Chipmunk: Better Synchronized Multi-Signatures from Lattices

Nils Fleischhacker[1]*, Gottfried Herold[2]**, Mark Simkin[2]***, and Zhenfei Zhang[2]†

[1] Ruhr University Bochum
[2] Ethereum Foundation

November 26, 2023

Abstract. Multi-signatures allow for compressing many signatures for the same message that were generated under independent keys into one small aggregated signature. This primitive is particularly useful for proof-of-stake blockchains, like Ethereum, where the same block is signed by many signers, who vouch for the block's validity. Being able to compress all signatures for the same block into a short string significantly reduces the on-chain storage costs, which is an important efficiency metric for blockchains.

In this work, we consider multi-signatures in the synchronized setting, where the signing algorithm takes an additional time parameter as input and it is only required that signatures for the same time step are aggregatable. The synchronized setting is simpler than the general multi-signature setting, but is sufficient for most blockchain related applications, as signers are naturally synchronized by the length of the chain.

We present Chipmunk, a concretely efficient lattice-based multi-signature scheme in the synchronized setting that allows for signing an a-priori bounded number of messages. Chipmunk allows for non-interactive aggregation of signatures and is secure against rogue-key attacks. The construction is plausibly secure against quantum adversaries as our security relies on the assumed hardness of the short integer solution problem.

We significantly improve upon the previously best known construction in this setting by Fleischhacker, Simkin, and Zhang (CCS 2022). Our aggregate signature size is 5.6× smaller and for 112 bits of security our construction allows for compressing 8192 individual signatures into a multi-signature of size around 136 KB. We provide a full implementation of Chipmunk and provide extensive benchmarks studying our construction's efficiency.

1 Introduction

Multi-signatures [IN83, MOR01] allow for compressing distinct signatures for the same message generated by different signers into one small aggregated signature. Such signature schemes are a powerful tool in distributed systems, like blockchains, where parties vouch for the validity of messages on the network by signing them. Rather than storing an amount of signatures that is linear in the number of parties that vouched for a specific messages, multi-signatures allow for storing a much shorter string that vouches for a message on behalf of all signers simultaneously. Popular proof-of-stake blockchains like Ethereum[3] and DFinity[4] employ multi-signatures at the core of their consensus layer.

The most popular multi-signature scheme used in practice is a construction due to Boneh, Gentry, Lynn, and Shacham [BGLS03] based on a signature scheme due to Boneh, Lynn, and

[*] mail@nilsfleischhacker.de. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

[**] gottfried.herold@ethereum.org

[***] mark.simkin@ethereum.org

[†] zhenfei.zhang@ethereum.org

[3] https://github.com/ethereum/annotated-spec/blob/master/phase0/beacon-chain.md#attestation

[4] https://dfinity.org/whitepaper.pdf

Shacham (BLS) [BLS01]. Their resulting multi-signatures are extremely small, but the security of their construction relies on the assumed hardness of computing discrete logarithms over pairing-friendly groups. It was shown by Shor [Sho94] that the discrete logarithm problem can be solved efficiently by quantum computers, meaning that any cryptographic primitive basing its security on such an assumption is insecure in the presence of a quantum adversary.

Luckily, not all computational hardness assumptions are created equal and some seem to remain hard in the presence of quantum adversaries. Building multi-signatures from computational hardness assumptions that withstand quantum adversaries is both a theoretically and practically important question. While it may not be clear when practically relevant quantum computers will appear, it is important to have secure alternatives for important cryptographic primitives ahead of time.

One class of cryptographic hardness assumptions that seems to be particularly resilient against quantum adversaries is lattice-based cryptography. Two of the three post-quantum signature schemes that were selected for standardization by NIST in 2022 base their security on hardness assumptions related to lattices and, not surprisingly, there has also been significant interest in constructing multi-signatures from lattice hardness assumptions [ES16, FH19, MJ19, PD20, KD20, FH20, DOTT21, BTT22, FSZ22a, BT23]. The current multi-signature constructions, however, do still have significant drawbacks that hinder their practical deployment. The constructions of El Bansarkhani and Sturm [ES16] and Ma and Jiang [MJ19] assume that the keys of all signers are generated honestly. This is not a realistic assumption as an adversarial signer could aim to perform a rogue-key attack by generating a malformed verification key that depends on honest signers' keys and allows for forging aggregated signatures, which falsely claim that both the malicious and the honest parties signed a message that was not actually signed by them. The scheme of Kansal and Dutta [KD20] was shown to be insecure by Liu et al. [LTT20]. The constructions of Fukumitsu and Hasegawa [FH19, FH20], Ma and Jiang [MJ19], and Peng and Du [PD20], and Boschini, Takahashi, and Tibouchi [BTT22] all require interaction between the signers for generating a joint multi-signature. Such an interaction between independent signers is difficult to realize in many distributed systems as the signers may be online at different times and may even not know of each others existence. The construction of Boudgoust and Takahashi [BT23] has aggregate signatures, which have a size that linearly depends on the number of aggregated signatures.

Recently, Fleischhacker, Simkin, and Zhang [FSZ22a] presented a lattice-based multi-signature construction named Squirrel, which allows for non-interactive aggregation and is secure against rogue-key attacks. They consider a simplified setting, where signer's keys are only able to sign an a-priori bounded number of messages and where signers are synchronized in the sense that aggregation only has to work for signatures that were generated for the same time step and same message. This simplified setting is still sufficiently strong for most blockchain applications, where signers do not sign more than one message per block and are naturally synchronized by the length of the current chain. While an a-priori bound on the number of messages that can be signed may seem like a strong limitation, one can simply set this number large enough, e.g. to 2^{24} which would allow a signer to sign a message every 10 seconds for 5 years non-stop. Aiming for 112 bits of security, their individual signatures are roughly 50 KB large and aggregating 4096 signatures results in a multi-signature that are 771 KB large.

Squirrel represents a significant step forward for multi-signature schemes that are plausibly secure in the presence of a quantum adversary and are concretely efficient. For real-world practical scenarios their aggregated signatures seem, however, still too large to be really used. As a point of reference, a full Ethereum block is on average less than 130 KB large[5], which would mean that one block could not even fit a single multi-signature.

1.1 Our Contribution

In this work we present Chipmunk[6], a multi-signature scheme in the synchronized setting [GR06, AGH10, HW18, DGNW20] with an a-priori bound on the number of signatures that can be issued per key. We aim for the exact same setting as Squirrel [FSZ22a], but provide both theoretical and practical improvements.

On the theoretical side, we strengthen the security notions for multi-signatures by requiring that aggregation involving malformed but verifying adversarial signatures will succeed with high probability. In Squirrel, aggregation was only required to work for honestly generated individual signatures. In principle, their security model would allow an adversary to perform a denial-of-service attack against the signature aggregation procedure by providing a single verifying, but malformed signature. In a real-world distributed system, such an attack on liveness would be highly problematic. We strengthen their security definitions to formally ensure that successfully verifying individual signatures will be successfully aggregated, even if they are chosen maliciously.

On the practical side, our scheme Chipmunk produces smaller individual and aggregated signatures, when compared to Squirrel. In terms of computational efficiency metrics, Chipmunk either significantly outperforms Squirrel or remains comparable in speed. In terms of bandwidth, Chipmunk's aggregate signatures are smaller by a factor of 5.6×, when compared to Squirrel. For keys that can generate 2^{21} signatures, an individual Chipmunk signatures is 37 KB and aggregating 8192 signatures results in an aggregate signature that is 136 KB large.

We have fully implemented Chipmunk and provide extensive benchmarks and comparisons to Squirrel in Section 7. Chipmunk currently is the most concretely efficient multi-signature known that is based on assumptions that are assumed to remain valid in the presence of a quantum adversary.

1.2 Technical Overview

Conceptually, Chipmunk closely follows the blueprint that was introduced by Fleischhacker, Simkin, and Zhang [FSZ22a]. Recall that in their work and in ours we only aim to issue an a-priori bounded number of signatures, meaning that key generation is parameterized by τ and produces a public key that can be used to sign 2^τ messages. Further recall that we are in the synchronized setting, meaning that we only aim to aggregate signatures for the same message that were issued at the same time step.

In Squirrel, each signer's public key pk is a homomorphic vector commitment of length 2^τ, where position i commits to pk^i, which is the public key of a key-homomorphic one-time signature scheme. To sign message m at time step i, the signer opens the commitment pk to pk^i at position i and uses the corresponding one-time signing key to sign the message m. The signature is a vector itself that consists of pk^i, the corresponding opening, and the signature of m under this one-time key. To verify that a message m was signed for time step i, the verifier checks that the given public key pk^i is a valid opening of the i-th position of the corresponding signer's public key pk and that the given signature verifies for message m under the public key pk^i.

[5] https://etherscan.io/chart/blocksize

[6] Smaller than squirrels, cuter than squirrels.

Aggregation of such signatures is performed by exploiting the homomorphic properties of the vector commitment and the one-time signature scheme. To aggregate signatures, roughly speaking, one simply adds up all the individual commitment openings, the one-time keys, and the corresponding one-time signatures. The homomorphism of the vector commitment scheme ensures that the sum of openings is a valid opening for the sum of committed messages, i.e. the one-time public keys, under the sum of commitments. The key-homomorphic property of the one-time signature scheme ensures that the sum of signatures for the same message verifies under the sum of one-time public keys. Chipmunk follows this blueprint, but improves upon all building blocks that are being used and thereby significantly reduces the multi-signature size of Chipmunk.

Key-Homomorphic One-Time Signatures. Squirrel uses a key-homomorphic one-time signature scheme that is similar to those of Boneh and Kim [BK20] and Lyubashevsky and Micciancio [LM08]. The details of the construction are not relevant for now. For Chipmunk, we use almost the exact same scheme, but observe by carefully inspecting their original security proof that a minor modification of the construction used in Squirrel allows for the proof to produce much tighter parameters and thus smaller signatures.

Homomorphic Vector Commitments. The vector commitment used by Squirrel is a homomorphic analogue of the classical Merkle tree construction. To make a Merkle tree homomorphic, the idea is to employ a homomorphic hash function to compute the node's values of the tree. Now when adding two trees node-wise, one obtains a new valid tree. Ajtai [Ajt99] introduced such a homomorphic hash function based on the short integer solution problem. The main difficulty with using this hash function is the fact that hash output values need to be transformed into valid hash input values in a way that is efficient and maintains the homomorphic properties we would like our tree to have. Without going into the details, the problem is that inputs for Ajtai's hash function need to have small norm, but outputs have potentially very large norms. Fleischhacker, Simkin, and Zhang [FSZ22a] solved this problem by effectively performing a binary decomposition of the hash function's output values, which resulted in vectors with infinity norm one, which could then again be used as hash function inputs. In Chipmunk, we generalize their trick of decomposing values into binary vectors to decomposition into vectors of small norm. While conceptually simple, we show that this change allows us to significantly reduce the size of our homomorphic vector commitment openings.

Encoded Openings. Given a Merkle tree, one can provide an opening for leaf i by revealing all nodes that are adjacent to those on the path from leaf i to the root. The nodes on the path can then be computed from the given information. Computing openings for the *unaggregated* homomorphic vector commitment construction of Squirrel as well as ours works essentially the same way. For Squirrel's and our *aggregated* vector commitments, the situation is unfortunately different. Once we start aggregating trees or openings, we need to explicitly compute the nodes on the path and provide them as part of the aggregated opening, which doubles the size of our openings if done naively.

In Chipmunk, we present a new compression algorithm, inspired by Babai's nearest plane algorithm [Bab86], which allows us to compress the size of our openings. Instead of additionally sending the nodes on the path, we only send the adjacent nodes and some small hints, which allow us reconstruct all needed node values. We believe that this technique may be of independent interest and could find applications outside of our construction.

Chipmunk Multi-Signatures. Our construction of multi-signatures from vector commitments and one-time signatures follows the blueprint that was already outlined above. One thing we glossed over so far are rogue-key attacks. If we were to simply add up individual signatures, then our scheme would be susceptible to an adversary that first sees the honest parties public keys and then generates a malicious public key that allows for forging multi-signatures involving honest signers for arbitrary messages. To avoid this type of attack, the individual signatures are multiplied by randomizer values before being added up. These randomizing values need to be from a space that is large enough to be unpredictable for the adversary, but cannot be too large as the multi-signature scheme's efficiency would deteriorate. In Squirrel, aggregation was a one-attempt process. In Chipmunk, we repeatedly choose fresh randomization values and attempt aggregation until the aggregated signature is "small enough". We show that doing this alllows us to reduce signature size without affecting the security or the performance of our construction.

We note that we grossly oversimplified many things in our above overview and that the precise construction and our improvements are more technically involved. While each individual improvement may seem conceptually simple, they all add up to significantly improved signature sizes, resulting in the by far most efficient multi-signature scheme from lattice assumptions to date.

New Results in Full Version. We note that the full version of this paper contains a new result, the encoding of vector commitment openings mentioned above, which is not present in our CCS 2023 proceedings version of this paper. This new result improves our signature sizes by roughly a factor of 1/5, when compared with the aggregate signature sizes we achieved in our proceedings paper. You are currently reading the full version.

2 Preliminaries

This section introduces notation, some basic definitions and a few basic lemmas that we will use throughout this work. We denote by $\lambda \in \mathbb{N}$ the security parameter and by $\mathsf{poly}(\lambda)$ any function that is bounded by a polynomial in λ. A function f in λ 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 $\mathsf{negl}(\lambda)$ any negligible function. An algorithm is PPT if it is modeled by a probabilistic Turing machine with a running time bounded by $\mathsf{poly}(\lambda)$.

Let S be a set. We write $x \leftarrow S$ for the process of sampling an element of S uniformly at random. Let T be a full binary tree of depth d. We denote the root node of T by the empty string ϵ, 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 S is represented by a labeling function $\mathsf{label}\colon \{0,1\}^{\leq d} \to S$.

Let $\boldsymbol{v}, \boldsymbol{u}$ be vectors of length m. We throughoutly use 1-based indices in this work. We write $\boldsymbol{v}^\intercal$ to denote the transpose of $\boldsymbol{v}$ and v_i to denote the i-th entry in the vector for $1 \leq i \leq m$. We generalize this notation and write $\boldsymbol{v}_{<i}$ to denote the $(i-1)$-length prefix of $\boldsymbol{v}$. We use the same notation for a bit-string s, denoting by s_i the i-th bit and by $s_{<i}$ the prefix consisting of the first $i-1$ bits of s. For $0 \leq t \leq 2^\tau - 1$ we denote by $\mathsf{bin}_\tau(t) \in \{0,1\}^\tau$ the big-endian binary decomposition of t (possibly with leading zeros to ensure a fixed length of τ). For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\}$.

Our concrete construction works over a power-of-two cyclotomic polynomial ring. Let $\Phi_{2n} = X^n + 1$ be the cyclotomic polynomial with n a power of two. We work in the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$. For the purpose of taking norms and transmitting data, we represent elements of $\mathcal{R}$ as n-dimensional vectors $\mathbb{Z}^n$ with $(c_1, \ldots, c_n)^\intercal \in \mathbb{Z}^n$ representing the ring element $\sum_{i=1}^n c_i X^{i-1}$. For any odd prime number q, we always represent $\mathbb{Z}_q$ by the set $\{-\frac{q-1}{2}, \ldots, \frac{q-1}{2}\}$ centered around 0. For an odd prime q, we denote by $\mathcal{R}_q$ the quotient ring of $\mathcal{R}$ modulo q, represented by vectors in $\mathbb{Z}_q^n$. Whenever we need to take representatives to view these as elements from $\mathcal{R}$, we do so by taking representatives centered around 0 as above. For efficiency reasons, our parameter choice will always satisfy $q \equiv 1 \bmod 2n$, so we can use more efficient NTT-based multiplication in $\mathcal{R}_q$. Let $\boldsymbol{c} \in \mathcal{R}$ be a ring element with coefficients $(c_1, \ldots, c_n)$. We work, unless specified otherwise, with the $\|.\|_\infty$-norm. We define $\|\boldsymbol{c}\| \coloneqq \|\boldsymbol{c}\|_\infty = \max_i |c_i|$ and $\|\boldsymbol{c}\|_1 = \sum_i |c_i|$ on $\mathcal{R}$ by taking the norm of the coefficient vector in the monomial basis. We extend these definitions to norms on $\mathcal{R}_q$ by taking representatives in $\mathcal{R}$ using coefficients in $\{-\frac{q-1}{2}, \ldots, \frac{q-1}{2}\}$. We also extend the definition of $\|.\|_\infty$ to $\mathcal{R}^m$ for any m by $\|\boldsymbol{c}\|_\infty = \max_i \|\boldsymbol{c}_i\|_\infty$.

Our convention is that mixed multiplication of an element from $\mathcal{R}$ with an element from $\mathcal{R}_q$ gives an element from $\mathcal{R}_q$, thereby viewing $\mathcal{R}_q$ as an $\mathcal{R}$-module (see Definition 3 below).

We denote by $\mathcal{B}_{\beta,q}$ the ball $\mathcal{B}_{\beta,q} = \{a \in \mathcal{R}_q \mid \|a\| \leq \beta\}$. We are only interested in the case $\beta < \frac{q}{2}$. By $\mathcal{T}_\alpha = \{a = (a_1 + a_2 \cdot X + \cdots + a_n X^{n-1}) \in \mathcal{R} \mid \|a\|_\infty \leq 1 \wedge \sum_{i=1}^n |a_i| = \alpha\}$ we denote the set of polynomials with ternary coefficients, i.e. coefficients from $\{-1, 0, 1\}$, and with exactly α non-zero coefficients.

Observe that for our choices of ring $\mathcal{R}$ and norm, for any $a \in \mathcal{R}$, we have $\|a\| = \|X \cdot a\|$, because multiplication by X acts on the coefficient vector as a cyclic shift (up to sign). For such rings and norms, we can make use of the following simple lemma that allows us to bound the norm of the product of two polynomials.

Lemma 1 ([Mic07]). *Let $a, b \in \mathcal{R}$ be two polynomials. Then $\|b \cdot a\| \leq \|a\|_1 \cdot \|b\|$.*

The security of our constructions relies on the hardness of the short integer solution problem defined over rings as follows.

Definition 2 (Ring Short Integer Solution Problem). *For a ring $\mathcal{R}$ and parameters $\mu, q, \beta \in \mathbb{N}$, the $\mathsf{SIS}_{\mathcal{R},q,\mu,\beta}$ problem is hard if for all PPT algorithms $\mathcal{A}$ it holds that*

$$\Pr[\boldsymbol{a} \leftarrow \mathcal{R}_q^\mu; \boldsymbol{s} \leftarrow \mathcal{A}(\boldsymbol{a})\colon \boldsymbol{s} \in \mathcal{B}_{\beta,q}^\mu \setminus \{\boldsymbol{0}\} \wedge \boldsymbol{a}^\intercal \boldsymbol{s} = 0] \leq \mathsf{negl}(\lambda)$$

$\mathcal{R}$-modules. In order to aggregate signatures, we will be taking linear combinations of individual elements, where for security reasons the coefficients need to be from a sufficiently large space. We use the ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ for those coefficients. This means that for both signatures and certain intermediate objects appearing during our constructions, we need to be able to both add them together and to multiply them with elements from $\mathcal{R}$. Recall that this is precisely captured by the notion of an $\mathcal{R}$-module, so let us recall some relevant notions here for convenience to the reader.

Definition 3 ($\mathcal{R}$-module). *For a commutative ring $\mathcal{R}$, an $\mathcal{R}$-module A is an abelian group (with addition denoted by $+$) together with a multiplication operation*

$$\cdot_A \colon \mathcal{R} \times A \to A, \quad (r, x) \mapsto r \cdot_A x$$

satisfying $(rs) \cdot_A x = r \cdot_A (s \cdot_A x)$ (associativity), $(r + s) \cdot_A x = (r \cdot_A x) + (s \cdot_A x)$ as well as $r \cdot_A (x + y) = (r \cdot_A x) + (r \cdot_A y)$ (distributivity) and $1 \cdot_A x = x$ for all $r, s \in \mathcal{R}, x, y \in A$.

This is really the same definition as a vector space over a field, except that we use a ring instead of a field. As opposed to (finite-dimensional) vector spaces, not every (finitely generated) $\mathcal{R}$-module is isomorphic to $\mathcal{R}^n$ for some n. Similar to vector spaces, the multiplication is often denoted by just $\cdot$ or even just concatenation; we only write $\cdot_A$ here for emphasis.

We will only consider $\mathcal{R}$-modules for the specific choice of ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$. The $\mathcal{R}$-modules we will need are typically of the form $\mathcal{R}^n$ or $\mathcal{R}_q^n$ for q prime and $n \in \mathbb{N}$ with module structures given by

$$\begin{aligned}\cdot_{\mathcal{R}^n}\colon\ \mathcal{R} \times \mathcal{R}^n \to \mathcal{R}^n, \quad (r,(x_1,\dots,x_n)) &\mapsto (r_1x_1,\dots,r_nx_n) \quad \text{resp.}\\ \cdot_{\mathcal{R}_q^n}\colon\ \mathcal{R} \times \mathcal{R}_q^n \to \mathcal{R}_q^n, \quad (r,(x_1,\dots,x_n)) &\mapsto (r_1x_1 \bmod q,\dots,r_nx_n \bmod q)\end{aligned}$$

It is straightforward to check that these are $\mathcal{R}$-modules.

We also need appropriate maps between modules that preserve this structure. Notably, a map $f\colon A \to B$ between $\mathcal{R}$-modules with the same $\mathcal{R}$ is called $\mathcal{R}$-linear, if $f(x+y) = f(x) + f(y)$ and $f(rx) = rf(x)$ holds for each $x, y \in A$, $r \in \mathcal{R}$. The composition of $\mathcal{R}$-linear maps gives a $\mathcal{R}$-linear map. Examples of $\mathcal{R}$-linear maps are the modular reduction map $\bmod q\colon \mathcal{R} \to \mathcal{R}_q$ and maps of the form $\mathcal{R}^n \to \mathcal{R}^m, \boldsymbol{v} \mapsto A \cdot \boldsymbol{v}$ for a fixed matrix $A \in \mathcal{R}^{m\times n}$. The former follows from compatibility of modular reduction with $+$ and $\cdot$. The latter holds because $A(\boldsymbol{v}+\boldsymbol{w}) = A\boldsymbol{v} + A\boldsymbol{w}$ and $Ar\boldsymbol{v} = rA\boldsymbol{v}$ for all $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{R}^n, r \in \mathcal{R}, A \in \mathcal{R}^{m\times n}$, since $\mathcal{R}$ is commutative.

Norm growth. To ensure that during aggregation, our norms don't grow too much, we will use the following auxiliary lemma to control the growth of the norm bounds.

Lemma 4. *Let n, α_w, ρ, β be positive integers such that n is a power of two. Let $\mathcal{R}$ be the polynomial ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$. Then for any $\ell \le \rho$, for any $\boldsymbol{x}_1, \dots, \boldsymbol{x}_\ell \in \mathcal{R}$ with $\|\boldsymbol{x}_i\| \le \beta$ and any growth factors $\zeta \ge 1$, we have*

$$\Pr\left[\boldsymbol{w}_1,\dots,\boldsymbol{w}_\ell \leftarrow \mathcal{T}_{\alpha_w} : \left\|\sum_{i=1}^{\ell} \boldsymbol{w}_i \cdot \boldsymbol{x}_i\right\| > \zeta \cdot \beta\right] < 2n \cdot \exp\left(-\frac{\zeta^2}{2\alpha\rho}\right) .$$

Proof. Recall that $\mathcal{T}_{\alpha_w}$ denotes ternary polynomials with weight exactly α_w. We will show that the claim holds even if we fix the positions of the non-zero entries in each $\boldsymbol{w}_i$ and only consider the randomness coming from the ±1-signs. Now observe that for each fixed k, the k-th coefficient y_k of $\sum_{i=1}^{\ell} \boldsymbol{w}_i \boldsymbol{x}_i$ is a sum of the form

$$y_k = \sum_{j=1}^{\alpha_w \cdot \ell} b_j c_j \ ,$$

where each b_j is some coefficient of some $\boldsymbol{x}_i$ and each $c_j \in \{-1, +1\}$ iid, corresponding to a sign choice of some coefficient of some $\boldsymbol{w}_i$ (everything depending on k). Thus, the expected value of y_k is 0 and $|b_j| \le \beta$, so changing any individual c_j out of the $\ell\alpha_w$ many c_j's can change the value of y_k by at most 2β. We can thus apply McDiarmid's inequality [McD89] to obtain

$$\Pr\Big[|y_k| > \zeta\beta\Big] \le 2\exp\left(\frac{-2(\zeta\beta)^2}{\ell\alpha_w(2\beta)^2}\right) \le 2\exp\left(-\frac{\zeta^2}{2\alpha_w\rho}\right) .$$

Taking a union bound over all n coefficients y_k of $\sum_{i=1}^{\ell} \boldsymbol{w}_i \boldsymbol{x}_i$ then gives the claim. □

We will construct a particular homomorphic vector commitment (HVC) denoted by $\mathsf{HVC}_0^{\mathrm{Chip}}$ in Section 3, then improve it to a more compact $\mathsf{HVC}_{\mathrm{Encoded}}^{\mathrm{Chip}}$ in Section 4. In Section 5, we construct a key-homomorphic one-time signature scheme (KOTS) denoted by $\mathsf{KOTS}^{\mathrm{Chip}}$. In Section 6, we combine those components to construct a synchronized aggregatable signature scheme. Our concrete constructions depends on a significant number of tunable parameters, whose choices affect both security, efficiency and functionality. Table 1 gives an overview and is intended as a reference for later, to aid the reader.

Parameter	Meaning
n	Dimension of the ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ we are working over; n is a power of two.
q	Prime number for HVCs. Our HVCs internally work modulo q, i.e. with $\mathcal{R}_q = \mathcal{R}/\langle q\rangle$.
q'	Prime number for KOTS. Our KOTS works modulo q', i.e. with $\mathcal{R}_{q'} = \mathcal{R}/\langle q'\rangle$. The HVC is used to commit to elements of $\mathcal{R}_{q'}$.
τ	Depth of our Merkle tree. 2^τ is the number of indices for the HVCs. This is also the number of time slots for the synchronized multi-signature.
ρ	Maximum number of homomorphic vector commitments or signatures that we support aggregating.
η	Arity parameter. Our constructions make use of $2\eta + 1$-ary decomposition.
κ	Number of limbs used in the decomposition of $\mathcal{R}q$ elements.
κ'	Number of limbs used in the decomposition of $\mathcal{R}q'$ elements.
ξ	Dimension (over $\mathcal{R}_{q'}$) of the elements our HVCs commit to.
γ	Dimension (over $\mathcal{R}_{q'}$) of public parameters and secret key components in the KOTS.
β_{agg}	Norm bound for HVCs after aggregation/homomorphic addition.
β_σ	Norm bound for KOTS signatures after aggregation/homomorphic addition.
β_{encode}	Norm bound for the encoded elements in the vector commitment openings.
α_w	Hamming weight for ring elements used as coefficients for homomorphic addition of our HVCs or KOTS.
α_H	Hamming weight for ring elements used as randomizers in the construction of individual KOTS signatures.
φ	Norm bound parameter for secret keys of the KOTS.
ε	Error bound. Individual aggregation attempts may fail with at most this probability for our HVCs or KOTS.
χ	Maximum number of aggregation attempts. Aggregation ultimately fails if χ individual attempts have failed.

Table 1: Parameters used in our concrete homomorphic vector commitment (HVC) and key-homomorphic one-time signature (KOTS) schemes. Since we combine those to a synchronized aggregatable signature scheme later, the parameters are related.

3 Homomorphic Vector Commitments

In this section, we define and instantiate homomorphic vector commitments, which allow for committing to a long vector with a short commitment value. Positions in the vector can be individually opened using a short opening value. We follow the definitions for vector commitments of Fleischhacker, Simkin, and Zhang [FSZ22a], but we require somewhat different and incomparable homomorphic properties. The definition of [FSZ22a] only requires honestly generated commitments to have homomorphic properties, whereas our definition requires the homomorphism to work for any individually verifying commitments and openings. On the other hand, [FSZ22a] requires that the homomorphism works with probability 1, whereas we allow some noticeable error. Among other things, this modification of the definition allows us to instantiate homomorphic vector commitments more compactly.

Definition 5. *Let $\tau \in \mathbb{N}$ be fixed. Let $\mathcal{R}$ be a ring and let A_{dom}, A_{com}, A_{op} be $\mathcal{R}$-modules. A homomorphic vector commitment scheme (HVC) for domain A_{dom} and vectors of length 2^τ is defined by six PPT algorithms* $\mathsf{HVC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{iVrfy}, \mathsf{sVrfy}, \mathsf{wVrfy})$.

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

$c \leftarrow \mathsf{Com}(\mathsf{pp}, \boldsymbol{m})$ *The commitment algorithm gets as input the public parameters and a vector $\boldsymbol{m} \in A_{\mathsf{dom}}^{2^\tau}$ and outputs a commitment $c \in A_{\mathsf{com}}$.*

$d \leftarrow \mathsf{Open}(\mathsf{pp}, c, \boldsymbol{m}, t)$ *The opening algorithm gets as input the public parameters, a commitment, the committed vector, and an index and outputs a decommitment $d \in A_{\mathsf{op}}$.*

$\boldsymbol{m}/\perp \leftarrow \mathsf{iVrfy}(\mathsf{pp}, c, t, d)$ *The individual verification algorithm takes as input public parameters, a commitment, an index, and a decommitment and outputs either $\boldsymbol{m} \in A_{\mathsf{dom}}$ or an error symbol.*

$\boldsymbol{m}/\perp \leftarrow \mathsf{sVrfy}(\mathsf{pp}, c, t, d)$ *The strong verification algorithm has the same input and output domains as the individual verification algorithm.*

$\boldsymbol{m}/\perp \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d)$ *The weak verification algorithm has the same input and output domains as the individual verification algorithm.*

For our purposes, $\mathcal{R}$ will always be $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ for n a power of two, as in Section 2. Our domain, commitment and opening space will always be of the form $A_{\mathsf{dom}} = \mathcal{R}_{q'}^{\ell_{\mathsf{dom}}}$, $A_{\mathsf{com}} = \mathcal{R}_q^{\ell_{\mathsf{com}}}$, $A_{\mathsf{op}} = \mathcal{R}^{\ell_{\mathsf{op}}}$ for some primes q, q'. Note that (correctly verifying) decommitments $d \in A_{\mathsf{op}}$ will have small coefficients and undergo arithmetic modulo q, so the reader may think of them as elements from $\mathcal{R}_q^{\ell_{\mathsf{op}}}$, as Squirrel [FSZ22a] does. However, we will impose some bounds on values that are not reduced modulo q later, so we need to formally treat them as elements from $\mathcal{R}^{\ell_{\mathsf{op}}}$ and write the modular reduction explicitly.

We can easily generalize this definition slightly and have A_{com} and A_{op} depend on the particular choice of $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, but will do not need that. Furthermore, the opening space A_{op} may depend on t. The latter is technically needed in Definition 26. To keep our notation simple, we only track that dependency if relevant. All our definitions and proofs directly apply to these generalization in a straightforward way.

Definition 6 (Individual Correctness). *Let* HVC *be a vector commitment scheme for domain A_{dom} and vector length 2^τ.* HVC *is individually correct, if for all security parameters $\lambda \in \mathbb{N}$, vectors $\boldsymbol{m} \in A_{\mathsf{dom}}^{2^\tau}$, indices $1 \leq t \leq 2^\tau$, parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, commitments $\boldsymbol{c} \leftarrow \mathsf{Com}(\mathsf{pp}, \boldsymbol{m})$, and decommitments $\boldsymbol{d} \leftarrow \mathsf{Open}(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$ it holds that*

$$\mathsf{iVrfy}\Big(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}\Big) = \boldsymbol{m}_t \ .$$

We require that individually verifying commitments and their respective decommitments can be homomorphically aggregated by computing a random $\mathcal{R}$-linear combination of them. Such aggregated commitments and decommitments should still *strongly* verify with high probability over the choice of the random linear combination, provided the coefficients of the linear combination are from some restricted subset W (such as a set of small elements).

Definition 7 (Probabilistic Homomorphism). *Let $A_{\mathsf{dom}}, A_{\mathsf{com}}, A_{\mathsf{op}}$ be $\mathcal{R}$-modules over some ring $\mathcal{R}$ and $\tau \in \mathbb{N}$. Let* HVC *be a vector commitment scheme for domain A_{dom} and vector length*

2^τ. Let $\rho \in \mathbb{N}$, $0 \leq \varepsilon \leq 1$ and $W \subseteq \mathcal{R}$. HVC *is (ρ, W, ε)-probabilistically homomorphic, if for all security parameters $\lambda \in \mathbb{N}$, number of aggregated commitments $\ell \leq \rho$, indices $1 \leq t \leq 2^\tau$, parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, commitments $\boldsymbol{c}^i \in A_{\mathsf{com}}$, and decommitments $\boldsymbol{d}^i \in A_{\mathsf{op}}$ with* $\mathsf{iVrfy}(\mathsf{pp}, \boldsymbol{c}^i, t, \boldsymbol{d}^i) = \boldsymbol{m}^i$ *such that $\boldsymbol{m}^i \neq \bot$ it holds that*

$$\Pr\left[w^1, \ldots, w^\ell \leftarrow W : \mathsf{sVrfy}\Big(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{c}^i, t, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{d}^i\Big) = \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{m}^i_t\right] \geq 1 - \varepsilon \ .$$

We additionally require that a further limited homomorphism still holds, even for maliciously *aggregated* commitments. For any two, even maliciously generated, commitments and their two respective openings that *strongly* verify, their difference will still *weakly* verify.

Definition 8 (Robust Homomorphism). *Let* HVC *be a vector commitment scheme for domain A_{dom} and vector length 2^τ.* HVC *is robustly homomorphic if for all security parameters $\lambda \in \mathbb{N}$, public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, indices $1 \leq t \leq 2^\tau$, (possibly malformed) commitments $\boldsymbol{c}^0, \boldsymbol{c}^1 \in A_{\mathsf{com}}$, and (possibly malformed) decommitments $\boldsymbol{d}^0, \boldsymbol{d}^1 \in A_{\mathsf{op}}$ with*

$$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) = \boldsymbol{m}^0 \quad \text{and} \quad \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) = \boldsymbol{m}^1$$

such that $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \bot$ it holds that

$$\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) = \boldsymbol{m}^0 - \boldsymbol{m}^1 \ .$$

Finally, we require the commitments to be position binding.

Definition 9 (Position-Binding). *Let* HVC *be a vector commitment scheme.* HVC *is position binding if for all security parameters λ and all PPT algorithms $\mathcal{A}$ it holds that*

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (\boldsymbol{c}, t, \boldsymbol{d}_0, \boldsymbol{d}_1) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \boldsymbol{m}_0 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d_0); \\ \boldsymbol{m}_1 \leftarrow \mathsf{wVrfy}(\mathsf{pp}, c, t, d_1) \end{array} : \boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\}\right] \leq \mathsf{negl}(\lambda) \ .$$

3.1 Squirrel's Homomorphic Vector Commitment

Since our homomorphic vector commitment is strongly based on Squirrel [FSZ22a], we recap their construction, albeit informally, in a bit more detail. Somewhat simplified, this commits to 2^τ (small) entries from $A_{\mathsf{dom}} = \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ by using a Merkle tree with a homomorphic hash function.[7] If we naively build a Merkle tree, this would mean that we construct a complete binary tree with 2^τ leaves, where each leaf corresponds to an entry we want to commit to. To each non-leaf node v, we associate the hash of its child nodes. See Figure 1 for a visualization, ignoring the bottom two rows for now. Concretely, the hash function utilized is Ajtai's hash function [Ajt99], which hashes child nodes $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ to

$$h_{\text{Ajtai}}(\boldsymbol{c}_1, \boldsymbol{c}_2) \coloneqq \boldsymbol{a}_1^\intercal \boldsymbol{c}_1 + \boldsymbol{a}_2^\intercal \boldsymbol{c}_2 \bmod q$$

[7] This is then extended to a scheme for (non-small) elements from $\mathcal{R}_{q'}^{\ell_{\mathsf{dom}}'}$. This exposition focuses on the $\mathcal{R}_q^{\ell_{\mathsf{dom}}}$-part of the construction and we set ${\ell_{\mathsf{dom}}}' = 1$ for notational simplicity.

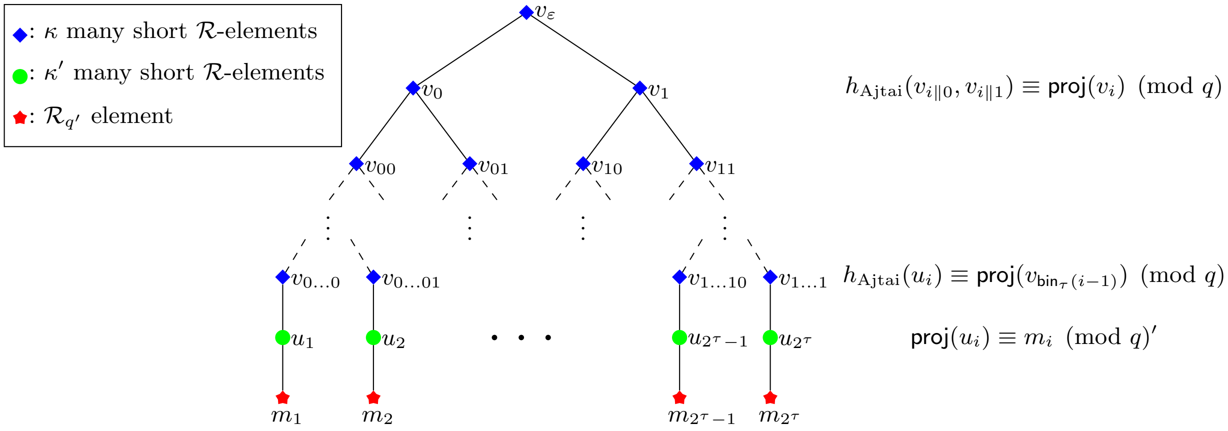

◆: κ many short $\mathcal{R}$-elements
●: κ' many short $\mathcal{R}$-elements
⬟: $\mathcal{R}_{q'}$ element
v_ε
v_0
v_1
v_{00}
v_{01}
v_{10}
v_{11}
$v_{0\ldots0}$
$v_{0\ldots01}$
$v_{1\ldots10}$
$v_{1\ldots1}$
u_1
u_2
$u_{2^\tau-1}$
u_{2^τ}
m_1
m_2
$m_{2^\tau-1}$
m_{2^τ}
$h_{\text{Ajtai}}(v_{i\|0}, v_{i\|1}) \equiv \mathsf{proj}(v_i) \pmod q$
$h_{\text{Ajtai}}(u_i) \equiv \mathsf{proj}(v_{\mathsf{bin}_\tau(i-1)}) \pmod q$
$\mathsf{proj}(u_i) \equiv m_i \pmod q)'$

Fig. 1: Squirrel's homomorphic vector commitment. The bottom 2 rows serve the purpose to commit to vectors of $\mathcal{R}_{q'}$ elements rather than to vectors of short $\mathcal{R}_q$- or $\mathcal{R}$-elements. The equations on the right are the constraints that link the layers together, ignoring shortness constraints. Note that all layers but the bottom one must contain short elements.

using a uniformly random $\mathcal{R}_q$-linear map given by fixed public uninform $\boldsymbol{a}_1, \boldsymbol{a}_2 \leftarrow \mathcal{R}_q^{\ell_{\mathsf{dom}}}$. Now, setting the relationship between parent node $\boldsymbol{p}$ and child nodes $\boldsymbol{c}_1, \boldsymbol{c}_2$ as $\boldsymbol{p} = h_{\text{Ajtai}}(\boldsymbol{c}_1, \boldsymbol{c}_2)$ does not quite work: firstly, the range of the hash function is not A_{dom}, which prevents iterating this construction. Secondly, this hash function is only binding (based on some appropriate ring-SIS assumption) if we restrict its input to small elements. To solve these issues, Squirrel chooses a second (public, fixed) linear function

$$\mathsf{proj}\colon \mathcal{R}_q^{\ell_{\mathsf{dom}}} \to \mathcal{R}_q$$

and sets the equation that relates the parent node $\boldsymbol{p} \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ with its children $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ as

$$h_{\text{Ajtai}}(\boldsymbol{c}_1, \boldsymbol{c}_2) = \mathsf{proj}(\boldsymbol{p}) \mod q \ . \tag{1}$$

One may view $\boldsymbol{p}$ as some kind of encoding of $\mathsf{proj}(\boldsymbol{p})$ here. Since $\boldsymbol{p}$ enters the hash function on the next layer of the tree as a child, it must be small (this is checked by the verification algorithms along with the linear relation above). So to construct the tree, we need to be able to find small preimages of proj. In lattice terms, this means we need to solve some close(st) vector problem for the kernel of proj. An important observation is that this construction actually works for any proj for which we can find short preimages: the homomorphic properties of the HVC are due to the fact that equation (1) above is phrased in terms of $\mathcal{R}_q$-linear maps, and the sum of small elements stays small. We emphasize that what primarily matters here is the linearity properties of proj and the *verification* equation. The map that finds the small preimage may be thought of as auxilliary and will not be linear.

Squirrel chooses proj as binary reconstruction $\mathsf{proj}(\boldsymbol{p}) = p_0 + 2p_1 + 4p_2 + \ldots$. An algorithm to find a short inverse is then given by binary decomposition.

To commit to the correct domain $\mathcal{R}_{q'}$, Squirrel adds some extra layers on the bottom of Figure 1.

The main improvement from Chipmunk over Squirrel comes from choosing a different map for proj and its inverse: we propose to instead use $(2\eta+1)$-ary decomposition rather than binary decomposition. This turns out to give significantly better parameters.

Some other differences in the actual construction are as follows:

- We define the commitment (corresponding to the root of the Merkle tree) to be in non-decomposed form.
- We define proj and the $2\eta+1$-adic decomposition as maps over $\mathcal{R}$ rather than $\mathcal{R}_q$.
- We constrain the size of $\mathsf{proj}(\boldsymbol{p})$ for any node of the tree.
- We use a more elaborate scheme to encode decommitments. This is explained in Section 4.

3.2 A Homomorphic Vector Commitment based on Ring-SIS

To construct a homomorphic vector commitment with the desired properties, we will define proj and an inverse, called decomposition, as described above. We use $(2\eta+1)$-ary decomposition for the latter, which allows us to map a ring element with possibly large norm to a vector of low norm ring elements. To be able to use the greatest arity while minimizing the infinity norm of decomposed elements, we use a *balanced* $(2\eta+1)$-ary decomposition, i.e. the decomposed elements have coefficients from $\{-\eta,\dots,+\eta\}$ centered around 0. We note that any even arity, such as the binary decomposition used by Squirrel [FSZ22a], is strictly worse than the next greater odd arity. We then show that the projection function has nice homomorphic properties.

Definition 10 (Projection onto $\mathcal{R}$ elements). *Let $\eta,\kappa\in\mathbb{N}$. For any $\boldsymbol{b}\in\mathbb{Z}^\kappa$ we define the function*

$$\mathsf{proj}_{\eta,\kappa}\colon\ \mathbb{Z}^\kappa\to\mathbb{Z},\quad \mathsf{proj}_{\eta,\kappa}(\boldsymbol{b})=\sum_{j=1}^{\kappa} b_j\cdot(2\eta+1)^{j-1}\ .$$

We can extend this to a map

$$\mathsf{proj}_{\eta,\kappa}\colon\ \mathcal{R}^\kappa\to\mathcal{R},\quad \mathsf{proj}_{\eta,\kappa}(\boldsymbol{b})=\sum_{j=1}^{\kappa} b_j\cdot(2\eta+1)^{j-1}\ .$$

Definition 11 (Balanced $(2\eta+1)$-ary decomposition of $\mathcal{R}$ elements). *Fix some odd arity $2\eta+1$ and let $\kappa\in\mathbb{N}$ be the number of limbs. Then we can uniquely decompose any $a\in\mathbb{Z}$ into a balanced $(2\eta+1)$-ary decomposition with κ limbs as*

$$a=\sum_{i=1}^{\kappa} a_i\cdot(2\eta+1)^{i-1}$$

where $a_i\in\{-\eta,\dots,\eta\}$ for all $1\le i<\kappa$. An algorithm and proof of this statement is given below in Figure 2 and Proposition 14. Note that it is notationally convenient to allow arbitrarily sized a in the definition and not bound a_κ, thereby putting all higher-order terms into a_κ. If we have the bound $|a|<\frac{(2\eta+1)^\kappa}{2}$, then the most significant limb a_κ will also be in $\{-\eta,\dots,\eta\}$.

We can extend this to a map on $\mathcal{R}$ by essentially decomposing each coefficient, uniquely mapping a polynomial $a\in\mathcal{R}$ to limbs $a_1\dots,a_\kappa\in\mathcal{R}$ such that

$$a=\sum_{i=1}^{\kappa} a_i\cdot(2\eta+1)^{i-1}$$

where $\|a_i\|_\infty\le\eta$ for all $1\le i<\kappa$. If $\|a\|_\infty<\frac{(2\eta+1)^\kappa}{2}$, we also have the bound $\|a_\kappa\|_\infty\le\eta$ for the most significant limb.

Matching the notation from Definition 10, we denote this decomposition map by $\mathsf{dec}_{\eta,\kappa}$*, giving a map*

$$\mathsf{dec}_{\eta,\kappa} \colon \mathcal{R} \to \mathcal{R}^{\kappa}, \quad a \mapsto (a_1, \ldots, a_\kappa) \ .$$

Definition 12 (Projection and Decomposition for $\mathcal{R}_q$). *Fix some odd arity* $(2\eta + 1)$ *and let* q *be prime. Set* $\kappa \coloneqq \lceil \log_{2\eta+1} q \rceil$ *We denote by*

$$\mathsf{proj}_q \colon \mathcal{R}^{\kappa} \to \mathcal{R}_q, \quad \mathsf{proj}_q(\boldsymbol{a}) \coloneqq \mathsf{proj}_{\eta,\kappa}(\boldsymbol{a}) \bmod q \in \mathcal{R}_q$$

and by

$$\mathsf{dec}_q \colon \mathcal{R}_q \to \mathcal{R}^{\kappa}, \quad \mathsf{dec}_q(a) \coloneqq \mathsf{dec}_{\eta,\kappa}(a'), \ \ ,$$

where a' *is the representative of* a *in* $\mathcal{R}$ *with coefficients in* $\{-\frac{q-1}{2}, \ldots, +\frac{q-1}{2}\}$.

We remark that the only difference between proj_q and $\mathsf{proj}_{\eta,\kappa}$ resp. between dec_q and $\mathsf{dec}_{\eta,\kappa}$ is whether the non-decomposed element is in $\mathcal{R}_q$ or $\mathcal{R}$. The decomposed elements are always from $\mathcal{R}^{\kappa}$. For proj_q and dec_q, the value of η is not denoted explicitly. This is done for notational consistency with Squirrel. In our constructions, all uses of proj_q and dec_q will use the same value for η, even if the values of q differ.

The following proposition immediately follow from the definitions (for $\mathcal{R}$-linearity, this follows from the examples given after Definition 3).

Proposition 13. *Let* q *be an odd integer and fix some odd arity* $2\eta + 1$*. The maps* proj_q *and* $\mathsf{proj}_{\eta,\kappa}$ *defined above are* $\mathcal{R}$*-linear. The map* $\mathsf{dec}_{\eta,\kappa}$ *is a one-sided inverse to* $\mathsf{proj}_{\eta,\kappa}$*, meaning that* $\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(a)) = a$ *for any* $a \in \mathcal{R}$*. Similarly,* dec_q *is a one-sided inverse to* proj_q*, meaning that* $\mathsf{proj}_q(\mathsf{dec}_q(a)) = a$ *for any* $a \in \mathcal{R}_q$*. For* $a \in \mathcal{R}_q$*, we also have* $\|\mathsf{dec}_q(a)\|_\infty \leq \eta$*.*

For the sake of readability we will at times abuse notation slightly and apply dec_q resp. $\mathsf{dec}_{\eta,\kappa}$ to *vectors* of $\mathcal{R}_q$ resp. $\mathcal{R}$ elements, which is to be understood as the component-wise application of dec_q resp. $\mathsf{dec}_{\eta,\kappa}$ with subsequent concatenation of the resulting vectors. Similarly, proj_q resp. $\mathsf{proj}_{\eta,\kappa}$ may be applied to vectors of a length that is a *multiple* of κ to result in a vector of $\mathcal{R}_q$ resp. $\mathcal{R}$ elements. The above discussion generalizes to this extension.

```
$\mathsf{dec}_{\eta,\kappa}(a)$
$r_1 \coloneqq a$
for $1 \leq i \leq \kappa - 1$
    Choose $a_i \in \{-\eta, \ldots, +\eta\}$ with $a_i \equiv r_i \bmod (2\eta + 1)$
    $r_{i+1} \coloneqq \frac{r_i - a_i}{2\eta+1}$    // Numerator is divisible by $2\eta + 1$
$a_\kappa \coloneqq r_\kappa$
return $(a_1, \ldots, a_\kappa)$
```

Fig. 2: Algorithm for balanced $(2\eta + 1)$-ary decomposition of integers $a \in \mathbb{Z}$. The corresponding algorithm for $a \in \mathcal{R}$ works by applying this coefficient-wise.

Proposition 14 (balanced $(2\eta+1)$-ary decomposition). *Let $\eta, \kappa \in \mathbb{N}$. The algorithm in Figure 2 runs in polynomial time. For any $a \in \mathbb{Z}$, it outputs the unique $(a_1, \dots, a_\kappa)$ with*

$$a = \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1} \tag{2}$$

and $a_i \in \{-\eta, \dots, +\eta\}$ for $1 \le i \le \kappa - 1$.

Proof. The algorithm is clearly polynomial time. $r - a_i$ is divisible by $2\eta + 1$ by construction of a_i. By definition, $a_1, \dots, a_{\kappa-1} \in \{-\eta, \dots, +\eta\}$. We show by induction that we have for all $1 \le i \le \kappa - 1$

$$a = r_i \cdot (2\eta+1)^{i-1} + \sum_{j=1}^{i-1} a_j (2\eta+1)^{j-1} \ .$$

This is clear for $i = 1$. Using induction, we compute

$$\begin{aligned} & r_{i+1} \cdot (2\eta+1)^{i} + \sum_{j=1}^{i} a_j (2\eta+1)^{j-1} \\ =& (r_i - a_i) \cdot (2\eta+1)^{i-1} + \sum_{j=1}^{i} a_j (2\eta+1)^{j-1} && \text{(Def. of } r_{i+1}\text{)} \\ =& r_i \cdot (2\eta+1)^{i-1} + \sum_{j=1}^{i-1} a_j (2\eta+1)^{j-1} = a && \text{(induction hypothesis)} \end{aligned}$$

For $i = \kappa$, this yields $a = \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1}$. For uniqueness, note that we just showed that the map $\{-\eta, \dots, \eta\}^{\kappa-1} \times \mathbb{Z} \to \mathbb{Z}$, $(a_1, \dots, a_\kappa) \mapsto \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1}$ is surjective. Taking this modulo $(2\eta+1)^{\kappa-1}$ gives us that $\{-\eta, \dots, +\eta\}^{\kappa-1} \to \mathbb{Z}_{(2\eta+1)^{\kappa-1}}$, $(a_1, \dots, a_{\kappa-1}) \mapsto \sum_{i=1}^{\kappa-1} a_i \cdot (2\eta+1)^{i-1} \bmod (2\eta+1)^{\kappa-1}$ is surjective, hence injective (because domain and range have the same finite size). So $a_1, \dots, a_{\kappa-1}$ are uniquely determined. Plugging this into Equation 2 shows that a_κ is uniquely determined as well. □

This lets us define a labeling function for a full binary tree matching Figure 1.

Definition 15 (Labeled Full Binary Tree). *Let $n, q, q', \xi \in \mathbb{N}$ with n a power of two and q, q' primes. Let $\boldsymbol{m} = (\boldsymbol{m}_1, \dots, \boldsymbol{m}_{2^\tau})^\intercal \in (\mathcal{R}_{q'}^{\xi})^{2^\tau}$, $\boldsymbol{g} \in \mathcal{R}_q^{\xi\lceil \log_{2\eta+1} q' \rceil}$ and $\boldsymbol{h}_0, \boldsymbol{h}_1 \in \mathcal{R}_q^{\lceil \log_{2\eta+1} q \rceil}$ be fixed. We define the labeling function $\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1} \colon (\mathcal{R}_{q'}^{\xi})^{2^\tau} \times \{0,1\}^{\le \tau} \to \mathcal{R}^{\lceil \log_{2\eta+1} q \rceil}$ for a labeled full binary tree of depth τ as*

$$\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, v) := \begin{cases} \mathsf{dec}_q(\boldsymbol{g}^\intercal \cdot \mathsf{dec}_{q'}(\boldsymbol{m}_{v+1})) & \text{if } |v| = \tau \\ \mathsf{dec}_q\begin{pmatrix} \boldsymbol{h}_0^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, v\|0) \\ +\boldsymbol{h}_1^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, v\|1) \end{pmatrix} & \text{if } |v| < \tau \end{cases} .$$

For this, remember that multiplication of elements from $\mathcal{R}_q$ and $\mathcal{R}$ is always understood[8] *to give an element in $\mathcal{R}_q$. For $\boldsymbol{m}_{v+1}$, we interpret $v \in \{0,1\}^\tau$ as an integer in $\{0 \dots, 2^\tau - 1\}$ in big-endian encoding and add 1 (i.e. the inverse of taking $\tilde{t} = \mathsf{bin}_\tau(t-1)$).*

[8] as opposed to taking some canonical representative of $\mathcal{R}_q$-elements in $\mathcal{R}$ and then multiplying in $\mathcal{R}$

Using the labeling function, we can define Chipmunk's HVC as in Figure 3.

Definition 16. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers such that n is a power of two and q, q' are primes. Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$ respectively. We define the homomorphic vector commitment $\mathsf{HVC}_0^{\mathrm{Chip}}$ for domain $A_{\mathsf{dom}} = \mathcal{R}_{q'}^{\xi}$ and vectors of length 2^τ by the algorithms given in Figure 3. Its commitments and openings are from $A_{\mathsf{com}} = \mathcal{R}_q$ and $A_{\mathsf{op}} = (\mathcal{R}^\kappa)^{2\tau} \times (\mathcal{R}^{\kappa'})^\xi$, where $\kappa = \lceil \log_{2\eta+1} q \rceil, \kappa' = \lceil \log_{2\eta+1} q' \rceil$.*

```
$\mathsf{Setup}(1^\lambda)$
  $\boldsymbol{g} \leftarrow \mathcal{R}_q^{\xi\kappa'}$
  $\boldsymbol{h}_0 \leftarrow \mathcal{R}_q^{\kappa}$
  $\boldsymbol{h}_1 \leftarrow \mathcal{R}_q^{\kappa}$
  return $(\boldsymbol{g}, \boldsymbol{h}_0, \boldsymbol{h}_1)$

$\mathsf{Com}(\mathsf{pp}, \boldsymbol{m})$
  $\boldsymbol{p}_0 := \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \epsilon)$
  $\boldsymbol{c} := \mathsf{proj}_q(\boldsymbol{p}_0)$
  return $\boldsymbol{c} \in \mathcal{R}_q$

$\mathsf{Open}(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$
  $\tilde{t} := \mathsf{bin}_\tau(t-1)$
  for $1 \le j \le \tau$
    $\boldsymbol{p}_j := \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j} \| \tilde{t}_j)$
    $\boldsymbol{s}_j := \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j} \| (\tilde{t}_j \oplus 1))$
  $\boldsymbol{u} := \mathsf{dec}_{q'}(\boldsymbol{m}_t)$
  return $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$

$\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \beta)$
  parse $\boldsymbol{d}$ as $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$
  $\tilde{t} := \mathsf{bin}_\tau(t-1)$
  if $\|\boldsymbol{u}\| > \beta$ or $\boldsymbol{g}^\intercal \cdot \boldsymbol{u} \neq \mathsf{proj}_q(\boldsymbol{p}_\tau)$
    return $\perp$
  if $\boldsymbol{c} \neq \boldsymbol{h}_{\tilde{t}_1}^\intercal \cdot \boldsymbol{p}_1 + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^\intercal \cdot \boldsymbol{s}_1$
    return $\perp$
  for $2 \le j \le \tau$
    if $\mathsf{proj}_q(\boldsymbol{p}_{j-1}) \neq \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_j$
      return $\perp$
  for $j \in \{1, \ldots, \tau\}$
    if $\|\boldsymbol{p}_j\| > \beta$ or $\|\boldsymbol{s}_j\| > \beta$
      return $\perp$
    if $\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j)\| > \frac{q\beta}{2\eta}$ or $\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j)\| > \frac{q\beta}{2\eta}$
      return $\perp$
  return $\mathsf{proj}_{q'}(\boldsymbol{u}) \in \mathcal{R}_{q'}^{\xi}$

$\mathsf{iVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$
  return $\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \eta)$

$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$
  return $\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \beta_{\mathsf{agg}})$

$\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$
  return $\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, 2\beta_{\mathsf{agg}})$
```

Fig. 3: The construction of the homomorphic vector commitment $\mathsf{HVC}_0^{\mathrm{Chip}}$ for message space $A_{\mathsf{dom}} = \mathcal{R}_{q'}^{\xi}$ based on a labeled binary tree, cf. Definition 16. Commitments $\boldsymbol{c}$ are in $A_{\mathsf{com}} = \mathcal{R}_q$. Openings are small elements in $A_{\mathsf{op}} = (\mathcal{R}^\kappa)^{2\tau} \times (\mathcal{R}^{\kappa'})^\xi$, where $\kappa = \lceil \log_{2\eta+1} q \rceil$, $\kappa' = \lceil \log_{2\eta+1} q' \rceil$. Let us clarify again that multiplication of $\mathcal{R}_q$ with $\mathcal{R}$ elements as done in the Ajtai hashes like $\boldsymbol{g}^\intercal \cdot \boldsymbol{u}$ is understood to give an element in $\mathcal{R}_q$, i.e. we perform modular reduction here.

Remark 1. Before proving security of $\mathsf{HVC}_0^{\mathrm{Chip}}$, let us give some remarks on the construction itself.

1. Chipmunk's final homomorphic vector commitment actually employs a space-efficient non-trivial way to encode and decode (verifying) decommitments $\boldsymbol{d} = (\boldsymbol{p}_1, \dots \boldsymbol{p}_\tau, \boldsymbol{s}_1, \dots, \boldsymbol{s}_\tau, \boldsymbol{u})$. To simplify the exposition, $\mathsf{HVC}_0^{\text{Chip}}$ in Figure 3 is described without these encoding and decoding schemes, which are formally part of the opening and verification algorithms. We describe this encoding and decoding separately in Section 4, giving an improved HVC denoted by $\mathsf{HVC}_{\text{Encoded}}^{\text{Chip}}$ there.
2. The tree labels constructed by the labeling function that constitute the Merkle path $\boldsymbol{p}_j$ with its sibling nodes $\boldsymbol{s}_j$ are *decomposed* elements, i.e. short elements in $\mathcal{R}$. For efficiency reasons, the commitment $\boldsymbol{c}$ itself is not $\boldsymbol{p}_0$, but rather in non-decomposed form. This is done to ensure the commitment is in $\mathcal{R}_q$ rather than $\mathcal{R}$, which is slightly more efficient when aggregating. Regarding analysis, observe that if we set $\boldsymbol{p}_0$ as in the definition of Com, the condition $\boldsymbol{c} = \boldsymbol{h}_{\tilde{t}_1}^\intercal \cdot \boldsymbol{p}_1 + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^\intercal \cdot \boldsymbol{s}_1$ is actually equivalent to
$$\mathsf{proj}_q(\boldsymbol{p}_0) = \boldsymbol{h}_{\tilde{t}_1}^\intercal \cdot \boldsymbol{p}_1 + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^\intercal \cdot \boldsymbol{s}_1 \ .$$
Hence, we may treat this condition as the special case $j = 1$ of the condition $\mathsf{proj}_q(\boldsymbol{p}_{j-1}) = \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_j$.
3. Let $\kappa := \lceil \log_{2\eta+1} q \rceil$. The inequality checks in the definition of Vrfy all compare elements from $\mathcal{R}_q$ and are to be taken modulo q. By contrast, the norm-bounds are to be taken in $\mathcal{R}$. For the individual verification, the condition that $\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j)\right\| \le \frac{q\beta}{2\eta}$ boils down to $\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j)\right\| \le \frac{q}{2}$. This is trivially satisfied by any decomposition of an element from $\mathcal{R}_q$ and just means that $\boldsymbol{p}_j = \mathsf{dec}_q(\mathsf{proj}_q(\boldsymbol{p}_j))$. If we did not require this, a dishonestly generated signature could choose $\boldsymbol{p}_j$ as the decomposition of an element whose coefficients are not in $\{-\frac{q-1}{2}, \dots, \frac{q-1}{2}\}$, but still bounded by $\frac{(2\eta+1)^\kappa - 1}{2}$. In particular, if q is significantly smaller than $(2\eta+1)^\kappa$, adding this condition actually gives a stronger shortness bound for the most significant limbs of the decomposition. These tighter bounds are not present in Squirrel or in the extended abstract of this work, but they significantly help to make our encoding of openings both more efficient and easier to analyze later.

Theorem 17. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers and $0 < \varepsilon \le 1$ such that n is a power of two, q, q' are prime, and*
$$\beta_{\mathsf{agg}} \ge \eta \sqrt{2\alpha_w \rho \big(\ln \tfrac{2n}{\varepsilon} + \ln(2\tau \lceil \log_{2\eta+1} q \rceil + \xi \lceil \log_{2\eta+1} q' \rceil + 2\tau)\big)} \ .$$
Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$ respectively. If the $\mathsf{SIS}_{\mathcal{R},q,2\lceil \log_{2\eta+1} q \rceil, 4\beta_{\mathsf{agg}}}$ problem and the $\mathsf{SIS}_{\mathcal{R},q,\xi\lceil \log_{2\eta+1} q' \rceil, 4\beta_{\mathsf{agg}}}$ problem are hard, then $\mathsf{HVC}_0^{\text{Chip}}$ is an individually correct, $(\rho, \mathcal{T}_{\alpha_w}, \varepsilon)$-probabilistically homomorphic, robustly homomorphic, and position binding HVC for domain $\mathcal{R}_{q'}^\xi$ and vector length 2^τ.

Proof. The theorem follows from Lemma 18, Lemma 19, Lemma 20, and Lemma 21 proven below. □

Lemma 18. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers and $0 < \varepsilon \le 1$, such that n is a power of two, q, q' are prime. Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$ respectively. Then $\mathsf{HVC}_0^{\text{Chip}}$ is an individually correct HVC for domain $\mathcal{R}_{q'}^\xi$ and vector length 2^τ.*

Proof. Let $\boldsymbol{m} \in (\mathcal{R}_{q'}^\xi)^{2^\tau}$, $\boldsymbol{c} = \mathsf{Com}(\mathsf{pp}, \boldsymbol{m})$, $t \in [2^\tau]$, $(\boldsymbol{p}_1, \dots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \dots, \boldsymbol{s}_\tau, \boldsymbol{u})^\intercal = \mathsf{Open}(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$. Let $\boldsymbol{p}_0, \tilde{t}$ be as in the definition of Com. We first observe that for all $j \in \{1, \dots, \tau\}$ it holds in $\mathcal{R}_q$

that

$$\begin{aligned}
\mathsf{proj}_q(\boldsymbol{p}_{j-1}) &= \mathsf{proj}_q\left(\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j})\right) && \text{(Def. of } \mathsf{Com} \text{ and } \mathsf{Open}\text{)}\\
&= \mathsf{proj}_q\left(\mathsf{dec}_q\left(\begin{matrix} \boldsymbol{h}_0^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j}\|0) \\ +\boldsymbol{h}_1^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j}\|1)\end{matrix}\right)\right) && \text{(Definition 15)}\\
&= \boldsymbol{h}_0^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j}\|0) + \boldsymbol{h}_1^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j}\|1) && \text{(Proposition 13)}\\
&= \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j}\|\tilde{t}_j) + \boldsymbol{h}_{\tilde{t}_j\oplus 1}^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j}\|(\tilde{t}_j \oplus 1))\\
&= \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j + \boldsymbol{h}_{\tilde{t}_j\oplus 1}^\intercal \cdot \boldsymbol{s}_j. && \text{(Def. of } \mathsf{Open}\text{)}
\end{aligned}$$

Observe that for $j = 1$, this gives $\boldsymbol{c} = \boldsymbol{h}_{\tilde{t}_1}^\intercal \cdot \boldsymbol{p}_1 + \boldsymbol{h}_{\tilde{t}_1\oplus 1}^\intercal \cdot \boldsymbol{s}_1$ in $\mathcal{R}_q$. Further it holds that

$$\begin{aligned}
\mathsf{proj}_q(\boldsymbol{p}_\tau) &= \mathsf{proj}_q(\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t})) && \text{(Def. of } \mathsf{Com} \text{ and } \mathsf{Open}\text{)}\\
&= \mathsf{proj}_q(\mathsf{dec}_q(\boldsymbol{g}^\intercal \cdot \mathsf{dec}_{q'}(\boldsymbol{m}_t))) && \text{(Definition 15)}\\
&= \boldsymbol{g}^\intercal \cdot \mathsf{dec}_{q'}(\boldsymbol{m}_t) && \text{(Proposition 13)}\\
&= \boldsymbol{g}^\intercal \cdot \boldsymbol{u}. && \text{(Def. of } \mathsf{Open}\text{)}
\end{aligned}$$

Therefore it only remains to check that the norm bounds are not violated. For every $j \in [\tau]$, $\boldsymbol{p}_j$ and $\boldsymbol{s}_j$ are outputs of the $\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}$ function and thus, by definition of $\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}$, decompositions of elements from $\mathcal{R}_q$. Similarly, $\boldsymbol{u}$ is the output of $\mathsf{dec}_{q'}$, applied to a vector of elements from $\mathcal{R}_{q'}$. By design, this implies that the resulting coefficients are in $\{-\eta, \dots, \eta\}$ and so the norm of each $\boldsymbol{p}_j$ and $\boldsymbol{s}_j$ as well as $\boldsymbol{u}$ is at most η. It also implies that applying $\mathsf{proj}_{\eta,\kappa}$ to $\boldsymbol{p}_j$ or $\boldsymbol{s}_j$ gives back the representative with coefficients in $\{-\frac{q-1}{2}, \dots, \frac{q-1}{2}\}$ that was decomposed. Consequently, we have $\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j)\|, \|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j)\| \leq \frac{q-1}{2}$. □

Lemma 19. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers and $0 < \varepsilon \leq 1$, such that n is a power of two, q, q' are prime, and*

$$\beta_{\mathsf{agg}} \geq \eta\sqrt{2\alpha_w\rho\left(\ln\tfrac{2n}{\varepsilon} + \ln(2\tau\kappa + \xi\kappa' + 2\tau)\right)}\ ,$$

where $\kappa = \lceil \log_{2\eta+1} q\rceil$ and $\kappa' = \lceil \log_{2\eta+1} q'\rceil$. Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$ respectively. Then $\mathsf{HVC}_0^{\mathrm{Chip}}$ is a $(\rho, \mathcal{T}_\alpha, \varepsilon)$-probabilistically homomorphic HVC for domain $\mathcal{R}_{q'}^\xi$ and vector length 2^τ.

Proof. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $\boldsymbol{c}^i \in \mathcal{R}_q^\kappa$, $1 \leq t \leq 2^\tau$, $\tilde{t} = \mathsf{bin}_\tau(t-1)$, $\boldsymbol{d}^i = (\boldsymbol{p}_1^i, \dots, \boldsymbol{p}_\tau^i, \boldsymbol{s}_1^i, \dots, \boldsymbol{s}_\tau^i, \boldsymbol{u})^\intercal \in \left(\mathcal{R}^{\lceil \log_{2\eta+1} q\rceil}\right)^{2\tau} \times \mathcal{R}^{\xi\lceil \log_{2\eta+1} q'\rceil}$ with $\mathsf{iVrfy}(\mathsf{pp}, \boldsymbol{c}^i, t, \boldsymbol{d}^i) = \boldsymbol{m}_t^i \neq \bot$ as specified in Definition 5. We first note that even for arbitrary $w^1, \dots, w^\ell \in \mathcal{T}_{\alpha_w}$ it holds for all $2 \leq j \leq \tau$ that

$$\begin{aligned}
\mathsf{proj}_q\Big(\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{p}_{j-1}^i\Big) &= \sum_{i=1}^{\ell} w^i \cdot \mathsf{proj}_q(\boldsymbol{p}_{j-1}^i) && \text{(Proposition 13)}\\
&= \sum_{i=1}^{\ell} w^i \cdot (\boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j^i + \boldsymbol{h}_{\tilde{t}_j\oplus 1}^\intercal \cdot \boldsymbol{s}_j^i) && \text{(Def. of } \mathsf{iVrfy}\text{)}\\
&= \sum_{i=1}^{\ell} \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot w^i \boldsymbol{p}_j^i + \boldsymbol{h}_{\tilde{t}_j\oplus 1}^\intercal \cdot w^i \boldsymbol{s}_j^i
\end{aligned}$$

$$= \boldsymbol{h}_{\tilde{t}_j}^{\mathsf{T}} \cdot \Big(\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{p}_j^i\Big) + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^{\mathsf{T}} \cdot \Big(\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{s}_j^i\Big).$$

and similarly

$$\begin{aligned}
\mathsf{proj}_q\Big(\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{p}_\tau^i\Big) &= \sum_{i=1}^{\ell} w^i \cdot \mathsf{proj}_q(\boldsymbol{p}_\tau^i) && \text{(Proposition 13)} \\
&= \sum_{i=1}^{\ell} w^i \cdot (\boldsymbol{g}^{\mathsf{T}} \cdot \boldsymbol{u}^i) && \text{(Def. of } \mathsf{iVrfy}\text{)} \\
&= \boldsymbol{g}^{\mathsf{T}} \cdot \sum_{i=1}^{\ell} w^i \boldsymbol{u}^i
\end{aligned}$$

and similarly that

$$\begin{aligned}
\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{c}^i &\equiv \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{h}_{\tilde{t}_1}^{\mathsf{T}} \cdot \boldsymbol{p}_1^i + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^{\mathsf{T}} \cdot \boldsymbol{s}_1^i \\
&\equiv \boldsymbol{h}_{\tilde{t}_1}^{\mathsf{T}} \cdot \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{p}_1^i + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^{\mathsf{T}} \cdot \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{s}_1^i
\end{aligned}$$

Therefore it only remains to verify that the norm-checks go through with sufficient probability. Writing out the conditions, this means that we need to show that

$$\begin{aligned}
P := \Pr\Big[w^1, \ldots, w^\ell \leftarrow \mathcal{T}_{\alpha_w}:\ \exists j \in [\tau].\ &\Big\|\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{p}_j^i\Big\| > \beta_{\mathsf{agg}} \vee \Big\|\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{s}_j^i\Big\| > \beta_{\mathsf{agg}} \vee \\
&\Big\|\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{u}^i\Big\| > \beta_{\mathsf{agg}} \vee \Big\|\sum_{i=1}^{\ell} w^i \cdot \mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j^i)\Big\| > \tfrac{q\beta_{\mathsf{agg}}}{2\eta} \vee \\
&\Big\|\sum_{i=1}^{\ell} w^i \cdot \mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j^i)\Big\| > \tfrac{q\beta_{\mathsf{agg}}}{2\eta}\Big] \leq \varepsilon\ .
\end{aligned}$$

Observe that this is an $\|.\|_\infty$-bound for a total of

$$N_{\text{bounds}} := \tau\ell\kappa + \tau\ell\kappa + \ell\xi\kappa' + \tau\ell + \tau\ell$$

many ring elements. For each of the N_{bounds} ring elements, we can individually apply Lemma 4 with the same growth factor $\zeta = \frac{\beta_{\mathsf{agg}}}{\eta}$. Taking a N_{bounds}-fold union bound then gives

$$P \leq N_{\text{bounds}} \cdot 2n \exp\Big(-\frac{\beta_{\mathsf{agg}}^2}{2\eta^2\alpha_w\rho}\Big)\ .$$

Our condition on β_{agg} is chosen exactly to guarantee that $\frac{\beta_{\mathsf{agg}}^2}{2\eta^2\alpha_w\rho} \geq \ln\big(2nN_{\text{bounds}} \cdot \frac{1}{\varepsilon}\big)$. This gives $P \leq \varepsilon$. It follows that with probability at least $1 - \varepsilon$, the strong verification algorithm outputs

$$\mathsf{proj}_{q'}\big(\sum_{i=1}^{\ell} w^i \cdot \boldsymbol{u}^i\big) = \sum_{i=1}^{\ell} w^i \cdot \mathsf{proj}_{q'}(\boldsymbol{u}^i) \qquad \text{(Proposition 13)}$$

$$= \sum_{i=1}^{\ell} w^i \cdot \mathsf{iVrfy}(\mathsf{pp}, \boldsymbol{c}^i, t, \boldsymbol{d}^i) \qquad \text{(Def. of iVrfy)}$$

$$= \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{m}_t^i \ ,$$

as required. □

Lemma 20. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers and $0 < \varepsilon \leq 1$, such that n is a power of two, q, q' are prime. Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$ respectively. Then $\mathsf{HVC}_0^{\text{Chip}}$ is a robustly homomorphic HVC.*

Proof. The proof of this lemma is taken almost verbatim from [FSZ22a]. It deviates only insofar as the full construction and proof was split in two in [FSZ22a], whereas it is combined in one here. Since the proof is short, we include it here for the sake of completeness. Let $\boldsymbol{c}^0, \boldsymbol{c}^1 \in \mathcal{R}_q^{\ell_{\mathsf{com}}}$, and $\boldsymbol{d}^0, \boldsymbol{d}^1 \in \mathcal{R}^{\ell_{\mathsf{op}}}$, and $1 \leq t \leq 2^\tau$, $\tilde{t} = \mathsf{bin}_\tau(t-1)$ be arbitrary, such that

$$\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) = \boldsymbol{m}^0 \quad \text{and} \quad \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) = \boldsymbol{m}^1 \tag{3}$$

with $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \perp$. Let $\boldsymbol{d}^i$ parse as $(\boldsymbol{p}_1^i, \ldots, \boldsymbol{p}_\tau^i, \boldsymbol{s}_1^i, \ldots, \boldsymbol{s}_\tau^i, \boldsymbol{u}^i)^\intercal$ for $i \in \{0, 1\}$. We first note that *if* $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \neq \perp$, then it holds in $\mathcal{R}_{q'}^{\xi}$ that

$$\begin{aligned}
&\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \\
&= \mathsf{proj}_{q'}(\boldsymbol{u}^0 - \boldsymbol{u}^1) && \text{(Def of sVrfy)} \\
&= \mathsf{proj}_{q'}(\boldsymbol{u}^0) - \mathsf{proj}_{q'}(\boldsymbol{u}^1) && \text{(Proposition 13)} \\
&= \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) - \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) && \text{(Def. of sVrfy)} \\
&= \boldsymbol{m}^0 - \boldsymbol{m}^1. && \text{(Equation 3)}
\end{aligned}$$

It thus remains to show that $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \neq \perp$. For this, let further $\boldsymbol{p}_0^i = \boldsymbol{c}^i$. By definition of the strong verification algorithm, and since $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \perp$ it holds that for $i \in \{0, 1\}$ and $j \in [\tau]$ that the following two conditions hold

$$\left\|\boldsymbol{p}_j^i\right\| \leq \beta_{\mathsf{agg}} \quad \text{and} \quad \left\|\boldsymbol{s}_j^i\right\| \leq \beta_{\mathsf{agg}} \tag{4}$$

$$\mathsf{proj}_q(\boldsymbol{p}_{j-1}^i) = \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j^i + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_j^i \ . \tag{5}$$

Similarly it holds that

$$\left\|\boldsymbol{u}^i\right\| \leq \beta_{\mathsf{agg}} \quad \text{and} \quad \mathsf{proj}_q(\boldsymbol{p}_\tau^i) = \boldsymbol{g}^\intercal \cdot \boldsymbol{u}^i \ . \tag{6}$$

We also get the bounds on the projections

$$\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j^i)\right\| \leq \frac{q\beta_{\mathsf{agg}}}{2\eta} \quad \text{and} \quad \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j^i)\right\| \leq \frac{q\beta_{\mathsf{agg}}}{2\eta} \ . \tag{7}$$

From Equation 4 and Equation 6 it follows that for all $j \in [\tau]$

$$\left\|\boldsymbol{p}_j^0 - \boldsymbol{p}_j^1\right\| \leq \left\|\boldsymbol{p}_j^0\right\| + \left\|\boldsymbol{p}_j^1\right\| \leq 2\beta_{\mathsf{agg}}$$

$$\left\|\boldsymbol{s}_j^0 - \boldsymbol{s}_j^1\right\| \leq \left\|\boldsymbol{s}_j^0\right\| + \left\|\boldsymbol{s}_j^1\right\| \leq 2\beta_{\mathsf{agg}}$$

and

$$\left\|\boldsymbol{u}^0 - \boldsymbol{u}^1\right\| \leq \left\|\boldsymbol{u}^0\right\| + \left\|\boldsymbol{u}^1\right\| \leq 2\beta_{\mathsf{agg}} \enspace .$$

From Equation 7 and linearity of $\mathsf{proj}_{\eta,\kappa}$, it follows that

$$\begin{aligned}\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j^0 - \boldsymbol{p}_j^1)\right\| = \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j^0) - \mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j^1)\right\| \leq \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j^0)\right\| + \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j^1)\right\| \leq \tfrac{q\beta_{\mathsf{agg}}}{\eta} \\ \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j^0 - \boldsymbol{s}_j^1)\right\| = \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j^0) - \mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j^1)\right\| \leq \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j^0)\right\| + \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j^1)\right\| \leq \tfrac{q\beta_{\mathsf{agg}}}{\eta} \enspace .\end{aligned}$$

By Equations 5 and 6 and the linearity of proj_q it follows that for all $j \in [\tau]$, it holds in $\mathcal{R}_q$ that

$$\begin{aligned}\mathsf{proj}_q(\boldsymbol{p}_{j-1}^0 - \boldsymbol{p}_{j-1}^1) &= \mathsf{proj}_q(\boldsymbol{p}_{j-1}^0) - \mathsf{proj}_q(\boldsymbol{p}_{j-1}^1) && \text{(Proposition 13)} \\ &= (\boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j^0 + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_j^0) - (\boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j^1 + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_j^1) && \text{(Equation 5)} \\ &= \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot (\boldsymbol{p}_j^0 - \boldsymbol{p}_j^1) + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot (\boldsymbol{s}_j^0 - \boldsymbol{s}_j^1) \enspace .\end{aligned}$$

and

$$\begin{aligned}\mathsf{proj}_q(\boldsymbol{p}_\tau^0 - \boldsymbol{p}_\tau^1) &= \mathsf{proj}_q(\boldsymbol{p}_\tau^0) - \mathsf{proj}_q(\boldsymbol{p}_\tau^1) && \text{(Proposition 13)} \\ &= (\boldsymbol{g}^\intercal \cdot \boldsymbol{u}^0 - \boldsymbol{g}^\intercal \cdot \boldsymbol{u}^1) && \text{(Equation 6)} \\ &= \boldsymbol{g}^\intercal \cdot (\boldsymbol{u}^0 - \boldsymbol{u}^1) \enspace .\end{aligned}$$

Thus, all checks in the weak verification algorithm go through and $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \neq \bot$. □

Lemma 21. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers and $0 < \varepsilon \leq 1$, such that n is a power of two, q, q' are prime. Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$ respectively. If the $\mathsf{SIS}_{\mathcal{R},q,2\lceil\log_{2\eta+1} q\rceil,4\beta_{\mathsf{agg}}}$ problem and the $\mathsf{SIS}_{\mathcal{R},q,\xi\lceil\log_{2\eta+1} q'\rceil,4\beta_{\mathsf{agg}}}$ problem are hard, then $\mathsf{HVC}_0^{\mathrm{Chip}}$ is position binding.*

Proof. This proof once again follows very closely the proof shown in [FSZ22a]. We will prove this lemma by leveraging that any pair of valid decommitments for different messages will lead to a collision somewhere in the generalized hash tree, which can be turned into a solution for one of the SIS instances.

Let $\mathcal{A}$ be an arbitrary PPT adversary against the position binding property of the construction. By the law of total probability it holds that

$$\begin{aligned}&\Pr[\boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\}] \\ =\ &\Pr[\boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\} \wedge \mathsf{proj}_q(\boldsymbol{p}_\tau^0) = \mathsf{proj}_q(\boldsymbol{p}_\tau^1)] \\ &+ \Pr[\boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\} \wedge \mathsf{proj}_q(\boldsymbol{p}_\tau^0) \neq \mathsf{proj}_q(\boldsymbol{p}_\tau^1)] \enspace .\end{aligned}$$

We now bound the two probabilities separately.

$$\begin{aligned}&\Pr[\boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\} \wedge \mathsf{proj}_q(\boldsymbol{p}_\tau^0) = \mathsf{proj}_q(\boldsymbol{p}_\tau^1)] \\ \leq\ &\Pr[\mathsf{proj}_{q'}(\boldsymbol{u}^0) \neq \mathsf{proj}_{q'}(\boldsymbol{u}^1) \wedge \boldsymbol{g}^\intercal \cdot \boldsymbol{u}^0 = \boldsymbol{g}^\intercal \cdot \boldsymbol{u}^1 \wedge \left\|\boldsymbol{u}^0\right\| \leq 2\beta_{\mathsf{agg}} \wedge \left\|\boldsymbol{u}^1\right\| \leq 2\beta_{\mathsf{agg}}] \quad \text{(Def. of wVrfy)}\end{aligned}$$

$$\begin{aligned}&\leq \Pr[\boldsymbol{u}^0 \neq \boldsymbol{u}^1 \wedge \boldsymbol{g}^\intercal \cdot (\boldsymbol{u}^0 - \boldsymbol{u}^1) = 0 \wedge \left\|\boldsymbol{u}^0 - \boldsymbol{u}^1\right\| \leq 4\beta_{\mathsf{agg}}] \\ &= \Pr[(\boldsymbol{u}^0 - \boldsymbol{u}^1) \in \mathcal{B}_{4\beta_{\mathsf{agg}},q}^{\xi\lceil \log_{2\eta+1} q'\rceil} \setminus \{\boldsymbol{0}\} \wedge \boldsymbol{g}^\intercal \cdot (\boldsymbol{u}^0 - \boldsymbol{u}^1) = 0] \\ &\leq \mathsf{negl}(\lambda) \ ,\end{aligned}$$

where the last inequality follows from the assumed hardness of the $\mathsf{SIS}_{\mathcal{R},q,\xi\lceil \log_{2\eta+1} q'\rceil,4\beta_{\mathsf{agg}}}$ problem and the fact that all involved algorithms are PPT.

We now analyze

$$\Pr[\boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\} \wedge \boldsymbol{p}_\tau^0 \bmod q \neq \boldsymbol{p}_\tau^1 \bmod q] \ .$$

We construct a PPT algorithm $\overline{\mathcal{A}}$ that solves the $\mathsf{SIS}_{\mathcal{R},q,2\lceil \log_{2\eta+1} q\rceil,4\beta_{\mathsf{agg}}}$ problem as follows. Upon input $\boldsymbol{a} = (a_1, \ldots, a_{2\lceil \log_{2\eta+1} q\rceil})^\intercal$, $\overline{\mathcal{A}}$ sets $\boldsymbol{h}_0 := (a_1, \ldots, a_{\lceil \log_{2\eta+1} q\rceil})^\intercal$ and $\boldsymbol{h}_1 := (a_{\lceil \log_{2\eta+1} q\rceil+1}, \ldots, a_{2\lceil \log_{2\eta+1} q\rceil})^\intercal$, samples $\boldsymbol{g} \leftarrow \mathcal{R}_q^{\xi\lceil \log_{2\eta+1} q'\rceil}$, sets $\mathsf{pp} := (\boldsymbol{g}, \boldsymbol{h}_0, \boldsymbol{h}_1)$ and runs $(\boldsymbol{c}, t, \boldsymbol{d}^0, \boldsymbol{d}^1) \leftarrow \mathcal{A}(\mathsf{pp})$. For $i \in \{0,1\}$ let $m^i := \mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}^i)$. If $\boldsymbol{m}^0 = \boldsymbol{m}^1$, $\bot \in \{\boldsymbol{m}^0, \boldsymbol{m}^1\}$, or $\mathsf{proj}_q(\boldsymbol{p}_\tau^0) = \mathsf{proj}_q(\boldsymbol{p}_\tau^1)$, $\overline{\mathcal{A}}$ aborts. Otherwise, parse $\boldsymbol{d}^i$ as $(\boldsymbol{p}_1^i, \ldots, \boldsymbol{p}_\tau^i, \boldsymbol{s}_1^i, \ldots, \boldsymbol{s}_\tau^i, \boldsymbol{u}^i)$, set $\boldsymbol{p}_0^i := \mathsf{dec}_q(\boldsymbol{c})$, $\tilde{t} := \mathsf{bin}_\tau(t-1)$.

Let $j^* \in [\tau+1]$ be the *largest* index, such that $\mathsf{proj}_q(\boldsymbol{p}_{j^*-1}^0) \neq \mathsf{proj}_q(\boldsymbol{p}_{j^*-1}^1)$. Note that such an index always exists, since $\boldsymbol{p}_0^0 = \mathsf{dec}_q(\boldsymbol{c}) = \boldsymbol{p}_0^1$, and that $j^* < \tau$, since $\mathsf{proj}_q(\boldsymbol{p}_\tau^0) \neq \mathsf{proj}_q(\boldsymbol{p}_\tau^1)$. If $\tilde{t}_{j^*-1} = 0$, $\overline{\mathcal{A}}$ outputs $\boldsymbol{z} := (\boldsymbol{p}_{j^*}^0, \boldsymbol{s}_{j^*}^0)^\intercal - (\boldsymbol{p}_{j^*}^1, \boldsymbol{s}_{j^*}^1)^\intercal$, if $\tilde{t}_{j^*-1} = 1$, $\overline{\mathcal{A}}$ outputs $\boldsymbol{z} := (\boldsymbol{s}_{j^*}^0, \boldsymbol{p}_{j^*}^0)^\intercal - (\boldsymbol{s}_{j^*}^1, \boldsymbol{p}_{j^*}^1)^\intercal$.

We now analyze the success probability of $\overline{\mathcal{A}}$. It holds that $\mathsf{proj}_q(\boldsymbol{p}_{j^*-1}^0) = \mathsf{proj}_q(\boldsymbol{p}_{j^*-1}^1)$ and by the definition of the weak verification algorithm that

$$\begin{aligned}&\boldsymbol{h}_{\tilde{t}_{j^*}}^\intercal \cdot \boldsymbol{p}_{j^*}^0 + \boldsymbol{h}_{\tilde{t}_{j^*}\oplus 1}^\intercal \cdot \boldsymbol{s}_{j^*}^0 = \boldsymbol{h}_{\tilde{t}_{j^*}}^\intercal \cdot \boldsymbol{p}_{j^*}^1 + \boldsymbol{h}_{\tilde{t}_{j^*}\oplus 1}^\intercal \cdot \boldsymbol{s}_{j^*}^1 \\ \iff &\boldsymbol{h}_{\tilde{t}_{j^*}}^\intercal \cdot (\boldsymbol{p}_{j^*}^0 - \boldsymbol{p}_{j^*}^1) + \boldsymbol{h}_{\tilde{t}_{j^*}\oplus 1}^\intercal \cdot (\boldsymbol{s}_{j^*}^0 - \boldsymbol{s}_{j^*}^1) = 0 \\ \iff &\boldsymbol{a}^\intercal \cdot \boldsymbol{z} = \boldsymbol{0} \ .\end{aligned}$$

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

$$\left\|\boldsymbol{p}_{j^*}^0\right\| \leq 2\beta_{\mathsf{agg}}, \quad \left\|\boldsymbol{s}_{j^*}^0\right\| \leq 2\beta_{\mathsf{agg}}, \quad \left\|\boldsymbol{p}_{j^*}^1\right\| \leq 2\beta_{\mathsf{agg}}, \quad \left\|\boldsymbol{s}_{j^*}^1\right\| \leq 2\beta_{\mathsf{agg}} \ .$$

Therefore, the norm of $\boldsymbol{z}$ can be bounded as

$$\|\boldsymbol{z}\| \leq \max\{\left\|\boldsymbol{p}_{j^*}^0\right\|, \left\|\boldsymbol{s}_{j^*}^0\right\|\} + \max\{\left\|\boldsymbol{p}_{j^*}^1\right\|, \left\|\boldsymbol{s}_{j^*}^1\right\|\} \leq 4\beta_{\mathsf{agg}} \ .$$

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

$$\mathsf{proj}_q(\boldsymbol{p}_{j^*-1}^0) = \mathsf{proj}_q(\boldsymbol{p}_{j^*-1}^1) \ ,$$

it holds that

$$\mathsf{proj}_q(\boldsymbol{p}_{j^*}^0) \neq \mathsf{proj}_q(\boldsymbol{p}_{j^*}^1)$$

and thereby that

$$\boldsymbol{p}_{j^*}^0 \neq \boldsymbol{p}_{j^*}^1 \ .$$

Therefore $\boldsymbol{z} \neq \boldsymbol{0}$. Thus, whenever $\mathcal{A}$ is successful, $\overline{\mathcal{A}}$ is successful with probability 1 and we can conclude that

$$\mathsf{negl}(\lambda) \geq \Pr[\boldsymbol{a} \leftarrow \mathcal{R}_q^{2\lceil \log_{2\eta+1} q\rceil}; \boldsymbol{z} \leftarrow \overline{\mathcal{A}}(\boldsymbol{a}) \colon \boldsymbol{z} \in \mathcal{B}_{4\beta_{\mathsf{agg}},q}^{2\lceil \log_{2\eta+1} q\rceil} \setminus \{\boldsymbol{0}\} \wedge \boldsymbol{a}^\intercal \boldsymbol{z} \equiv 0]$$

$$= \Pr[\boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\} \wedge \boldsymbol{p}_\tau^0 \bmod q \neq \boldsymbol{p}_\tau^1 \bmod q] \ .$$

Combining the above, it follows that

$$\Pr[\boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \bot \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\}] \leq \mathsf{negl}(\lambda) \ ,$$

as required. □

4 Encoding HVC openings

To reduce the size needed to transmit openings in our final HVC construction, we employ a nontrivial encoding scheme. Let us first sketch the idea and how it relates to lattice enumeration, before defining it more formally in Figure 4.

Our HVC construction is, except for projections and decompositions, a Merkle tree with a homomorphic hash function. Time slots t correspond to paths in the Merkle tree and our openings contain the labels $\boldsymbol{p}_i$ along the path, together with the sibling nodes' labels $\boldsymbol{s}_i$. Usually, when opening a path of a Merkle tree, it is not necessary to actually include most of the nodes $\boldsymbol{p}_i$ along the Merkle path in the opening, but only the sibling nodes $\boldsymbol{s}_i$ (ignoring possible special handling at the root or leaf). The reason is that for any valid opening of a usual Merkle tree, we have

$$H(\boldsymbol{p}_i, \boldsymbol{s}_i) = \boldsymbol{p}_{i-1} \quad \text{or} \quad H(\boldsymbol{s}_i, \boldsymbol{p}_i) = \boldsymbol{p}_{i-1} \ ,$$

where H is the hash function used in the construction (concretely for us, Ajtai's hash function h_{Ajtai}). This allows the verifier to compute $\boldsymbol{p}_{i-1}$ from $\boldsymbol{p}_i$ by itself, if given $\boldsymbol{s}_i$.

For us, the corresponding relation (ignoring smallness constraints) instead reads

$$H(\boldsymbol{p}_i, \boldsymbol{s}_i) = \mathsf{proj}_q(\boldsymbol{p}_{i-1}) \quad \text{or} \quad H(\boldsymbol{s}_i, \boldsymbol{p}_i) = \mathsf{proj}_q(\boldsymbol{p}_{i-1})$$

throwing proj_q, i.e. $\mathsf{proj}_{\eta,\kappa}$ and reduction modulo q, in the mix, which complicates things.

Now, for individually verifying openings, the above idea still works out due the size constraints: the bounds $\|\boldsymbol{p}_i\| \leq \eta$ and $\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_i)\right\| \leq \frac{q-1}{2}$ for individually verifying openings imply that $\boldsymbol{p}_i$ is actually uniquely determined by $\mathsf{proj}_q(\boldsymbol{p}_i)$. Indeed, $\boldsymbol{p}_i$ is given by $\boldsymbol{p}_i = \mathsf{dec}_q(\mathsf{proj}_q(\boldsymbol{p}_i))$, leading to

$$\boldsymbol{p}_{i-1} = \mathsf{dec}_q(H(\boldsymbol{p}_i, \boldsymbol{s}_i)) \quad \text{or} \quad \boldsymbol{p}_{i-1} = \mathsf{dec}_q(H(\boldsymbol{s}_i, \boldsymbol{p}_i)) \ .$$

For aggregate openings, this unfortunately no longer holds: for any given output of Ajtai's hash function, which we will denote as hint $\in \mathcal{R}_q$ in our algorithm, the equation $\mathsf{proj}_q(\boldsymbol{p}_i) = \text{hint}$ can have many solutions that satisfy the more relaxed size constraints that we impose on aggregate openings.

Ignoring any size constraints, for a given hint $\in \mathcal{R}_q$, the set of solutions to $\mathsf{proj}_q(\boldsymbol{p}_i) = \text{hint}$ is a lattice coset $\mathcal{C}$ of the form $\mathcal{C} = \Lambda_{\mathcal{R},q} + \boldsymbol{t}$, where $\Lambda_{\mathcal{R},q} := \{\boldsymbol{x} \in \mathcal{R}^\kappa \mid \mathsf{proj}_{\eta,\kappa}(\boldsymbol{x}) \bmod q = 0\}$, with $\kappa = \lceil \log_{2\eta+1} q \rceil$. The vector $\boldsymbol{t}$ depends on hint. Note that while the coset $\mathcal{C}$ is uniquely determined by hint, there are multiple possible choices for $\boldsymbol{t}$. The different possible choices differ exactly by elements from $\Lambda_{\mathcal{R},q}$. Our task now boils down to efficiently encode small elements $\boldsymbol{p}_i$ (in $\|.\|_\infty$-norm) from this lattice coset $\mathcal{C}$. Both encoder and decoder know hint (and hence $\mathcal{C}$) from hashing the child nodes; note that for notational convenience, our encoder defined in Figure 4 instead determines hint by computing $\mathsf{proj}_q(\boldsymbol{p}_i)$.

Before formally defining our encoding and decoding algorithms, let us give some more informal remark that explains the basic idea and how it relates to Babai's algorithm and lattice enumeration. Note that our actual algorithm in Figure 4 and its formal analysis given below will be fully self-contained and do not rely on this remark in any way.

Remark 2 (Lattice enumeration). Let us now explain how our encoding and decoding is connected to Babai's algorithm [Bab86] (more precisely, the generalization in [?]) and lattice enumeration. The problem we need to solve is to encode some $\boldsymbol{v} \in \mathcal{C} = \boldsymbol{t} + \mathit{\Lambda}$ via an encoding $\overline{\boldsymbol{v}}$, where short $\boldsymbol{v}$ should correspond to short(er) $\overline{\boldsymbol{v}}$. Here, $\mathit{\Lambda}$ can be an arbitrary (full-rank) lattice at first that we will later take to be $\mathit{\Lambda}_{\mathcal{R},q}$. Let us assume we have some pre-agreed basis $\mathcal{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots\}$ for $\mathit{\Lambda}$, either over $\mathbb{Z}$ or (if $\mathit{\Lambda}$ is a free $\mathcal{R}$-module) over our ring $\mathcal{R}$.

The most simple idea, corresponding to what's called Babai rounding, is to deterministically (so both encoder and decoder agree on it) determine some $\boldsymbol{t}_{\mathrm{ref}}$ from hint, or, equivalently, from $\mathcal{C}$, such that $\mathcal{C} = \boldsymbol{t}_{\mathrm{ref}} + \mathit{\Lambda}$. Then $\boldsymbol{v} - \boldsymbol{t}_{\mathrm{ref}} \in \mathit{\Lambda}$ and we can encode $\boldsymbol{v}$ by the coordinate vector $(\alpha_1, \alpha_2 \ldots)$ of $\boldsymbol{v} - \boldsymbol{t}_{\mathrm{ref}}$ with respect to the basis $\mathcal{B}$.

How good this is (i.e. how small the α_i are) depends on both how "good" the basis $\mathcal{B}$ is and how we choose $\boldsymbol{t}_{\mathrm{ref}}$. For the latter, we want $\boldsymbol{v} - \boldsymbol{t}_{\mathrm{ref}}$ to be small, so $\boldsymbol{t}_{\mathrm{ref}}$ should itself be small. One way (phrased for a $\mathbb{Z}$-basis for simplicity) called Babai rounding to choose $\boldsymbol{t}_{\mathrm{ref}}$ is to set $\boldsymbol{t}_{\mathrm{ref}} = \sum_i t_i \boldsymbol{b}_i$ with real-valued coefficients $-\frac{1}{2} < t_i \leq \frac{1}{2}$. Observe that we have $\boldsymbol{v} = \sum_i (t_i + \alpha_i)\boldsymbol{b}_i$. This means that this approach essentially computes the α_i by writing $\boldsymbol{v}$ with respect to $\mathcal{B}$ and rounding the coefficients to the nearest integers.

Note that this approach first computes a short reference $\boldsymbol{t}_{\mathrm{ref}} \in \mathcal{C}$ in some way and then separately encodes $\boldsymbol{v}$ by encoding the difference. An equivalent, useful view, is to consider this as a single algorithm akin to lattice enumeration: we want to output not just a single short vector $\boldsymbol{t}_{\mathrm{ref}}$ of a lattice coset, but rather enumerate (candidate) short vectors $\boldsymbol{v}$. For this, we don't set a single t_i's with $-\frac{1}{2} < t_i \leq \frac{1}{2}$, but rather enumerate possible candidate short vectors by also trying larger values of t_i, parameterized by α_i's. In lattice enumeration, where the goal is to find a vector as short as possible, we usually try a large number of such candidates and settle for the shortest one we found (usually, this takes super-polynomial time). Here, we are given $\boldsymbol{v}$ and we encode $\boldsymbol{v}$ by the branch (parameterized by α_i) a lattice enumeration algorithm would need to take to output $\boldsymbol{v}$.

This point of view lets us use Babai's algorithm proper rather than the more naive Babai rounding: here, we (greedily) choose coefficients wrt. a given basis $\mathcal{B}$, but differently to Babai rounding, we choose coefficients one-by-one and each choice is affected by the previous choices.

For a recursive description of Babai's algorithm / enumeration, pick one[9] of the basis vectors $\boldsymbol{b}_* \in \mathcal{B}$ and decompose $\mathcal{B}$ into a disjoint union $\mathcal{B} = \{\boldsymbol{b}_*\} \cup \mathcal{B}'$. This decomposes $\mathit{\Lambda}$ as $\mathit{\Lambda} = \mathit{\Lambda}' \oplus \mathrm{Span}\, \boldsymbol{b}_*$, where $\mathit{\Lambda}'$ is the lattice generated by $\mathcal{B}'$. We now choose $\boldsymbol{t}_{\mathrm{ref}}^{(1)} \in \mathcal{C}$ and then pick the unique α_* such that $\boldsymbol{v} \in \alpha_* \boldsymbol{b}_* + \boldsymbol{t}_{\mathrm{ref}}^{(1)} + \mathit{\Lambda}'$. This is similar to the approach before, except that now we only determine a single coefficient α_*. Note that the choice of $\boldsymbol{t}_{\mathrm{ref}}^{(1)}$ only matters modulo $\mathit{\Lambda}'$, so only the single (real-valued) $\boldsymbol{b}_*$-component of $\boldsymbol{t}_{\mathrm{ref}}^{(1)}$ matters. This is hence a 1-dimensional problem and Babai's algorithm (which only works for $\mathbb{Z}$-coefficients and is designed for the $\|.\|_2$-norm) chooses $\boldsymbol{t}_{\mathrm{ref}}^{(1)}$ such that the $\|.\|_2$-distance between $\boldsymbol{t}_{\mathrm{ref}}^{(1)}$ and $\mathrm{Span}_{\mathbb{R}}\, \mathcal{B}'$ is minimized (with some arbitrary deterministic tie-breakers). We then recurse into the new problem instance given by $\boldsymbol{v} \in \mathcal{C}'$ with $\mathcal{C}' = \mathit{\Lambda}' + \alpha_* \boldsymbol{b}_* + \boldsymbol{t}_{\mathrm{ref}}^{(1)}$

[9] The choice of $\boldsymbol{b}_*$ matters. This algorithm is typically applied to bases $(\boldsymbol{b}_1, \ldots)$ obtained from lattice reduction. Lattice reduction outputs an ordered basis and with the ordering convention from lattice reduction, the appropriate choice for Babai's algorithm is to choose the *last* basis element as $\boldsymbol{b}_*$

of dimension 1 less than the original. Note that to get a full-rank lattice, we may orthogonally project out the orthogonal complement to $\operatorname{Span}_{\mathbb{R}} \varLambda'$. The 0-dimensional base case is trivial. Babai's algorithm itself only considers $\alpha_* = 0$, lattice enumeration branches into several candidate α_* and our approach sets α_* from $\boldsymbol{v}$. Importantly, the next $\boldsymbol{t}_{\text{ref}}^{(2)}$ chosen in the next step during the recursion depends on $\mathcal{C}'$ and hence on the previous choice of α_*.

For our problem with $\varLambda = \varLambda_{\mathcal{R},q}$ and $\mathcal{C}$ determined by hint $\in \mathcal{R}_q$, we do not work over the $\|.\|_2$-norm, but rather the $\|.\|_\infty$-norm. Fortunately, our lattice has a very good basis for this norm (close to the coordinate axes). We take a very similar general approach, but instead of choosing $\boldsymbol{t}_{\text{ref}}^{(1)}$ via a $\|.\|_2$-minimization problem, we directly choose $\boldsymbol{t}_{\text{ref}}^{(1)} := \mathsf{dec}_q(\text{hint})$. Note that this equals $\boldsymbol{t}_{\text{ref}}^{(1)} = \mathsf{dec}_q(\mathsf{proj}_q(\boldsymbol{v}))$. In the next recursion step, we set $\boldsymbol{t}_{\text{ref}}^{(2)}$ as $\mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}))$. Note here that our basis element $\boldsymbol{b}_*$ in the first step is given by $(q, 0, 0, \ldots)$. Essentially, determining α_* in the first recursion step allows us to "update" the information hint we are given from something modulo q to something unreduced, which helps the algorithm by choosing a better $\boldsymbol{t}_{\text{ref}}^{(2)}$. Our algorithm does not need to perform any orthogonal projections and works over $\mathcal{R}$. We also only perform this recursive step once and use the Babai rounding approach after one step; the reason is that the shape of our basis is so good that this is sufficient.

Let us now proceed to define our encoding and decoding formally. For this, we need bases of the relevant lattices.

Proposition 22. *Let q, η be positive integers with q prime. Set $\kappa := \lceil \log_{2\eta+1} q \rceil$ and define lattices*

$$\begin{aligned}\varLambda_{\mathcal{R}} &:= \{\boldsymbol{v} \in \mathcal{R}^\kappa \mid \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) = 0\}\\ \varLambda_{\mathcal{R},q} &:= \{\boldsymbol{v} \in \mathcal{R}^\kappa \mid \mathsf{proj}_q(\boldsymbol{v}) = 0\}\end{aligned}$$

for the kernels of $\mathsf{proj}_{\eta,\kappa}$ and proj_q, respectively. Define vectors $\boldsymbol{b}_*$ and $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1} \in \mathcal{R}^\kappa$ as

$$\begin{aligned}\boldsymbol{b}_* &= (q, 0, 0, \ldots, 0)\\ \boldsymbol{b}_1 &= (-(2\eta+1), 1, 0, 0, \ldots, 0)\\ \boldsymbol{b}_2 &= (0, -(2\eta+1), 1, 0, \ldots, 0)\\ &\cdots\\ \boldsymbol{b}_{\kappa-1} &= (0, 0, \ldots, 0, -(2\eta+1), 1)\ .\end{aligned}$$

Then $\varLambda_{\mathcal{R}}$ and $\varLambda_{\mathcal{R},q}$ are $\mathcal{R}$-module lattices (i.e. lattices that are also free $\mathcal{R}$-modules). A basis (over $\mathcal{R}$) for $\varLambda_{\mathcal{R}}$ is given by $\mathcal{B} := \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}\}$ and a basis over $\mathcal{R}$ for $\varLambda_{\mathcal{R},q}$ is given by $\{\boldsymbol{b}_*, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}\}$.

Proof. Note that $\mathcal{R}$ is a free $\mathbb{Z}$-module, i.e. $\mathcal{R} \cong \mathbb{Z}^n$ as a $\mathbb{Z}$-module (but not as a ring), so the whole notion of $\mathcal{R}$-module lattice even makes sense. Being kernels of appropriate $\mathcal{R}$-linear maps, $\varLambda_{\mathcal{R}}$ and $\varLambda_{\mathcal{R},q}$ are clearly $\mathcal{R}$-module lattices. Recall that $\mathsf{proj}_{\eta,\kappa}$ is defined as

$$\mathsf{proj}_{\eta,\kappa}\colon\ \mathcal{R}^\kappa \to \mathcal{R}, \quad \mathsf{proj}_{\eta,\kappa}(v_1, \ldots, v_\kappa) = \sum_{i=1}^{\kappa} (2\eta+1)^{i-1} \cdot v_i\ .$$

We easily compute that $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{b}_*) = q$ and $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{b}_i) = 0$ for $1 \leq i \leq \kappa - 1$. Hence, all $\boldsymbol{b}_i$ and $\boldsymbol{b}_*$ are in the appropriate lattices. They are also clearly linearly independent due to the triangular shape of $\{\boldsymbol{b}_*\} \cup \mathcal{B}$.

We now need to show that they also span $\Lambda_{\mathcal{R}}$ resp. $\Lambda_{\mathcal{R},q}$, i.e. that $\Lambda_{\mathcal{R}} \subset \operatorname{Span}_{\mathcal{R}} \mathcal{B}$ and $\Lambda_{\mathcal{R},q} \subset \operatorname{Span}_{\mathcal{R}}(\mathcal{B} \cup \{\boldsymbol{b}_*\})$. For this, consider any $\boldsymbol{x} \in \mathcal{R}^\kappa$. Without the first column, $\mathcal{B}$, viewed as a matrix, is a $(\kappa-1)\times(\kappa-1)$ lower triangular matrix with 1's on the diagonal. This implies that the projection of $\operatorname{Span}_{\mathcal{R}} \mathcal{B}$ onto the last $\kappa-1$ coefficients (over $\mathcal{R}$) is all of $\mathcal{R}^{\kappa-1}$, and this projection is bijective. This means we can find a unique $\tilde{\boldsymbol{x}} \in \operatorname{Span}_{\mathcal{R}} \mathcal{B}$ matching $\boldsymbol{x}$ on the last $\kappa-1$ coefficients, giving a unique decomposition

$$\boldsymbol{x} = (x', 0, \ldots,\ 0) + \tilde{\boldsymbol{x}}$$

with $x' \in \mathcal{R}, \tilde{\boldsymbol{x}} \in \operatorname{Span}_{\mathcal{R}} \mathcal{B}$. Since $\operatorname{Span}_{\mathcal{R}} \mathcal{B} \subset \Lambda_{\mathcal{R}}$, we have

$$\mathsf{proj}_{\eta,\kappa}(\boldsymbol{x}) = \mathsf{proj}_{\eta,\kappa}((x', 0, \ldots, 0)) + \mathsf{proj}_{\eta,\kappa}(\tilde{\boldsymbol{x}}) = x' + 0 = x' \ .$$

Consequently, if $\boldsymbol{x} \in \Lambda_{\mathcal{R}}$, we have by definition $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{x}) = 0$, so $x' = 0$ and $\boldsymbol{x} = \tilde{\boldsymbol{x}}$. This means $\boldsymbol{x} = \tilde{\boldsymbol{x}} \in \operatorname{Span}_{\mathcal{R}} \mathcal{B}$, giving $\Lambda_{\mathcal{R}} \subset \operatorname{Span}_{\mathcal{R}} \mathcal{B}$. Similarly, if $\boldsymbol{x} \in \Lambda_{\mathcal{R},q}$, we have $\mathsf{proj}_{\eta,\kappa}\boldsymbol{x} \equiv 0 \mod q$, so $x' \bmod q = 0$. This gives $(x', 0, \ldots, 0) \in \operatorname{Span}_{\mathcal{R}} \boldsymbol{b}_*$. Together with $\tilde{\boldsymbol{x}} \in \operatorname{Span}_{\mathcal{R}} \mathcal{B}$, this implies $\boldsymbol{x} \in \operatorname{Span}_{\mathcal{R}}(\mathcal{B} \cup \{\boldsymbol{b}_*\})$.

□

Formally, we define encoding and decoding algorithms $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$ and $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}$ as in Figure 4.

```
$\mathsf{Encode}^{\mathrm{B}}_{\eta,q}(\boldsymbol{v})$
$\kappa := \lceil \log_{2\eta+1} q \rceil$
hint $:= \mathsf{proj}_q(\boldsymbol{v}) \in \mathcal{R}_q$
Represent hint by $\mathrm{hint}' \in \mathcal{R}$, $\|\mathrm{hint}'\| \le \frac{q-1}{2}$
$\alpha_* := \dfrac{\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) - \mathrm{hint}'}{q} \in \mathcal{R}$   // numerator divisible by $q$
$\boldsymbol{\delta}_v := \boldsymbol{v} - \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})) \in \mathcal{R}^\kappa$
Find $\alpha_1, \ldots, \alpha_{\kappa-1} \in \mathcal{R}$ s.t.   // Exist by Lemma 23
  $\boldsymbol{\delta}_v = \alpha_1 \boldsymbol{b}_1 + \ldots + \alpha_{\kappa-1}\boldsymbol{b}_{\kappa-1}$
return $\overline{\boldsymbol{v}} = (\alpha_*, \alpha_1, \ldots, \alpha_{\kappa-1})$
```

```
$\mathsf{Decode}^{\mathrm{B}}_{\eta,q}(\overline{\boldsymbol{v}}, \mathrm{hint})$
$\kappa := \lceil \log_{2\eta+1} q \rceil$
Represent hint $\in \mathcal{R}_q$ by $\mathrm{hint}' \in \mathcal{R}$, $\|\mathrm{hint}'\| \le \frac{q-1}{2}$
parse $\overline{\boldsymbol{v}}$ as $(\alpha_*, \alpha_1, \ldots, \alpha_{\kappa-1})$
$h'' := \mathrm{hint}' + q \cdot \alpha_*$   // We show $h'' = \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})$
$\boldsymbol{\delta}_v := \alpha_1 \boldsymbol{b}_1 + \ldots + \alpha_{\kappa-1}\boldsymbol{b}_{\kappa-1}$
return $\mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v \in \mathcal{R}^\kappa$
```

Fig. 4: Algorithms for encoding and decoding an element $\boldsymbol{v} \in \mathcal{R}^\kappa$. Decoding requires hint $= \mathsf{proj}_q(\boldsymbol{v})$. The vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}$ are the basis of $\Lambda_{\mathcal{R}}$ as defined in Proposition 22.

Remark 3. We mention that our formal description in Figure 4 of the algorithm is self-contained and actually makes no explicit mention of $\boldsymbol{t}^{(1)}_{\mathrm{ref}}$ or $\boldsymbol{t}^{(2)}_{\mathrm{ref}}$ as defined in Remark 2. However, it is easy to see that (in the notation given in the algorithm) $\boldsymbol{\delta}_v = \boldsymbol{v} - \boldsymbol{t}^{(2)}_{\mathrm{ref}}$ and we just encode that by coefficients as in Babai rounding. With $\boldsymbol{t}^{(1)}_{\mathrm{ref}} = \mathsf{dec}_q(\mathrm{hint})$, the definition of α_* given in Remark 2 means that this α_* is such that $\boldsymbol{v} = \alpha_* \boldsymbol{b}_* + \mathsf{dec}_q(\mathrm{hint}) + \boldsymbol{v}'$ with $\boldsymbol{v}' \in \Lambda' = \Lambda_{\mathcal{R}}$.

Observe that $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{b}_*) = q$, $\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_q(\mathrm{hint})) = \mathrm{hint}'$ and $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}') = 0$. This gives

$$\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) = \mathsf{proj}_{\eta,\kappa}(\alpha_* \boldsymbol{b}_*) + \mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_q(\mathrm{hint})) + \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}') = \alpha_* q + \mathrm{hint}' \ .$$

From this, it follows that the definition of α_* from Figure 4 as $\alpha_* := \frac{\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) - \mathrm{hint}'}{q}$ and the one from Remark 2 actually coincide.

Lemma 23 (Properties of $\mathsf{Encode}^{\mathbf{B}}_{\eta,q}$ and $\mathsf{Decode}^{\mathbf{B}}_{\eta,q}$). *Let q, η, κ be positive integers with q prime and $\kappa = \lceil \log_{2\eta+1} q \rceil$. Then the deterministic encoding and decoding algorithms defined in Figure 4 satisfy the following properties:*

1. *Coefficients $\alpha_*, \alpha_1 \ldots, \alpha_{\kappa-1}$ as required in $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$ exist, are unique and can be found in polynomial time.*
2. *For any $\boldsymbol{v} \in \mathcal{R}^\kappa$, $\mathrm{hint} = \mathsf{proj}_q(\boldsymbol{v})$ and $\overline{\boldsymbol{v}} \leftarrow \mathsf{Encode}^{\mathrm{B}}_{\eta,q}(\boldsymbol{v})$ we have $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}(\overline{\boldsymbol{v}}, \mathrm{hint}) = \boldsymbol{v}$.*
3. *For any $\overline{\boldsymbol{v}} \in \mathcal{R}^\kappa$, $\mathrm{hint} \in \mathcal{R}_q$ and $\boldsymbol{v} \leftarrow \mathsf{Decode}^{\mathrm{B}}_{\eta,q}(\overline{\boldsymbol{v}}, \mathrm{hint})$, we have $\mathsf{proj}_q(\boldsymbol{v}) = \mathrm{hint}$ and $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}(\boldsymbol{v}) = \overline{\boldsymbol{v}}$.*
4. *For any $\boldsymbol{v} \in \mathcal{R}^\kappa$ and $(\alpha_*, \alpha_1, \ldots, \alpha_{\kappa-1}) \leftarrow \mathsf{Encode}^{\mathrm{B}}_{\eta,q}(\boldsymbol{v})$, we have*

$$\|\alpha_*\| < \frac{\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})\|}{q} + \frac{1}{2} \qquad \textit{and}$$
$$\|\alpha_i\| < \frac{\|\boldsymbol{v}\|}{2\eta} + \frac{1}{2} \qquad \textit{for } 1 \le i \le \kappa - 1 \ .$$

Proof. For each of the individual claims, let notation be as in the definitions of the algorithms in Figure 4. Note that variables $\alpha_*, \boldsymbol{\delta}_v, \mathrm{hint}, \mathrm{hint}', \alpha_1 \ldots, \alpha_{\kappa-1}$ appearing in both $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$ and $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}$ with the same name actually have the same value as far as this proof is concerned. This only matters for item 2, where it is obvious, and for item 3, where we actually need to prove it for $\boldsymbol{\delta}_v, \alpha_*, \alpha_1, \ldots, \alpha_{\kappa-1}$.

Let us now prove each individual claim in order.

For item 1, note that $\mathrm{hint}' \equiv \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) \mod q$ by definition, so the division by q makes sense in $\mathcal{R}$. For the α_i, note that $\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(\boldsymbol{x})) = \boldsymbol{x}$ for all $\boldsymbol{x} \in \mathcal{R}$. This implies that

$$\mathsf{proj}_{\eta,\kappa}(\boldsymbol{\delta}_v) = \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) - \mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}))) = \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) - \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) = 0 \ .$$

So $\boldsymbol{\delta}_v \in \Lambda_{\mathcal{R}}$ and, by Proposition 22, coefficients α_i exist and are unique. They can clearly be found by solving an (overdetermined) system of linear equations (over $\mathcal{R}$). In fact, $\mathcal{B}$ is already in appropriate echelon form, so this can even be done without divisions in $\mathcal{R}$ and is clearly polynomial time. We will write down another solution explicitly alongside the proof of item 4.

Let us now tackle item 2. During decoding, note that

$$h'' = \mathrm{hint}' + q \cdot \alpha_* = \mathrm{hint}' + q \cdot \frac{\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) - \mathrm{hint}'}{q} = \mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) \ .$$

It follows that

$$\mathsf{Decode}^{\mathrm{B}}_{\eta,q}(\overline{\boldsymbol{v}}, h) = \mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v = \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})) + (\boldsymbol{v} - \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}))) = \boldsymbol{v} \ ,$$

as desired.

For item 3, let $\alpha_*, \boldsymbol{\delta}_v, \alpha_i, \mathrm{hint}, \mathrm{hint}'$ denote the values used during the computation by $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}$. While the equally named values in $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$ are actually the same, we need to prove this. First, note that $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{b}_i) = 0$ for all $1 \leq i \leq \kappa - 1$. By $\mathcal{R}$-linearity, it follows that $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{\delta}_v) = 0$. From this, we get

$$\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) = \mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v) = h'' = \mathrm{hint}' + q \cdot \alpha_* \equiv \mathrm{hint}' \equiv \mathrm{hint} \mod q \ ,$$

so $\mathsf{proj}_q(\boldsymbol{v}) = \mathrm{hint}$. In particular, the values of hint and hint' used in $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$ match those in $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}$. During the computation of $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}(\boldsymbol{v})$ with $\boldsymbol{v} = \mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v$ output by $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}$, we compute

$$\frac{\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) - \mathrm{hint}'}{q} = \frac{\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v) - \mathrm{hint}'}{q} = \frac{h'' - \mathrm{hint}'}{q} = \alpha_* \ ,$$

using again linearity and that $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{\delta}_v) = 0$. So the value of α_* recovered inside $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$ is the same as the value of α_* in $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}$. Similarly, during the computation in $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$, we have

$$\begin{aligned}
\boldsymbol{v} - \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})) &= \mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v - \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v)) \\
&= \mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v - \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(h'')) + \mathsf{proj}_{\eta,\kappa}(\boldsymbol{\delta}_v)) \\
&= \mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v - \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(h'')) + 0) \\
&= \mathsf{dec}_{\eta,\kappa}(h'') + \boldsymbol{\delta}_v - \mathsf{dec}_{\eta,\kappa}(h'') = \boldsymbol{\delta}_v \ ,
\end{aligned}$$

so the value for $\boldsymbol{\delta}_v$ obtained in $\mathsf{Encode}^{\mathrm{B}}_{\eta,q}$ is the same as that in $\mathsf{Decode}^{\mathrm{B}}_{\eta,q}$. Since $\mathcal{B}$ is an $\mathcal{R}$-basis, it follows that the values of the α_i are also the same, which proves this item.

We now tackle the last item 4, giving bounds on the encodings. The first bound is just an easy application of the triangle inequality:

$$\|\alpha_*\| = \left\| \frac{\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}) - \mathrm{hint}'}{q} \right\| \leq \frac{1}{q}\Big(\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})\right\| + \left\|\mathrm{hint}'\right\| \Big) \leq \frac{1}{q}\Big(\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})\right\| + \tfrac{q-1}{2} \Big) < \frac{\left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})\right\|}{q} + \frac{1}{2}$$

For the other bound, let us look at the individual components of $\boldsymbol{v}$ and $\mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v})) \in \mathcal{R}^\kappa$. For this, set $(w_1, \dots, w_\kappa) \coloneqq \mathsf{dec}_{\eta,\kappa}(\mathsf{proj}_{\eta,\kappa}(\boldsymbol{v}))$ and $(v_1, \dots, v_\kappa) \coloneqq \boldsymbol{v}$ with $v_i, w_i \in \mathcal{R}$. By definition of $\mathsf{dec}_{\eta,\kappa}$, we have $\|w_i\| \leq \eta$ for $1 \leq i < \kappa$. Note that this bound excludes the most significant limb. Writing the equation $\boldsymbol{\delta}_v = \alpha_1 \boldsymbol{b}_1 + \ldots + \alpha_{\kappa-1}\boldsymbol{b}_{\kappa-1}$ in its components, using the definition of $\mathcal{B}$ gives the following linear system of equations (over $\mathcal{R}$) in unknowns $\alpha_1, \dots, \alpha_{\kappa-1}$.

$$\begin{aligned}
v_1 - w_1 &= -(2\eta+1)\alpha_1 \\
v_2 - w_2 &= \alpha_1 - (2\eta+1)\alpha_2 \\
v_3 - w_3 &= \alpha_2 - (2\eta+1)\alpha_3 \\
&\dots \\
v_{\kappa-1} - w_{\kappa-1} &= \alpha_{\kappa-2} - (2\eta+1)\alpha_{\kappa-1} \\
v_\kappa - w_\kappa &= \alpha_{\kappa-1}
\end{aligned}$$

We now prove the bound for $\|\alpha_i\|$ by induction over i, using those equations[10][11].

[10] We have more equations than variables α_i and we will not use the last equation.

[11] It may be helpful to think of these equations as equations between polynomials where we allow *rational* coefficients. By item 1, we know a priori that the (unique) rational solution for the α_i will turn out integral.

For $i = 1$, the first equation above gives $\alpha_1 = -\frac{v_1 - w_1}{2\eta+1}$. This implies

$$\|\alpha_1\| \leq \frac{\|v_1\| + \|w_1\|}{2\eta+1} \leq \frac{\|\boldsymbol{v}\|}{2\eta+1} + \frac{\eta}{2\eta+1} < \frac{\|\boldsymbol{v}\|}{2\eta} + \frac{1}{2} \ .$$

For $1 \leq i \leq \kappa - 1$, we have $\alpha_i = -\frac{v_i - w_i - \alpha_{i-1}}{2\eta+1}$. By induction, we can bound this as

$$\|\alpha_i\| \leq \frac{\|v_i\| + \|w_i\| + \|\alpha_{i-1}\|}{2\eta+1} < \frac{\|\boldsymbol{v}\| + \eta + \frac{\|\boldsymbol{v}\|}{2\eta} + \frac{1}{2}}{2\eta+1} = \frac{\|\boldsymbol{v}\|}{2\eta} + \frac{1}{2} \ ,$$

which finishes the proof. □

Definition 24. *Let $n, \eta, q, q', \xi \in \mathbb{N}$ with n a power of two, q, q' primes. Consider the HVC construction $\mathsf{HVC}_0^{\mathrm{Chip}}$ from Figure 3 with openings from $A_{\mathsf{op}} = (\mathcal{R}^{\kappa})^{2\tau} \times (\mathcal{R}^{\kappa'})^{\xi}$, where $\kappa = \lceil \log_{2\eta+1} q \rceil$, $\kappa' = \lceil \log_{2\eta+1} q' \rceil$. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda})$ be fixed, defining coefficients $\boldsymbol{g}, \boldsymbol{h}_0, \boldsymbol{h}_1$ for Ajtai's hash functions.*

We call an opening $\boldsymbol{d} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$ linearly verifying for time slot $t, 1 \leq t \leq 2^{\tau}$ iff the following conditions hold

$$\begin{aligned} \boldsymbol{g}^{\mathsf{T}} \cdot \boldsymbol{u} &= \mathsf{proj}_q(\boldsymbol{p}_\tau) \\ \mathsf{proj}_q(\boldsymbol{p}_{j-1}) &= \boldsymbol{h}_{\tilde{t}_j}^{\mathsf{T}} \cdot \boldsymbol{p}_j + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^{\mathsf{T}} \cdot \boldsymbol{s}_j \quad \textit{for all } 2 \leq j \leq \tau \ , \end{aligned}$$

where $\tilde{t} = \mathsf{bin}_\tau(t-1)$ is the binary decomposition of $t-1$. We define $A_{\mathsf{op,lin}}^{t} \subset A_{\mathsf{op}}$ to be the subset of all linearly verifying openings for t. Since $A_{\mathsf{op,lin}}^{t}$ is defined via $\mathcal{R}$-linear constraints, $A_{\mathsf{op,lin}}^{t}$ is an $\mathcal{R}$-submodule of A_{op}.

By construction, any opening that passes either individual, weak or strong verification must be linearly verifying. Whether a given opening $\boldsymbol{d}$ is linearly verifying or not can be checked in polynomial time, given only $\boldsymbol{d}$ and public data pp and t.

We now define an efficient encoding scheme for linearly verifying openings in Figure 5.

```
$\mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \boldsymbol{d})$
parse $\boldsymbol{d}$ as $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$
if $\boldsymbol{d} \notin A_{\mathsf{op,lin}}^{t}$
  return $\perp$
for $j \in \{1, \ldots, \tau\}$
  $\overline{\boldsymbol{p}}_j := \mathsf{Encode}_{\eta,q}^{\mathrm{B}}(\boldsymbol{p}_j)$
$\overline{\boldsymbol{d}} := (\overline{\boldsymbol{p}}_1, \ldots, \overline{\boldsymbol{p}}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$
return $\overline{\boldsymbol{d}}$
```

```
$\mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})$
$\tilde{t} := \mathsf{bin}_\tau(t-1)$
parse $\overline{\boldsymbol{d}}$ as $(\overline{\boldsymbol{p}}_1, \ldots, \overline{\boldsymbol{p}}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$
$\mathrm{hint}_\tau := \boldsymbol{g}^{\mathsf{T}} \cdot \boldsymbol{u} \in \mathcal{R}_q$
for $j \in \{\tau, \ldots, 1\}$   // loop downward
  $\boldsymbol{p}_j := \mathsf{Decode}_{\eta,q}^{\mathrm{B}}(\overline{\boldsymbol{p}}_j, \mathrm{hint}_j)$
  $\mathrm{hint}_{j-1} := \boldsymbol{h}_{\tilde{t}_j}^{\mathsf{T}} \cdot \boldsymbol{p}_j + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^{\mathsf{T}} \cdot \boldsymbol{s}_j \in \mathcal{R}_q$
  // Note: $\mathrm{hint}_0$ is unused
$\boldsymbol{d} := (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$
return $\boldsymbol{d}$
```

Fig. 5: Algorithms for encoding and decoding openings for a given time slot.

Theorem 25 (Efficient encoding of decommitments). *Let $n, q, q', \eta, \xi \in \mathbb{N}$ with n a power of two, q, q' primes. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ be fixed, defining coefficients $\boldsymbol{g}, \boldsymbol{h}_0, \boldsymbol{h}_1$ for Ajtai's hash functions. Fix a time slot t with $1 \leq t \leq 2^\tau$. Define $A^t_{\mathsf{op,lin}} \subset A_{\mathsf{op}}$ as in Definition 24, where*[12] *$A_{\mathsf{op}} = \mathcal{R}^{\ell_{\mathsf{op}}}$. Then $\mathsf{Encode}_{\mathrm{op}}$ and $\mathsf{Decode}_{\mathrm{op}}$, as defined in Figure 5 satisfy the following properties.*

1. *$\mathsf{Encode}_{\mathrm{op}}$ and $\mathsf{Decode}_{\mathrm{op}}$ are deterministic polynomial time algorithms.*
2. *For any $\boldsymbol{d} \in A_{\mathsf{op}}, \overline{\boldsymbol{d}} := \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \boldsymbol{d})$, we have $\overline{\boldsymbol{d}} = \perp$ iff $\boldsymbol{d} \notin A^t_{\mathsf{op,lin}}$.*
3. *For any $\overline{\boldsymbol{d}} \in \mathcal{R}^{\ell_{\mathsf{op}}}, \boldsymbol{d} := \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})$, we have $\boldsymbol{d} \in A^t_{\mathsf{op,lin}}$.*
4. *For fixed pp and time slot t, the functions*
$$\mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, .)\colon\ A^t_{\mathsf{op,lin}} \to \mathcal{R}^{\ell_{\mathsf{op}}},\ \boldsymbol{d} \mapsto \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \boldsymbol{d})$$
$$\mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, .)\colon\ \mathcal{R}^{\ell_{\mathsf{op}}} \to A^t_{\mathsf{op,lin}},\ \overline{\boldsymbol{d}} \mapsto \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})$$
are inverses to each other.
5. *Let $\boldsymbol{d} \in A_{\mathsf{op}}$ and $(\overline{\boldsymbol{p}}_1, \ldots, \overline{\boldsymbol{p}}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u}) := \overline{\boldsymbol{d}} := \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \boldsymbol{d})$. Let $\boldsymbol{c} \in A_{\mathsf{com}}$ be any (possibly maliciously generated) commitment and let Vrfy be as in Figure 3. If $\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \beta) \neq \perp$ for some β, then*
$$\|\overline{\boldsymbol{p}}_i\| < \frac{\beta}{2\eta} + \frac{1}{2} \quad \textit{for all } i \ .$$

We remark that for individually verifying openings, we have $\beta = \eta$ above, so the bound reads $\|\overline{\boldsymbol{p}}_i\| < 1$ for this case, meaning that $\overline{\boldsymbol{p}}_i = 0$. This just captures the fact that in this case, we can use the usual Merkle tree trick of not transmitting the $\boldsymbol{p}_i$, but letting the verifier compute them.

Proof. For item 1 and for item 2, there is nothing to show, really.

Let us show item 3 now: by Lemma 23, item 3, the $\boldsymbol{p}_i$ constructed by $\mathsf{Decode}_{\mathrm{op}}$ satisfy $\mathsf{proj}_q(\boldsymbol{p}_j) = \mathrm{hint}_j$ for all $1 \leq j \leq \tau$. With the way $\mathsf{Decode}_{\mathrm{op}}$ chooses hint_j, the definition of what it means for an opening to be linearly verifying precisely reads $\mathsf{proj}_q(\boldsymbol{p}_j) = \mathrm{hint}_j$. So $\boldsymbol{d}$ output by $\mathsf{Decode}_{\mathrm{op}}$ is linearly verifying.

For item 4, let us first show that $\mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \boldsymbol{d})) = \boldsymbol{d}$ for all $\boldsymbol{d} \in A^t_{\mathsf{op,lin}}$. To disambiguate, we temporarily denote the equally named values $\boldsymbol{p}_j$ and $\boldsymbol{d}$ appearing in both $\mathsf{Encode}_{\mathrm{op}}$ and $\mathsf{Decode}_{\mathrm{op}}$ by $\boldsymbol{p}_j^{\mathrm{enc}}$ and $\boldsymbol{d}^{\mathrm{enc}}$ resp. $\boldsymbol{p}_j^{\mathrm{dec}}$ and $\boldsymbol{d}^{\mathrm{dec}}$. Of course, these are equal, but that's precisely what we need to show here. For this, we show that

- hint_j constructed by $\mathsf{Decode}_{\mathrm{op}}$ satisfies $\mathrm{hint}_j = \mathsf{proj}_q(\boldsymbol{p}_j^{\mathrm{enc}})$ and
- $\boldsymbol{p}_j^{\mathrm{enc}} = \boldsymbol{p}_j^{\mathrm{dec}}$

for each $\tau \geq j \geq 1$ by induction over j (starting at $j = \tau$ and going down):

For $j = \tau$, since $\boldsymbol{d}^{\mathrm{enc}}$ is linearly verifying, $\mathsf{proj}_q(\boldsymbol{p}_\tau^{\mathrm{enc}}) = \boldsymbol{g}^\intercal \cdot \boldsymbol{u}$. From this we get $\mathsf{proj}_q(\boldsymbol{p}_\tau^{\mathrm{enc}}) = \mathrm{hint}_\tau$. Using Lemma 23, item 2, this gives $\boldsymbol{p}_\tau^{\mathrm{dec}} = \boldsymbol{p}_\tau^{\mathrm{enc}}$. For $\tau > j \geq 1$, we have

$$\begin{aligned} \mathsf{proj}_q(\boldsymbol{p}_j^{\mathrm{enc}}) &= \boldsymbol{h}_{t_{j+1}}^\intercal \cdot \boldsymbol{p}_{j+1}^{\mathrm{enc}} + \boldsymbol{h}_{t_{j+1}\oplus 1}^\intercal \cdot \boldsymbol{s}_{j+1} && (\boldsymbol{d}^{\mathrm{enc}} \text{ linearly verifying}) \\ &= \boldsymbol{h}_{t_{j+1}}^\intercal \cdot \boldsymbol{p}_{j+1}^{\mathrm{dec}} + \boldsymbol{h}_{t_{j+1}\oplus 1}^\intercal \cdot \boldsymbol{s}_{j+1} && (\text{induction hypothesis}) \end{aligned}$$

[12] We write the domain of $\mathsf{Encode}_{\mathrm{op}}$ as A_{op} resp. $A^t_{\mathsf{op,lin}}$ and the range as $\mathcal{R}^{\ell_{\mathsf{op}}}$. Even though those are the same as sets, we only view the domain as an $\mathcal{R}$-module in the usual way.

$$= \mathbf{hint}_j \ . \qquad \text{(Definition of hint}_j \text{ in } \mathsf{Decode}_{\mathrm{op}})$$

Again, using Lemma 23, item 2, we can conclude $\boldsymbol{p}_j^{\mathrm{dec}} = \boldsymbol{p}_j^{\mathrm{enc}}$, finishing the induction. This gives $\boldsymbol{d}^{\mathrm{dec}} = \boldsymbol{d}^{\mathrm{enc}}$, showing $\mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \boldsymbol{d})) = \boldsymbol{d}$.

Now consider the other direction, i.e. that $\mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})) = \overline{\boldsymbol{d}}$ for all $\overline{\boldsymbol{d}} \in \mathcal{R}^{\ell_{\mathsf{op}}}$. By Lemma 23, item 3, we get $\mathsf{proj}_q(\boldsymbol{p}_j) = \mathrm{hint}_j$ (so $\boldsymbol{d}$ is linearly verifying and $\mathsf{Encode}_{\mathrm{op}}$ does not abort) and that the values of $\overline{\boldsymbol{p}}_j$ constructed by $\mathsf{Encode}_{\mathrm{op}}$ are the same as those in $\mathsf{Decode}_{\mathrm{op}}$, which shows the claim.

For item 5, write $\boldsymbol{d} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$. Recall that $\mathsf{Vrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \beta) \neq \perp$ checks among other things that

$$\left\|\boldsymbol{p}_j\right\| \leq \beta \quad \text{and} \quad \left\|\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j)\right\| \leq \frac{q\beta}{2\eta} \quad \text{for all } 1 \leq i \leq \tau \ .$$

Plugging this into item 4 of Lemma 23 directly gives $\left\|\overline{\boldsymbol{p}}_j\right\| < \frac{\beta}{2\eta} + \frac{1}{2}$ for all j. □

We can use $\mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, .)$ and $\mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, .)$ to store and transmit openings: By item 5 of Theorem 25, the encoded openings have smaller norm than the unencoded versions, which can be used to save space. The restriction that $\mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, .)$ only works for linearly verifying openings is immaterial, as openings violating this condition will never be valid anyway.

Formally, we can define an encoded version of Chipmunk's HVC as follows:

Definition 26. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers such that n is a power of two, q, q' are prime. Let $\mathsf{HVC}_0^{\mathrm{Chip}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{iVrfy}, \mathsf{sVrfy}, \mathsf{wVrfy})$ be the HVC from Figure 3 for its domain A_{dom} and vectors of length 2^τ. Denote by A_{op} and A_{com} the $\mathcal{R}$-modules where the openings and commitments are from. Recall that A_{op} has the form $A_{\mathsf{op}} = \mathcal{R}^{\ell_{\mathsf{op}}}$.*

We can then define an encoded version $\mathsf{HVC}_{\mathrm{Encoded}}^{\mathrm{Chip}} = (\mathsf{Setup}', \mathsf{Com}', \mathsf{Open}', \mathsf{iVrfy}', \mathsf{sVrfy}', \mathsf{wVrfy}')$ by simply encoding/decoding the openings as follows:

$\mathsf{Setup}'(1^\lambda)$**:** *Identical to* Setup.
$\mathsf{Com}'(\mathsf{pp}, \boldsymbol{m})$**:** *Identical to* Com.
$\mathsf{Open}'(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$**:** *Run $\boldsymbol{d} \leftarrow \mathsf{Open}(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$ and output $\overline{\boldsymbol{d}} = \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \boldsymbol{d})$.*
$\mathsf{iVrfy}'(\mathsf{pp}, \boldsymbol{c}, t, \overline{\boldsymbol{d}})$**:** *Run $\boldsymbol{d} \leftarrow \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})$. Output whatever $\mathsf{iVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$ outputs.*
$\mathsf{sVrfy}'(\mathsf{pp}, \boldsymbol{c}, t, \overline{\boldsymbol{d}})$**:** *Run $\boldsymbol{d} \leftarrow \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})$. Output whatever $\mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$ outputs.*
$\mathsf{wVrfy}'(\mathsf{pp}, \boldsymbol{c}, t, \overline{\boldsymbol{d}})$**:** *Run $\boldsymbol{d} \leftarrow \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})$. Output whatever $\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})$ outputs.*

Note that the opening space of $\mathsf{HVC}_{\mathrm{Encoded}}^{\mathrm{Chip}}$ is $\mathcal{R}^{\ell_{\mathsf{op}}}$. However, to perform homomorphic operations on openings, we need to operate on the *unencoded* values, since encoding/decoding is not a $\mathcal{R}$-linear operation with the usual $\mathcal{R}$-module structure on $\mathcal{R}^{\ell_{\mathsf{op}}}$. To formally satisfy the homomorphism requirements, we therefore need to endow the set $\mathcal{R}^{\ell_{\mathsf{op}}}$ with a (non-standard) $\mathcal{R}$-module structure $A_{\mathsf{op}}^t = (\mathcal{R}^{\ell_{\mathsf{op}}}, \odot, \oplus)$, where scalar multiplication $\odot$ by ring elements and the addition $\oplus$ are given by

$$\begin{aligned} \odot\colon\ \mathcal{R} \times A_{\mathsf{op}}^t \to A_{\mathsf{op}}^t, &\quad w \odot \overline{\boldsymbol{d}} := \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, w \cdot \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}})) \\ \oplus\colon\ A_{\mathsf{op}}^t \times A_{\mathsf{op}}^t, &\quad \overline{\boldsymbol{d}}_1 \oplus \overline{\boldsymbol{d}}_2 := \mathsf{Encode}_{\mathrm{op}}(\mathsf{pp}, t, \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}_1) + \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}_2)) \end{aligned}$$

This gives an $\mathcal{R}$-module A_{op}^t, which is the opening space of $\mathsf{HVC}_{\mathrm{Encoded}}^{\mathrm{Chip}}$. Note that it depends on t.

Theorem 27. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers and $0 < \varepsilon \leq 1$ such that n is a power of two and q, q' prime. Let $\mathsf{HVC}_0^{\text{Chip}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{iVrfy}, \mathsf{sVrfy}, \mathsf{wVrfy})$ be the homomorphic vector commitment from Figure 3 and $\mathsf{HVC}_{\text{Encoded}}^{\text{Chip}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}', \mathsf{iVrfy}', \mathsf{sVrfy}', \mathsf{wVrfy}')$ be the encoded version from Definition 26 based on it for those parameters. Then $\mathsf{HVC}_{\text{Encoded}}^{\text{Chip}}$ is individually correct, $(\rho, \mathcal{T}_{\alpha_w}, \varepsilon)$-probabilistically homomorphic, robustly homomorphic and position binding for domain $\mathcal{R}_{q'}^{\xi}$ and vector length 2^τ, provided $\mathsf{HVC}_0^{\text{Chip}}$ has those properties.*

Proof. There is really not much to show here.

Individual correctness follows directly from item 4 of Theorem 25.

The robust homomorphism properties also follows from this, together with the way we defined A_{op}^t. Notably, assume we have $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $1 \leq t \leq 2^\tau$, commitments $\boldsymbol{c}^0, \boldsymbol{c}^1 \in A_{\mathsf{dom}}$ and $\overline{\boldsymbol{d}}^0, \overline{\boldsymbol{d}}^1 \in A_{\mathsf{op}}^t$ with

$$\mathsf{sVrfy}'(\mathsf{pp}, \boldsymbol{c}^0, t, \overline{\boldsymbol{d}}^0) = \boldsymbol{m}^0 \quad \text{and} \quad \mathsf{sVrfy}'(\mathsf{pp}, \boldsymbol{c}^1, t, \overline{\boldsymbol{d}}^1) = \boldsymbol{m}^1$$

such that $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \perp$. We need to show that

$$\mathsf{wVrfy}'(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \overline{\boldsymbol{d}}^0 \ominus \overline{\boldsymbol{d}}^1) = \boldsymbol{m}^0 - \boldsymbol{m}^1 \ ,$$

where $\ominus$ is the subtraction in A_{op}^t corresponding to $\oplus$.

Let $\boldsymbol{d}^0 := \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}^0), \boldsymbol{d}^1 := \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}^1)$. By definition of sVrfy', we have

$$\boldsymbol{m}^0 = \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) \quad \text{and} \quad \boldsymbol{m}^1 = \mathsf{sVrfy}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) \ .$$

Since HVC is robustly homomorphic, this yields

$$\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) = \boldsymbol{m}^0 - \boldsymbol{m}^1 \ .$$

By definition, $\mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}^0 \ominus \overline{\boldsymbol{d}}^1) = \boldsymbol{d}^0 - \boldsymbol{d}^1$. Putting these together, we obtain

$$\begin{aligned} &\mathsf{wVrfy}'(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \overline{\boldsymbol{d}}^0 \ominus \overline{\boldsymbol{d}}^1) \\ =&\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}^0 \ominus \overline{\boldsymbol{d}}^1)) \\ =&\mathsf{wVrfy}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) \\ x =&\boldsymbol{m}^0 - \boldsymbol{m}^1 \ . \end{aligned}$$

For the probabilistic homomorphism property, let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $\ell < \rho$ and $1 \leq t \leq 2^\tau$.

For $1 \leq i \leq \ell$, consider commitments $\boldsymbol{c}^i \in A_{\mathsf{com}}$, decommitments $\overline{\boldsymbol{d}}^i \in A_{\mathsf{op}}^t$ with $\mathsf{iVrfy}'(\mathsf{pp}, \boldsymbol{c}^i, t, \overline{\boldsymbol{d}}^i) = \boldsymbol{m}^i$ such that $\boldsymbol{m}^i \neq \perp$. We need to show that

$$\Pr\left[w^1, \ldots, w^\ell \leftarrow W\colon\ \mathsf{sVrfy}'\Big(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{c}^i, t, \bigoplus_{i=1}^{\ell} w^i \odot \overline{\boldsymbol{d}}^i\Big) = \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{m}_t^i\right] \geq 1 - \varepsilon \ .$$

For this, set $\boldsymbol{d}^i := \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}^i)$. By definition of iVrfy', we have $\mathsf{iVrfy}(\mathsf{pp}, \boldsymbol{c}^i, t, \boldsymbol{d}^i) = \boldsymbol{m}^i$. Then we get

$$\mathsf{sVrfy}'\Big(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{c}^i, t, \bigoplus_{i=1}^{\ell} w^i \odot \overline{\boldsymbol{d}}^i\Big)$$

$$=\mathsf{sVrfy}\Big(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{c}^i, t, \mathsf{Decode}_{\mathrm{op}}\Big(\mathsf{pp}, t, \bigoplus_{i=1}^{\ell} w^i \odot \overline{\boldsymbol{d}}^i\Big)\Big)$$
$$=\mathsf{sVrfy}\Big(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{c}^i, t, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{d}^i\Big)$$

The claim then follows from the probabilistic homomorphism property of HVC.

Let us look at the position-binding property. For any adversary $\mathcal{A}$ against the position-binding property of $\mathsf{HVC}^{\mathrm{Chip}}_{\mathrm{Encoded}}$, we construct an adversary $\mathcal{B}$ against HVC as follows.

$\mathcal{B}(\mathsf{pp})$ runs $(\boldsymbol{c}, t, \overline{\boldsymbol{d}}_0, \overline{\boldsymbol{d}}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ and outputs $(\boldsymbol{c}, t, \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}_0), \mathsf{Decode}_{\mathrm{op}}(\mathsf{pp}, t, \overline{\boldsymbol{d}}_1))$. It is easy to see that $\mathcal{A}$ is successful iff $\mathcal{B}$ is. □

Remark 4. Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers such that n is a power of two and q, q' prime. Let us collect in Table 2 the individual components of our HVC constructions and look at the bit-sizes of commitments and (individually or strongly verifying) openings as functions of the parameters. Note that the strongly verifying case will correspond to the contribution for the size of aggregated signatures later in Section 6, and this size is the most important metric we want to minimize.

A commitment is a (non-short) single element from $\mathcal{R}_q$. This means we can use $n\lceil\log q\rceil$ bits to store it.[13] In the variant without our elaborate encodings, an opening consists of $\ell_{\mathsf{op}} = 2\tau\lceil\log_{2\eta+1} q\rceil + \xi\lceil\log_{2\eta+1} q'\rceil$ many elements from $\mathcal{R}$. Each element is $\|.\|_\infty$-bounded: for individually verifying openings, the bound is η, giving $n\ell_{\mathsf{op}}\lceil\log(2\eta+1)\rceil$ bits. For strongly verifying opening, the bound is β_{agg}, giving $n\ell_{\mathsf{op}}\lceil\log(2\beta_{\mathsf{agg}}+1)\rceil$ bits.

For $\mathsf{HVC}^{\mathrm{Chip}}_{\mathrm{Encoded}}$, an opening consists of the same number of elements from $\mathcal{R}$, but we have tighter size constraints for $\tau\lceil\log_{2\eta+1} q\rceil$ of them. For the individually verifying case, those elements are actually 0. In the strongly verifying case, the (non-attained) bound is $\frac{\beta_{\mathsf{agg}}}{2\eta} + \frac{1}{2}$, giving $n\tau\lceil\log_{2\eta+1} q\rceil \left\lceil\log(2\lfloor\frac{\beta_{\mathsf{agg}}}{2\eta} + \frac{1}{2}\rfloor + 1)\right\rceil \leq n\tau\lceil\log_{2\eta+1} q\rceil \left\lceil\log(\lfloor\frac{\beta_{\mathsf{agg}}}{\eta}\rfloor + 2)\right\rceil$ many bits for the $\overline{\boldsymbol{p}}_i$'s.

			size in bits
commitments			$n\lceil\log q\rceil$
opening	$\mathsf{HVC}^{\mathrm{Chip}}_{0}$	individually verifying	$(2\tau\kappa + \xi\kappa')n \cdot \lceil\log(2\eta+1)\rceil$
		strongly verifying	$(2\tau\kappa + \xi\kappa')n \cdot \lceil\log(2\beta_{\mathsf{agg}}+1)\rceil$
	$\mathsf{HVC}^{\mathrm{Chip}}_{\mathrm{Encoded}}$	individually verifying	$(\tau\kappa + \xi\kappa')n \cdot \lceil\log(2\eta+1)\rceil$
		strongly verifying	$(\tau\kappa + \xi\kappa')n \cdot \lceil\log(2\beta_{\mathsf{agg}}+1)\rceil + \tau\kappa n\lceil\log(\lfloor\frac{\beta_{\mathsf{agg}}}{\eta}\rfloor + 2)\rceil$

Table 2: bitlength of our HVC constructions. We denote by $\kappa = \lceil\log_{2\eta+1} q\rceil$ and $\kappa' = \lceil\log_{2\eta+1} q'\rceil$ the number of limbs for the decompositions of $\mathcal{R}_q$ resp. $\mathcal{R}_{q'}$ elements.

[13] In principle, we could do $\lceil n\log q\rceil$ by using some clever arithmetic encoding; however, for simplicity, we assume here that every coefficient is stored individually.

5 Key-Homomorphic One-Time Signatures

In this section, we define and instantiate key-homomorphic one-time signatures, which are a weak form of a digital signature scheme that is only guaranteed to be unforgeable, if at most one signature is published under any given public key. A one-time signature is called key-homomorphic, if the linear combination of separate signatures for the same message verifies under the linear combination of the corresponding public keys.

Our definitions again follow the definitions of [FSZ22a] closely, but are incomparable just as in Section 3. As with the vector commitments, we have the stronger requirement that the homomorphism works for any individually verifying signature, not just honestly created ones. But, this homomorphism is allowed to have a noticeable correctness error.

The construction presented in this section is a modification of the construction of [FSZ22a], which itself was a modification of the one-time signature schemes by Boneh and Kim [BK20] and Lyubashevsky and Micciancio [LM08].

Definition 28 (Key-Homomorphic One-Time Signature). *Let $\mathcal{R}$ be a ring. Let A_{opk} and A_{sig} be $\mathcal{R}$-modules denoting the spaces where the public keys and signatures are from. A key-homomorphic one-time signature scheme (KOTS) over $\mathcal{R}$ with public key space A_{opk} and signatures from A_{sig} is defined by six PPT algorithms* $\mathsf{KOTS} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{iVrfy}, \mathsf{sVrfy}, \mathsf{wVrfy})$.

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

$(\mathsf{osk}, \mathsf{opk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$ *The key generation algorithm takes as input the public parameters and outputs a key pair with* $\mathsf{opk} \in A_{\mathsf{opk}}$.

$\boldsymbol{\sigma} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{osk}, m)$ *The signing algorithm takes as input the public parameters, a one-time signing key, and a message and outputs a signature* $\sigma \in A_{\mathsf{sig}}$.

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

$b \leftarrow \mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$ *The strong verification algorithm has the same input and output domains as the individual verification algorithm.*

$b \leftarrow \mathsf{wVrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$ *The weak verification algorithm has the same input and output domains as the individual verification algorithm.*

Note that for us, we will always have $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ for n a power of two and $A_{\mathsf{sig}} = \mathcal{R}_{q'}^{\ell_{\mathsf{sig}}}$, $A_{\mathsf{opk}} = \mathcal{R}_{q'}^{\ell_{\mathsf{opk}}}$ for some prime q'.

Definition 29 (Individual Correctness). *Let* KOTS *be a key-homorphic one-time signature scheme.* KOTS *is individually correct, if for all security parameters* $\lambda \in \mathbb{N}$, *parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, *key pairs* $(\mathsf{osk}, \mathsf{opk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$, *messages* $m \in \{0,1\}^*$, *and signatures* $\boldsymbol{\sigma} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{osk}, m)$ *it holds that*

$$\mathsf{iVrfy}(\mathsf{pp}, \mathsf{opk}, m, \sigma) = 1 \ .$$

We require that individually verifying signatures can be homomorphically aggregated by computing a random linear combination of them. Such aggregated signatures should still *strongly* verify with high probability over the choice of the random linear combination.

Definition 30 (Probabilistic Homomorphism). *Let* KOTS *be a one-time signature scheme over a ring* $\mathcal{R}$ *with public key space* A_{opk} *and signatures from* A_{sig}*. Let* $\rho \in \mathbb{N}$*, error bound* $0 \leq \varepsilon \leq 1$ *and* $W \subseteq \mathcal{R}$*.* KOTS *is* (ρ, W, ε)*-probabilistically homomorphic, if for all security parameters* $\lambda \in \mathbb{N}$*, number of aggregated signatures* $\ell \in [\rho]$*, parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, public keys* $\mathsf{opk}^i \in A_{\mathsf{opk}}$*, messages* $m \in \{0,1\}^*$ *and signatures* $\boldsymbol{\sigma}^i \in A_{\mathsf{sig}}$ *with* $\mathsf{iVrfy}(\mathsf{pp}, \mathsf{opk}^i, m, \boldsymbol{\sigma}^i)$ *it holds that*

$$\Pr\left[w^1, \dots, w^\ell \leftarrow W \colon \mathsf{sVrfy}(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \mathsf{opk}^i, m, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{\sigma}^i) = 1\right] \geq 1 - \varepsilon \ .$$

As with the vector commitments from the previous section, we additionally require that a further limited homomorphism still holds, even for maliciously *aggregated* signatures. For any two, even maliciously generated, signatures that *strongly* verify under potentially maliciously generated public keys, their difference will still *weakly* verify.

Definition 31 (Robust Homomorphism). *Let* KOTS *be a key-homomorphic one-time signature scheme over a ring* $\mathcal{R}$ *with public key space* A_{opk} *and signatures from* A_{sig}*.* KOTS *is robustly homomorphic if for all security parameters* $\lambda \in \mathbb{N}$*, public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, messages* $m \in \{0,1\}^*$*, (possibly malformed) public keys* $\mathsf{opk}^0, \mathsf{opk}^1 \in A_{\mathsf{opk}}$*, and (possibly malformed) signatures* $\boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1 \in A_{\mathsf{sig}}$ *with*

$$\mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}^0, m, \boldsymbol{\sigma}^0) = 1 \quad \textit{and} \quad \mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}^1, m, \boldsymbol{\sigma}^1) = 1$$

it holds that

$$\mathsf{wVrfy}(\mathsf{pp}, \mathsf{opk}^0 - \mathsf{opk}^1, m, (\boldsymbol{\sigma}^0 - \boldsymbol{\sigma}^1)) = 1.$$

The following definition of a multi-user version of (one-time) existential unforgeability under rerandomized keys is taken directly from [FSZ22a].

Definition 32 (Multi-User Existential Unforgeability under Rerandomized Keys). *A* (ρ, W, ε)*-homomorphically correct KOTS is* W'*-existentially unforgeable under rerandomized keys (EUF-RK), if for all security parameters* λ*, any* $T = \mathsf{poly}(\lambda) \in \mathbb{N}$ *and all stateful PPT algorithms* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ \forall i \in [T].\, (\mathsf{osk}_i, \mathsf{opk}_i) \leftarrow \mathsf{KGen}(\mathsf{pp}); \\ (i^*, m^*, \sigma^*, w^*) \leftarrow \mathcal{A}^{\widetilde{\mathsf{Sign}}(\cdot,\cdot)}(\mathsf{pp}, \mathsf{opk}_1, \dots, \mathsf{opk}_T) \end{array} : \begin{array}{r} \mathsf{wVrfy}(\mathsf{pp}, w^* \cdot \mathsf{opk}_{i^*}, m^*, \sigma^*) = 1 \\ \wedge\, m^* \notin Q_{i^*} \wedge |Q_{i^*}| \leq 1 \wedge w^* \in W' \end{array}\right] \leq \mathsf{negl}(\lambda) \ ,$$

where the oracle $\widetilde{\mathsf{Sign}}(\cdot,\cdot)$ *is defined as* $\widetilde{\mathsf{Sign}}(i, m) := \mathsf{Sign}(\mathsf{osk}_i, m)$ *and* Q_i *denotes the set of messages for which a signing query with index* i *has been made.*

Figure 6 shows the construction of the KOTS we will use. The construction is almost identical to the construction from [FSZ22a] but differs from it in its choice of the ball from which the secret keys are chosen. Specifically, the components of the secret keys are allowed to have a larger infinity norm. This is beneficial, because the security proof partially relies on fact that the function mapping secret keys to public keys and signatures is highly compressing. With a larger secret key-space the compression ratio increases, allowing us to reduce the size of other parameters, ultimately decreasing the size of the signatures.

```
$\mathsf{Setup}(1^\lambda)$
    $\boldsymbol{a} \leftarrow \mathcal{R}_{q'}^{\gamma}$
    return $\boldsymbol{a}$

$\mathsf{KGen}(\mathsf{pp})$
    $\boldsymbol{s}_0 \leftarrow \mathcal{B}_{\varphi,q'}^{\gamma}$
    $\boldsymbol{s}_1 \leftarrow \mathcal{B}_{\varphi\cdot\alpha_H,q'}^{\gamma}$
    $v_0 := \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0$
    $v_1 := \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1$
    return $((\boldsymbol{s}_0, \boldsymbol{s}_1), (v_0, v_1))$

$\mathsf{Sign}(\mathsf{pp}, \mathsf{osk}, m)$
    parse $\mathsf{osk}$ as $(\boldsymbol{s}_0, \boldsymbol{s}_1)$
    $\boldsymbol{\sigma} := \boldsymbol{s}_0 \cdot H(m) + \boldsymbol{s}_1$
    return $\boldsymbol{\sigma}$

$\mathsf{iVrfy}(\mathsf{pp}, \mathsf{opk}, m, \boldsymbol{\sigma})$
    return $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{opk}, m, \boldsymbol{\sigma}, 2\varphi\alpha_H)$

$\mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}, m, \boldsymbol{\sigma})$
    return $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{opk}, m, \boldsymbol{\sigma}, \beta_\sigma)$

$\mathsf{wVrfy}(\mathsf{pp}, \mathsf{opk}, m, \boldsymbol{\sigma})$
    return $\mathsf{Vrfy}(\mathsf{pp}, \mathsf{opk}, m, \boldsymbol{\sigma}, 2\beta_\sigma)$

$\mathsf{Vrfy}(\mathsf{pp}, \mathsf{opk}, m, \boldsymbol{\sigma}, \beta')$
    parse $\mathsf{opk}$ as $(v_0, v_1)$
    if $\|\boldsymbol{\sigma}\| > \beta'$
        return $0$
    if $\boldsymbol{a}^\intercal \cdot \boldsymbol{\sigma} \neq v_0 \cdot H(m) + v_1$
        return $0$
    return $1$
```

Fig. 6: Description of our key-homomorphic one-time signature scheme $\mathsf{KOTS}^{\mathrm{Chip}}$ from Definition 33. H is a collision-resistant hash function mapping bit-strings to $\mathcal{T}_{\alpha_H}$. Our key space is $A_{\mathsf{opk}} = \mathcal{R}_{q'}^2$. The signature space is $A_{\mathsf{sig}} = \mathcal{R}_{q'}^{\gamma}$.

Definition 33. *Let $n, q', \alpha_H, \varphi, \gamma, \beta_\sigma$ be integers and H be a hash function mapping bit strings to $\mathcal{T}_{\alpha_H}$. Let $\mathcal{R} = \mathbb{Z}[X]/\langle X^n+1\rangle$ and $\mathcal{R}_{q'} = \mathbb{Z}_{q'}[X]/\langle X^n+1\rangle$. We define $\mathsf{KOTS}^{\mathrm{Chip}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{iVrfy}, \mathsf{sVrfy}, \mathsf{wVrfy})$ as the key-homomorphic one-time scheme over $\mathcal{R}$ as in Figure 6. Its public key space is $A_{\mathsf{opk}} = \mathcal{R}_{q'}^2$ and its signature space is $A_{\mathsf{sig}} = \mathcal{R}_{q'}^{\gamma}$.*

Theorem 34. *Let $\lambda, \alpha_w, \alpha_H, \varphi, \gamma, \rho, \beta_\sigma, n, q'$ be integers and $0 < \varepsilon < 1$ such that q' is prime and $q' > 16\alpha_w\alpha_H\varphi$, n is a power of two and there exists δ with*

$$\begin{aligned} 2^{2\lambda} &\leq |\mathcal{T}_{\alpha_H}| \leq 2^{2\lambda+\delta}, \\ \beta_\sigma &\geq 4\varphi\alpha_H\sqrt{\tfrac{1}{2}\alpha_w\rho \cdot \ln\tfrac{2n\gamma}{\varepsilon}} \\ \gamma &\geq ((3\lambda+\delta)/n + \log_2 q')\log_2^{-1}(\varphi + \tfrac{1}{2}) \ . \end{aligned}$$

Let $H\colon \{0,1\}^* \to \mathcal{T}_{\alpha_H}$ be a hash function. Let $W' = \{w_0 - w_1 \mid w_0, w_1 \in \mathcal{T}_{\alpha_w} \wedge w_0 \neq w_1\}$. If the $\mathsf{SIS}_{\mathcal{R},q',\gamma,2\beta_\sigma+4\alpha_w\alpha_H\varphi}$ problem is hard and H is collision resistant, then the construction $\mathsf{KOTS}^{\mathrm{Chip}}$ from Figure 6 is an individually correct, $(\rho, \mathcal{T}_{\alpha_w}, \varepsilon)$-probabilistically homomorphic, robustly homomorphic KOTS that is W'-multi-user existentially unforgeable under rerandomized keys.

Proof. The theorem follows from Lemma 35, Lemma 36, Lemma 37, and Lemma 38. □

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

Lemma 35. *Let $\lambda, \alpha_w, \alpha_H, \varphi, \gamma, \rho, \beta_\sigma, n, q'$ be positive integers, such that n is a power of two, q' is prime. Let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n+1\rangle$. Let $H\colon \{0,1\}^* \to \mathcal{T}_{\alpha_H}$ be a hash function. Then $\mathsf{KOTS}^{\mathrm{Chip}}$ as in Figure 6 is a individually correct one time signature scheme.*

Proof. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{osk}, \mathsf{opk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$, $m \in \{0,1\}^*$ and $\boldsymbol{\sigma} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{osk}, m)$ be arbitrary. We first observe that the check on the *value* of the signature goes through, as

$$\begin{aligned} \boldsymbol{a}^\intercal \boldsymbol{\sigma} &= \boldsymbol{a}^\intercal (\boldsymbol{s}_0 \cdot H(m) + \boldsymbol{s}_1) && \text{(Def. of } \mathsf{Sign}) \\ &= \boldsymbol{a}^\intercal \boldsymbol{s}_0 \cdot H(m) + \boldsymbol{a}^\intercal \boldsymbol{s}_1 && \text{(Distributivity)} \\ &= v_0 \cdot H(m) + v_1. && \text{(Def. of } \mathsf{KGen}) \end{aligned}$$

The signature also does not violate the norm bound, as

$$\begin{aligned} \|\boldsymbol{\sigma}\| &= \|\boldsymbol{s}_0 \cdot H(m) + \boldsymbol{s}_1\| && \text{(Def. of } \mathsf{Sign}) \\ &\leq \|\boldsymbol{s}_0 \cdot H(m)\| + \|\boldsymbol{s}_1\| \\ &\leq \|\boldsymbol{s}_0\| \cdot \|H(m)\|_1 + \|\boldsymbol{s}_1\| && \text{(Lemma 1)} \\ &= 2\varphi\alpha_H. && \text{(Def. of } \mathsf{KGen}) \end{aligned}$$

The lemma thus follows. □

Lemma 36. *Let λ, α_w, α_H, φ, γ, ρ, β_σ, n, q' be positive integers and $0 < \varepsilon < 1$, such that*

$$\beta_\sigma \geq 4\varphi\alpha_H \sqrt{\tfrac{1}{2}\alpha_w \rho \cdot \ln \tfrac{2n\gamma}{\varepsilon}} \ .$$

Let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$. Let $H \colon \{0,1\}^* \to \mathcal{T}_{\alpha_H}$ be a hash function. Then $\mathsf{KOTS}^{\mathrm{Chip}}$ as in Figure 6 is a $(\rho, \mathcal{T}_{\alpha_w}, \varepsilon)$-probabilistically homomorphic one time signature scheme.

Proof. Let $\ell \in [\rho]$, $m \in \{0,1\}^*$, and $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and for $i \in [\ell]$ let $\mathsf{opk}^i = (v_0, v_1) \in \mathcal{R}_{q'}^2$ and $\boldsymbol{\sigma}^i \in \mathcal{R}_{q'}^\gamma$ be arbitrary such that for all $i \in [\ell]$, $\mathsf{iVrfy}(\mathsf{pp}, \mathsf{opk}^i, m, \boldsymbol{\sigma}^i) = 1$.

We first note that even for arbitrary $w_1, \ldots, w_\ell \in \mathcal{T}_\alpha$ it holds that

$$\begin{aligned} \boldsymbol{a}^\intercal \cdot \sum_{i=1}^{\ell-1} w^i \cdot \boldsymbol{\sigma}^i &= \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{a}^\intercal \boldsymbol{\sigma}^i && \text{(Distributivity)} \\ &= \sum_{i=1}^{\ell} w^i \cdot (v_0^i \cdot H(m) + v_1^i) && \text{(Def. of } \mathsf{iVrfy}) \\ &= \Big(\sum_{i=1}^{\ell} w^i v_0^i\Big) \cdot H(m) + \Big(\sum_{i=1}^{\ell} w^i v_1^i\Big). && \text{(Distributivity)} \end{aligned}$$

Therefore, it only remains to verify that the norm-check goes through with sufficient probability. That means we need to show that

$$P \coloneqq \Pr\left[w^1, \ldots, w^\ell \leftarrow \mathcal{T}_{\alpha_w} : \Big\| \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{\sigma}^i \Big\| > \beta_\sigma \right] \leq \varepsilon \ .$$

For each individual $\boldsymbol{\sigma}^i$, it holds by the definition of iVrfy that $\|\boldsymbol{\sigma}^i\| \leq 2\varphi\alpha_H$. What we need show here is a norm bound in $\mathcal{R}^\gamma$, i.e. the bound holds even if we do not reduce modulo q'. Using Lemma 4 with $\zeta = \frac{\beta_\sigma}{2\varphi\alpha_H}$ and taking a union bound over all γ entries immediately gives

$$P \leq \gamma \cdot 2n \exp\Big(-\frac{\beta_\sigma^2}{8\varphi^2\alpha_H^2\alpha_w \ell}\Big) \leq 2\gamma n \exp\Big(-\frac{\beta_\sigma^2}{8\varphi^2\alpha_H^2\alpha_w \rho}\Big) \ . \qquad (8)$$

Our condition $\beta_\sigma \geq 4\varphi\alpha_H\sqrt{\frac{1}{2}\alpha_w\rho\cdot\ln\frac{2n\gamma}{\varepsilon}}$ is chosen as to be equivalent to

$$\frac{\beta_\sigma^2}{8\varphi^2\alpha_H^2\alpha_w\rho} \geq \ln\Big(\frac{2\gamma n}{\varepsilon}\Big) \ .$$

Plugging this into Equation 8 directly gives $P \leq \varepsilon$. It thus follows that with probability at least $1-\varepsilon$ the strong verification algorithm outputs 1 as required.

Lemma 37. *Let λ, α_H, φ, γ, β_σ, q', n be positive integers. As usual, let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n+1\rangle$. Let $H\colon \{0,1\}^* \to \mathcal{T}_{\alpha_H}$ be a hash function. Then $\mathsf{KOTS}^{\mathrm{Chip}}$ as in Figure 6 is robustly homomorphic.*

Proof. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $m \in \{0,1\}^*$, $\mathsf{opk}^0 = (v_0^0, v_1^0), \mathsf{opk}^1 = (v_0^1, v_1^1) \in \mathcal{R}_q^2$, and $\boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1 \in \mathcal{R}_{q'}^\gamma$ be arbitrary such that $\mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}^0, m, \boldsymbol{\sigma}^0) = 1$ and $\mathsf{sVrfy}(\mathsf{pp}, \mathsf{opk}^1, m, \boldsymbol{\sigma}^1) = 1$.

By the definition of the strong verification algorithm, it holds that

$$\left\|(\boldsymbol{\sigma}^0 - \boldsymbol{\sigma}^1)\right\| \leq \left\|\boldsymbol{\sigma}^0\right\| + \left\|\boldsymbol{\sigma}^1\right\| \leq 2\beta_\sigma \ ,$$

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

$$\begin{aligned}
\boldsymbol{a}^\intercal \cdot (\boldsymbol{\sigma}^0 - \boldsymbol{\sigma}^1) &= \boldsymbol{a}^\intercal \cdot \boldsymbol{\sigma}^0 - \boldsymbol{a}^\intercal \cdot \boldsymbol{\sigma}^1 \\
&= (v_0^0 \cdot H(m) + v_1^0) - (v_0^1 \cdot H(m) + v_1^1) && (\text{Def of } \mathsf{sVrfy}) \\
&= (v_0^0 - v_0^1) \cdot H(m) + (v_1^0 - v_1^1) \ .
\end{aligned}$$

Therefore, the lemma statement follows. □

Lemma 38. *Let $n, \gamma, q', \alpha_H, \alpha_w, \lambda$ be positive integers with q' prime and n a power of two, with $q' > 16\alpha_w\alpha_H\varphi$. Let $H\colon \{0,1\}^* \to \mathcal{T}_{\alpha_H}$ be a hash function. If the $\mathsf{SIS}_{\mathcal{R},q',\gamma,2\beta_\sigma+4\alpha_w\alpha_H\varphi}$ problem is hard and H is collision resistant, then $\mathsf{KOTS}^{\mathrm{Chip}}$ as in Figure 6 is existentially unforgeable under rerandomized keys.*

Proof. Let $\mathcal{A}$ be an arbitrary adversary against the multi-user W'-existentially unforgeability under rerandomized keys with success probability $\nu(\lambda)$. We construct an algorithm $\overline{\mathcal{A}}$ that solves $\mathsf{SIS}_{\mathcal{R},q',\gamma,2\beta_\sigma+4\alpha_w\alpha_H\varphi}$ as follows. Given $\boldsymbol{a} \in \mathcal{R}_{q'}^\gamma$, $\overline{\mathcal{A}}$ honestly chooses secret keys $(\boldsymbol{s}_0^i, \boldsymbol{s}_1^i) \in \mathcal{B}_{\varphi,q'}^\gamma \times \mathcal{B}_{\varphi\alpha_H,q'}^\gamma$ uniformly at random for $i \in [T]$ and invokes $\mathcal{A}$ on public keys (v_0^i, v_1^i), with $v_b^i := \boldsymbol{a}^\intercal \cdot s_b^i$. Whenever $\mathcal{A}$ sends a signing query (i, m), $\overline{\mathcal{A}}$ will respond with the honestly computed signature $\boldsymbol{\sigma} := \boldsymbol{s}_0^i \cdot H(m) + \boldsymbol{s}_1^i$. Eventually $\mathcal{A}$ outputs a candidate forgery $(i^*, m^*, \boldsymbol{\sigma}^*, w^*)$ and $\overline{\mathcal{A}}$ will compute a signature on the same message as $\boldsymbol{\sigma'} := w^* \cdot \boldsymbol{s}_0^{i^*} \cdot H(m^*) + w^* \cdot \boldsymbol{s}_1^{i^*}$. It then outputs $\boldsymbol{\sigma}^* - \boldsymbol{\sigma'}$.

To analyze the success probability of $\overline{\mathcal{A}}$ suppose that $\mathcal{A}$ outputs a *valid* forgery. I.e., at most a single query was asked for index i^*, said query was *not* m^*, $w^* \in W'$ and $\mathsf{wVrfy}(\boldsymbol{a}, (w^* v_0^{i^*}, w^* v_1^{i^*}), m^*, \boldsymbol{\sigma}^*) = 1$. From this and the definition of $\boldsymbol{\sigma'}$ above it follows that

$$\begin{aligned}
\boldsymbol{a}^\intercal \cdot (\boldsymbol{\sigma}^* - \boldsymbol{\sigma'}) &= \boldsymbol{a}^\intercal \boldsymbol{\sigma}^* - \boldsymbol{a}^\intercal \boldsymbol{\sigma'} \\
&= (w^* \cdot v_0^{i^*} H(m) + w^* \cdot v_1^{i^*}) - \boldsymbol{a}^\intercal (w^* \cdot \boldsymbol{s}_0^{i^*} \cdot H(m^*) + w^* \cdot \boldsymbol{s}_1^{i^*}) \\
&= (w^* \cdot v_0^{i^*} H(m) + w^* \cdot v_1^{i^*}) - (w^* \cdot \boldsymbol{a}^\intercal \boldsymbol{s}_0^{i^*} \cdot H(m^*) + w^* \cdot \boldsymbol{a}^\intercal \boldsymbol{s}_1^{i^*}) \\
&= (w^* \cdot v_0^{i^*} \cdot H(m) + w^* \cdot v_1^{i^*}) - (w^* \cdot v_0^{i^*} \cdot H(m) + w^* \cdot v_1^{i^*}) = 0.
\end{aligned}$$

as required for a solution to the SIS problem.

Next, to argue that $\|\boldsymbol{\sigma}^* - \boldsymbol{\sigma}'\| \leq 2\beta_\sigma + 4\alpha_w\alpha_H\varphi$, note that the weak verification algorithm guarantees that $\|\boldsymbol{\sigma}^*\| \leq 2\beta_\sigma$. Further, since $w^* \in W'$ there exist $w_0, w_1 \in \mathcal{T}_{\alpha_w}$ such that $w^* = w_0 - w_1$ and $\|w^*\|_1 \leq \|w_0\|_1 + \|w_1\|_1 = 2\alpha_w$. We can thus bound the norm of $\boldsymbol{\sigma}'$ as

$$\begin{aligned}
\|\boldsymbol{\sigma}'\| &= \left\|w^* \cdot \boldsymbol{s}_0^{i^*} \cdot H(m^*) + w^* \cdot \boldsymbol{s}_1^{i^*}\right\| && \text{(Def. of } \mathsf{Sign}) \\
&= \left\|w^* \cdot \boldsymbol{s}_0^{i^*} \cdot H(m^*)\right\| + \left\|w^* \cdot \boldsymbol{s}_1^{i^*}\right\| && \text{(Triangle Inequality)} \\
&= \|w^*\|_1 \cdot \|H(m^*)\|_1 \cdot \left\|\boldsymbol{s}_0^{i^*}\right\| + \|w^*\|_1 \cdot \left\|\boldsymbol{s}_1^{i^*}\right\| && \text{(Lemma 1)} \\
&= 4\alpha_w\alpha_H\varphi \ . && (w^* \in W' \text{ and } H(m^*) \in \mathcal{T}_{\alpha_H})
\end{aligned}$$

It follows that $\|\boldsymbol{\sigma}^* - \boldsymbol{\sigma}'\| \leq \|\boldsymbol{\sigma}^*\| + \|\boldsymbol{\sigma}'\| \leq 2\beta_\sigma + 4\alpha_w\alpha_H\varphi$ as required.

Finally, we need to argue that $\boldsymbol{\sigma}^* - \boldsymbol{\sigma}' \neq 0$. This is the case iff $\boldsymbol{\sigma}^* \neq \boldsymbol{\sigma}'$. It thus suffices to bound the probability, that $\boldsymbol{\sigma}^* = \boldsymbol{\sigma}'$.

To this end, we observe by Lemma 39 that, since $\mathcal{A}$ has learned at most a single signature under $(v_0^{i^*}, v_1^{i^*})$, the corresponding $(\boldsymbol{s}_0^{i^*}, \boldsymbol{s}_1^{i^*})$ remains information-theoretically hidden from $\mathcal{A}$ among at least 2 possible secret keys. Once $\mathcal{A}$ outputs a valid forgery $(i^*, m^*, \boldsymbol{\sigma}^*, w^*)$ the signing key used for the forgery becomes uniquely determined by Lemma 40 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 $1/2 - \mathsf{negl}(\lambda)$. Therefore, the success probability of our reduction $\overline{\mathcal{A}}$ is $(1/2 - \mathsf{negl}(\lambda))\nu(\lambda)$ and since the SIS problem is assumed to be hard, $\nu(\lambda)$ must therefore be negligible in λ. □

Lemma 39. *Let $n, \gamma, q', \alpha_H, \varphi, \lambda$ be positive integers such that there exists δ with $\gamma \geq ((3\lambda + \delta)/n + \log_2 q)\log_2^{-1}(\varphi + \frac{1}{2})$ and $|\mathcal{T}_{\alpha_H}| \leq 2^{2\lambda+\delta}$, let $\mathcal{R}_{q'} = \mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$. Then for any $\boldsymbol{a} \in \mathcal{R}_{q'}^\gamma$ and uniformly chosen $(\boldsymbol{s}_0, \boldsymbol{s}_1) \in \mathcal{B}_{\varphi,q'}^\gamma \times \mathcal{B}_{\varphi\alpha_H,q'}^\gamma$ it holds with probability at least $1 - 2^{-\lambda}$ that for every $c \in \mathcal{T}_{\alpha_H}$ there exists $(\boldsymbol{s}_0', \boldsymbol{s}_1') \in \mathcal{B}_{\varphi,q'}^\gamma \times \mathcal{B}_{\varphi\alpha_H,q'}^\gamma$ such that $(\boldsymbol{s}_0', \boldsymbol{s}_1') \neq (\boldsymbol{s}_0, \boldsymbol{s}_1)$, $(\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0', \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1') = (\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0, \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1)$ and $\boldsymbol{s}_0' \cdot c + \boldsymbol{s}_1' = \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1$.*

Proof. We define a function $f_{\boldsymbol{a},c}$ that maps any secret key $(\boldsymbol{s}_0, \boldsymbol{s}_1)$ to a pair of public key and signature defined as $((\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0, \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1), \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1)$. 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 $(2\varphi + 1)^{n\gamma} \cdot (2\varphi\alpha_H + 1)^{n\gamma}$. The number of possible signatures is at most $(4\varphi\alpha_H + 1)^{n\gamma}$. For fixed values $\boldsymbol{a}, c, \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1$, we observe that once $\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0$ is fixed, the second component $\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1 = \boldsymbol{a}^\intercal \cdot ((\boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1) - \boldsymbol{s}_0 \cdot c)$ is uniquely determined. Thus for a fixed signature, there are at most q'^n many possible public keys and therefore the size of the range of $f_{\boldsymbol{a},c}$ is at most $(4\varphi\alpha_H + 1)^{n\gamma} \cdot q'^n$. We observe that

$$\begin{aligned}
\frac{(2\varphi + 1)^{n\gamma} \cdot (2\varphi\alpha_H + 1)^{n\gamma}}{(4\varphi\alpha_H + 1)^{n\gamma} \cdot q'^n} &\geq \frac{(2\varphi + 1)^{n\gamma} \cdot (2\varphi\alpha_H + 1)^{n\gamma}}{(4\varphi\alpha_H + 2)^{n\gamma} \cdot q'^n} \\
&= \frac{(2\varphi + 1)^{n\gamma}}{2^{n\gamma} \cdot q'^n} \\
&= \left(\varphi + \tfrac{1}{2}\right)^{n\gamma} \cdot \frac{1}{q'^n} \\
&= 2^{\log_2(\varphi + \frac{1}{2}) \cdot n\gamma - n\log_2 q'} \ .
\end{aligned}$$

Using the condition on γ from the lemma statement, one can see that

$$\log_2(\varphi + \tfrac{1}{2}) \cdot n\gamma - n \log_2 q' \geq n\big(\tfrac{3\lambda+\delta}{n} + \log_2 q\big) - n \log_2 q' = 3\lambda + \delta$$

and thus, as claimed the domain of $f_{\boldsymbol{a},c}$ is at least $2^{3\lambda+\delta}$ times larger than its range.

Using Lemma 4.1 from [LM08], the probability, over a uniformly chosen secret key, that there exists $(\boldsymbol{s}'_0, \boldsymbol{s}'_1) \in \mathcal{B}^{\gamma}_{1,q'} \times \mathcal{B}^{\gamma}_{\beta_s,q'}$ such that $(\boldsymbol{s}'_0, \boldsymbol{s}'_1) \neq (\boldsymbol{s}_0, \boldsymbol{s}_1)$, $(\boldsymbol{a}^\intercal \cdot \boldsymbol{s}'_0, \boldsymbol{a}^\intercal \cdot \boldsymbol{s}'_1) = (\boldsymbol{a}^\intercal \cdot \boldsymbol{s}_0, \boldsymbol{a}^\intercal \cdot \boldsymbol{s}_1)$ and $\boldsymbol{s}'_0 \cdot c + \boldsymbol{s}'_1 = \boldsymbol{s}_0 \cdot c + \boldsymbol{s}_1$ is at least $1 - 2^{-3\lambda-\delta}$. By union bounding over all possible hash values $c \in \mathcal{T}_{\alpha_H}$ and observing that $|\mathcal{T}_{\alpha_H}| \leq 2^{2\lambda+\delta}$ the lemma statement follows. □

Lemma 40. *Let $n, \gamma, q', \alpha_H, \alpha_w$ be positive integers with q' prime and n a power of two such that $q' > 16\alpha_w\alpha_H\varphi$ and let $\mathcal{R}_{q'} = \mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$. Let $\boldsymbol{a} \in \mathcal{R}^{\gamma}_{q'}$, $c_0, c_1 \in \mathcal{T}_{\alpha_H}$, $w_0, w_1 \in \mathcal{T}_{\alpha_w}$, and $\sigma_0, \sigma_1 \in \mathcal{R}$ 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 $(\boldsymbol{s}_0, \boldsymbol{s}_1) \in \mathcal{B}^{\gamma}_{\varphi,q'} \times \mathcal{B}^{\gamma}_{\varphi\alpha_H,q'}$, such that*

$$\boldsymbol{s}_0 \cdot c_0 + \boldsymbol{s}_1 = \sigma_0 \quad \textit{and} \quad (w_0 - w_1) \cdot (\boldsymbol{s}_0 \cdot c_1 + \boldsymbol{s}_1) = \sigma_1 \ .$$

Proof. Let $(\boldsymbol{s}_0, \boldsymbol{s}_1) \in \mathcal{B}^{\gamma}_{\varphi,q'} \times \mathcal{B}^{\gamma}_{\varphi\alpha_H,q'}$ and $(\boldsymbol{s}'_0, \boldsymbol{s}'_1) \in \mathcal{B}^{\gamma}_{\varphi,q'} \times \mathcal{B}^{\gamma}_{\varphi\alpha_H,q'}$ be two secret keys, such that

$$\boldsymbol{s}_0 \cdot c_0 + \boldsymbol{s}_1 = \boldsymbol{s}'_0 \cdot c_0 + \boldsymbol{s}'_1 \implies (\boldsymbol{s}_0 - \boldsymbol{s}'_0) \cdot c_0 + (\boldsymbol{s}_1 - \boldsymbol{s}'_1) = 0 \tag{9}$$

and

$$\begin{aligned} (w_0 - w_1) \cdot (\boldsymbol{s}_0 \cdot c_1 + \boldsymbol{s}_1) &= (w_0 - w_1) \cdot (\boldsymbol{s}'_0 \cdot c_1 + \boldsymbol{s}'_1) \\ \implies (w_0 - w_1)((\boldsymbol{s}_0 - \boldsymbol{s}'_0) \cdot c_1 &+ (\boldsymbol{s}_1 - \boldsymbol{s}'_1)) = 0 \ . \end{aligned} \tag{10}$$

Equation 9 implies that

$$(w_0 - w_1)((\boldsymbol{s}_0 - \boldsymbol{s}'_0) \cdot c_0 + (\boldsymbol{s}_1 - \boldsymbol{s}'_1)) = 0.$$

Combined with Equation 10, we get that in $\mathcal{R}_{q'}$

$$(w_0 - w_1)(\boldsymbol{s}_0 - \boldsymbol{s}'_0)(c_0 - c_1) = 0 \ . \tag{11}$$

Since $w_0, w_1 \in \mathcal{T}_{\alpha_w}$, $\boldsymbol{s}_0, \boldsymbol{s}'_0 \in \mathcal{B}^{\gamma}_{\varphi,q'}$, and $c_0, c_1 \in \mathcal{T}_{\alpha_H}$, it holds by Lemma 1 that

$$\big\|(w_0 - w_1)(\overline{\boldsymbol{s}}_0 - \overline{\boldsymbol{s}}'_0)(c_0 - c_1)\big\| \leq \|w_0 - w_1\|_1 \cdot \|c_0 - c_1\|_1 \cdot \big\|(\overline{\boldsymbol{s}}_0 - \overline{\boldsymbol{s}}'_0)\big\| \leq 8\alpha_w\alpha_H\varphi \leq \tfrac{q'-1}{2} \ ,$$

where $\overline{\boldsymbol{s}}_i \in \mathcal{R}$ is the representative of $\boldsymbol{s}_i \in \mathcal{R}_{q'}$ with coefficients in $\{-\frac{q'-1}{2}, \ldots, +\frac{q'-1}{2}\}$. Therefore Equation 11 also holds in $\mathcal{R}$. Since $w_0 \neq w_1$, $c_0 \neq c_1$, and $\mathcal{R}$ is an integral domain, it follows that $\boldsymbol{s}_0 = \boldsymbol{s}'_0$. By Equation 9, it must therefore hold that $(\boldsymbol{s}_0, \boldsymbol{s}_1) = (\boldsymbol{s}'_0, \boldsymbol{s}'_1)$. □

6 Synchronized Multi-Signatures

In this section, we show how the tools developed in the previous sections can be combined to yield a synchronized multi-signature with the desired properties. We do this in a manner that is almost identical to the way Squirrel [FSZ22a] does it, except that our aggregation is modified to use a rejection sampling technique that allows us to reduce the signature size. Roughly speaking, a public key in the multi-signature scheme is a vector commitment to a vector of independent one-time signature public keys. To sign a message at time t, the signer publishes an opening to the key in vector position t and signs the message with that key.

To aggregate these signatures, the construction computes a random linear combination of them, using weights derived using a random oracle. The uniform distribution of weights allows us to leverage the probabilistic homomorphism of the KOTS and HVC schemes, such that this aggregation procedure will be successful with probability at least $1 - 2\varepsilon$. By rejecting unsuccessful attempts and retrying a number of times, the overall probability of an aggregation failure can be made negligible.

We will now formally define the requirements for a synchronized multi-signature scheme. Once again, our definitions follow the definitions of Fleischhacker, Simkin, and Zhang [FSZ22a]. In contrast to their work, however, we define a significantly stronger notion of correctness for aggregated signatures. More concretely, [FSZ22a] only required that aggregation is successful for honestly generated keys and signatures. We, on the other hand, require that any sequence of *individually valid* signatures can be successfully aggregated.[14]

Definition 41 (Synchronized Multi-Signatures). *A synchronized ρ-wise multi-signature scheme for 2^τ time periods is defined by six PPT algorithms* $\mathsf{MSIG} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Aggregate}, \mathsf{iVrfy}, \mathsf{aVrfy})$.

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

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

$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, t, m)$ *The signing algorithm takes as input the public parameters, a secret key, a time period $1 \leq t \leq 2^\tau$, and a message and outputs a signature.*

$\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S})$ *The aggregation algorithm takes as input the public parameters, a list of public keys, a time period $1 \leq t \leq 2^\tau$, a message, and a list of signatures, where $|\mathcal{P}| = |\mathcal{S}| \leq \rho$ and outputs an aggregated signature or an error $\perp$.*

$b \leftarrow \mathsf{iVrfy}(\mathsf{pp}, \mathsf{pk}, t, m, \sigma)$ *The deterministic individual verification algorithm takes as input the public parameters, a public key, a time period $1 \leq t \leq 2^\tau$, a message, and a signature and outputs a bit indicating acceptance/rejection.*

$b \leftarrow \mathsf{aVrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}})$ *The deterministic aggregated verification algorithm takes as input the public parameters, a list of public keys, a time period $1 \leq t \leq 2^\tau$, a message, and an aggregated signature and outputs a bit indicating acceptance/rejection.*

Definition 42 (Individual Correctness). *Let* MSIG *be a synchronized ρ-wise multi-signature scheme for 2^τ time periods.* MSIG *is individually correct if for all security parameters $\lambda \in \mathbb{N}$, public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, *key pairs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$, *time periods $1 \leq t \leq 2^\tau$, message $m \in \{0,1\}^*$, and signatures* $\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, t, m)$ *it holds that*

$$\mathsf{iVrfy}(\mathsf{pp}, \mathsf{pk}, t, m, \sigma) = 1 \ .$$

Definition 43 (Aggregation Correctness with Rogue Keys and Signatures). *Let* MSIG *be a synchronized ρ-wise multi-signature scheme for 2^τ time periods.* MSIG *has correct aggregations in the presence of rogue keys and signatures if for all security parameters $\lambda \in \mathbb{N}$, public parameters* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, *number of aggregated signatures $\ell \in [\rho]$, time periods $1 \leq t \leq 2^\tau$, messages $m \in \{0,1\}^*$, public keys $\mathcal{P} = (\mathsf{pk}^1, \ldots, \mathsf{pk}^\ell)$ and signatures $\mathcal{S} = (\sigma^1, \ldots, \sigma^\ell)$, such that for all $i \in [\ell]$,* $\mathsf{iVrfy}(\mathsf{pp}, \mathsf{pk}^i, t, m, \sigma^i) = 1$ *it holds that*

$$\Pr[\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S}) \colon \mathsf{aVrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}}) = 1] = 1 - \mathsf{negl}(\lambda) \ .$$

[14] It is worth noting, that the *construction* of Squirrel [FSZ22a] actually satisfies this stronger notion. It was just never defined or proven.

Definition 44 (Unforgeability). *Let* MSIG *be a synchronized ρ-wise multi-signature scheme for 2^τ time periods.* MSIG *is unforgeable if for all security parameters $\lambda \in \mathbb{N}$, and all PPT algorithms $\mathcal{A}$ it holds that*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KGen}(\mathsf{pp}); \\ (\mathcal{P}, t, m, \sigma_{\mathsf{agg}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}^*, \cdot, \cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{array} : \begin{array}{l} \mathsf{aVrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}}) = 1 \\ \wedge\, \mathsf{pk}^* \in \mathcal{P} \\ \wedge\, \forall\, (t', m', \sigma') \in \mathcal{Q}.\ (t', m') \neq (t, m) \\ \wedge\, \forall\, t'.\ |\mathcal{Q}_{t'}| \leq 1 \end{array}\right] \leq \mathsf{negl}(\lambda)\ ,$$

where $\mathcal{Q}$ denotes the set of signing queries made by $\mathcal{A}$ and $\mathcal{Q}_{t'}$ denotes the set of signing queries made for timeslot t'.

6.1 Construction

For ease of notation we define the function zip that "zips" up two vectors into a single vector of pairs, i.e.

$$\mathsf{zip}(\boldsymbol{a}, \boldsymbol{b}) := \begin{pmatrix} (a_1, b_1) \\ \vdots \\ (a_\ell, b_\ell) \end{pmatrix}.$$

The following theorem now states the security of our construction presented in Figure 7.

Theorem 45. *Let $\lambda, n, q', \xi, \chi, \tau$ be positive integers and $0 < \varepsilon < \frac{1}{2}$ with n being a power of two, q' being prime, and $\chi \geq \lambda / \log_2(\frac{1}{2\varepsilon})$. Let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$. Let $W \subseteq \mathcal{R}$ be a set such that $|W| > 2^\lambda$ and let $W' := \{w^0 - w^1 | w^0, w^1 \in W\}$. Let $H\colon \{0,1\}^* \to W^\rho$ be a random oracle. Let* KOTS *be a key homomorphic one-time signature scheme with public keys in $\mathcal{R}_{q'}^\xi$ and let* HVC *be a homomorphic vector commitment for domain $\mathcal{R}_{q'}^\xi$.*

If KOTS *is individually correct, (ρ, W, ε)-probabilistically homomorphic, robustly homomorphic, and W'-multi-user existentially unforgeable under rerandomized keys and* HVC *is individually correct, (ρ, W, ε)-probabilistically homomorphic, robustly homomorphic, and position-binding, then the construction from Figure 7 is an unforgeable synchronized ρ-wise multi-signature scheme that is individually correct and has correct aggregations in the presence of rogue keys and signatures.*

Proof. The theorem follows immediately from Lemma 47, Lemma 48, Lemma 49 below. □

Our concrete proposal is to use $\mathsf{HVC}^{\mathrm{Chip}}_{\mathrm{Encoded}}$ and $\mathsf{KOTS}^{\mathrm{Chip}}$ to thereby construct the Chipmunk synchronized multi-signature.

Definition 46 (Chipmunk synchronized multi-signatures). *Let $n, q, q', \eta, \tau, \rho, \alpha_H, \alpha_w, \gamma$ be positive integers, with n being a power of two, q, q' prime. Let $\mathcal{R}, \mathcal{R}_q, \mathcal{R}_{q'}$ be as usual. We define the synchronized multi-signature Chipmunk, denoted $\mathsf{MSIG}^{\mathrm{Chip}}$, by instanciating the construction from Figure 7 with $\mathsf{HVC}^{\mathrm{Chip}}_{\mathrm{Encoded}}$ with $\xi = 2$ and $\mathsf{KOTS}^{\mathrm{Chip}}$.*

As a corollary of Theorem 45, we obtain that $\mathsf{MSIG}^{\mathrm{Chip}}$ is an unforgeable synchronized ρ-wise multi-signature that is individually correct and has correct aggregations in the presence of rogue keys and signatures, provided $\mathsf{HVC}^{\mathrm{Chip}}_{\mathrm{Encoded}}$ and $\mathsf{KOTS}^{\mathrm{Chip}}$ satisfy the appropriate security properties. The latter are guaranteed by Theorem 27 and Theorem 34, provided we set parameters appropriately

```
$\mathsf{Setup}(1^\lambda)$
$\mathsf{pp}_{\mathsf{KOTS}} \leftarrow \mathsf{KOTS.Setup}(1^\lambda)$
$\mathsf{pp}_{\mathsf{HVC}} \leftarrow \mathsf{HVC.Setup}(1^\lambda)$
return $\mathsf{pp} := (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}})$
```

```
$\mathsf{KGen}(\mathsf{pp})$
foreach $1 \le i \le 2^\tau$
  $(\mathsf{osk}^i, \mathsf{opk}^i) \leftarrow \mathsf{KOTS.KGen}(\mathsf{pp}_{\mathsf{KOTS}})$
$\mathsf{OSS} = (\mathsf{osk}^1, \dots, \mathsf{osk}^{2^\tau})$
$\mathsf{OPK} = (\mathsf{opk}^1, \dots, \mathsf{opk}^{2^\tau})$
$\boldsymbol{c} \leftarrow \mathsf{HVC.Com}(\mathsf{pp}_{\mathsf{HVC}}, \mathsf{OPK})$
return $(\mathsf{sk}, \mathsf{pk}) := ((\mathsf{OSS}, \mathsf{OPK}), \boldsymbol{c})$
```

```
$\mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S})$
if $|\mathcal{S}| \neq |\mathcal{P}|$
  return $\perp$
for $(\mathsf{pk}, \sigma) \in \mathsf{zip}(\mathcal{P}, \mathcal{S})$
  if $\mathsf{iVrfy}(\mathsf{pp}, \mathsf{pk}, t, m, \sigma) = 0$
    return $\perp$
$j := 0$
do
  $j := j + 1$
  $(w^0, \dots, w^{|\mathcal{P}|}) := H(t, m, \mathcal{P}, j)$
  $\boldsymbol{\sigma}' := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{\sigma}'^i$
  $\boldsymbol{d} := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{d}^i$
while $j < \chi$ and $\mathsf{aVrfy}(\mathsf{pp}, \mathcal{P}, t, m, (\boldsymbol{\sigma}', \boldsymbol{d}, j)) = 0$
return $\sigma_{\mathsf{agg}} := (\boldsymbol{\sigma}', \boldsymbol{d}, j)$
```

```
$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, t, m)$
$\boldsymbol{\sigma}' \leftarrow \mathsf{KOTS.Sign}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{osk}_t, m)$
$\boldsymbol{d} \leftarrow \mathsf{HVC.Open}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, \mathsf{OPK}, t)$
return $\sigma := (\boldsymbol{\sigma}', \boldsymbol{d})$
```

```
$\mathsf{iVrfy}(\mathsf{pp}, \mathsf{pk}, t, m, \sigma)$
$\mathsf{opk} \leftarrow \mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, t, \boldsymbol{d})$
if $t > 2^\tau$ or $\mathsf{opk} = \perp$
  return $0$
else
  return $\mathsf{KOTS.sVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}, m, \boldsymbol{\sigma}')$
```

```
$\mathsf{aVrfy}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}})$
$(w^1, \dots, w^{|\mathcal{P}|}) := H(t, m, \mathcal{P}, j)$
$\boldsymbol{c} := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{c}_i$
$\mathsf{opk} \leftarrow \mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, t, \boldsymbol{d})$
if $|\mathcal{P}| > \rho$ or $\mathsf{opk} = \perp$
  return $0$
else
  return $\mathsf{KOTS.sVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}, m, \boldsymbol{\sigma}')$
```

Fig. 7: A synchronized multi-signature scheme based on homomorphic vector commitments and key-homomorphic one-time signatures.

and the appropriate Ring-SIS problems are hard. We collect the neccessary conditions in Table 3. Note that W from Theorem 45 corresponds to $W = \mathcal{T}_{\alpha_w}$.

We now proceed to show Lemma 47, Lemma 48 and Lemma 49 to actually prove Theorem 45.

Lemma 47. *Let $\lambda, n, q', \xi, \chi, \rho, \tau$ be positive integers with n being a power of two, q' being prime. Let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n+1\rangle$. Let KOTS be a key-homomorphic one-time signature scheme with public keys in $\mathcal{R}_{q'}^{\xi}$ and let HVC be a homomorphic vector commitment for domain $\mathcal{R}_{q'}^{\xi}$.*

If both KOTS and HVC are individually correct, then the construction from Figure 7 is individually correct.

Proof. Let $\mathsf{pp} = (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}}) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{pk}) = ((\mathsf{OSS}, \mathsf{OPK}), \boldsymbol{c}) \leftarrow \mathsf{KGen}(\mathsf{pp})$, $1 \leq t \leq 2^\tau$,$m \in \{0,1\}^*$, and $\sigma = (\boldsymbol{\sigma}', \boldsymbol{d}) \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, t, m)$. By definition of the signing algorithm it holds that

$$\boldsymbol{\sigma}' \leftarrow \mathsf{KOTS.Sign}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{osk}^t, m) \quad \text{and} \quad \boldsymbol{d} \leftarrow \mathsf{HVC.Open}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, \mathsf{OPK}, t) \ .$$

By definition of the key generation algorithm it further holds that

$$(\mathsf{osk}^t, \mathsf{opk}^t) \leftarrow \mathsf{KOTS.KGen}(\mathsf{pp}_{\mathsf{KOTS}}) \ .$$

From the individual correctness of HVC and the definition of the individual verification algorithm it follows that

$$\mathsf{opk}^t = \mathsf{opk} \leftarrow \mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, t, \boldsymbol{d}) \ ,$$

which finally implies by the individual correctness of KOTS that

$$\mathsf{KOTS.sVrfy}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}, m, \boldsymbol{\sigma}') = 1 \ .$$

Individual correctness thus follows. □

Lemma 48. *Let $\lambda, n, q', \xi, \chi, \rho, \tau$ be positive integers and $0 < \varepsilon < \frac{1}{2}$ with n being a power of two, q' being prime and $\chi \geq \lambda/\log(\frac{1}{2\varepsilon})$. Let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n+1\rangle$. Let $W \subseteq \mathcal{R}$ be a set and let $W' := \{w^0 - w^1 | w^0, w^1 \in W\}$. Let $H\colon \{0,1\}^* \to W^\rho$ be a random oracle. Let KOTS be a key-homomorphic one-time signature scheme with public keys in $\mathcal{R}_{q'}^{\xi}$ and let HVC be a homomorphic vector commitment for domain $\mathcal{R}_{q'}^{\xi}$.*

If both KOTS and HVC are (ρ, W, ε)-probabilistically homomorphic, then the construction from Figure 7 has correct aggregations in the presence of rogue keys and signatures.

Proof. Let $\mathsf{pp} = (\mathsf{pp}_{\mathsf{HVC}}, \mathsf{pp}_{\mathsf{KOTS}}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\ell \in [\rho]$, $1 \leq t \leq 2^\tau$, $m \in \{0,1\}^*$, $\mathcal{P} = (\boldsymbol{c}^1, \ldots, \boldsymbol{c}^\ell)$, and $\mathcal{S} = (\sigma^1, \ldots, \sigma^\ell)$ with $\sigma^i = (\boldsymbol{\sigma}', \boldsymbol{d})$, be arbitrary, such that for all $i \in [\ell]$, $\mathsf{iVrfy}(\mathsf{pp}, \boldsymbol{c}^i, t, m, \sigma^i)$.

The aggregation algorithm makes up to χ attempts to aggregate the signature and will only output an *invalid* signature, if all χ attempts fail. It thus suffices to analyse the probability with which all attempts fail.

Attempt j is performed by computing weights $(w^1, \ldots, w^\ell) := H(t, m, \mathcal{P}, j)$ and computing

$$\boldsymbol{\sigma}' := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{\sigma}'_i \quad \text{and} \quad \boldsymbol{d} := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{d}_i \ .$$

Let $\boldsymbol{c} = \sum_{i\in[\ell]} w^i \cdot \boldsymbol{c}^i$. Since the signatures individually verify, there exists a well-defined $\mathsf{opk}^i := \mathsf{HVC.iVrfy}(\mathsf{pp}, \boldsymbol{c}^i, t, \boldsymbol{d}^i)$ for all $i \in [\ell]$. Since further H is a random oracle we can apply the (ρ, W, ε)-probabilistic homomorphism of both HVC of KOTS to conclude that

$$\Pr\Big[\mathsf{HVC.sVrfy}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, t, \boldsymbol{d}) \neq \sum_{i\in[\ell]} w^i \cdot \mathsf{opk}^i\Big] \leq \varepsilon \ .$$

and

$$\Pr\Big[\mathsf{KOTS.sVrfy}\Big(\mathsf{pp}_{\mathsf{KOTS}}, \sum_{i\in[\ell]} w^i \cdot \mathsf{opk}^i, m, \boldsymbol{\sigma}'\Big) = 0\Big] \leq \varepsilon \ .$$

The aggregation attempt fails if either of these conditions is violated. Therefore, by a union bound, each individual attempt fails with probability at most 2ε. Since each attempt is an independent Bernoulli trial the probability of overall failure of all $\chi \geq \lambda / \log_2(\frac{1}{2\varepsilon})$ attempts can be bounded by $(2\varepsilon)^\chi \leq 2^{-\lambda}$. Hence, aggregation will succeed with overwhelming probability. □

Lemma 49. *Let $\lambda, n, q', \xi, \chi, \rho, \tau$ be positive integers with n being a power of two, q' being prime. Let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$. Let $W \subseteq \mathcal{R}$ be a set such that $|W| > 2^\lambda$ and let $W' := \{w^0 - w^1 | w^0, w^1 \in W\}$. Let $H \colon \{0,1\}^* \to W^\rho$ be a random oracle. Let KOTS be a key-homomorphic one-time signature scheme with public keys in $\mathcal{R}_{q'}^\xi$ and let HVC be a homomorphic vector commitment for domain $\mathcal{R}_{q'}^\xi$.*

If KOTS is W'-multi-user existentially unforgeable under rerandomized keys and HVC is position-binding, then the construction from Figure 7 is unforgeable.

Proof. The proof for this lemma remains essentially identical to the proof of unforgeability for Squirrel [FSZ22a]. The entire argument is only concerned with the *aggregated verification* algorithm, the unforgeability of KOTS and the position binding of HVC. None of the differences between Chipmunk and Squirrel affect these parts, with the tiny exception that the random oracle during verification now takes the additional input j. Literally, the only necessary change in the proof is, therefore, that during the technically tedious forking lemma setup, the simulated random oracle needs to also take $j \in [\chi]$ as input. As such, we omit the proof here and refer the interested reader to the full version of the original Squirrel paper [FSZ22b]. We stress that the proof of unforgeability in [FSZ22b] relies on a variant [BN06] of the forking lemma [PS96], which uses a rewinding strategy that does not apply to quantum algorithms. □

7 Benchmarks

In this section, we define the parameters with which we instantiate Chipmunk and we provide various benchmarks, showing that our new construction significantly outperforms the previous Squirrel construction of Fleischhacker, Simkin, and Zhang [FSZ22a].

Our concretely proposed construction uses the key-homomorphic one-time signature scheme for Figure 6 and $\mathsf{HVC}^{\text{Chip}}_{\text{Encoded}}$ for the homomorphic vector commitment.

7.1 Parameters and Security Estimates

The dimension of the ring $\mathcal{R}$ is fixed to $n = 512$. We choose q, q', s.t. $q, q' \equiv 1 \mod 2n$, in order speed up multiplications in $\mathcal{R}_q$ and $\mathcal{R}_{q'}$ by using NTT. The constraints that need to be satisfied by our parameters are summarized in Table 3. The concrete efficiency for a given set of parameters is determined by Table 4, which also spells out where the contributions come from. To find such concrete parameters we used a script[15], which enumerates possible parameters that satisfy all constraints, that lead to hard ring-SIS problems, and that allow for efficient NTT evaluations. For any choice of $\lambda \in \{112, 128\}$, $\rho \in \mathbb{N}$, and $\tau \in \mathbb{N}$ our script finds the parameter set that allows for the smallest possible signature size. For convenience, the results of running the script for a range of reasonable input parameters are shown in Table 8 in Appendix A.

#	Source	Constraint
1	Lemma 19	$\beta_{\mathsf{agg}} \geq \eta\sqrt{2\alpha_w\rho(\ln\frac{2n}{\varepsilon} + \ln(2\tau\kappa + \xi\kappa' + 2\tau))}$
2	Lemma 19	$\kappa = \lceil \log_{2\eta+1} q \rceil$
3	Lemma 19	$\kappa' = \lceil \log_{2\eta+1} q' \rceil$
4	Theorem 25	$\beta_{\mathsf{encode}} < \frac{\beta_{\mathsf{agg}}}{2\eta} + 1/2$
5	Lemma 36	$\beta_\sigma \geq 4\varphi\alpha_H\sqrt{\frac{1}{2}\alpha_w\rho \cdot \ln\frac{2n\gamma}{\varepsilon}}$
6	Lemma 38	$\lvert\mathcal{T}_{\alpha_H}\rvert \geq 2^{2\lambda}$
7	Lemma 39	$\gamma \geq ((3\lambda + \delta)/n + \log_2 q')\log_2^{-1}(\varphi + \frac{1}{2})$
8	Lemma 39	$\lvert\mathcal{T}_{\alpha_H}\rvert \leq 2^{2\lambda+\delta}$
9	Lemma 40	$q' > 16\alpha_w\alpha_H\varphi$
10	Definition 46	$\xi = 2$
11	Lemma 48	$\chi \geq \lambda/\log(\frac{1}{2\varepsilon})$
12	Lemma 49	$\lvert\mathcal{T}_{\alpha_w}\rvert \geq 2^\lambda$

Table 3: The constraints a set of Chipmunk parameters needs to satisfy to ensure that the proofs are applicable. The parameters additionally need to be chosen such that the associated Ring-SIS problems are hard.

Contribution		Size in Bits
public parameters	HVC: Ajtai's hash functions	$n(\xi\kappa' + 2\kappa)\lceil\log q\rceil$
	KOTS	$n\gamma\lceil\log q'\rceil$
public key	HVC commitment	$n\lceil\log q\rceil$
secret key	2^τ KOTS keys	(may regenerate on the fly)[17]
signatures (individual)	KOTS signature	$n\gamma\lceil\log(4\varphi\alpha_H + 1)\rceil$
	HVC opening	$(\tau\kappa + \xi\kappa')n\lceil\log(2\eta + 1)\rceil$
aggregate signatures	agg. KOTS signature	$n\gamma\lceil\log(2\beta_\sigma + 1)\rceil$
	agg. HVC opening	$(\tau(\kappa - 1) + (\xi\kappa' - 1))n\lceil\log(2\beta_{\mathsf{agg}} + 1)\rceil + \tau\kappa n\lceil\log(\lfloor\frac{\beta_{\mathsf{agg}}}{\eta}\rfloor + 2)\rceil$
	index j of aggregation attempt	$\lceil\log\chi\rceil$

Table 4: Space efficiency of Chipmunk as a function of the tunable parameters. $\kappa := \lceil\log_{2\eta+1} q\rceil$, $\kappa' := \lceil\log_{2\eta+1} q'\rceil$

Let us briefly explain how our script works. First, to ensure security of Chipmunk, the specific ring-SIS problems in Lemma 19 and 36 need to be hard. We use the same approach to derive

[15] https://github.com/GottfriedHerold/Chipmunk

the security of the parameters as was used in Squirrel [FSZ22a]. We adopt the so-called "realistic model" from [ADPS16]. For a BKZ of block size β, the cost in this model is estimated by $2^{0.292\beta+16.4+\log(\#\texttt{SVP calls})}$. The LWE-estimator [APS15] shows that for a root Hermite factor of 1.005 we expect a block size of 286, which yields 112 bits of security under the above model. Similarly, a root Hermite factor of 1.004 yields 128 bits of security.

The output length of the hash function that is used for hashing messages needs to be large enough to prevent meet-in-the-middle type of attacks and from the security proofs we also need that for given public parameter $\boldsymbol{a}$, a fixed hash digest in $\mathcal{T}_{\alpha_H}$, and a signature $\boldsymbol{\sigma}$ for the one time signature scheme, there exists at least two short corresponding $(\boldsymbol{s}_0, \boldsymbol{s}_1)$ with overwhelming probability. Lastly, we also require every randomizer to be unguessable, and therefore $|\mathcal{T}_{\alpha_w}| \geq 2^{\lambda}$.

7.2 Implementation

Chipmunk was implemented in Rust. The source code, as well as the scripts for parameter derivation are released to the open domain[15].

Comparison. For evaluating the performance of Chipmunk, there are two natural points of reference, which are the trivial solution of just storing a list of ρ Falcon signatures naively and using Squirrel [FSZ22a], the previous state-of-the-art construction. The data for Chipmunk and Squirrel are collected over a same benchmark platform, an AMD 5900x with 24 threads and 32 Gigabytes of memory. The data for Falcon-512 is collected from the official website[16]. All three candidates are instantiated to yield 112 bits security.

In Table 5, we compare the three solutions for a fixed parameters set, where we only change the number of signatures that are being aggregated. The comparison with the trivial Falcon solution is quite straightforward as the size of the naive solution's aggregated signatures grows linearly in the number of signers. For an aggregate signature involving 8192 signers, Chipmunk outperforms the trivial solution by a factor of 40 in terms of aggregate signature size. Obviously, the improvement only gets larger as the number of signers increases, In comparison to Squirrel, we see that for both 1024 and 8192 aggregated signatures, our scheme is better in all metrics. There are two main obstacles, namely the key generation time and the aggregate signature size, that would prevent Squirrel from being widely deployable. Our benchmarks show that Chipmunk's key generation time is better by a factor of 7.4 and that the size of the aggregate signatures is smaller by a factor of 5.6.

Cost of encoding. We benchmark the cost of encoding mechanism. On a single thread, the encoding algorithm takes 7.3μs to convert a single node into its encoded form and the decoding algorithm takes 6.6μs for the reverse direction. Even though the costs of encoding and decoding are negligible themselves, they do affect the overall performance of the verification significantly. Without our encoding algorithm, during verification each layer in the HVC opening can be verified in parallel. With our encoding algorithm a serial dependency is introduced and for this reason the verifier needs to compute the hint from the previous layer before decoding the current layer. This introduces a trade-off between verification cost and the aggregated signature size, as captured in Table 6.

[16] https://falcon-sign.info/

# signers		Falcon	Squirrel	Chipmunk	Imp. Falcon	Imp. Squirrel
	Key Generation	8.6 ms	4 min	32.3 sec	-	7.4×
	Signing	0.17 ms	2.1 ms	0.4 ms	-	5.2×
	Fresh Sig. Size	666 Bytes	45 KB	32 KB	-	1.4×
1024	Aggregation	-	1.2 sec	0.57 sec	-	2.1×
	Batch Verification	36.7 ms	19.5 ms	7.7 ms	4.8×	2.5×
	Agg. Sig. size	682 KB	572 KB	118 KB	5.7×	4.8×
8192	Aggregation	-	9.6 sec	4.6 sec	-	2.1×
	Batch Verification	294 ms	53 ms	45 ms	6.5×	1.2×
	Agg. Sig. size	5.5 MB	762 KB	136 KB	40 ×	5.6×

Table 5: Comparison of Chipmunk, Squirrel, and a trivial solution of just concatenating all individual signer's Falcon-512 signatures. Squirrel and Chipmunk are instantiated with $\lambda = 112$ and $\tau = 21$.

Tree Height	With Encoding		Without Encoding	
	Signature size	Verification time	Signature size	Verification time
21	118 KB	8.6 ms	142 KB	7.3 ms
24	133 KB	8.9 ms	160 KB	7.5 ms
26	143 KB	9.4 ms	172 KB	7.1 ms

Table 6: Trade-off between signature size and verification cost via encoding algorithm. Instantiated with $\lambda = 112$ and $\rho = 1024$.

Note that with encoding, although the verification algorithm becomes serial across different layers of the tree, the main computation (i.e., the ring multiplications) within each layer is still parallelization friendly. Overall, for the platform that we tested (with 24 threads), we only observe a slight decrease in the verification speed compared to non-encoding method. We conclude that it is always beneficial to use our encoding mechanism.

Scalability in terms of τ. To investigate Chipmunk's practicality, we take a closer look at the key generation time and the aggregated signature size. For this purpose, we conducted a benchmark over a typical server that is equipped with an AMD 7773x with 64 cores and 1 Terabytes of memory. In Table 7, we show how the efficiency of Chipmunk behaves, when the tree height τ grows. Notice that this platform is different from that of Section 7.2 and that it more accurately simulates the computational power of real-world nodes running blockchains.

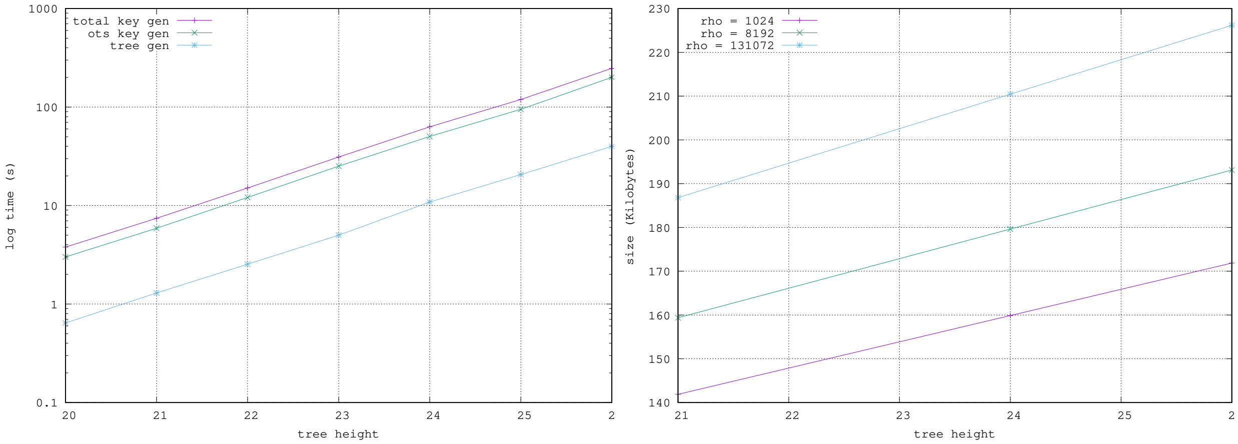
In Figure 8, we also plotted the key generation times and aggregate signature sizes of Chipmunk for a growing tree height τ. One can see that both of these benchmarks scale linearly in the number of leaves of the tree as expected. For the largest parameter set with $\tau = 26$ we can generate a keypair in just 4 minutes, significantly improving over the 2 hour key generation time of Squirrel.

[17] Similar to Squirrel [FSZ22a], Chipmunk is an online-offline signature scheme, since the opening of the vector commitment can be computed ahead of time without knowing the message to be signed. This means that online signing only consists of computing the one-time signature. The secret key of Chipmunk is a large tree, which can be re-derived from a pseudorandom seed whenever needed. This is computationally quite expensive, but a signer can trade storage size against offline signing speed by caching the top layers of the tree. The reported times for signing correspond to the *online* signing time.

Tree Height	Key Generation	Online Signing[17]	# Signers	Aggregation	Verification	Agg. Sig. Size
21	9.1 sec	0.40 ms	1024	468 ms	8.6 ms	118 KB
			8192	3.8 sec	42.6 ms	136 KB
24	37.4 sec	0.44 ms	1024	516 ms	8.9 ms	133 KB
			8192	4.1 sec	46 ms	153 KB
26	4 min	0.44 ms	1024	630 ms	9.4 ms	143 KB
			8192	4.6 sec	51 ms	164 KB

Table 7: Benchmark results

Such a key would be sufficient for 21 years of usage assuming each block takes 10 seconds to finalize as in the case of Ethereum. We conclude that the key generation time is no longer a bottleneck for practical deployment.



(a) Chipmunk key generations time.

(b) Chipmunk aggregated signature size

Fig. 8: Plots showing the scaling characteristics of the key generation time and aggregated signature size of Chipmunk.

One Aggregate Signature under the Microscope. To better understand the size of Chipmunk aggregate signatures, we also inspected the sizes of the aggregate signature's individual components. For the sake of concreteness, let us just consider $\tau = 21$ and $\rho = 1024$. An aggregated signature of size approx 118 Kilobytes, fitting inside a single ethereum block whose peak size is around 130 Kilobytes[18]. It consists of the following three components.

First, an encoding of the aggregated path and its adjacent nodes. The aggregated path and its adjacent nodes belong to the homomorphic vector commitment, i.e. $2\tau\kappa$ polynomials in $\mathcal{R}$ with an infinity norm bound $\beta_{\mathtt{agg}}$. We use the encoding method to encode half of those ring elements. Therefore, all these nodes can be represented with $\tau\kappa$ polynomials bounded by $\beta_{\mathtt{encode}}$; and another $\tau\kappa$ polynomials bounded by $\beta_{\mathtt{agg}}$. The total size of the path is $\tau\kappa n((\log(\beta_{\mathtt{encode}})+1)+(\log(\beta_{\mathtt{agg}})+1)) = 102$ Kilobytes.

[18] https://etherscan.io/chart/blocksize

The aggregated decomposed public keys for the one time signature scheme, i.e., $2\kappa'$ polynomials in $\mathcal{R}$ with a same norm bound β_{agg}. This requires $2\kappa' n(\log(\beta_{\mathsf{agg}})+1) = 8$ Kilobytes.

The last component is the aggregated one time signature, i.e., γ polynomials in $\mathcal{R}$ with norm bound $\beta_\sigma < 2^{20}$, that constitutes $\gamma n(\log(\beta_\sigma)+1) = 8$ Kilobytes.

The total size of the aggregated signature is therefore $102 + 8 + 8 = 118$ Kilobytes.

References

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 327–343, Austin, TX, USA, August 10–12, 2016. USENIX Association. 7.1

AGH10. Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 473–484, Chicago, Illinois, USA, October 4–8, 2010. ACM Press. 1.1

Ajt99. Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP 99: 26th International Colloquium on Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 1–9, Prague, Czech Republic, July 11–15, 1999. Springer, Heidelberg, Germany. 1.2, 3.1

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015. 7.1

Bab86. László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Comb.*, 6(1):1–13, 1986. 1.2, 2

BGLS03. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany. 1

BK20. Dan Boneh and Sam Kim. One-time and interactive aggregate signatures from lattices. https://crypto.stanford.edu/~skim13/agg_ots.pdf, 2020. 1.2, 5

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany. 1

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. 6.1

BT23. Katharina Boudgoust, , and Akira Takahashi. Sequential half-aggregation of lattice-based signatures. In *Computer Security-ESORICS 2023: 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023. Proceedings.* Springer, 2023. 1

BTT22. Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. MuSig-L: Lattice-based multi-signature with single-round online phase. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 276–305, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. 1

DGNW20. Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 2093–2110. USENIX Association, August 12–14, 2020. 1.1

DOTT21. Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 99–130, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany. 1

ES16. Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 140–155, Milan, Italy, November 14–16, 2016. Springer, Heidelberg, Germany. 1

FH19. Masayuki Fukumitsu and Shingo Hasegawa. A tightly-secure lattice-based multisignature. In *6th ASIA Public-Key Cryptography Workshop*, page 3–11, Auckland, New Zealand, 2019. Association for Computing Machinery. 1

FH20. Masayuki Fukumitsu and Shingo Hasegawa. A lattice-based provably secure multisignature scheme in quantum random oracle model. In Khoa Nguyen, Wenling Wu, Kwok-Yan Lam, and Huaxiong Wang, editors, *ProvSec 2020: 14th International Conference on Provable Security*, volume 12505 of *Lecture Notes in Computer Science*, pages 45–64, Singapore, November 29 – December 1, 2020. Springer, Heidelberg, Germany. 1

FSZ22a. Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. Squirrel: Efficient synchronized multi-signatures from lattices. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 1109–1123, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. 1, 1.1, 1.2, 1.2, 3, 3, 3.1, 3.2, 3.2, 3.2, 5, 5, 5, 6, 14, 6.1, 7, 7.1, 7.2, 17

FSZ22b. Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. Squirrel: Efficient synchronized multi-signatures from lattices. Cryptology ePrint Archive, Report 2022/694, 2022. https://eprint.iacr.org/2022/694. 6.1

GR06. Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 257–273, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany. 1.1

HW18. Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 197–229, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. 1.1

IN83. Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983. 1

KD20. Meenakshi Kansal and Ratna Dutta. Round optimal secure multisignature schemes from lattice with public key aggregation and signature compression. In Abderrahmane Nitaj and Amr M. Youssef, editors, *AFRICACRYPT 20: 12th International Conference on Cryptology in Africa*, volume 12174 of *Lecture Notes in Computer Science*, pages 281–300, Cairo, Egypt, July 20–22, 2020. Springer, Heidelberg, Germany. 1

LM08. Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 37–54, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany. 1.2, 5, 5

LTT20. Zi-Yuan Liu, Yi-Fan Tseng, and Raylin Tso. Cryptanalysis of a round optimal lattice-based multisignature scheme. Cryptology ePrint Archive, Report 2020/1172, 2020. https://eprint.iacr.org/2020/1172. 1

McD89. Colin McDiarmid. On the method of bounded differences. In Johannes Siemons, editor, *Surveys in Combinatorics, 1989: Invited Papers at the Twelfth British Combinatorial Conference*, volume 141 of *London Mathematical Society Lecture Note Series*, pages 148–188, Norwich, UK, July 3–7 1989. Cambridge University Press. 2

Mic07. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *computatinal complexity*, 16(4):365–411, December 2007. 1

MJ19. Changshe Ma and Mei Jiang. Practical lattice-based multisignature schemes for blockchains. *IEEE Access*, 7:179765–179778, 2019. 1

MOR01. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 245–254, Philadelphia, PA, USA, November 5–8, 2001. ACM Press. 1

PD20. Chunyan Peng and Xiujuan Du. New lattice-based digital multi-signature scheme. In *6th International Conference of Pioneering Computer Scientists, Engineers and Educators*, volume 1258 of *CCIS*, pages 129–137, Taiyuan, China, September 2020. Springer, Heidelberg, Germany. 1

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany. 6.1

Sho94. Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994. 1

A Concrete Parameters

We used a script[19] to find concrete parameters that allow for instantiating Chipmunk based on a hard ring-SIS problem. We have used a fixed ring dimension $n = 512$. A selection of possible parameter choices is given in Table 8.

Parameter Sets					KOTS Parameters					HVC Parameters			**Agg. Sig. Size**
λ	τ	ρ	α_w	χ	α_H	φ	γ	β_σ	q'	η	β_{agg}	q	(Kilobytes)
112	21	1024	16	12	37	13	6	761464	3115009	29	24750	202753	118 KB
		8192	16	12	37	16	6	2650762	10684417	49	118278	962561	136 KB
		131072	16	12	37	13	7	8649632	34676737	98	946220	7591937	159 KB
	23	1024	16	12	37	13	6	761464	3115009	29	24797	202753	128 KB
		8192	16	12	37	16	6	2650762	10684417	49	118506	962561	147 KB
		131072	16	12	37	13	7	8649632	34676737	98	948044	7591937	172 KB
	24	1024	16	12	37	13	6	761464	3115009	29	24820	202753	133 KB
		8192	16	12	37	16	6	2650762	10684417	49	118613	962561	153 KB
		131072	16	12	37	13	7	8649632	34676737	98	948899	7591937	179 KB
	26	1024	16	12	37	13	6	761464	3115009	29	24862	202753	143 KB
		8192	16	12	37	16	6	2650762	10684417	49	118814	962561	164 KB
		131072	16	12	37	13	7	8649632	34676737	98	950510	7616513	192 KB
128	21	1024	19	14	44	9	7	685898	2836481	31	28830	249857	120 KB
		8192	19	14	44	8	8	1730419	7026689	12	31766	270337	168 KB
		131072	19	14	44	9	8	7786884	31221761	17	180007	1454081	197 KB
	23	1024	19	14	44	9	7	685898	2836481	31	28886	249857	129 KB
		8192	19	14	44	8	8	1730419	7026689	12	31827	270337	182 KB
		131072	19	14	44	9	8	7786884	31221761	17	180351	1454081	214 KB
	24	1024	19	14	44	9	7	685898	2836481	31	28912	249857	134 KB
		8192	19	14	44	8	8	1730419	7026689	12	31855	270337	189 KB
		131072	19	14	44	9	8	7786884	31221761	17	180512	1454081	222 KB
	26	1024	19	14	44	9	7	685898	2836481	31	28961	249857	144 KB
		8192	19	14	44	8	8	1730419	7026689	12	31909	270337	203 KB
		131072	19	14	44	9	8	7786884	31221761	17	180816	1454081	238 KB

Table 8: Parameter sets for Chipmunk for a fixed ring dimension $n = 512$.

[19] https://github.com/GottfriedHerold/Chipmunk