecon	31,866
	Sign	6,174,442
	SignRecon	29,123
	SignVerify	81,723,543

son for this discrepancy is that VT-Dilithium uses one instance of VT-UOV.TSS.VectorShare, while VT-UOV requires only a single invocation of TSS.IntShare.

Regarding signature verification, VT-BLS is the least efficient due to the high computational cost of the pairing operation. In contrast, VT-UOV achieves the fastest verification performance among all schemes. This is mainly due to the inherently lightweight verification process of the UOV algorithm.

5 CONCLUSION AND FUTURE WORK

In this work, we introduced VT-UOV, a Verifiable Timed Signature scheme based on the quantum-secure Salt-UOV Digital Signature Algorithm. We integrated UOV into a VTS framework by enabling secret key, public key, and signature sharing using Shamir's Threshold Secret Sharing. Despite UOV's large parameter sizes and complex equations, we successfully achieved the desired functionality. Future research will incorporate the NIST-standardized UOV scheme to analyze its impact on key and signature sizes, commitment sizes, and proof efficiency. Additionally, we will evaluate the computational performance of VT-UOV, comparing post-quantum multivariate cryptography with other VTS schemes to assess efficiency trade-offs.

References

1. Alagic, G., Bros, M., Ciadoux, P., Cooper, D., Dang, Q., Dang, T., Kelsey, J.M., Lichtinger, J., Miller, C.A., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., Waller, N., Liu, Y.K.: Status report on the first round of the additional digital signature schemes for the nist post-quantum cryptography

standardization process (2024-10-24 04:10:00 2024). https://doi.org/https://doi.org/10.6028/NIST.IR.8528, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=958723
2. Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM Journal on Computing **20**(6), 1084–1118 (1991)
3. 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)
4. Fischlin, M., Mittelbach, A.: An overview of the hybrid argument. Cryptology ePrint Archive (2021)
5. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**, 36–63 (2001)
6. Katz, J.: Digital signatures, vol. 1. Springer (2010)
7. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 206–222. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
8. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
9. Sakumoto, K., Shirai, T., Hiwatari, H.: On provable security of uov and hfe signature schemes against chosen-message attack. In: Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4. pp. 68–82. Springer (2011)
10. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**, 161–174 (1991)
11. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
12. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)
13. 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 Security Proofs

We give the formal security proof of VT-UOV scheme.

Theorem 1 (Privacy). *Assuming that our NIZK and TLP systems rely on the assumptions provided in [13]. VT-UOV satisfies privacy as described in [13, Definition 3] in the random oracle model.*

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 [4] to create a sequence of hybrids akin to the privacy proof in VTS Schemes [13].

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 $|I| = t-1$, 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 $\mathcal{H}_{3+n+1} \dots \mathcal{H}_{3+2n-t+1}$: $\forall i \in [n-1]$, 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 $\mathcal{H}_{3+2n-t+2}$: In this hybrid, let prover samples uniform shared secret keys and public keys

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

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

$$pk \coloneqq \mathsf{VT\text{-}UOV.PkRecon}(\mathsf{pk.Shares}).$$

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.

Theorem 2 (Soundness). *Assuming that our NIZK and TLP systems rely on the assumptions provided in [13]. VT-UOV satisfies soundness as described in [13, Definition 2] in the random oracle model.*

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 $(Z_1, \dots, Z_n)$ 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.

$$\mathsf{VT\text{-}UOV.SignVerify}(\mu, \mathcal{P}^{(i)}) \neq 1.$$

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 $(Z_1, \dots, Z_n)$ 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.