Squirrel: Efficient Synchronized Multi-Signatures from Lattices

Nils Fleischhacker\textsuperscript{1}, Mark Simkin\textsuperscript{2}, and Zhenfei Zhang\textsuperscript{2,⋆⋆}

\textsuperscript{1} Ruhr University Bochum
\textsuperscript{2} Ethereum Foundation

Abstract. The focus of this work are multi-signatures schemes in the synchronized setting. A multi-
signature scheme allows multiple signatures for the same message but from independent signers to be
compressed into one short aggregated signature, which allows verifying all of the signatures simultane-
ously. In the synchronized setting, the signing algorithm takes the current time step as an additional
input. It is assumed that no signer signs more than one message per time step and we aim to aggregate
signatures for the same message and same time step. This setting is particularly useful in the context
of blockchains, where validators are naturally synchronized by the blocks they sign.

We present Squirrel, a concretely efficient lattice-based multi-signature scheme in the synchronized
setting that works for a bounded number of $2^\tau$ time steps and allows for aggregating up to $\rho$ signatures
at each step, where both $\tau$ and $\rho$ are public parameters upon which the efficiency of our scheme depends.
Squirrel allows for non-interactive aggregation of independent signatures and is proven secure in the
random oracle model in the presence of rogue-key attacks assuming the hardness of the short integer
solution problem in a polynomial ring.

We provide a careful analysis of all parameters and show that Squirrel can be instantiated with good
concrete efficiency. For $\tau = 24$ and $\rho = 4096$, a signer could sign a new message every 10 seconds for 5
years non-stop. Assuming the signer has a cache of 112 MB, signing takes 68 ms and verification of an
aggregated signature takes 36 ms. The size of the public key is 1 KB, the size of an individual signature
is 52 KB, and the size of an aggregated signature is 771 KB.

1 Introduction

A multi-signature scheme [IN83, MOR01] allows for compressing multiple signatures for the same
message, generated under independent keys, into one short aggregated signature. Given the corre-
sponding public keys, the message, and the aggregated signature, anyone can verify the validity of
all signatures simultaneously.

Such signature schemes are particularly useful in the context of cryptocurrencies, where a set
of validators maintain a public append-only ledger. The ledger should only contain valid data and
minimizing the amount of data stored on the ledger is crucial for the overall efficiency of the
cryptocurrency. In regular time intervals, new candidate data blocks appear that may or may not
be added to the ledger. If a validator deems a data block eligible for addition to the ledger, they will
vouch for it by signing it. If enough validators have signed a specific data block, then it is added
to the ledger along with all the signatures vouching for it. In this setting multi-signatures allow for
storing less data on the ledger by replacing all individual signatures with the aggregated signature.

It has been shown that multi-signatures can be constructed from a variety of assumptions, such
as the RSA assumption [IN83, OO93], discrete logarithm assumptions [BDD+00, MOR01, BN06,
BCJ08, NRS21], and pairings-based assumptions [BGLS02, Bol03, LOS+06, BDN18, DGNW20].

\textsuperscript{1} mail@nilsfleischhacker.de. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Founda-
tion) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972.

\textsuperscript{2} mark.simkin@ethereum.org

\textsuperscript{⋆ ⋆ ⋆} zhenfei.zhang@ethereum.org
Unfortunately, all of the above assumptions are susceptible to quantum attacks [Sho94] and there has been little work in multi-signatures schemes that plausibly remain secure in the presence of a quantum adversary.

Two recent works [ES16, DOTT21] present multi-signatures schemes, whose security rely on the hardness of lattice assumptions which are currently considered hard even against quantum adversaries. Both of these works, however, have significant drawbacks that hinder their potential practical deployment. The scheme of El Bansarkhani and Sturm [ES16] is only proven secure in a setting where all signing keys, even the adversarial ones, are generated honestly. In reality, an adversary could attempt to perform a so-called rogue key attack, where maliciously formed keys are chosen depending on the honest keys, such that they can forge aggregated signatures that supposedly correspond to a set of keys consisting of both, honest and malicious keys. Further, both works require interaction between the independent signers to aggregate the signatures. In applications such as the one sketched above the signers may be online at different times and are potentially not even aware of who the others signers are, thus making interactive aggregation problematic. Ideally, aggregation of signatures should be non-interactive in the sense that it does not require further interaction between any of the signers and the aggregating entity. From a security perspective the aggregated signature should remain unforgeable even if malicious keys are included and aggregation is performed by the adversary.

1.1 Our Contribution

In this work, we focus on multi-signature schemes in the synchronized setting [GR06, AGH10, HW18, DGNW20]. Here, the signing algorithm is given an additional time step $t$ as input along with the message and the secret key. It is assumed that no signer produces more than one signature per time step. Rather than aiming to aggregate any set of signatures we aim to aggregate signatures by independent signers for the same message and same time step. Going back to our previous append-only ledger example, we observe that the validators are naturally synchronized and only aim to aggregate signatures for the same data block which can be associated with a time step $t$.

We present Squirrel$^3$, a concretely efficient lattice-based multi-signature scheme in the synchronized setting that works for a bounded number of $2^\tau$ time steps and allows for aggregating up to $\rho$ signatures at each step, where both $\tau$ and $\rho$ are public parameters upon which the efficiency of our scheme depends. Squirrel allows for non-interactive aggregation of signatures and is secure against rogue key attacks in the random oracle model assuming the hardness of the short integer solution problem in a polynomial ring.

It may seem that having an upper bound on the number of signatures is a severe restriction that limits the practical usefulness of our results. To see that this is not the case in many settings, we note that even with a $\tau$ as small as 24, a single signing key supports signing a new message every 10 seconds for 5 years non-stop.

Squirrel is both asymptotically and concretely efficient in most parameters as can be seen in Table 1. Keys and signature sizes are reasonably small and verification of an aggregated signature only takes a few tens of milliseconds. The main (theoretical) bottleneck of Squirrel is the asymptotic worst-case signing cost. Fortunately, our construction possesses several nice features that alleviate the asymptotic inefficiency in practice. Our construction is an online/offline signature scheme [EGM90] which means that the majority of the computational cost of the signing

---

3 Our construction, just like our rodent friends from the Sciuridae family, heavily rely on (binary) trees.
procedure can be preprocessed before the message to be signed is known. The amortized overall computational cost per signature is exponentially smaller than the worst-case cost, which means that signing is computationally cheap most of the time and only rarely requires a larger computational effort. Lastly, we show that the concrete computational worst-case costs of the signing procedure can be significantly reduced by storing somewhat larger secret keys. As an exemplary data point, one can see in Table 1 that a 2 GB secret key allows for signing times below 4 ms. We stress that storing such a “large” secret key with modern hardware does not pose a problem in the absolute majority of use-cases, for instance, where signers are blockchain validators with adequate resources. Verification of the aggregated signature, which is 771 KB large, only takes 36 ms.

A naive construction of a lattice-based multi-signature scheme with non-interactive aggregation is to simply append individual signatures of a plain lattice-based signature scheme. Such a scheme can, for instance, be instantiated with Dilithium signatures [DKL+ 18] or Falcon signatures [PFH+20]. When comparing such a solution to ours, for the parameters from above, the naive scheme with Dilithium requires roughly 3 KB per signature and 0.2 ms per verification. For 4096 aggregated signatures, the size would be around 12 MB and verification would take around 800 ms. In comparison, our solution requires 771 KB for the aggregated signature, reducing size by 94%, and verification is faster by a factor of 20. For Falcon signatures, we observe smaller gains, reducing 71% in size, and accelerating verification by a factor of 4. In terms of verification times, our result is also on-par with pre-quantum algorithms, since it takes roughly 200 ms to verify 4096 ECDSA signatures [lib] and 2 ms to verify 4096 BLS signatures [pai]. When there are more signatures to aggregate, our benchmark shows that our signature size scales sub-logarithmically with regard to $\rho$. We provide a detailed discussion of the concrete efficiency of our scheme in Section 6.

### 1.2 Limitations of our Work

Since Squirrel is proven secure under the assumed hardness of the short integer solution problem in a polynomial ring, it does not directly fall victim to attacks by a quantum adversary. However, our security proof relies on a variant [BN06] of the forking lemma [PS96], and therefore uses a rewinding strategy that does not apply to quantum algorithms. Although it is plausible that our scheme is secure against quantum attackers, we do not currently know how to prove this and leave such a proof as an open question. In this context, it may be noteworthy, that a proof of security

<table>
<thead>
<tr>
<th>Asymptotic Efficiency</th>
<th>Computational</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline Sign</td>
<td>Online Sign</td>
<td>Verify</td>
</tr>
<tr>
<td>Worst</td>
<td>$O(2^\tau)$</td>
<td>$O(\tau)$</td>
</tr>
<tr>
<td>Average</td>
<td>$\tilde{O}(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concrete Efficiency</th>
<th>Computational</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$0.4$ s</td>
<td>$25$ ms</td>
</tr>
<tr>
<td></td>
<td>$1.6$ ms</td>
<td>$1.6$ ms</td>
</tr>
</tbody>
</table>

Table 1. The asymptotic worst-case and average-case along with concrete worst-case costs of Squirrel. Here $\lambda$ denotes the security parameter and $\rho$ the maximum number of signatures that can be aggregated. The maximum number of signatures that can be issued under one key pair is $2^\tau$. The column $\sigma$ specifies the size of an individual signature while $\sigma_{agg}$ specifies the size of an aggregated signature. Asymptotic worst-case cost is measured in terms of ring multiplications. The $\tilde{O}(\cdot)$ notation hides logarithmic dependencies. Concrete costs are measured for $\tau = 24$ and $\rho = 4096$ with $\lambda = 112$. 
against quantum attackers in the classical random oracle model would be sufficient, because such a proof could be lifted to the quantum random oracle model [BDF+11] using the work of Yamakawa and Zhandry [YZ21].

1.3 Technical Overview

Let us start with a very simple solution. Assume we are already given a one-time multi-signature scheme, i.e. a scheme, where a signer can sign exactly once under a given public key. To create a signature scheme that allows for signing $2^\tau$ many times, a signer can generate $2^\tau$ many independent one-time signature key pairs and publish a public key, which is the concatenation of all one-time public keys. To sign at time $t$, the signer signs using the $t$-th secret key. Such a scheme would already constitute a valid multi-signature solution in the synchronized setting for a bounded number of signatures. The main drawback of this approach is that the public key grows linearly in $2^\tau$, which is completely unacceptable.

As a subsequent iteration of the idea above, one can attempt to publish the root node $v$ of a Merkle tree that is computed on top of all the public keys. The tree serves as a commitment to the vector of individual public keys. To sign a message at time $t$, we would now publish a signature under the $t$-th key pair along with the $t$-th public key and a membership proof, which shows that the key is indeed the $t$-th leaf of the tree with the root node $v$. Unfortunately, this solution breaks the aggregation property, since one-time signatures can still be aggregated, but the membership proofs of the separate Merkle trees cannot. Luckily for us, the idea of Merkle trees with some homomorphic properties has already been studied by Papamanthou et al. [PSTY13]. In principle, their construction of a “homomorphic Merkle tree” is sufficient to make the simple idea from above work. Using these trees, one can now aggregate both the one-time multi-signatures and the membership proofs. To make this solution secure against rogue-key attacks, we can not just sum up separate signatures, but instead compute a random linear combination thereof, where the weights are chosen via a random oracle.

The main issue with the work of Papamanthou et al. [PSTY13] is the large concrete costs associated with their tree construction. When trying to realize our approach with their work, one obtains signatures sizes in the gigabyte range which would be prohibitively expensive for practical scenarios. On a very high level the main issue, among others, with their construction is that their security relies on a lattice-based assumption where the parameters grow linearly in the number of leaves of their tree. The parameters of the used assumption further deteriorate, when the random weights are applied to the membership proofs.

In this work, we present a new construction of such a Merkle tree with homomorphic properties, which does not have the drawbacks of the previous work, and is concretely efficient. Our tree construction is conceptually similar to that of Papamanthou et al., but there are some important differences. Specifically, their construction computes the labels of internal nodes as weighted sums of all leaves rooted in that node. In contrast, our construction is essentially a standard Merkle Tree instantiated with Ajtai’s hash function [Ajt99] with an additional decomposition step to map hash values into the domain of the hash function. Our tree construction does not achieve certain properties that were relevant to the work of Papamanthou et al., but in exchange our simpler construction allows for much better concrete parameters and is fully sufficient for our work here.

We then present an appropriate one-time multi-signature scheme that works in combination with this tree as outlined above. We stress that even though our construction is simple on a conceptual level, realizing the idea and making it concretely efficient is far from it.
Paper Outline. We define some notation and review some existing definitions that will be used throughout the paper in Section 2. We formally define our notion of a homomorphic vector commitment and show how to instantiate it with a construction that resembles a Merkle tree in Section 3. We define the notion of a one-time multi-signature scheme that we need and instantiate it in Section 4. Our multi-signature scheme is presented in Section 5. Finally, we discuss all relevant concrete parameters and provide extensive benchmarks of our construction in Section 6.

2 Preliminaries

This section introduces notation, some basic definitions and lemmas that we will use throughout this work. We denote by $\lambda \in \mathbb{N}$ the security parameter and by $\text{poly}(\lambda)$ any function that is bounded by a polynomial in $\lambda$. A function $f$ in $\lambda$ is negligible, if for every $c \in \mathbb{N}$, there exists some $N \in \mathbb{N}$, such that for all $\lambda > N$ it holds that $f(\lambda) < 1/\lambda^c$. We denote by $\negl(\lambda)$ any negligible function. An algorithm is PPT if it is modeled by a probabilistic Turing machine with a running time bounded by $\text{poly}(\lambda)$.

Let $X$ be a set. We write $x \leftarrow X$ to denote the process of sampling an element of $X$ uniformly at random. Let $n \in \mathbb{N}$, we denote by $[n]$ the set $\{0,\ldots,n\}$. Let $T$ be a full binary tree of depth $d$. We denote the root node of $T$ by the empty string $\epsilon$, and for any node $v$, $v\|0$ and $v\|1$ denotes the left and right child of $v$ respectively. In particular, $\{0,1\}^d$ is the set of leaves of $T$. A labeled full binary tree with labels in $X$ is represented by a labeling function $\text{label} : \{0,1\}^{\leq d} \rightarrow X$.

Let $v$ be a vector. We write $v^\top$ to denote its transpose and $v_i$ to denote the $i$-th entry in the vector for $i \in [\|v\| - 1]$. Further, $v_{<i}$ denotes the $i$-length prefix of $v$. Similarly for a bit-string $s$, $s_i$ denotes the $i$-th bit of $s$ and $s_{<i}$ denotes the prefix consisting of the first $i$ bits of $s$. Note that vectors and bit-strings are zero-indexed. From time to time we will slightly abuse this notation and use a bit-string $s$ as an index. In this case the index is to be understood as the canonical interpretation of $s$ as an integer in little-endian encoding.

Without loss of generality, we work on a power-of-two cyclotomic polynomial ring. Let $\Phi_{2n} = x^n+1$ the cyclotomic polynomial with $n$ a power of 2. We work in a polynomial ring $\mathcal{R} = \mathbb{Z}[x]/(x^n+1)$ and represent elements of $\mathcal{R}$ as $n$-dimensional vectors $\mathbb{Z}^n$ with $(c_0, \ldots, c_{n-1})^\top \in \mathbb{Z}^n$ representing the ring element $\sum_{i=0}^{n-1} x^i c_i$. Let $q$ be some prime such that $q \equiv 1 \mod 2n$. $\mathcal{R}_q$ refers to the subset of $\mathcal{R}$ represented by vectors in $\mathbb{Z}^n_q$. Let $x = c \in \mathcal{R}$ be a ring element. We denote $\|x\| = \|c\|_\infty = \max_{i \in [n-1]}|x_i|$. For an element $a \in \mathcal{R}_q$ we denote by $\|a\|$ the norm over $\mathcal{R}$.

We denote by $B_\beta$ the ball $B_\beta = \{a \in \mathcal{R}_q \mid \|a\| < \beta\}$ and by $\mathcal{T}_\alpha = \{a = (a_0 + a_1 \cdot x + \cdots + a_{n-1} x^{n-1}) \in \mathcal{R} \mid \|a\| = 1 \land \sum_{i=0}^{n-1}|a_i| = \alpha\}$ the set of polynomials with ternary coefficients, i.e. coefficients from $\{-1,0,1\}$, with exactly $\alpha$ non-zero coefficients.

The following simple lemma bounds the norm of a ring element with small coefficients that is multiplied by a ternary polynomial with a bounded number of non-zero coefficients.

Lemma 1. Let $a \in \mathcal{R}^d$ be a vector of polynomials with coefficients in $\{0,1\}$ and let $b \in \mathcal{T}_\alpha$. Then $\|b \cdot a\| \leq \alpha$.

Proof. Since $\| \cdot \|$ is the infinity norm, it is sufficient to show that $\|b \cdot a\| \leq \alpha$ where $a$ is a binary polynomial over $\mathcal{R}$. Note that for $a, b \in \mathcal{R}$, denote $c = a \cdot b$. The ring multiplication can be expressed
as a vector-matrix multiplication, where \(a_i, b_i,\) and \(c_i\) are the coefficients of \(a, b,\) and \(c,\) respectively:

\[(c_0, c_1, \ldots, c_{n-1}) = (b_0, b_1, \ldots, b_{n-1}) \times \begin{bmatrix}
a_0 & a_1 & \ldots & a_{n-1} \\
-a_{n-1} & a_0 & \ldots & a_{n-2} \\
\vdots & \vdots & \ddots & \vdots \\
-a_1 & -a_2 & \ldots & a_0
\end{bmatrix}\]

Observe that each column of the “\(a\) matrix” has no more than \(\alpha\) non-zero entries. Also, given that \(b_i\)-s are ternary, we know that each \(c_i\) will be no greater than \(\alpha\) in absolute value.

The computationally hard problem upon which the security of our constructions relies is the short integer solution problem defined over rings as follows.

**Definition 1 (Ring Short Integer Solution Problem).** For a ring \(\mathcal{R}\) and parameters \(\mu, q, \beta \in \mathbb{N},\) the SIS\(_{\mathcal{R}, q, \mu, \beta}\) problem is hard if for all PPT algorithms \(A\) it holds that

\[
\Pr\left[\begin{aligned}
a &\leftarrow \mathcal{R}_q^\mu; \\
b &\leftarrow A(a): s \in \mathcal{B}_\beta^\mu \setminus \{0\} \land a^\top s = 0
\end{aligned}\right] \leq \text{negl}(\lambda)
\]

We will be using a minor variation of the general forking lemma as introduced by Bellare and Neven [BN06] that explicitly deals with oracle algorithms.

**Lemma 2 (General Forking Lemma).** Fix an integer \(p \geq 1\) and a set \(H\) of size \(h \geq 2.\) Let \(A\) be a randomized oracle algorithm with randomness space \(R\) that on input \(x, h_0, \ldots, h_{p-1}\) and given access to an oracle \(O\) returns a triple, the first element of which is a bit, the second element of which is an integer in the range \(0, \ldots, p-1\) and the third element of which we refer to as a side output. Let \(IG\) be a randomized algorithm that we call the input generator. The accepting probability of \(A,\) denoted \(\epsilon,\) is defined as

\[
\epsilon := \Pr \left[\begin{aligned}
(x, y) &\leftarrow IG; \\
h_0, \ldots, h_{p-1} &\leftarrow H; \ b = 1
\end{aligned}\right]
\]

The forking algorithm \(F_A\) associated with \(A\) is the randomized oracle algorithm that takes input \(x,\) is given access to an oracle \(O\) and proceeds as follows:

\[
\begin{array}{l}
F_A^{O(y,)}(x) \\
\hline
r \leftarrow R \\
h_0, \ldots, h_{p-1} \leftarrow H \\
(b, i, \sigma) \leftarrow A^{O(y,)}(x, h_0, \ldots, h_{p-1}; r) \\
\text{if } b = 0 \\
\quad \text{return } (0, \bot, \bot) \\
h'_0, \ldots, h'_{p-1} \leftarrow H \\
(b', i', \sigma') \leftarrow A^{O(y,)}(x, h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_{p-1}; r) \\
\text{if } b' = 1 \text{ and } i = i' \text{ and } h'_i \neq h_i \\
\quad \text{return } (1, \sigma, \sigma') \\
\text{else} \\
\quad \text{return } (0, \bot, \bot)
\end{array}
\]

Then

\[
\epsilon \leq \frac{p}{h} + \sqrt{p \cdot \Pr[(x, y) \leftarrow IG; (b, \sigma, \sigma') \leftarrow F_A^{O(y,)}(x) : b = 1].}
\]
3 Homomorphic Vector Commitment

In this section, we formally define the notion of a homomorphic vector commitment that we will need in our main construction. This primitive, on an intuitive level, allows for committing to a long vector by publishing a short commitment value. Individual positions of the vector can then be opened individually with short openings. The commitment scheme should be homomorphic, meaning that a linear combination of individual commitments different vectors be opened to the linear combination of the entries of the individual vectors.

Definition 2. Let \( \mathcal{R} \) be a ring and let \( q = q(\lambda) \in \mathbb{N} \). A homomorphic vector commitment scheme (HVC) for domain \( \mathcal{R}_q^{\ell_{dom}} \) is defined by four PPT algorithms (\( \text{Setup}, \text{Com}, \text{Open}, \text{Vrfy} \)).

\[
\begin{align*}
\text{pp} &\leftarrow \text{Setup}(1^\lambda, \tau) \quad \text{The setup algorithm takes as input the security parameter and the binary logarithm of the length of the committed vectors and outputs public parameters.} \\
c_i &\leftarrow \text{Com}(\text{pp}, m) \quad \text{The commitment algorithm gets as input the public parameters and a vector } m \in (\mathcal{R}^{\ell_{dom}}_q)^{2^\tau} \text{ and outputs a commitment } c \in \mathcal{R}^{\ell_{com}}_q. \\
d &\leftarrow \text{Open}(\text{pp}, c, m, t) \quad \text{The opening algorithm gets as input the public parameters, a commitment, the committed vector, and an index and outputs a decommitment } d \in \mathcal{R}^{\ell_{dec}}_q. \\
m_i / \bot &\leftarrow w\text{Vrfy}(\text{pp}, c, t, d) \quad \text{The weak verification algorithm takes as input public parameters, a commitment, an index, and a decommitment and outputs either } m \in \mathcal{R}^{\ell_{dom}}_q \text{ or an error symbol.} \\
m_i / \bot &\leftarrow s\text{Vrfy}(\text{pp}, c, t, d) \quad \text{The strong verification algorithm takes as input public parameters, a commitment, an index, and a decommitment and outputs either } m \in \mathcal{R}^{\ell_{dom}}_q \text{ or an error symbol.}
\end{align*}
\]

Let \( \rho \in \mathbb{N} \) and \( W \subseteq \mathcal{R} \). A vector commitment is \((\rho, W)\)-homomorphically correct, if for all security parameters \( \lambda \in \mathbb{N} \), vector lengths \( 2^\tau = \text{poly}(\lambda) \), \( \ell \in [\rho] \), vectors \( m_0, \ldots, m_{\ell-1} \in (\mathcal{R}^{\ell_{dom}}_q)^{2^\tau} \), ring elements \( w^0, \ldots, w^{\ell-1} \in W \), and indices \( t \in [2^\tau - 1] \) it holds that

\[
\Pr \left[ \begin{array}{l}
\text{pp} \leftarrow \text{Setup}(1^\lambda, \tau); \\
c^i \leftarrow \text{Com}(\text{pp}, m^i); \\
d^i \leftarrow \text{Open}(\text{pp}, c^i, m^i, t) \\
w\text{Vrfy}(\text{pp}, c^i, t, d^i) = \sum_{i=0}^{\ell-1} w^i \cdot m^i 
\end{array} \right] = 1
\]

Remark 1. Note that the homomorphic correctness definition above implies regular correctness of unaggregated commitments with \( \ell = 1 \) and \( 1 \in W \).

Definition 3 (Position-Binding). An HVC is position binding if for all security parameters \( \lambda \) and all PPT algorithms \( \mathcal{A} \) it holds that

\[
\Pr \left[ \begin{array}{l}
\text{pp} \leftarrow \text{Setup}(1^\lambda, \tau); \\
(c, t, d_0, d_1) \leftarrow \mathcal{A}(\text{pp}); \\
m_0 \neq m_1 \\
m_0 \leftarrow w\text{Vrfy}(\text{pp}, c, t, d_0); \\
\bot \notin \{m_0, m_1\} \\
m_1 \leftarrow w\text{Vrfy}(\text{pp}, c, t, d_1) 
\end{array} \right] \leq \text{negl}(\lambda).
\]

We require that a limited homomorphism even for malicious commitments. For any two, even malicious, commitments and their two respective openings that strongly verify, their difference will still weakly verify.
**Definition 4.** Let HVC be a vector commitment scheme (HVC) for domain $R_q^{\text{dom}}$ with commitment length $\ell_{\text{com}}$ and decommitment length $\ell_{\text{dec}}$. HVC is robustly homomorphic if for all security parameters $\lambda \in \mathbb{N}$, vector lengths $2^\tau = \text{poly}(\lambda)$, public parameters $\text{pp} \leftarrow \text{Setup}(1^\lambda, \tau)$, indices $t \in [2^\tau - 1]$, (possibly malformed) commitments $c^0, c^1 \in R_q^{\text{com}}$, and (possibly malformed) decommitments $d^0, d^1 \in R_q^{\text{dec}}$ such that $s_{\text{Vrfy}}(\text{pp}, c^0, t, d^0) = m^0$ and $s_{\text{Vrfy}}(\text{pp}, c^1, t, d^1) = m^1$ with $m^0, m^1 \neq \perp$ it holds that

$$w_{\text{Vrfy}}(\text{pp}, c^0 - c^1, t, d^0 - d^1) = m^0 - m^1.$$ 

### 3.1 Homomorphic Vector Commitment for $R_q$

Having formally defined the primitive we want, we now show how to construct it. We first focus on constructing a vector commitment with domain $R_q$. In Section 3.2 we will show how to leverage this into a more general construction for domain $R^\xi_q$. On a high level, our construction follows a blueprint initially presented by Papamanthou et al. [PSTY13]. We will instantiate our homomorphic vector commitments by constructing a Merkle tree with a “sufficiently” homomorphic hash functions at the internal nodes. Our hash function will have different input and output domains and for that reason, we will need to apply a decomposition function on the hash outputs at the internal nodes before they can be used as inputs in the computation of the parents nodes’ values. In contrast to the work of Papamanthou et al., our approach is conceptually simpler, since we require somewhat different properties to what they needed. We put a particular focus on picking the domains of the hash function as well as the decomposition function in a manner that optimizes the concrete efficiency of our final construction.

We now define a decomposition function that allows us to map a ring element with possibly large norm to a vector of low norm ring elements and we show that this function has nice homomorphic properties.

**Definition 5 (Binary decomposition of $R_q$ elements).** For any $a = \sum_{i=0}^{n-1} a_i \cdot x^i \in R_q$, denote by $(a_{i,0}, \ldots, a_{i,\log q - 1})^T \in \{0, 1\}^{\log q}$ the binary decomposition of $a_i$, i.e.,

$$a_i := \sum_{j=0}^{\log q - 1} a_{i,j} \cdot 2^j.$$ 

We define the following decomposition of $a$ into binary polynomials:

$$\text{bin}_q : R_q \to R_q^{\log q}, \quad \text{bin}_q(a) = \left(\sum_{i=0}^{n-1} a_{i,0} \cdot x^i, \ldots, \sum_{i=0}^{n-1} a_{i,\log q - 1} \cdot x^i\right).$$

**Definition 6 (Projection onto $R_q$ elements).** For any $b \in R_q^{\log q}$ we define the function

$$\text{proj}_q : R_q^{\log q} \to R_q, \quad \text{proj}(b) = \sum_{j=0}^{\log q - 1} 2^j \cdot b_j.$$
For the sake of readability we will omit $q$ and simply write $\text{bin}$ and $\text{proj}$ whenever the modulus is clear from context.

The following two simple lemmas effectively states that the projection function is the inverse of the decomposition function and that the projection function is linear.

**Lemma 3.** For all $a = \sum_{i=0}^{n-1} a_i \cdot x^i \in \mathcal{R}_q$, it holds that $\text{proj}(\text{bin}(a)) = a$.

**Proof.**

$$\text{proj}(\text{bin}(a)) = \text{proj}\left( \sum_{i=0}^{n-1} a_{i,0} \cdot x^i, \ldots, \sum_{i=0}^{n-1} a_{i,[\log q]-1} \cdot x^i \right)$$

$$= \sum_{j=0}^{[\log q]-1} \sum_{i=0}^{n-1} a_{i,j} \cdot x^i$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{[\log q]-1} 2^j \cdot a_{i,j}$$

$$= \sum_{i=0}^{n-1} x^i \cdot \sum_{j=0}^{[\log q]-1} 2^j \cdot a_{i,j}$$

$$= \sum_{i=0}^{n-1} x^i \cdot a_i = a \quad \square$$

**Lemma 4.** The projection function $\text{proj}$ is linear, i.e., for any $b_0, b_1 \in \mathcal{R}_q^{[\log q]}$ and any $w^0, w^1 \in \mathcal{R}_q$, $\text{proj}(w^0 \cdot b_0 + w^1 \cdot b_1) = w^0 \cdot \text{proj}(b_0) + w^1 \cdot \text{proj}(b_1)$.

**Proof.**

$$\text{proj}(w^0 \cdot b_0 + w^1 \cdot b_1) = \sum_{j=0}^{[\log q]-1} 2^j \cdot (w^0 \cdot b^0_j + w^1 \cdot b^1_j)$$

$$(\text{Definition 6})$$

$$= w^0 \cdot \left( \sum_{j=0}^{[\log q]-1} 2^j \cdot b^0_j \right) + w^1 \cdot \left( \sum_{j=0}^{[\log q]-1} 2^j \cdot b^1_j \right)$$

$$= w^0 \cdot \text{proj}(b_0) + w^1 \cdot \text{proj}(b_1) \quad (\text{Definition 6})$$

$$\square$$

We extend the definitions of $\text{bin}$ and $\text{proj}$ to vectors of ring elements in the natural sense. That is, let $a \in \mathcal{R}_q^x$ and $b \in \mathcal{R}_q^{[\log q]}$ with $b_i := (b_{i,[\log q]}, \ldots, b_{i+[\log q]-1})^T$, then $\text{bin}(a) := (\text{bin}(a_0), \ldots, \text{bin}(a_x))$ and $\text{proj}(b) := (\text{proj}(b_0), \ldots, \text{proj}(b_{[\log q]-1}))$. It is easy to verify that Lemma 3 and Lemma 4 also apply to this extension.

Equipped with the decomposition and projection functions, we are now ready to define how the labels of the nodes in our tree construction will be computed.

**Definition 7 (Labelled full binary tree).** Let $m = (m_0, \ldots, m_{2^\tau-1})^T \in \mathcal{R}_q^{2^\tau}$ and $h_0, h_1 \in \mathcal{R}_q^{[\log q]}$ be fixed. We define the labeling function $\text{label} : \{0,1\}^{\leq \tau} \rightarrow \mathcal{R}_q^{[\log q]}$ of for a labeled full binary tree of depth $\tau$ as

$$\text{label}(h_0, h_1, m, v) := \begin{cases} \text{bin}(m_v) & \text{if } |v| = \tau \\ \text{bin} \left( h_0^T \cdot \text{label}(h_0, h_1, m, v|0) + h_1^T \cdot \text{label}(h_0, h_1, m, v|1) \right) & \text{if } |v| < \tau \end{cases}$$
Let \( \lambda \) be the smallest integer, such that \( \binom{n}{\alpha} \cdot 2^\alpha \geq 2^\lambda \). If the SIS\(_{\mathbb{R}_q,2[\log q],2poly} \) problem is hard, then the construction from Figure 1 is a \((\rho, \mathcal{T}_\alpha)\)-homomorphically correct, robustly homomorphic, and position binding vector commitment scheme (HVC) for \( \mathbb{R}_q \).

**Proof.** The theorem follows from Lemma 6, Lemma 8, and Lemma 9 proven below.

### Lemma 6

The construction from Figure 1 is a \((\rho, \mathcal{T}_\alpha)\)-homomorphically correct vector commitment scheme (HVC) for \( \mathbb{R}_q \).

**Proof.** Let \( m^0, \ldots, m^{t-1} \in \mathbb{R}_q^\mathcal{T}, p_i^0 = \text{Com}(\text{pp}, m^i), t \in [2^\tau - 1], (p_i^1, \ldots, p_i^t, s_i^1, \ldots, s_i^t)^T = \text{Open}(\text{pp}, p_i^0, m^i, t), \) and \( w^0, \ldots, w^{t-1} \in \mathcal{T}_\alpha \) as specified in Definition 2. We will first prove a claim about the individual honestly computed commitments and decommitments.

**Claim 7.** For all \( j \in [\tau - 1] \) it holds that \( \text{proj}(p_j^i) = h_{t_j}^i \cdot p_{j+1}^i + h_{t_{j+1}}^i \cdot s_{j+1}^i \).

**Proof.** We observe that for all \( j \in [\tau - 1] \) it holds that

\[
\text{proj}(p_j^i)
\]
The construction from Figure 1 is a robustly homomorphic vector commitment scheme.

Lemma 8. The construction from Figure 1 is a robustly homomorphic vector commitment scheme.
Proof. Let $c^0, c^1 \in \mathcal{R}_q^{\ell_{con}}$, and $d^0, d^1 \in \mathcal{R}_q^{\ell_{sec}}$, and $t \in [2^\tau - 1]$ be arbitrary, such that
\[ s_{\text{Vrfy}}(pp, c^0, t, d^0) = m^0 \quad \text{and} \quad s_{\text{Vrfy}}(pp, c^1, t, d^1) = m^1 \] (1)
with $m^0, m^1 \neq \perp$. Let $d^i$ parse as $(p^i_1, \ldots, p^i_i, s^i_1, \ldots, s^i_r)^T$ for $i \in \{0, 1\}$. We first note that if $w_{\text{Vrfy}}(pp, c^0 - c^1, t, d^0^1 - d^1) \neq \perp$, then
\[
w_{\text{Vrfy}}(pp, c^0 - c^1, t, d^0 - d^1) = \text{proj}(p^0_i - p^1_i) = \text{proj}(p^0_i) - \text{proj}(p^1_i) = s_{\text{Vrfy}}(pp, c^0, t, d^0) - s_{\text{Vrfy}}(pp, c^1, t, d^1) \] (Def. of Vrfy)
\[ = m^0 - m^1. \] (Equation 1)

It thus remains to show that $w_{\text{Vrfy}}(pp, c^0 - c^1, t, d^0^1 - d^1) \neq \perp$. For this, let further $p^i_0 = c^i$. By definition of the strong verification algorithm, and since $m^0, m^1 \neq \perp$ it holds that for $i \in \{0, 1\}$ and $j \in [\tau - 1]$
\[
\|p^i_{j+1}\| \leq \rho \cdot \alpha \quad \text{and} \quad \|s^i_{j+1}\| \leq \rho \cdot \alpha \quad \text{(2)}
\]
\[
\text{proj}(p^i_j) = h^T_{i_j} \cdot p^i_{j+1} + h^T_{1-i_j} \cdot s^i_{j+1}. \quad \text{(3)}
\]

From Equation 2 it follows that for all $j \in [\tau - 1]$
\[
\|p^0_j - p^1_j\| \leq \|p^0_j\| + \|p^1_j\| \leq 2\rho \cdot \alpha
\]
\[
\|s^0_j - s^1_j\| \leq \|s^0_j\| + \|s^1_j\| \leq 2\rho \cdot \alpha.
\]

From Equation 3 and the linearity of proj it follows that for all $j \in [\tau - 1]$
\[
\text{proj}(p^0_j - p^1_j) = \text{proj}(p^0_j) - \text{proj}(p^1_j) \] (Lemma 4)
\[= (h^T_{i_j} \cdot p^0_{j+1} + h^T_{1-i_j} \cdot s^0_{j+1}) - (h^T_{i_j} \cdot p^1_{j+1} + h^T_{1-i_j} \cdot s^1_{j+1}) \] (Equation 3)
\[= h^T_{i_j} \cdot (p^0_{j+1} - p^1_{j+1}) + h^T_{1-i_j} \cdot (s^0_{j+1} - s^1_{j+1}).
\]

Therefore, all checks in the weak verification algorithm go through and $w_{\text{Vrfy}}(pp, c^0 - c^1, t, d^0^1 - d^1) \neq \perp$.

\[\square\]

Lemma 9. If the $\text{SIS}_{\mathcal{R}, q, 2[\log q], 4\rho \cdot \alpha}$ problem is hard then the construction from Figure 1 is position binding.

Proof. We will prove this lemma by leveraging that any pair of valid decommitments will lead to a collision somewhere in the generalized hash tree, which can be turned into a solution for the SIS instance. Let $A$ be an arbitrary PPT adversary against the position binding property of the construction. We construct a PPT algorithm that solves the $\text{SIS}_{\mathcal{R}, q, 2[\log q], 4\rho \cdot \alpha}$ problem as follows. Upon input $a = (a_0, \ldots, a_{2[\log q] - 1})^T$, $B$ sets $h_0 := (a_0, \ldots, a_{[\log q] - 1})^T$ and $h_1 := (a_{[\log q]}, \ldots, a_{2[\log q] - 1})^T$ and runs $(c, t, d^0, d^1) \leftarrow A((h_0, h_1))$. For $i \in \{0, 1\}$ let $m^i := w_{\text{Vrfy}}((h_0, h_1), c, t, d^i)$. If $m^0 \neq m^1$ or $\perp \in \{m^0, m^1\}$, $B$ aborts. Otherwise, parse $d^i$ as $(p^i_1, \ldots, p^i_r, s^i_1, \ldots, s^i_r)$, set $p^i_0 := c$. 

12
Let $j^* \in [\tau]$ be the \textit{largest} index, such that $\proj(p_{j^*}^0) = \proj(p_{j^*}^1)$. Note that such an index always exists, since $p_0^0 = c = p_1^0$ and that $j^* < \tau$, since $\proj(p_{j^*}^0) = m_0^0 \neq m_1^0 = \proj(p_{j^*}^1)$. If $i_{j^*} = 0$, $B$ outputs $z := (p_{j^*+1}^0, s_{j^*+1}^0)^t - (p_{j^*+1}^1, s_{j^*+1}^1)^t$, if $i_{j^*} = 1$, $B$ outputs $z := (s_{j^*+1}^0, p_{j^*+1}^0)^t - (s_{j^*+1}^1, p_{j^*+1}^1)^t$.

We now analyze the success probability of $B$. It holds that $\proj(p_{j^*}^0) = \proj(p_{j^*}^1)$ and by the definition of the weak verification algorithm that

$$h_{i_{j^*}}^T \cdot p_{j^*+1}^0 + h_{i_{j^*}+1}^T \cdot s_{j^*+1}^0 = h_{i_{j^*}}^T \cdot p_{j^*+1}^1 + h_{i_{j^*}+1}^T \cdot s_{j^*+1}^1$$

$$\iff h_{i_{j^*}}^T \cdot (p_{j^*+1}^0 - p_{j^*+1}^1) + h_{i_{j^*}+1}^T \cdot (s_{j^*+1}^0 - s_{j^*+1}^1) = 0$$

$$\iff a^T \cdot z = 0$$

It further holds by the definition of the weak verification algorithm that

$$\|p_{j^*+1}^0\| \leq 2\rho \cdot \alpha, \quad \|s_{j^*+1}^0\| \leq 2\rho \cdot \alpha, \quad \|p_{j^*+1}^1\| \leq 2\rho \cdot \alpha, \quad \|s_{j^*+1}^1\| \leq 2\rho \cdot \alpha.$$ 

Therefore, the norm of $z$ can be bounded as

$$\|z\| \leq \max\{|\|p_{j^*+1}^0\|, \|s_{j^*+1}^0\|\} + \max\{|\|p_{j^*+1}^1\|, \|s_{j^*+1}^1\|\} \leq 4\rho \cdot \alpha.$$ 

It remains to show that $z \neq 0$. Since $j^*$ is the \textit{largest} index such that

$$\proj(p_{j^*}^0) = \proj(p_{j^*}^1)$$

it holds that

$$\proj(p_{j^*+1}^0) \neq \proj(p_{j^*+1}^1)$$

and thereby that

$$p_{j^*+1}^0 \neq p_{j^*+1}^1.$$ 

Therefore $z \neq 0$. Thus, whenever $A$ is successful, $B$ is successful with probability 1 and we can conclude that

$$\negl(\lambda) \geq \Pr[a \leftarrow R_q^{2[\log q]}; s \leftarrow B(a) : z \in B_q^{2[\log q]} \setminus \{0\} \land a^T s = 0]$$

$$= \Pr\left[\begin{array}{l}
(c, t, d_1, d_2) \leftarrow \mathcal{A}(pp); \quad m_1 \neq m_2 \\
\qquad m_1 \leftarrow w\text{Vrfy}(pp, c, t, d_1); \quad \perp \notin \{m_1, m_2\} \\
\qquad m_2 \leftarrow w\text{Vrfy}(pp, c, t, d_2)
\end{array}\right]$$

\[\square\]

### 3.2 Homomorphic Vector Commitment for $R_q^\xi$

In the previous section we constructed an HVC for domain $R_q$ for some $\bar{q} = \text{poly}(\lambda)$. For our application however, this is however not ideal for two reasons. In our main construction of a synchronized multi-signature scheme, the committed values are public keys of a one-time signature scheme. These are not individual ring elements, but pairs of $R_q$ elements for some $q = \text{poly}(\lambda)$ leading to a domain mismatch. The simplest solution of choosing $\bar{q} = q$ and always decommitting to pairs of leaves works but turns out to be inefficient. We therefore want the freedom to choose
The theorem follows from Lemma 11, Lemma 12, and Lemma 13 proven below.

Proof. Let \(\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)\) and \(\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)\) be polynomial rings parameterized by \(n = \text{poly}(\lambda)\) and \(q = \text{poly}(\lambda)\), \(\bar{q} = \text{poly}(\lambda)\) respectively and let \(\xi \in \mathbb{N}\). If \(\text{HVC}\) is a \((\rho, \mathcal{T}_\alpha)\)-homomorphically correct, robustly homomorphic, and position binding vector commitment scheme (HVC) for \(\mathcal{R}_q\) and if the SIS \(\mathcal{R}_q, \xi [\log q], 4p_0\) problem is hard, then the construction from Figure 2 is a \((\rho, \mathcal{T}_\alpha)\)-homomorphically correct, robustly homomorphic, and position binding vector commitment scheme (HVC) for \(\mathcal{R}_q^\xi\).

Proof. The theorem follows from Lemma 11, Lemma 12, and Lemma 13 proven below.

**Lemma 11.** If \(\text{HVC}\) is a \((\rho, \mathcal{T}_\alpha)\)-homomorphically correct vector commitment scheme (HVC) for \(\mathcal{R}_q\), then the construction from Figure 2 is a \((\rho, \mathcal{T}_\alpha)\)-homomorphically correct vector commitment scheme (HVC) for \(\mathcal{R}_q^\xi\).

Proof. Let \(\mathbf{m}^0, \ldots, \mathbf{m}^{\ell-1} \in (\mathcal{R}_q^\xi)^{2^\ell}, \mathbf{c}^i = \text{Com}(\mathbb{pp}, \mathbf{m}^i), t \in [2^\ell - 1], (\mathbf{d}^i, \mathbf{b}^i)^T = \text{Open}(\mathbb{pp}, \mathbf{c}^i, \mathbf{m}^i, t),\) and \(w^0, \ldots, w^{\ell-1} \in \mathcal{T}_\alpha\) as specified in Definition 2.

By the \((\rho, \mathcal{T}_\alpha)\)-homomorphic correctness of \(\text{HVC}\) and the definition of \(\mathbf{b}^i\), it holds that

\[
\text{sVrfy}(\mathbb{pp}, \sum_{i=0}^{\ell-1} w^i \cdot \mathbf{c}^i, t, \sum_{i=0}^{\ell-1} w^i \cdot \mathbf{d}^i) = \sum_{i=0}^{\ell-1} w^i \cdot (\mathbf{h}^T \cdot \text{bin}(\mathbf{m}^i)) = \mathbf{h}^T \cdot \sum_{i=0}^{\ell-1} w^i \cdot \mathbf{b}^i
\]
Further, since all $b^i$ are binary and $w^i \in T_\alpha$, it holds that
\[
\left\| \sum_{i=0}^{\ell-1} w^i \cdot b^i \right\| \leq \ell \cdot \alpha \leq \rho \cdot \alpha.
\]

Therefore, all checks in the strong verification algorithm go through and it outputs
\[
\text{proj} \left( \sum_{i=0}^{\ell-1} w^i \cdot b^i \right) = \sum_{i=0}^{\ell-1} w^i \cdot \text{proj}(b^i)
\]
\[
= \sum_{i=0}^{\ell-1} w^i \cdot \text{proj}(\text{bin}(m_i))
\]
\[
= \sum_{i=0}^{\ell-1} w^i \cdot m_i^i
\]
as required. \hfill \Box

**Lemma 12.** If $\mathbb{HV}C$ is robustly homomorphic, then the construction from Figure 2 is a robustly homomorphic vector commitment scheme ($\mathbb{HVC}$).

*Proof.* Let $c^0, c^1 \in \mathcal{R}_q^{\mathsf{com}}$, $(d^0, b^0), (d^1, b^1) \in \mathcal{R}_q^{\mathsf{dsc}}$, and $t \in [2^r - 1]$ be arbitrary, such that
\[
\text{sVrfy}(pp, c^0, t, (d^0, b^0)) = m^0 \quad \text{and} \quad \text{sVrfy}(pp, c^1, t, (d^1, b^1)) = m^1
\]
with $m^0, m^1 \neq \perp$. We first note that if $\text{wVrfy}(pp, c^0 - c^1, t, (d^0 - d^1, b^0 - b^1)) \neq \perp$, then
\[
\text{wVrfy}(pp, c^0 - c^1, t, (d^0 - d^1, b^0 - b^1))
\]=\text{proj}(b^0 - b^1) \quad \text{(Def of wVrfy)}
\]=\text{proj}(b^0) - \text{proj}(b^1) \quad \text{(Lemma 4)}
\]=\text{sVrfy}(pp, c^0, t, (d^0, b^0)) - \text{sVrfy}(pp, c^1, t, (d^1, b^1)) \quad \text{(Def. of sVrfy)}
\]=m^0 - m^1. \quad \text{(Equation 4)}

It thus remains to show that $\text{wVrfy}(pp, c^0 - c^1, t, (d^0 - d^1, b^0 - b^1)) \neq \perp$.

We first note that the norm check goes through, since
\[
\|b^0 - b^1\| \leq \|b^0\| + \|b^1\| \leq 2\rho\alpha,
\]
where the last inequality follows from the definition of the strong verification algorithm and the fact that $m^0, m^1 \neq \perp$. From the same observation it also follows that
\[
\overline{sVrfy}(pp, c^0, t, d^0) = h^\top \cdot b^0 \quad \text{and} \quad \overline{sVrfy}(pp, c^1, t, d^1) = h^\top \cdot b^1.
\]

Therefore, since $\mathbb{HV}C$ is robustly homomorphic it follows that
\[
\overline{wVrfy}(pp, c^0 - c^1, t, d^0 - d^1) = h^\top \cdot b^0 - h^\top \cdot b^1 = h^\top \cdot (b^0 - b^1).
\]

Since all checks go through, it follows that $\text{wVrfy}(pp, c^0 - c^1, t, (d^0 - d^1, b^0 - b^1)) \neq \perp$. \hfill \Box

**Lemma 13.** If $\mathbb{HV}C$ is position binding and if the $\text{SIS}_{\mathcal{R}, q, \xi, [\log q], \lambda, \rho, \alpha}$ problem is hard then the construction from Figure 2 is position binding.
Proof. Let $A$ be an arbitrary PPT adversary against the position binding property of the construction. By applying the definition of the weak verification algorithm and finally splitting the probability depending on $\vec{m}_0 = \vec{m}_1$ we obtain

$$
\begin{align*}
\Pr & \left[\begin{array}{l}
\text{pp} \leftarrow \text{Setup}(1^\lambda, \tau); \\
(c, t, (d_0, b_0), (d_1, b_1)) \leftarrow A(\text{pp}); \quad m_0 \neq m_1 \\
m_0 \leftarrow \text{wVrfy}(\text{pp}, c, t, (d_0, b_0)); \quad \land \perp \notin \{m_0, m_1\} \\
m_1 \leftarrow \text{wVrfy}(\text{pp}, c, t, (d_1, b_1))
\end{array}\right] = \\
\Pr & \left[\begin{array}{l}
\text{pp} \leftarrow \text{Setup}(1^\lambda, \tau); \\
(c, t, (d_0, b_0), (d_1, b_1)) \leftarrow A(\text{pp}); \quad \|b_0\| \leq 2\rho \lambda \wedge \|b_1\| \leq 2\rho \lambda \\
m_0 \leftarrow \text{wVrfy}(\text{pp}, c, t, d_0); \quad \land \hat{m}_0 = h^T \cdot b_0 \wedge \hat{m}_1 = h^T \cdot b_1 \\
m_1 \leftarrow \text{wVrfy}(\text{pp}, c, t, d_1)
\end{array}\right]; \\
\Pr & \left[\begin{array}{l}
\text{pp} \leftarrow \text{Setup}(1^\lambda, \tau); \\
(c, t, (d_0, b_0), (d_1, b_1)) \leftarrow A(\text{pp}); \quad m_0 \leftarrow \text{wVrfy}(\text{pp}, c, t, d_0); \quad \land \hat{m}_0 = h^T \cdot b_0 \wedge \hat{m}_1 = h^T \cdot b_1 \\
m_1 \leftarrow \text{wVrfy}(\text{pp}, c, t, d_1)
\end{array}\right]; \\
\Pr & \left[\begin{array}{l}
\text{pp} \leftarrow \text{Setup}(1^\lambda, \tau); \\
(c, t, (d_0, b_0), (d_1, b_1)) \leftarrow A(\text{pp}); \quad \|b_0\| \leq 2\rho \lambda \wedge \|b_1\| \leq 2\rho \lambda \\
m_0 \leftarrow \text{wVrfy}(\text{pp}, c, t, d_0); \quad \land \hat{m}_0 = h^T \cdot b_0 \wedge \hat{m}_1 = h^T \cdot b_1 \\
m_1 \leftarrow \text{wVrfy}(\text{pp}, c, t, d_1)
\end{array}\right]
\end{align*}
\right) \cdot (5)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\right)
\r
Since both probabilities are thus bounded by negligible functions, the lemma follows. \qed

## 4 Key-Homomorphic One-Time Signatures

In this section, we define and instantiate the notion of a key-homomorphic one-time signature scheme that we will need in our final construction. Intuitively, a one-time signature is unforgeable as long as at most one signature for some message is published under a given public key. We call such a scheme homomorphic, if the a linear combination of separate signatures for the same message verifies under the linear combination of the corresponding public keys, while still being unforgeable. We present a construction of this primitive, which is similar to previous one-time signature schemes by Boneh and Kim [BK20] and Lyubashevsky and Micciancio [LM08].

**Definition 8 (One-Time Signature).** Let $\mathcal{R}$ be a ring. A key-homomorphic one-time signature scheme (KOTS) over $\mathcal{R}$ with public key length $\ell_{\text{opk}}$ and signature length $\ell_{\text{sig}}$ is defined by four PPT algorithms KOTS = ($\text{Setup}$, $\text{KGen}$, $\text{Sign}$, $\text{Vrfy}$).

$\text{pp} \leftarrow \text{Setup}(1^\lambda)$ \ The setup algorithm takes as input the security parameter and outputs public parameters.

$(\text{osk}, \text{opk}) \leftarrow \text{KGen}(\text{pp})$ \ The key generation algorithm takes as input the public parameters and outputs a key pair with $\text{opk} \in \mathcal{R}_{\ell_{\text{opk}}}^\lambda$.

$\sigma \leftarrow \text{Sign}(\text{pp}, \text{osk}, m)$ \ The signing algorithm takes as input the public parameters, a one-time signing key, and a message and outputs a signature $\sigma \in \mathcal{R}_{\ell_{\text{sig}}}^\lambda$.

$b \leftarrow \text{wVrfy}(\text{pp}, \text{opk}, m, \sigma)$ \ The weak verification algorithm takes as input the public parameters, a verification key, a message, and a candidate signature and outputs a bit indicating acceptance/rejection.

$b \leftarrow \text{sVrfy}(\text{pp}, \text{opk}, m, \sigma)$ \ The strong verification algorithm takes as input the public parameters, a verification key, a message, and a candidate signature and outputs a bit indicating acceptance/rejection.

A one-time signature is $(\rho, W)$-homomorphically correct, if for all security parameters $\lambda \in \mathbb{N}$, $\ell \in [\rho]$, messages $m \in \{0, 1\}^\star$, and ring elements $w_1, \ldots, w_\ell \in W$ it holds that

$$
\Pr \left[ \text{pp} \leftarrow \text{Setup}(1^\lambda); \\
(\text{osk}^i, \text{opk}^i) \leftarrow \text{KGen}(\text{pp}); \\
s^i \leftarrow \text{Sign}(\text{pp}, \text{osk}^i, m) \\
\sum_{i=1}^\ell w^i \cdot \text{opk}^i, m, \sum_{i=1}^\ell w^i \cdot s^i = 1 \\
\right] = 1
$$

**Remark 2.** Note that again the homomorphic correctness definition above implies regular correctness of unaggregated signatures with $\ell = 1$ and $W = \{1\}$.  

17
As with the vector commitments from the previous section, we want our signature scheme to be robustly homomorphic in the sense that the difference of two maliciously generated signatures under malicious public keys will verify, if the individual signatures verify.

**Definition 9.** Let KOTS be a \((\varrho, W)\)-homomorphically correct one-time signature scheme over \(R\) with public key length \(\ell_{\text{opk}}\) and signature length \(\ell_{\text{sig}}\). KOTS is robustly homomorphic if for all \(\lambda \in \mathbb{N}\), \(pp \leftarrow \text{Setup}(1^\lambda)\), \(m \in \{0, 1\}^*\), \(\text{opk}^0, \text{opk}^1 \in R_{q}^{\ell_{\text{opk}}}\), and \(\sigma^0, \sigma^1 \in R_{q}^{\ell_{\text{sig}}}\) such that
\[
\text{svrfy}(pp, \text{opk}^0, m, \sigma^0) = 1 \quad \text{and} \quad \text{svrfy}(pp, \text{opk}^1, m, \sigma^1) = 1
\]
it holds that
\[
\text{wvrfy}(pp, \text{opk}^0 - \text{opk}^1, m, (\sigma^0 - \sigma^1)) = 1.
\]

We define a multi-user version of (one-time) existential unforgeability, this will allow for a tighter proof of the synchronized multi-signature scheme. The definition is further strengthened by allowing the adversary to produce forgeries not just under one of the given public keys, but also under mildly rerandomized public key.

**Definition 10 (Multi-User Existential Unforgeability under Rerandomized Keys).** A \((\varrho, W)\)-homomorphically correct KOTS is \(W\)-existentially unforgeable under rerandomized keys \((\text{EUF-RK})\), if for all security parameters \(\lambda\), any \(T = \text{poly}(\lambda) \in \mathbb{N}\) and all stateful PPT algorithms \(A\) it holds that
\[
\Pr \left[ \begin{array}{c}
    \forall i \in [T - 1], (\text{osk}_i, \text{opk}_i) \leftarrow \text{KGen}(pp); \\
    \text{wvrfy}(pp, w^* \cdot \text{opk}_i^*, m^*, \sigma^*) = 1 \\
    (i^*, m^*, \sigma^*, w^*) \leftarrow A(\text{Sign}(\cdot))(pp, \text{opk}_0, \ldots, \text{opk}_{T-1}); \\
\end{array} \right] \leq \text{negl}(\lambda),
\]
where the oracle \(\text{Sign}(\cdot)\) is defined as \(\text{Sign}(i, m) := \text{Sign}(\text{osk}_i, m)\) and \(Q_i\) denotes the set of messages for which a signing query with index \(i\) has been made.

Our construction presented here closely follows a construction that appeared previously in the work of Boneh and Kim [BK20].

**Theorem 14.** Let \(R_q = \mathbb{Z}[x]/(x^n + 1)\) be a polynomial ring parameterized by \(n = \text{poly}(\lambda)\) and \(q = \text{poly}(\lambda)\). Let \(\alpha\) be the smallest integer, such that \(\binom{n}{\alpha} \cdot 2^\alpha \geq 2^\lambda\). Let \(W' = \{w_0 - w_1 \mid w_0, w_1 \in \mathcal{T}_\alpha \land w_0 \neq w_1\}\). If the \(\text{SIS}_{R_q, \gamma, (4\rho + 4)\alpha \beta_s}\) problem is hard and \(H : \{0, 1\}^* \rightarrow \mathcal{T}_\beta_s\) is collision resistant, then the construction from Figure 3 is a \((\varrho, W)\)-homomorphically correct KOTS that is multi-user existentially unforgeable under rerandomized keys.

**Proof.** The theorem follows from Lemma 15, Lemma 16, and Lemma 17.

The following three lemmas state that our construction satisfies the desired homomorphic properties and that it is unforgeable.

**Lemma 15.** Let \(\beta_s, \alpha, \varrho \in \mathbb{N}\) and let \(H : \{0, 1\}^* \rightarrow \mathcal{T}_\beta_s\) be a hash function. Let \(\beta_\alpha = 2\rho\alpha\beta_s\). The construction from Figure 3 is a \((\varrho, \mathcal{T}_\alpha)\)-homomorphically correct one time signature scheme.
Fig. 3. Description of the key-homomorphic one-time signature scheme. $H$ is a collision-resistant hash function mapping bit-strings to $\mathbb{T}^{\beta_s}$.

Proof. Let $\lambda \in \mathbb{N}$, $\ell \in [\rho]$, $m \in \{0, 1\}^*$, and $w_1, \ldots, w_\ell \in \mathbb{T}_s$ as in Definition 8. Let $pp \leftarrow \text{Setup}(1^\lambda)$ and for $i \in [\ell]$, let $(\text{osk}_i, \text{opk}_i) \leftarrow \text{KGen}(pp)$ and $\sigma_i = \text{Sign}(pp, \text{osk}_i, m)$. We need to show that both checks in the verification algorithm go through for $\beta' = \beta_\sigma$. We first observe that the norm check goes through:

$$\sum_{i=0}^{\ell-1} w_i \cdot \sigma_i = \sum_{i=0}^{\ell-1} w_i \cdot (s_{0i} \cdot H(m) + s_{1i})$$

(Def of Sign)

$$\leq \sum_{i=0}^{\ell-1} \|w_i \cdot (s_{0i} \cdot H(m) + s_{1i})\|$$

(Triangle Inequality)

$$\leq \sum_{i=0}^{\ell-1} \alpha \cdot (\|s_{0i} \cdot H(m)\| + \|s_{1i}\|)$$

(Lemma 1)

$$\leq \sum_{i=0}^{\ell-1} \alpha \cdot (\beta_s + \beta_s)$$

(Lemma 1)

$$= 2\ell \alpha \beta_s \leq 2 \rho \alpha \beta_s = \beta_\sigma.$$  

It remains to verify that the second check goes through:

$$a^\top \sum_{i=0}^{\ell-1} w_i \cdot \sigma_i = a^\top \sum_{i=0}^{\ell-1} w_i \cdot (s_{0i} \cdot H(m) + s_{1i})$$

(Def of Sign)

$$= \sum_{i=0}^{\ell-1} w_i \cdot (a^\top \cdot s_{0i} \cdot H(m) + a^\top \cdot s_{1i})$$

(Distributivity)

$$= \sum_{i=0}^{\ell-1} w_i \cdot (v_{0i} \cdot H(m) + v_{1i})$$

(Def of KGen)
\[ (\sum_{i=0}^{\ell-1} w^i \cdot v_0^i) \cdot H(m) + (\sum_{i=1}^{\ell} w^i \cdot v_1^i) \]

The lemma statement thus follows. \( \square \)

**Lemma 16.** Let \( \beta_s \in \mathbb{N} \) and let \( H : \{0,1\}^* \rightarrow T_{\beta_s} \) be a hash function. Then the construction from Figure 3 is a robustly homomorphic.

**Proof.** Let \( \lambda \in \mathbb{N} \), \( p \leftarrow \text{Setup}(1^\lambda) \), \( m \in \{0,1\}^* \), \( \text{opk}^0, \text{opk}^1 \in \mathcal{R}_q \), and \( \sigma^0, \sigma^1 \in \mathcal{R}_q^\gamma \) as specified in Definition 9. To conclude that the scheme is robustly homomorphic we need to verify that both checks in the verification algorithm go through for \( \beta' = 2\beta_\sigma \).

Since both \( \sigma^0 \) and \( \sigma^1 \) strongly verify, it holds that

\[ \| (\sigma^0 - \sigma^1) \| \leq \| \sigma^0 \| + \| \sigma^1 \| \leq 2\beta_\sigma, \]

even though the norm check goes through. It remains to verify that the second check also goes through.

\[ a^\top \cdot (\sigma^0 - \sigma^1) = a^\top \cdot \sigma^0 - a^\top \cdot \sigma^1 \]
\[ = (v_0^0 \cdot H(m) + v_0^1) - (v_1^1 \cdot H(m) + v_1^1) \] (Def of sVrfy)
\[ = (v_0^0 - v_1^1) \cdot H(m) + (v_0^1 - v_1^1). \]

Therefore, the lemma statement follows. \( \square \)

**Lemma 17.** Let \( n, \gamma, q, \beta_s, \alpha, \delta \) be positive integers with \( q \) prime and \( n \) a power of 2, such that \( q > 8\alpha\beta_s, 2^{(3\lambda+\delta)/n\gamma} \cdot q^{1/\gamma} \leq 3/2, \) and \( 2^{2\lambda} \leq |\mathcal{T}_{\beta_s}| \leq 2^{2\lambda+\delta} \). Let \( H : \{0,1\}^* \rightarrow \mathcal{T}_{\beta_s} \) be a hash function. Let \( \beta_\sigma = 2\rho \alpha \beta_s. \) If the SIS\( _{\mathcal{R},n,\gamma,(4\rho+4)\alpha\beta_s} \) problem is hard and \( H \) is collision resistant, then the construction from Figure 3 is existentially unforgeable under rerandomized keys.

**Proof.** Let \( \mathcal{A} \) be an arbitrary adversary against the multi-user existentially unforgeable under rerandomized keys with success probability \( \epsilon = \epsilon(\lambda) \). We construct an algorithm \( \mathcal{B} \) that solves \( (\mathcal{R}, q, \gamma, (4\rho + 8)\alpha\beta_s) \)-SIS as follows. Given \( a \in \mathcal{R}_q^\gamma \), \( \mathcal{B} \) chooses secret keys \( (s_0^i, s_1^i) \in \mathcal{B}_1^i \times \mathcal{B}_2^i \) uniformly at random for \( i \in [T - 1] \) and invokes \( \mathcal{A} \) on public keys \( v_0^i, v_1^i \), with \( v_0^i := a^\top \cdot s_0^i \). Whenever \( \mathcal{A} \) sends a signing query \( i, m \), \( \mathcal{B} \) will respond by sending the honestly computed signature \( \sigma := s_0^i \cdot H(m) + s_1^i. \) Eventually \( \mathcal{A} \) will then output a supposed forgery \( (i^*, m^*, \sigma^*, w^*) \) and \( \mathcal{B} \) will compute the signature on the same message as \( \sigma' := w^* \cdot s_0^i \cdot H(m^*) + w^* \cdot s_1^i \). It then outputs \( \sigma^* - \sigma' \).

To analyze the success probability of \( \mathcal{B} \), suppose that \( \mathcal{A} \) outputs a valid forgery. We first note that, if \( \sigma^* \neq \sigma' \), it holds that \( \sigma^* - \sigma' \neq 0 \) and further, since both signatures verify, that
\[ a^\top \cdot (\sigma^* - \sigma') = a^\top \cdot \sigma^* - a^\top \cdot \sigma' \]
\[ = (w^* \cdot v_0^i \cdot H(m) + w^* \cdot v_1^i) - (w^* \cdot v_0^i \cdot H(m) + w^* \cdot v_1^i) = 0. \]

Since \( \sigma^* \) weakly verifies, it must hold that \( \| \sigma^* \| \leq 2\beta_\sigma \). Further, \( \sigma' \) is an honestly computed signature for a secret key rerandomized with \( w^* \in W' \). Therefore, there exist \( w_0, w_1 \in \mathcal{T}_\alpha \), such that \( w^* = w_0 - w_1 \) and it holds that
\[ \| \sigma' \| = \| w^* \cdot s_0^i \cdot H(m^*) + w^* \cdot s_1^i \| \] (Def. of Sign)

20
It thus holds that \( \| \sigma^* - \sigma' \| \leq \| \sigma^* \| + \| \sigma' \| \leq 2 \cdot \beta_\sigma + 4 \alpha \beta_s = 4 \rho \alpha \beta_s + 4 \alpha \beta_s = (4 \rho + 4) \alpha \beta_s \). Thus \( \sigma^* - \sigma' \) is always a valid solution to the SIS-instance. Thus, it remains to bound the probability, that \( \sigma^* = \sigma' \).

For this, we observe by Lemma 18 that the secret key for which \( \mathcal{A} \) chooses to forge is information-theoretically hidden from \( \mathcal{A} \) among at least 2 possible secret keys. Once \( \mathcal{A} \) outputs a valid forgery \((i^*, m^*, \sigma^*, w^*)\), the signing key used for the forgery becomes uniquely determined by Lemma 19 as long as \( H(m^*) \neq H(m) \), which is guaranteed with overwhelming probability by the collision resistance of \( H \). It follows that \( \sigma^* \neq \sigma' \) with probability at least \( 1/2 - \text{negl}(\lambda) \). Therefore, the success probability of our reduction \( \mathcal{B} \) is \( (1/2 - \text{negl}(\lambda)) \epsilon \) and since the SIS problem is hard, \( \epsilon \) must be negligible in \( \lambda \).

**Lemma 18.** Let \( n, \gamma, q, \beta_s, \delta \) be positive integers such that \( 2^{(3\lambda+\delta)/n\gamma} \cdot q^{1/\gamma} \leq 3/2 \) and \( 2^{2\lambda} \leq |T_{\beta_s}| \leq 2^{2\lambda+\delta} \), let \( \mathcal{R} = \mathbb{Z}[x]/(x^n + 1) \). Then for any \( \alpha \in \mathcal{R}^\gamma \) and uniformly chosen \( (s_0, s_1) \in \mathcal{B}_1^\gamma \times \mathcal{B}_2^\gamma \) it holds with probability at least \( 1 - 2^{-\lambda} \) that for every \( c \in T_{\beta_s} \) there exists \( (s'_0, s'_1) \in \mathcal{B}_1^\gamma \times \mathcal{B}_2^\gamma \) such that \( (s'_0, s'_1) \neq (s_0, s_1) \), \( (\alpha^\top \cdot s'_0, \alpha^\top \cdot s'_1) = (\alpha^\top \cdot s_0, \alpha^\top \cdot s_1) \) and \( s'_0 \cdot c + s'_1 = s_0 \cdot c + s_1 \).

**Proof.** Our proof closely follows the proof from [LM08, Lemma 4.9]. We define a function \( f_{a,c} \) that maps any secret key \((s_0, s_1)\) to a pair of the public key and signature of message \( c \) defined as \((\alpha^\top \cdot s_0, \alpha^\top \cdot s_1), s_0 \cdot c + s_1\). We will show that the domain of this function is at least \( 2^{3\lambda+\delta} \) times larger than the range. The number of possible secret keys is \( 3^{n\gamma} \cdot (2\beta_s + 1)^{n\gamma} \). The number of possible signatures is at most \( (4\beta_s + 1)^{n\gamma} \). For fixed values \( a, c, s_0 \cdot c + s_1 \), we observe that once \( \alpha^\top \cdot s_0 \) is fixed, the second component \( \alpha^\top \cdot s_1 = \alpha^\top \cdot ((s_0 \cdot c + s_1) - s_0 \cdot c) \) is uniquely determined. Thus for a fixed signature, there are at most \( q^n \) many possible public keys and therefore the range of \( f_{a,c} \) is at most \( (4\beta_s + 1)^{n\gamma} \cdot q^n \). We observe that

\[
\frac{3^{n\gamma} \cdot (2\beta_s + 1)^{n\gamma}}{(4\beta_s + 1)^{n\gamma} \cdot q^n} \geq \frac{3^{n\gamma} \cdot (2\beta_s + 1)^{n\gamma}}{(4\beta_s + 2)^{n\gamma} \cdot q^n} = \frac{3^{n\gamma}}{2^{n\gamma} \cdot q^n}
\]

Using the inequality from the lemma statement, one can see that

\[
2^{(3\lambda+\delta)/n\gamma} \cdot q^{1/\gamma} \leq \frac{3}{2} \implies 2^{3\lambda+\delta} \cdot q^n \leq \left(\frac{3}{2}\right)^{n\gamma} \implies 2^{3\lambda+\delta} \leq \frac{3^{n\gamma}}{2^{n\gamma} \cdot q^n}
\]

Using Lemma 4.1 from [LM08], the probability, over a uniformly chosen secret key, that there exists \((s'_0, s'_1) \in \mathcal{B}_1^\gamma \times \mathcal{B}_2^\gamma \) such that \((s'_0, s'_1) \neq (s_0, s_1), (\alpha^\top \cdot s'_0, \alpha^\top \cdot s'_1) = (\alpha^\top \cdot s_0, \alpha^\top \cdot s_1) \) and \( s'_0 \cdot c + s'_1 = s_0 \cdot c + s_1 \) is at least \( 1 - 2^{-3\lambda-\delta} \). By union bounding over all possible hash values \( c \in T_{\beta_s} \) and observing that \( T_{\beta_s} \leq 2^{2\lambda+\delta} \) the lemma statement follows.
Lemma 19. Let $n, q, \beta_s, \alpha$ be positive integers with $q$ prime and $n$ a power of two such that $q > 8\alpha\beta_s$ and let $R = \mathbb{Z}[x]/(x^n + 1)$. Let $\alpha \in R_q^\times$, $c_0, c_1 \in T_{\beta_s}$, $w_0, w_1 \in T_\alpha$, and $\sigma_0, \sigma_1 \in R$ be arbitrary ring elements such that $c_0 \neq c_1$ and $w_0 \neq w_1$. Then there exists at most a single pair of vectors $(s_0, s_1) \in B_1^\times \times B_0^\times$, such that

$$s_0 \cdot c_0 + s_1 = \sigma_0 \quad \text{and} \quad (w_0 - w_1) \cdot (s_0 \cdot c_1 + s_1) = \sigma_1.$$  

Proof. Let $(s_0, s_1)$ and $(s_0', s_1')$ be two secret keys, such that

$$s_0 \cdot c_0 + s_1 = s_0' \cdot c_0 + s_1' \implies (s_0 - s_0') \cdot c_0 + (s_1 - s_1') = 0 \quad (7)$$

and

$$(w_0 - w_1) \cdot (s_0 \cdot c_1 + s_1) = (w_0 - w_1) \cdot (s_0' \cdot c_1 + s_1')$$
$$\implies (w_0 - w_1)((s_0 - s_0') \cdot c_1 + (s_1 - s_1')) = 0 \quad (8)$$

Equation 7 implies that

$$(w_0 - w_1)((s_0 - s_0') \cdot c_0 + (s_1 - s_1')) = 0.$$

Combined with Equation 8, we get that

$$(w_0 - w_1)(s_0 - s_0')(c_0 - c_1) = 0$$

We now observe that

$$\| (w_0 - w_1)((s_0 - s_0') \cdot c_0 + (s_1 - s_1')) \| \leq 8\alpha\beta_s$$

and

$$\| (w_0 - w_1)((s_0 - s_0') \cdot c_1 + (s_1 - s_1')) \| \leq 8\alpha\beta_s.$$

Since $w_0 \neq w_1$, $c_0 \neq c_1$, $q > 8\alpha\beta_s$, and $R_q$ is an integral domain, it follows that $s_0 = s_0'$. By Equation 7, it must therefore hold that $(s_0, s_1) = (s_0', s_1')$. $\square$

5 Synchronized Multi-Signatures

In this section, we present the main construction of this work. In a nutshell, our construction will produce a public key, which is a vector commitment to a vector of independent one-time signature public keys. To sign a message at time $t$, the signer will publish an opening to the key in vector position $t$ and then sign the corresponding message with that key. The (non-interactive) aggregation of multiple independent signatures for the same message, will heavily rely on the homomorphic properties of the used vector commitment and one-time signature scheme. Let us now first formally define what a synchronized multi-signature scheme is.

Definition 11 (Synchronized Multi-Signatures). A synchronized $\rho$-wise multi-signature scheme for a bounded number of time periods is defined by five PPT algorithms $(\text{Setup, KGen, Sign, Aggregate, Vrfy})$.

$(\text{pp}) \leftarrow \text{Setup}(1^\lambda, 1^\tau)$ The setup algorithm takes as input the security parameter and the maximum number of time periods and outputs public parameters $\text{pp}$.

$(\text{sk}, \text{pk}) \leftarrow \text{KGen}(\text{pp})$ The key generation algorithm takes as input the public parameters and outputs a key-pair.

$\sigma \leftarrow \text{Sign}(\text{pp, sk, t, m})$ The signing algorithm takes as input the public parameters, a secret key, a time period $t \in [\tau - 1]$, and a message and outputs a signature.
The deterministic aggregation algorithm takes as input the public parameters, a list of public keys, a time period \( t \in [\tau - 1] \), a message, and a list of signatures, where \(|P| = |S| \leq \rho\) and outputs an aggregated signature or an error \( \perp \).

The deterministic verification algorithm takes as input the public parameters, a list of public keys, a time period \( t \in [\tau - 1] \), a message, and an aggregated signature and outputs a bit indicating acceptance/rejection.

A synchronized \( \rho \)-wise multi-signature scheme is correct, if for all \( \lambda, \tau \in \mathbb{N} \), \( \ell \in [\rho] \setminus \{0\} \), \( t \in [\tau - 1] \), and \( m \in \{0,1\}^* \) it holds that

\[
\Pr \left[ \begin{array}{l}
pp \leftarrow \text{Setup}(1^\lambda, 1^\tau), \\
(\sk_i, \pk_i) \leftarrow \text{KGen}(pp), \mathcal{P} := (\pk_0, \ldots, \pk_{\ell-1}) \\
\sigma_i \leftarrow \text{Sign}(pp, \sk_i, t, m), S := (\sigma_0, \ldots, \sigma_{\ell-1}) \\
\sigma_{\text{agg}} \leftarrow \text{Aggregate}(pp, \mathcal{P}, t, m, S)
\end{array} \right] = 1
\]

for some negligible function \( \negl(\lambda) \), where \( \mathcal{Q} \) denotes the set of signing queries made by \( \mathcal{A} \).

The following lemma will be useful for proving the security of our construction in Theorem 21, specifically it will be useful during the security reduction to the underlying one-time signature scheme. Intuitively, the lemma shows that two valid aggregate signatures that are created using vectors of random weights that differ in one position, allow for extracting a valid one-time signature and key.

**Lemma 20.** Let \( pp \leftarrow \text{Setup}(1^\lambda, 1^\tau) \) and \((\sk^*, \pk^*) \leftarrow \text{KGen}(pp)\) be fixed. Let \( \ell \in [\rho] \setminus \{0\}, \ t \in [\tau - 1], m \in \{0,1\}^*, \mathcal{P} = (\pk_0, \ldots, \pk_{\ell-1}) \) with \( \pk_j = \pk^* \), \( \sigma_{\text{agg}}^0 = (\sigma_0^0, d_0), \sigma_{\text{agg}}^1 = (\sigma_1^1, d_1) \), and let \( H_0, H_1 \) be two random oracles, such that

\[
(w_0, \ldots, w_{\ell-1}) := H_0(t, m, \mathcal{P})
\]
\[
(w_0, \ldots, w_{j-1}, w'_j, w_{j+1}, \ldots, w_{\ell-1}) := H_1(t, m, \mathcal{P})
\]

with \( w_j \neq w'_j \) and

\[
\text{Vrfy}^{H_0}(pp, \mathcal{P}, t, m, \sigma_{\text{agg}}^0) = 1 \quad \text{and} \quad \text{Vrfy}^{H_1}(pp, \mathcal{P}, t, m, \sigma_{\text{agg}}^1) = 1.
\]

Then, for \( opk^* \leftarrow \text{HVC.wVrfy}(pp_{\text{HVC}}, c^* \cdot (w_j - w'_j), t, d_0 - d_1) \) it holds that

\[
opk^* \neq \perp \quad \text{and} \quad \text{KOTS.wVrfy}(pp_{\text{KOTS}}, opk^*, m, \sigma_0^0 - \sigma_1^1) = 1.
\]
Proof. Since
\[ \text{Vrfy}^H_0(pp, P, t, m, \sigma^0_{agg}) = 1 \quad \text{and} \quad \text{Vrfy}^H_1(pp, P, t, m, \sigma^1_{agg}) = 1, \]
it must hold by definition of the verification algorithm that
\[ \text{HVC.sVrfy}(pp_{HVC}, \sum_{i \in [\ell] \setminus \{0\}} w_i \cdot c_i, t, d_0) = \text{opk}_0 \]
and
\[ \text{HVC.sVrfy}(pp_{HVC}, w'_j \cdot c_j + \sum_{i \in [\ell] \setminus \{0,j\}} w_i \cdot c_i, t, d_1) = \text{opk}_1 \]
for \( \text{opk}_0, \text{opk}_1 \neq \bot \). Thus by Definition 4 it holds that
\[
\text{opk}^* = \text{HVC.wVrfy}(pp_{HVC}, c^* \cdot (w_j - w'_j), t, d_0 - d_1) = \text{HVC.wVrfy}(pp_{HVC}, \left( \sum_{i \in [\ell] \setminus \{0\}} w_i \cdot c_i \right) - \left( w'_j \cdot c_j + \sum_{i \in [\ell] \setminus \{0,j\}} w_i \cdot c_i \right), t, d_0 - d_1) = (\text{opk}_0 - \text{opk}_1).
\]

Fig. 4. The synchronized multi-signature scheme Squirrel based on homomorphic vector commitments and key-homomorphic one-time signatures.
Further, by definition of the verification algorithm it must also hold that

\[ \text{KOTS.sVrfy}(pp_{\text{KOTS}}, opk_0, m, \sigma'_0) = 1 \quad \text{and} \quad \text{KOTS.sVrfy}(pp_{\text{KOTS}}, opk_1, m, \sigma'_1) = 1 \]

Thus, by definition Definition 9 it holds that

\[ \text{KOTS.wVrfy}(pp_{\text{KOTS}}, opk^*, m, \sigma'_0 - \sigma'_1) = \text{KOTS.wVrfy}(pp_{\text{KOTS}}, opk_0 - opk_1, m, \sigma'_0 - \sigma'_1) = 1 \]

Theorem 21. Let \( \mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1) \) be a polynomial ring parameterized by \( n = \text{poly}(\lambda) \) and \( q = \text{poly}(\lambda) \). Let \( W \subseteq \mathcal{R}_q \) be a set and let \( W' := \{ w_0 - w_1 | w_0, w_1 \in W \} \). Let \( \text{KOTS} \) be a \((\rho, W')\)-homomorphically correct one-time signature scheme with public keys in \( \mathcal{R}_q^\mathcal{E} \) and let \( \text{HVC} \) be a \((\rho, W)\)-homomorphically correct vector commitment for domain \( \mathcal{R}_q^\mathcal{E} \). If \( \text{KOTS} \) is robustly homomorphic and multi-user existentially unforgeable under rerandomized keys and \( \text{HVC} \) is robustly homomorphic and position-binding, then Squirrel, shown in Figure 4, is a correct and unforgeable synchronized \( p \)-wise multi-signature.

**Proof.** It is easy to verify, that the homomorphic correctness of the vector commitment and the one-time signature scheme imply that the scheme is correct.

To prove unforgeability, let \( \mathcal{A} \) be an arbitrary PPT algorithm that makes at most \( p = \text{poly}(\lambda) \) queries to the random oracle and for which it holds that

\[
\Pr \left[ \begin{array}{l}
pp \leftarrow \text{Setup}(1^\lambda, \rho, \tau); \\
(\text{sk}^*, \text{pk}^*) \leftarrow \text{KGen}(pp); \\
(\mathcal{P}^*, t^*, m^*, \sigma^*_{\text{agg}}) \leftarrow \text{Sign}(pp, \text{sk}^*; \cdot \cdot \cdot ) (pp, \text{pk}^*) \land \forall t'. |\{(m', \sigma) | (t', m', \sigma) \in \mathcal{Q}\}| \leq 1 \\
\text{Vrfy}(pp, \mathcal{P}^*, t^*, m^*, \sigma^*_{\text{agg}}) = 1 \\
\text{Vrfy}(\mathcal{P}^*, \mathcal{P}^*, t^*, m^*, \sigma^*_{\text{agg}}) = 1
\end{array} \right] = \epsilon(\lambda) \quad (9)
\]

We assume without loss of generality that \( \mathcal{A} \) always queries \((\mathcal{P}^*, t^*, m^*)\) to the random oracle.

**Claim 22.** Let \( \text{IG} \) and \( \mathcal{B} \) be as defined in Figure 5. Then it holds that

\[
\Pr \left[ \begin{array}{l}
((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(h_0, \ldots, h_{p-1}) \leftarrow W; b = 1
\end{array} \right] = \epsilon(\lambda).
\]

**Proof.** The input generation algorithm \( \text{IG} \) performs exactly the same setup expected by \( \mathcal{A} \) and then \( \mathcal{B} \) simply executes \( \mathcal{A} \), perfectly simulating the random oracle by lazy sampling. The only interesting part of the simulation is the fact that \( \mathcal{B} \) reorders the used randomness if a query includes the challenge public key \( \text{pk}^* \). This is necessary to later make use of Lemma 2 without modification, but does not impact the simulation at all: the random values \( h_0 \) through \( h_{p-1} \) are all distributed
\[
\begin{align*}
&\text{IG}(1^\lambda) \\
&\text{pp} \leftarrow \text{Setup}(1^\lambda, \rho, \tau) \\
&(sk^*, c^*) \leftarrow \text{KGen}(\text{pp}) \\
&\text{return } ((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \\
&H(t, m, \mathcal{P}) \\
&\text{if } ((t, m, \mathcal{P}), w) \in \mathcal{L} \\
&\text{return } w \\
&\text{parse } \mathcal{P} \text{ as } (c_0, \ldots, c_{\ell-1}) \\
&w := (h_{i_0}, \ldots, h_{i_{\ell-1}}) \\
&\text{for } j \in [\ell - 1] \\
&\text{if } c_j = \text{pk}^* \\
&w := (w_0, \ldots, w_{j-1}, w_{j-1}, \ldots, w_{\ell-2}) \\
i := i + 1 \\
&\mathcal{L} := \mathcal{L} \cup ((t, m, \mathcal{P}), w) \\
&\text{return } w \\
&\mathcal{B}^{\text{Sign}}(\text{pp}, \text{pk}^*, \ldots, (\text{pp}, \text{pk}^*)) \text{ (pp, c^*, h_0, \ldots, h_{p-1})} \\
i := 0 \\
&\mathcal{L} := \emptyset \\
((\mathcal{P}^*, t^*, m^*, \sigma^*_{\text{agg}})) \leftarrow \mathcal{A}^{\text{Sign}}(\text{pp}, \text{pk}^*, \ldots, (\text{pp}, \text{pk}^*)) \\
\text{if } \text{Vrfy}^{\mathcal{H}(\cdot)}(\text{pp}, \mathcal{P}^*, t^*, m^*, \sigma^*_{\text{agg}}) = 0 \\
\text{return } (0, 0, \bot) \\
\text{if } \text{pk}^* \notin \mathcal{P}^* \text{ or } \exists \sigma. ((t, m, \sigma) \in \mathcal{Q}) \\
\text{return } (0, 0, \bot) \\
\text{for } t' \in [2^\lambda - 1] \\
\text{if } |\{(m', \sigma) \mid (t', m', \sigma) \in \mathcal{Q}\}| > 1 \\
\text{return } (0, 0, \bot) \\
\text{for } j \in [i - 1] \\
((t_j, m_j, \mathcal{P}_j), w_j) = \mathcal{L}_j \\
\text{if } (t_j, m_j, \mathcal{P}_j) = (t^*, m^*, \mathcal{P}^*) \\
\text{return } (1, j + |\mathcal{P}^*| - 1, (\mathcal{P}^*, t^*, m^*, \sigma^*_{\text{agg}}, w_j)) \\
\text{return } (0, 0, \bot)
\end{align*}
\]

Fig. 5. The setup for the forking lemma based on attacker \( \mathcal{A} \).

The algorithm \( \mathcal{B} \) checks whether \( \mathcal{A} \) would have been successful according to Definition 12 and outputs \( b = 1 \) if \( \mathcal{A} \) was successful. Therefore, the claim follows.

By combining Claim 22 with Lemma 2 we can conclude that

\[
\epsilon \leq \frac{p}{|\mathcal{L}|} + \sqrt{p \cdot \Pr \left[ ((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(b_0, \omega^0, \omega^1) \leftarrow \mathcal{F}_B^{\text{Sign}}(\text{pp}, \text{pk}^*); \quad b = 1 \right] \right]. \tag{10}
\]

It remains to bound the probability

\[
\Pr \left[ ((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(b, \omega^0, \omega^1) \leftarrow \mathcal{F}_B^{\text{Sign}}(\text{pp}, \text{sk}^*); \quad b = 1 \right] 
\]

For any output \( (1, (\mathcal{P}^0, t^0, m^0, (\sigma^0, d^0), w^0), (\mathcal{P}^1, t^1, m^1, (\sigma^1, d^1), w^1)) \) of the forking algorithm it holds by definition of \( \mathcal{B} \) and \( \mathcal{F}_B \) that \( (\mathcal{P}^0, e^0, m^0) = (\mathcal{P}^1, t^1, m^1) \) because these are inputs to the random oracle before the fork occurs. To improve readability we thus introduce a modified forking algorithm \( \bar{\mathcal{F}}_B \) defined in Figure 6. Obviously, it holds that

\[
\Pr \left[ ((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(b_0, \omega^0, \omega^1) \leftarrow \bar{\mathcal{F}}_B^{\text{Sign}}(\text{pp}, \text{sk}^*); \quad b = 1 \right] 
\]

\[
= \Pr \left[ ((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(b, \mathcal{P}, t, m, \sigma^0_{\text{agg}}, \omega^0, \sigma^1_{\text{agg}}, \omega^1) \leftarrow \bar{\mathcal{F}}_B^{\text{Sign}}(\text{pp}, \text{sk}^*); \quad b = 1 \right] \tag{11}
\]
Let \((b, \mathcal{P}, t, m, (\sigma^0, d^0), (\sigma^1, d^1)) \leftrightarrow \overline{\text{Sign}(pp, sk^*; \cdot)}(pp, pk^*)\) be an execution of the modified forking algorithm with

\[
\text{pk}^* = c^* \quad \text{and} \quad \text{sk}^* = ((\text{osk}_0^*, \ldots, \text{osk}_{2^r-1}^*), (\text{opk}_0^*, \ldots, \text{opk}_{2^r-1}^*)).
\]

Let \(j\) denote the index, such that \(\mathcal{P}_j = \text{pk}^*\). We then define

\[
\tilde{w} := (w_j^0 - w_j^1) \quad \text{and} \quad \text{opk} := \text{HVC}.wVrfy(pp_{\text{HVC}}, \tilde{w} \cdot c^*, t, (d^0 - d^1)).
\]

The probability from Equation 11 can then be split as

\[
\Pr \left[ \begin{array}{c}
((\text{pp, pk}^*), (\text{pp, sk}^*)) \leftarrow \overline{\text{IG}(1^\lambda)}; \\
\left( (b, \mathcal{P}, t, m, \sigma^0_{\text{agg}}, w^0, \sigma^1_{\text{agg}}, w^1) \leftarrow \overline{\text{Sign}(pp, sk^*; \cdot)}(pp, pk^*) : b = 1 \right)
\end{array} \right] =
\Pr \left[ \begin{array}{c}
((\text{pp, pk}^*), (\text{pp, sk}^*)) \leftarrow \overline{\text{IG}(1^\lambda)}; \\
\left( (b, \mathcal{P}, t, m, \sigma^0_{\text{agg}}, w^0, \sigma^1_{\text{agg}}, w^1) \leftarrow \overline{\text{Sign}(pp, sk^*; \cdot)}(pp, pk^*) : \text{opk} = \tilde{w} \cdot \text{opk}_j^* \right)
\end{array} \right] \quad (12)
\]

and we can bound the two parts separately.

Consider \(\mathcal{R}_0\) described in Figure 7 as an adversary against the position binding of HVC. To analyse the success probability of \(\mathcal{R}_0\), consider an execution \((c, t, d_0, d_1) \leftarrow \mathcal{R}_0(pp_{\text{HVC}}, \tau)\). According to the definitions above, it holds that

\[
\text{HVC}.wVrfy(pp_{\text{HVC}}, c, t, d_0)
= \text{HVC}.wVrfy(pp_{\text{HVC}}, c, w_j^0 \cdot c^* - w_j^1 \cdot c^*, t^0, w_j^0 \cdot \text{Open}(pp_{\text{HVC}}, c, \text{OPK}^*, t) - w_j^0 \cdot \text{Open}(pp_{\text{HVC}}, c, \text{OPK}^*, t))
\]

\[
= \text{HVC}.sVrfy(pp_{\text{HVC}}, c, w_j^0 \cdot c^*, t^0, w_j^0 \cdot \text{Open}(pp_{\text{HVC}}, c, \text{OPK}^*, t))
- \text{HVC}.sVrfy(pp_{\text{HVC}}, c, w_j^1 \cdot c^*, t^0, w_j^1 \cdot \text{Open}(pp_{\text{HVC}}, c, \text{OPK}^*, t)) \quad \text{(Robust Homomorphism)}
\]

\[
= w_j^0 \cdot \text{opk}_j^* - w_j^1 \cdot \text{opk}_j^* = \tilde{w} \cdot \text{opk}_j^*
\]

(Homomorphic correctness)

and

\[
\text{HVC}.wVrfy(pp_{\text{HVC}}, c, t, d_1) = \text{HVC}.wVrfy(pp_{\text{HVC}}, \tilde{w} \cdot c^*, t, (d^0 - d^1)) = \text{opk}
\]

We then have that

\[
\Pr \left[ \begin{array}{c}
\text{pp}_{\text{HVC}} \leftarrow \text{HVC}.\text{Setup}(1^\lambda, \tau); \\
(c, t, d_0, d_1) \leftarrow \mathcal{R}_0(pp); \\
\text{HVC}.wVrfy(pp, c, t, d_0) \quad : m_0 \neq m_1 \wedge \bot \notin \{m_0, m_1\}; \\
\text{HVC}.wVrfy(pp, c, t, d_1)
\end{array} \right]
\]

Fig. 6. The modified forking algorithm.
\[ \mathcal{R}_0(\mathsf{pp}_{\mathsf{HVC}}, \tau) \]
\[ \mathsf{pp}_{\mathsf{KOTS}} \leftarrow \mathsf{KOTS} \cdot \mathsf{Setup}(1^\lambda) \]
\[ \mathsf{pp} := (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau) \]
\[ ((\mathsf{OSS}^*, \mathsf{OPK}^*), c^*) \leftarrow \mathsf{KGen}(\mathsf{pp}) \]
\[ (b, \mathcal{P}, t, m, \sigma^0_{agg}, w^0, \sigma^1_{agg}, w^1) \leftarrow F_B^{\mathsf{Sign}(\mathsf{pp}, (\mathsf{OSS}^*, \mathsf{OPK}^*), \cdots)}(\mathsf{pp}, c^*) \]
\[ \text{if } b = 1 \]
\[ \text{parse } \mathcal{P} \text{ as } (c_0, \ldots, c_\ell) \]
\[ \text{parse } \sigma^0_{agg} \text{ as } (\sigma^0, d^0) \]
\[ \text{parse } \sigma^1_{agg} \text{ as } (\sigma^1, d^1) \]
\[ \text{for } j \in [\ell - 1] \]
\[ \text{if } c_j = c^* \]
\[ \text{return } (\bar{w} \cdot c^*, t, \bar{w} \cdot \mathsf{Open}(\mathsf{pp}_{\mathsf{HVC}}, c^*, \mathsf{OPK}^*, t), (d^1 - d^d)) \]

Fig. 7. A reduction that uses the forking algorithm \( F_B \) to attack the position binding of HVC.

\[ \mathsf{pp} = (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau) \]
\[ ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \]
\[ (b, \mathcal{P}, t, m, \sigma^0_{agg}, w^0, \sigma^1_{agg}, w^1) \leftarrow F_B^{\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, \cdots)}(\mathsf{pp}, \mathsf{pk}^*); \]
\[ m_0 \neq m_1 \]
\[ \wedge \bot \notin \{m_0, m_1\} \]

(13)

\[ \mathsf{pp} = (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau) \]
\[ ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \]
\[ \mathsf{opk} = \mathsf{opk}^*_b \wedge \mathsf{opk} \neq \bot \]

(14)

\[ \mathsf{pp} = (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}, \tau) \]
\[ ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \]
\[ \mathsf{opk} 
\wedge b = 1 \]

(15)

where Equation 13 follows because the inputs \( \mathcal{R}_0 \) provides to the forking algorithm are distributed identically to those sampled by \( \mathsf{IG} \). Equation 14 follows because by the above observation \( (w^0_j - w^1_j) \cdot \mathsf{opk}_b^* \) cannot be \( \bot \). Finally Equation 15 follows from Lemma 20 and the fact that the forking algorithm only outputs \( b = 1 \) if both multi-signatures verify.

Since the homomorphic vector commitment is assumed to be position binding, it thus follows that

\[ \text{negl}(\lambda) \geq \Pr \left[ ((\mathsf{pp}, \mathsf{pk}^*), (\mathsf{pp}, \mathsf{sk}^*)) \leftarrow \mathsf{IG}(1^\lambda); \mathsf{opk} \neq \bar{w} \cdot \mathsf{opk}_b^* \right] \]

(16)

Consider now \( \mathcal{R}_1 \) described in Figure 8 as an adversary against the existential unforgeability under rerandomized keys of \( \mathsf{KOTS} \). We analyze the success probability of \( \mathcal{R}_1 \) by observing that

\[ \Pr \left[ \forall i \in [T - 1], (\mathsf{osk}_i, \mathsf{opk}_i) \leftarrow \mathsf{KOTS} \cdot \mathsf{KGen}(\mathsf{pp}); \mathsf{KOTS} \cdot \mathsf{wVrfy}(\mathsf{pp}, \mathsf{opk}_i, m^*, \sigma^*) = 1 \wedge m^* \notin Q_i \wedge \bar{w} \in \mathcal{W}' \right] \]

(17)
\[ \mathcal{R}_1(\text{pp}_{\text{KOTS}}, \text{opk}_3, \ldots, \text{opk}_{2^\tau-1}) \]

\[
\begin{align*}
\text{pp}_{\text{HVC}} & \leftarrow \text{HVC.Setup}(1^\lambda) \\
\text{pp} & := (\text{pp}_{\text{KOTS}}, \text{pp}_{\text{HVC}}, \tau) \\
\text{OPK}^* & := (\text{opk}_0, \ldots, \text{opk}_{2^\tau-1}) \\
c^* & \leftarrow \text{HVC.Com}(\text{pp}_{\text{HVC}}, \text{OPK}^*) \\
(b, \mathcal{P}, t, m, \sigma_{\text{agg}}^0, w^0, \sigma_{\text{agg}}^1, w^1) & \leftarrow \mathcal{F}_B^\text{Sign}(\cdot)(\text{pp}, c^*) \\
\text{if } b = 1 \\
\text{parse } \mathcal{P} & \text{ as } (c_0, \ldots, c_\ell) \\
\text{parse } \sigma_{\text{agg}}^0 & \text{ as } (\sigma^0, d^0) \\
\text{parse } \sigma_{\text{agg}}^1 & \text{ as } (\sigma^1, d^1) \\
\text{for } j \in [\ell - 1] \\
\text{if } c_j = c^* \\
\text{return } (t, m, (\sigma^0 - \sigma^1), w^0_j - w^1_j) \\
\text{return } \bot
\end{align*}
\]

Fig. 8. A reduction that uses the forking algorithm \( \mathcal{F}_B \) to attack the multi-user existential unforgeability under rerandomized keys of KOTS.

\[ \geq \Pr \left[ \begin{array}{c}
((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(b, \mathcal{P}, t, m, \sigma_{\text{agg}}^0, w^0, \sigma_{\text{agg}}^1, w^1) \leftarrow \mathcal{F}_B^\text{Sign}(\cdot)(\text{pp}, \text{pk}^*): \\
\text{KOTS.wVrfy}(\text{pp}, \tilde{w} \cdot \text{opk}, m, \sigma^0 - \sigma^1) = 1 \\
\land (\exists \sigma'. (m, t, \sigma') \in Q)
\end{array} \right) \]  \hspace{1cm} (18)

\[ \geq \Pr \left[ \begin{array}{c}
((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(b, \mathcal{P}, t, m, \sigma_{\text{agg}}^0, w^0, \sigma_{\text{agg}}^1, w^1) \leftarrow \mathcal{F}_B^\text{Sign}(\cdot)(\text{pp}, \text{pk}^*): \\
\text{KOTS.wVrfy}(\text{pp}, \tilde{\text{opk}}, m, \sigma^0 - \sigma^1) = 1 \\
\land (\exists \sigma'. (m, t, \sigma') \in Q)
\end{array} \right] \]  \hspace{1cm} (19)

\[ \geq \Pr \left[ \begin{array}{c}
((\text{pp}, \text{pk}^*), (\text{pp}, \text{sk}^*)) \leftarrow \text{IG}(1^\lambda); \\
(b, \mathcal{P}, t, m, \sigma_{\text{agg}}^0, w^0, \sigma_{\text{agg}}^1, w^1) \leftarrow \mathcal{F}_B^\text{Sign}(\cdot)(\text{pp}, \text{pk}^*): \\
\tilde{\text{opk}} = \tilde{w} \cdot \text{opk}_t
\end{array} \right] \]  \hspace{1cm} (20)

Here Equation 18 follows by observing that the inputs \( \mathcal{R}_1 \) provides to the forking algorithm are distributed identically to those sampled by IG and \( \mathcal{R}_1 \) is successful whenever the difference of the two signatures verifies. Finally, Equation 20 follows from Lemma 20 and the fact that the forking algorithm only outputs \( b = 1 \) if both multi-signatures verify.
Since KOTS is assumed to be multi-user existentially unforgeable under rerandomized keys, it thus follows that

\[
\text{negl}(\lambda) \geq \Pr \left[ \left( (pp, pk^*), (pp, sk^*) \right) \leftarrow \text{IG}(1^\lambda); \begin{array}{c}
(b, P, t, m, \sigma_0^0, w_0^0, \sigma_1^1, w_1^1) \\
\leftarrow \text{Sign}(pp, sk^*, \cdot, \cdot) (pp, pk^*) \end{array}; \tilde{opk} = \tilde{w} \cdot \tilde{opk}_t, \land \tilde{b} = 1 \right] \quad (21)
\]

Combining Equations 16 and 21 with Equation 10 we can conclude that

\[
\epsilon \leq \frac{p}{|T_\alpha|} + \sqrt{p \cdot \text{negl}(\lambda)}
\]

which implies that \( \epsilon \) is negligible and concludes the proof.

\[\square\]

6 Parameters and performances

Let us first set up our stage with Ethereum blockchain as a running example. It is reported that there are over 300,000 nodes in total [MEJ20], and an Ethereum block is agreed by around 2500 active validators within 10 seconds [eth]. We therefore target \( \rho = 4096 \) signature aggregations which is more than enough to aggregate all the votes from those validators. To illustrate the scalability of our scheme, we also present data for \( \rho = 1024 \) and 8192, respectively.

Our synchronized multi-signature scheme “Squirrel” uses a time parameter \( \tau \), which also defines the height for our labeled binary tree. We give parameters for \( \tau \in \{21, 24, 26\} \), that roughly translate to 0.66, 5 and 21 years of life time for public keys, if we assume each block takes 10 seconds to generate.

6.1 Parameters and space complexity

<table>
<thead>
<tr>
<th>( \rho )</th>
<th>1024</th>
<th>4096</th>
<th>8192</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>( q_{HVC} )</td>
<td>12289</td>
<td>61441</td>
<td>249857</td>
</tr>
<tr>
<td>( q_{KOTS} )</td>
<td>6694913</td>
<td>28930049</td>
<td>57673729</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>( \beta_s )</td>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>( \beta_{agg} )</td>
<td>2048</td>
<td>4096</td>
<td>8192</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>41</td>
<td>44</td>
<td>46</td>
</tr>
</tbody>
</table>

Table 2. Parameter sets

We propose three parameter sets each targeting 112 bits security as in Table 2. For the rest of the section, we will use \( \rho = 4096 \) as an example. We set \( q_{KOTS} = 28930049 \) and \( q_{HVC} = 61441 \) respectively; both are NTT friendly for our choice of \( n = 512 \). This implies that our \( \text{bin}(\cdot) \) maps an \( \mathcal{R}_{HVC} \) element into 16 elements; and maps an \( \mathcal{R}_{KOTS} \) element into 25 elements. Note that \( \text{bin}(\cdot) \) does not map a random elements in \( \mathcal{R}_{HVC} \) uniformly to binary polynomials space; neither does our scheme require such a uniformity. It takes \( n \left\lfloor \log q_{HVC} \right\rfloor = 8192 \) bits, or 1 kilobyte to represent an element in \( \mathcal{R}_{HVC} \). A Squirrel signature consists of three components:
– an HVC decommitment of $2\tau$ path nodes and adjacent nodes, where each node consists of $\lceil \log q_{HVC} \rceil$ many $R_{q_{HVC}}$ elements with bounded norm $\beta_{agg}$;
– an KOTS public key and its sibling public key, which are hashed into the committed leaves. This consists of $4\lceil \log q_{KOTS} \rceil$ many $R_{q_{KOTS}}$ elements with bounded norm $\beta_{agg}$;
– an KOTS signature, that consists of $\gamma$ many $R_{q_{KOTS}}$ elements with bounded norm $\beta_{\sigma}$.

For a fresh signature (prior to aggregation), the polynomials in each node are all binary, derived from a decomposition of a single $R_{q_{HVC}}$ element. It is therefore sufficient to represent the node with 1 kilobyte of data. In addition, since the signature has not been aggregated, one will be able to derive the nodes along the path with the adjacent leaf and $\tau$ adjacent nodes. This reduces the required number of nodes to $\tau$, excluding the root (a.k.a. the public key). In total, we require $\tau$ kilobytes storage for a path when the signature is not aggregated.

During aggregation, we multiply the binary polynomials from different users but at a same position from the tree with randomizers, and sum up the products. This gives us a total number of $2\tau\lceil \log q_{HVC} \rceil$ polynomials, where each polynomial has an infinity norm bound $\alpha \rho$. In practice, it is possible to derive a better bound $\beta_{agg} = 4096$ if we assume that the polynomials in the decommitments are all binary. We defer this discuss to Section 6.3. An aggregated path requires a maximum $2\tau\lceil \log q_{HVC} \rceil n(\log \beta_{agg} + 1)$ bits, or $26\tau$ KB of data.

For the KOTS, prior to aggregation, each public key consists of $2 R_{q_{KOTS}}$ elements, of a combined size of 3.1 KB. During aggregation, each public key is decomposed into $\lceil \log q_{KOTS} \rceil$ many $R_{q_{HVC}}$ elements. The aggregated polynomials also have a same norm bound of $\beta_{agg}$. That is, an aggregated KOTS public key requires a maximum $2\lceil \log q_{KOTS} \rceil n(\log \beta_{agg} + 1)$ bits, or 40.6 KB of data.

A non-aggregated KOTS signature requires $\gamma$ ring elements with a norm bound of $2\beta_{s}$, or $n \gamma(\lceil \log(2\beta_{s}) \rceil + 1) = 22$ KB. We defer to Section 6.3 for how $\beta_{s}$ is chosen. An aggregated KOTS signature requires $\gamma$ ring elements with a norm bound $\beta_{\sigma} = 2\rho \alpha \beta_{s}$, which is $n \gamma(\lceil \log(\beta_{\sigma}) \rceil + 1) = 66$ KB.

Putting everything together, our scheme’s public key is the root of the tree that uses 1 kilobytes. An un-aggregated signature requires $\tau + 28$ kilobyte, consists of the path to the root, which is $\tau$ nodes; two KOTS public keys of 6.2 kilobytes, and a KOTS signature that requires 22 KB. An aggregated signature requires $2\tau + 147$ kilobytes, consists of the HVC decommitment, which is $2\tau$ number of nodes; two aggregated KOTS public keys of 81.2 KB, and an KOTS signature of 66 KB.

We summarize the characteristics of our scheme in Table 3.

<table>
<thead>
<tr>
<th>$\rho$: $#$ sig</th>
<th>$\tau$: tree height</th>
<th>Life cycle</th>
<th>PK size</th>
<th>Sig size</th>
<th>Max AggSig size</th>
<th>Improvement*</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>8 months</td>
<td>45 KB</td>
<td>572 KB</td>
<td>14%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>5 years</td>
<td>48 KB</td>
<td>635 KB</td>
<td>5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>21 years</td>
<td>50 KB</td>
<td>677 KB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>8 months</td>
<td>49 KB</td>
<td>693 KB</td>
<td>74%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>5 years</td>
<td>52 KB</td>
<td>771 KB</td>
<td>71%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>21 years</td>
<td>54 KB</td>
<td>823 KB</td>
<td>69%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>8 months</td>
<td>53 KB</td>
<td>762 KB</td>
<td>85%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>5 years</td>
<td>57 KB</td>
<td>850 KB</td>
<td>84%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>21 years</td>
<td>59 KB</td>
<td>908 KB</td>
<td>83%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Improvement over $\rho$ signatures of Falcon-512 with signature size of 666 bytes.

Table 3. Space complexity of Squirrel.
6.2 Computational complexity and benchmarks

We implement our scheme and release the source code to the open domain\(^4\). We report benchmark result for the case of \(\rho \in \{1024, 4096\}\) and \(\tau = 21\); and give estimations for performance of \(\tau = 24\) and 26. We run the benchmark over an AMD 5900x CPU with 12 cores, and with parallelization option turned on.

<table>
<thead>
<tr>
<th></th>
<th>(\rho = 1024)</th>
<th>(\rho = 4096)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathcal{R}_\text{sync NTT})</td>
<td>4.1 (\mu s)</td>
<td>6.9 (\mu s)</td>
</tr>
<tr>
<td>(\mathcal{R}_\text{sync NTT mul.})</td>
<td>197 (n s)</td>
<td>260 (n s)</td>
</tr>
<tr>
<td>(\mathcal{R}_\text{sync NTT})</td>
<td>5.8 (\mu s)</td>
<td>5.43 (\mu s)</td>
</tr>
<tr>
<td>(\mathcal{R}_\text{sync NTT mul.})</td>
<td>508 (n s)</td>
<td>413 (n s)</td>
</tr>
<tr>
<td>ter-bin mul.</td>
<td>1.5 (\mu s)</td>
<td></td>
</tr>
<tr>
<td>HVC hash</td>
<td>69 (\mu s)</td>
<td>107 (\mu s)</td>
</tr>
<tr>
<td>KOTS hash</td>
<td>111 (\mu s)</td>
<td>143 (\mu s)</td>
</tr>
<tr>
<td>gen randomizer</td>
<td>1.8 (\mu s)</td>
<td></td>
</tr>
<tr>
<td>path randomization</td>
<td>274 (\mu s)</td>
<td>283 (\mu s)</td>
</tr>
<tr>
<td>1024 paths aggregation</td>
<td>680 (m s)</td>
<td>834 (m s)</td>
</tr>
<tr>
<td>1024 paths batch verification</td>
<td>20 (m s)</td>
<td>30 (m s)</td>
</tr>
</tbody>
</table>

Table 4. Microbenchmarks

**Microbenchmarks** We report the computation cost in Table 4. The main units of computations are

- A generic \(\mathcal{R}_\text{sync}\) multiplication consists of converting both input polynomials into their NTT form \((O(n \log n))\), and conducting a coordinate-wise multiplication \((O(n))\), and convert the result back to integer polynomials. Denote this cost by \(c_1\).
- A generic \(\mathcal{R}_\text{sync}\) multiplication consists of converting both input polynomials into their NTT form \((O(n \log n))\), and conducting a coordinate-wise multiplication \((O(n))\), and convert the result back to integer polynomials. Denote this cost by \(c_2\).
- Multiply a binary polynomial with a fixed weight ternary polynomial. Denote this cost by \(c_3\).

As examples, our hash function takes \(2\lceil \log q_{\text{HVC}} \rceil\) number of generic ring multiplications; randomizing a node takes \(\lceil \log q_{\text{HVC}} \rceil\) ternary ring multiplications. Concretely, our implementation reports that

- Hashing two child nodes into a parent node takes 107 microsecond;
- Hashing an KOTS public key into a leaf node takes 143 microsecond.

This is a lot better than \(2 \log q\) number of multiplications due to a) parallelization, and b) the fact that hash parameters are already in the NTT form already; and that we only need to perform a single inverse NTT at the end.

**Full Picture** Similar to hash based signature schemes \([\text{BDH11, BHK}^+19]\), the key generation stage is the most expensive one in our case. It involves generating \(2^\tau\) KOTS keys, each costs \(2^\gamma\) generic ring multiplications; and the whole tree, at a cost of \(2^\tau\) node hashes and \(2^\tau\) leave hashes. Overall cost is \(2^\tau(2\lceil \log q_{\text{HVC}} \rceil + 2\lceil \log q_{\text{KOTS}} \rceil)c_1 + 2^{\tau+1}\gamma c_2 = 2^{\tau+1}((\lceil \log q_{\text{HVC}} \rceil + \lceil \log q_{\text{KOTS}} \rceil)c_1 + \gamma c_2)\).

\(^4\) https://github.com/zhenfeizhang/sync-multi-sig
Table 5. Estimated cost with cache

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\tau$</th>
<th>Offline signing</th>
<th>Offline signing with cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ammortized</td>
<td>worst-case</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$2^\tau$ nodes</td>
<td>$2^\tau$ nodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$4 \log q_{\text{HVC}}</td>
<td>c_1$</td>
</tr>
<tr>
<td>1024</td>
<td>storage</td>
<td>7 MB</td>
<td>112 MB</td>
</tr>
<tr>
<td>21</td>
<td>41 $\mu$s</td>
<td>43 sec</td>
<td>42 ms</td>
</tr>
<tr>
<td>26</td>
<td>23 min</td>
<td>1.3 sec</td>
<td>83 ms</td>
</tr>
<tr>
<td>4096</td>
<td>storage</td>
<td>8 MB</td>
<td>128 MB</td>
</tr>
<tr>
<td>21</td>
<td>52 sec</td>
<td>0.4 sec</td>
<td>25 ms</td>
</tr>
<tr>
<td>24</td>
<td>7 min</td>
<td>1.6 sec</td>
<td>99 ms</td>
</tr>
<tr>
<td>26</td>
<td>28 min</td>
<td>1.6 sec</td>
<td>99 ms</td>
</tr>
</tbody>
</table>

Table 6. Benchmark results and estimations

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\tau$</th>
<th>Key Generation</th>
<th>Online signing</th>
<th>Aggregation*</th>
<th>Verification*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$2^{\tau+1}(\lceil \log q_{\text{HVC}} +</td>
<td>\log q_{\text{KOTS}}</td>
<td>c_1 + \gamma c_2 \rceil) \gamma c_3 \gamma c_3 + \rho(2\tau + 2)</td>
<td>\log q_{\text{HVC}}</td>
</tr>
<tr>
<td>1024</td>
<td>21</td>
<td>4 min</td>
<td>2.1 ms</td>
<td>1.2 sec</td>
<td>19.5 ms</td>
</tr>
<tr>
<td>24</td>
<td>32 min</td>
<td>1.4 sec</td>
<td>22 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>2 hour</td>
<td>1.5 sec</td>
<td>24 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>21</td>
<td>4.5 min</td>
<td>2.3 ms</td>
<td>1.4 sec</td>
<td>31 ms</td>
</tr>
<tr>
<td>24</td>
<td>36 min</td>
<td>1.6 sec</td>
<td>36 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>2.4 hour</td>
<td>1.8 sec</td>
<td>38 ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*: Aggregate and batch verify 1024 signatures. **: Estimations based on extrapolating $\tau = 21$ data.

Squirrel is an online/offline signature scheme. A speed sensitive signer may store the whole tree and avoid the entire offline phase. The online signing time becomes simply generating the OTS signature, which takes $\gamma$ generic ring multiplications at a cost of $\gamma c_2$. The signer will need to store the whole tree which consists of $2^\tau$ nodes and $2^\tau$ leaves, which translates into 5.1 gigabytes, 41 gigabytes and 164 gigabytes of data for each of the parameter settings respectively.

A space sensitive signer may store the last used path (and its adjacent nodes); and update it to its current path on-the-fly. Observe that any node will not be computed more than twice: the first time is during tree generation, and the second time is when it is firstly required in a path (and its adjacent nodes). Once a node is no longer required by a path nor the adjacent nodes, it will never be required again. Therefore, the amortized cost for each signatures will be 2 hashes (total number of nodes divided by total number of leaves). Since our hash function uses $2\lceil \log q_{\text{HVC}} \rceil = 32$ generic ring multiplications, the amortized cost is 64 ring multiplications to update the path, and $\gamma = 44$ generic ring multiplications for KOTS signing.

In practice, the real bottleneck is the worst-case scenario, in which the signer will need to generate the signature for leaf with index $2^{\tau-1}$ (i.e., the first leaf of the second sub-tree) within a block interval. Concretely, the signer will need to generate $2^\tau - 2$ nodes, or equivalently, conduct $(2^{\tau+1} - 4) \log q_{\text{HVC}} \approx 2^{\tau+1} \log q_{\text{HVC}}$ generic ring multiplications. There are a few straightforward method to alleviate the situation. First, as an online/offline scheme, the signer always knows exactly when it will use leave $2^{\tau-1}$. Therefore, it will be able to pre-compute this path offline. Secondly, if the signer is allowed some cache, it can store the top $h$ levels of the tree, or $2^{h+1} - 2$ nodes, excluding the root. Accordingly, at the worst-case, the signer will need to online compute two sub-trees whose roots are the nodes at $h$-th level. That is $2(2^{\tau-h+1} - 1) \approx 2^{\tau-h+2}$ nodes, or $2^{\tau-h+3} \log q_{\text{HVC}}$ generic ring multiplications in total. It also implies that the worst-case complexity will be reduced by half for every additional level of nodes we cache. Table 6 gives a rough estimation of cache versus signing time.
To aggregate $\rho$ signatures, the aggregator will need to multiply each path with some randomizers. There are $\rho(2\tau + 2)$ number of nodes, leaves and KOTS public key nodes, combined; each requires $\lceil \log q_{\text{HVC}} \rceil$ ternary ring multiplications. The aggregator will also need to randomize-then-aggregate KOTS signatures, which also incurs $\rho$ generic ring multiplications. The total cost will be $\rho c_2 + \rho (2\tau + 2) \lceil \log q_{\text{HVC}} \rceil c_3$.

To verify an (aggregated) signature, the verifier will need to check that the path is valid with regard to the root of the tree. This takes $2 \lceil \log q_{\text{HVC}} \rceil \tau$ number of multiplications to check the path; and $\rho$ number of multiplications to aggregate the public keys. In addition, the KOTS verification also uses $2\gamma$ ring multiplications.

### 6.3 Security estimation

**Combinatorials** First we need the randomizers to be sampled from a large space as per Lemma 2, i.e., $\binom{n}{r} 2^a \geq \lambda$. Setting $\alpha = 20$, i.e., the randomizer are sampled from the set of ternary polynomials with 20 non-zero entries, we have $\binom{n}{r} \cdot 2^a > 2\lambda$.

Then, we discuss how we arrived to $\beta_{\text{agg}} = 4096$. We assume that the aggregator may be malicious, that is, it can cherry pick their signatures so that, for a given node (i.e., an $R_q$ element) for a given path, all the signatures will have 1s at a same index. Even so, the randomizers are outputs from the random oracle, where there are $\alpha$ number of $\pm 1$s with equal probability. Therefore, for an aggregated polynomial, each coefficient can be seen as a sum of $\alpha \rho$ number of random elements in $\{-1, 1\}$. We need to set a bound $\beta_{\text{agg}}$ such that, the probability that all coefficients for all nodes are bounded by $\beta_{\text{agg}}$ in absolute value with overwhelming probability, i.e.,

$$2\tau \lceil \log q_{\text{HVC}} \rceil n \Pr \left( \forall i \in [\alpha \rho], b_i \in \{-1, 1\} : \left| \sum_{i=1}^{\alpha} b_i \beta_{\text{agg}} \right| \leq 2^{-\lambda} \right) \leq 2^{-\lambda}$$

For $\alpha = 20$ we are able to set $\beta_{\text{agg}} = 4096$. Additionally, we require that $2\beta_{\text{agg}} < q_{\text{HVC}}/2$ so that in Lemma 9 the extracted vector is indeed a short solution to the SIS problem.

The messages are hashed into $T_{\beta_s}$. Therefore, we need to set $\beta_s = 44$ so that $|T_{\beta_s}| > 2^{2\lambda}$. Note that we need $(4\rho + 8)\alpha \beta_s < q_{\text{KOTS}}/2$ so that in Lemma 17 the extracted vector is indeed a short solution to the SIS problem. Per Lemma 17, we then need to set $\gamma = 44$ such that $2^{(3\lambda+1)/n_\gamma} q_{\text{KOTS}}^{1/\gamma} \leq 3/2$.

**Lattice attacks** For a root Hermite factor $c \leq 1.005$, the LWE-estimator [APS15] reported that BKZ [CN11] will be able to find a short vector for a block size of 286. Such a lattice reduction requires 112 bits operations under a realistic sieving model [BDGL16].

Our HVC scheme requires that the $\text{SIS}_{R_q,2\lceil \log q_{\text{HVC}} \rceil,4\rho\alpha}$ problem is hard as per Theorem 5 and Lemma 9, for $q_{\text{HVC}} = 61441$, $\rho \in [4096]$ and $\alpha = 20$. An SIS becomes easier when the target solution is longer, therefore it is sufficient to analyze the case $\rho = 4096$. This instantiation yields a lattice of dimension $(2 \lceil \log q_{\text{HVC}} \rceil + 1)n$ and determinant $q_{\text{HVC}}^n$. As per [DM08], a lattice reduction algorithm will find a short vector of $2^{\sqrt{n} \log q_{\text{HVC}} \log \rho} \alpha \beta_s$ for some root Hermite factor $c$ that depends on the lattice reduction algorithm. In the meantime, the vector we are searching for has an infinity norm of $\beta_{\text{agg}}$, which means its $\ell_2$ norm is bounded by $t_{\text{HVC}} = \sqrt{2n} \log q_{\text{HVC}} \beta_{\text{agg}}$. With our choice of parameters, a lattice reduction algorithm will be able to find this target vector for $c < 1.005$.

Last, we analysis the hardness of the $\text{SIS}_{R_{\text{KOTS}},\gamma,(4\rho+8)\alpha \beta_s}$ assumption for our KOTS scheme as per Lemma 17. This follows a similar analysis as the above SIS analysis. Here, we have a
lattice of dimension $(\gamma + 1)n$ and determinant $q^{KOTS}$. An aggregated signature has an infinity norm bound of $(4\rho + 8)\alpha\beta_s$, which implies $t_{HOST} = \sqrt{n}(4\rho + 8)\alpha\beta_s$ in $\ell_2$ norm. We also require that $t_{HOST} < c \dim^{2\sqrt{n} \log q^{KOTS} \log e}$ so that BKZ cannot solve this instance of SIS problem. With our parameter sets we have $c < 1.004$.

References


BK20. Dan Boneh and Sam Kim. One-time and interactive aggregate signatures from lattices. 2020. 4. 4


