WAVE

Round 1 Submission

Gustavo Banegas
Qualcomm, France

Kévin Carrier
CY Cergy-Paris University, France

André Chailloux
Inria, France

Alain Couvreur
Inria, and Laboratoire d'Informatique de l'École polytechnique (LIX),
UMR 7161, CNRS, École polytechnique, Insitut Polytechnique de Paris, Palaiseau, France

Thomas Debris-Alazard
Inria, and Laboratoire d'Informatique de l'École polytechnique (LIX),
UMR 7161, CNRS, École polytechnique, Insitut Polytechnique de Paris, Palaiseau, France

Philippe Gaborit
University of Limoges, France

Pierre Karpman
Université Grenoble Alpes, France

Johanna Loyer
Inria, France

Ruben Niederhagen
Academia Sinica, Taiwan, and University of Southern Denmark, Denmark

Nicolas Sendrier
Inria, France

Benjamin Smith
Inria, and Laboratoire d'Informatique de l'École polytechnique (LIX),
UMR 7161, CNRS, École polytechnique, Insitut Polytechnique de Paris, Palaiseau, France

Jean-Pierre Tillich
Inria, France

Contact: wave-contact@inria.fr

Coordinator: thomas.debris@inria.fr

Website: https://wave-sign.org

Version: 1

Release Date: June 1, 2023

Contents

Notation and Conventions

$x := y$	x is defined to be equal to y
$\#S$	Cardinality of the finite set S
$\mathbb{Z}$	The ring of integers
$[a, b]$	$[a, b] := \{i \in \mathbb{Z} \mid a \leq i \leq b\}$
$[a, b)$	$[a, b) := \{i \in \mathbb{Z} \mid a \leq i < b\}$
$\mathbb{F}_q$	The q-ary finite field ($q = 3$ in all of our concrete parameter sets)
$\mathbf{x} \in \mathbb{F}_q^n$	$\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_q^n$, vectors *in row notation* generally use bold letters
$\mathbf{x}(i)$	Alternative notation for x_i, convenient when not a single letter, *e.g.* $\mathbf{e}_V(i)$
$\mathbf{x}_{\mathcal{J}}$	$(x_i)_{i \in \mathcal{J}}$ for $\mathcal{J} \subseteq [0, n)$
$(\mathbf{x} \parallel \mathbf{y})$	Concatenation of vectors $\mathbf{x}$ and $\mathbf{y}$
$\mathrm{Supp}(\mathbf{x})$	Support of $\mathbf{x}$, the set $\{i \in [0, n),\ x_i \neq 0\}$
$\lvert\mathbf{x}\rvert$	Hamming weight of $\mathbf{x} \in \mathbb{F}_q^n$, $\#\,\mathrm{Supp}(\mathbf{x})$
$a \cdot \mathbf{x}$	Scalar product $a\,\mathbf{x} := (ax_i)_{0 \leq i < n}$ for $a \in \mathbb{F}_q$ and $\mathbf{x} \in \mathbb{F}_q^n$
$\mathbf{x} \star \mathbf{y}$	Component-wise product $\mathbf{x} \star \mathbf{y} := (x_i y_i)_{0 \leq i < n}$ for $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$
$\langle \mathbf{x}, \mathbf{y} \rangle$	Inner product $\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{i=1}^{n} x_i y_i \in \mathbb{F}_q$ for $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{F}_q^n$
$\mathbf{M} \in \mathbb{F}_q^{r \times n}$	$(M_{i,j})_{0 \leq i < r, 0 \leq j < n}$, $r \times n$ matrix over $\mathbb{F}_q$. Matrices generally use capital bold letters
$\mathbf{M}(i, j)$	Alternative notation for $M_{i,j}$, convenient when not a single letter, *e.g.* $\mathbf{M}_V(i, j)$
$\mathrm{row}(\mathbf{M}, i)$	i-th row of $\mathbf{M}$
$\mathrm{col}(\mathbf{M}, i)$	i-th column of $\mathbf{M}$
M_i	Alternative notation for $\mathrm{row}(\mathbf{M}, i)$
$\mathbf{M}_{\mathcal{J}}$	$(M_{i,j})_{0 \leq i < r, j \in \mathcal{J}}$ for $\mathbf{M} \in \mathbb{F}_q^{r \times n}$ and $\mathcal{J} \subseteq [0, n)$
$\langle \mathbf{M} \rangle$	Raw span of $\mathbf{M}$
$\mathbf{x} \star \mathbf{M}$	Row-wise star product, $\mathbf{x} \star \mathbf{M} := (x_j M_{i,j})_{0 \leq i < r, 0 \leq j < n}$
$\mathbf{M}^\top$	Transposition of $\mathbf{M}$
$\mathfrak{S}_n$	Group of permutations of $[0, n)$
$\mathbf{x}^\pi$	$\mathbf{x}^\pi := (x_{\pi(i)})_{i \leq 0 < n}$, for $\mathbf{x} \in \mathbb{F}_q^n$ and $\pi \in \mathfrak{S}_n$
$\mathbf{M}^\pi$	$\mathbf{M}^\pi := (M_{i,\pi(j)})_{0 \leq i < r, 0 \leq j < n}$ for $\mathbf{M} \in \mathbb{F}_q^{r \times n}$ and $\pi \in \mathfrak{S}_n$
$x \xleftarrow{\mathcal{D}} X$	Variable x sampled from the set X according to the distribution $\mathcal{D}$
$x \xleftarrow{\$} X$	Variable x sampled uniformly at random from the finite set X

Information set

An *information set* for a matrix $\mathbf{M} \in \mathbb{F}_q^{(r+g)\times n}$ (where $g \geq 0$) is a set $\mathcal{J} \subseteq [0, n)$ of cardinality r such that $\mathbf{M}_{\mathcal{J}} \in \mathbb{F}_q^{(r+g)\times r}$ has full rank.

Systematic form

A matrix $\mathbf{M} \in \mathbb{F}_q^{r\times n}$ is in *systematic form* if $\mathbf{M} = (\mathbf{Id}_r \mid \mathbf{R})$ where $\mathbf{Id}_r$ is the $r \times r$ identity matrix.

Linear code

An $[n, k]_q$-code $\mathcal{C}$ is defined to be a k-dimensional subspace of $\mathbb{F}_q^n$.

Generator matrix

A *generator matrix* for a linear $[n, k]_q$-code $\mathcal{C}$ is a matrix $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ whose rows form a basis of $\mathcal{C}$; that is,

$$\mathcal{C} = \left\{ \mathbf{xG} : \mathbf{x} \in \mathbb{F}_q^k \right\} .$$

Dual code

The *dual* of an $[n, k]_q$-code $\mathcal{C}$ is the $[n, n-k]_q$-code defined by

$$\mathcal{C}^\perp := \left\{ \mathbf{c}^\perp \in \mathbb{F}_q^n : \forall \mathbf{c} \in \mathcal{C},\ \langle \mathbf{c}, \mathbf{c}^\perp \rangle = 0 \right\} .$$

Parity-check matrix

A *parity-check matrix* for an $[n, k]_q$-code $\mathcal{C}$ is a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ whose rows form a basis of the dual $\mathcal{C}^\perp$. Note that

$$\mathcal{C} = \left\{ \mathbf{c} \in \mathbb{F}_q^n \ : \ \mathbf{Hc}^\top = \mathbf{0} \right\} .$$

1. Introduction

Wave is a code-based hash-and-sign signature scheme introduced in [DST19]. Wave instantiates the theoretical framework of Gentry, Peikert and Vaikuntanathan [GPV08] using a novel code-based trapdoor: its security is proven to inherit from the hardness of two well-identified problems for which the best known attacks rely on generic decoding algorithms [DST19]. With appropriate parameters, Wave can therefore offer high security against classical and quantum adversaries.

This document specifies the Wave scheme, and proposes highly conservative parameter sets for signatures targeting NIST post-quantum security levels I, III, and V. It also describes a portable C reference implementation for Wave instances with those parameters.

Wave enjoys **short signatures** and **fast verification**, even with conservative parameters. Table 2 gives a preview of our results: Wave signature sizes are highly competitive with structured lattice-based signatures such as Falcon [FHK+17] and CRYSTALS/Dilithium [LDK+20], and signatures can be verified in milliseconds on a PC platform. Wave public keys are generator matrices for random-looking linear codes, so they are decidedly on the large side (especially given our conservative parameter choices): this is the main drawback of Wave. However, in use-cases where large public keys can be stored, Wave can be a strong candidate for high-security quantum-safe signatures.

Table 2. Signature length and verification speed results for Wave instances. Timings count millions of cycles used on average by the (non-optimized) reference implementation, running on an Intel Core i5-1135G7 platform at 2.40GHz. See §5 for more detailed figures.

Wave instance Post-quantum security target	Wave822 Level I	Wave1249 Level III	Wave1644 Level V
Signature length[1] (Bytes)	**822**	**1249**	**1644**
Public key size (Bytes)	3 677 390	7 867 598	13 632 308
Key generation (MCycles)	14 468	47 222	108 642
Signing (MCycles)	1 161	3 507	7 397
Verification (MCycles)	**205.8**	**464.1**	**813.3**
Verification[2] (MCycles)	**1.231**	**2.580**	**4.329**

[1] Signatures are compressed to variable-length byte arrays, which may in fact be shorter than this bound.

[2] Verification where the public key is pre-loaded in bitsliced format, and does not require conversion from the transport format: see §5 for details.

We give a brief high-level overview of the scheme in §2, and explain the design rationale in §3. We formally specify the scheme in §4, present performance results for the reference implementation in §5, and link to Known Answer Tests in §6. To justify the security of our proposed instances, we summarize the security proof for Wave in §7 and survey the best known attacks in §8.

2. The Wave Signature Scheme – Sketch

Wave is based on the GPV framework [GPV08]; as such, it is built on a trapdoor function. The Wave trapdoor, detailed in §3, is based on *permuted generalized* $(U|U+V)$*-codes*.

Definition 1. *Let* n, k_U, *and* k_V *be integers with* n *even and* $k_U, k_V \leq n/2$. *Let* U *be an* $[n/2, k_U]_q$*-code with generator (resp. parity-check) matrix* $\mathbf{G}_U$ *(resp.* $\mathbf{H}_U$*), and let* V *be an* $[n/2, k_V]_q$*-code with generator (resp. parity-check) matrix* $\mathbf{G}_V$ *(resp.* $\mathbf{H}_V$*). Let* π *be a permutation in* $\mathfrak{S}_n$*, and let* $\mathbf{b}$ *and* $\mathbf{c}$ *be vectors in* $\mathbb{F}_q^{n/2}$ *with* $\mathbf{c}(i) \neq 0$ *for all all* $i \in [0, n/2)$.

The permuted generalized $(U|U+V)$*-code associated to* $(U, V, \pi, \mathbf{b}, \mathbf{c})$ *is the* $[n, k_U + k_V]_q$*-code admitting the generator matrix* $\mathbf{G}$ *and parity-check matrix* $\mathbf{H}$ *defined by*

$$\mathbf{G} = \left(\begin{array}{c|c} \mathbf{G}_U & \mathbf{c} \star \mathbf{G}_U \\ \hline \mathbf{b} \star \mathbf{G}_V & \mathbf{d} \star \mathbf{G}_V \end{array}\right)^{\pi} \quad \textit{and} \quad \mathbf{H} = \left(\begin{array}{c|c} \mathbf{d} \star \mathbf{H}_U & -\mathbf{b} \star \mathbf{H}_U \\ \hline -\mathbf{c} \star \mathbf{H}_V & \mathbf{H}_V \end{array}\right)^{\pi}$$

where $\mathbf{d} := \mathbf{1} + \mathbf{b} \star \mathbf{c}$.

A Wave public key is a parity-check matrix, in systematic form, of a permuted generalized $(U|U+V)$-code over $\mathbb{F}_3$. The associated private key is the underlying structure: the codes U and V, vectors $\mathbf{b}$ and $\mathbf{c}$, and permutation π. Table 3 sketches the Wave signature scheme; the full specification begins in §4.

Table 3. Sketch of the Wave signature scheme.

System parameters:

field size	sec. level	codelength	weight	U-dimension	V-dimension	dimension
$q = 3$	λ	n	w	k_U	k_V	$k := k_U + k_V$

Private key: $(U, V, \pi, \mathbf{b}, \mathbf{c})$ defining a permuted generalized $(U|U+V)$-code.

Public key: $\mathbf{R} \in \mathbb{F}_3^{(n-k)\times k}$ such that $(\mathbf{Id}_{n-k} \mid \mathbf{R})$ is a parity-check matrix of the permuted generalized $(U|U+V)$-code associated to the secret key.

Signature on a message $\mathbf{m}$ under a public key $\mathbf{R}$: $\sigma = (\mathtt{salt}, \mathbf{e}) \in \mathbb{F}_2^{2\lambda} \times \mathbb{F}_3^k$ such that

$$|\mathbf{e}| + \left|\mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) - \mathbf{e}\mathbf{R}^\top\right| = w\,. \tag{1}$$

Verification of a signature $\sigma = (\mathtt{salt}, \mathbf{e})$ on a message $\mathbf{m}$ under a public key $\mathbf{R}$: Accept if (and only if) Equation (1) holds.

The binary vector $\mathtt{salt}$ in the signature is a (random) salt required by the security proof. Given a signature $(\mathtt{salt}, \mathbf{e})$ on $\mathbf{m}$ under $\mathbf{R}$, the vector $\mathbf{x} := \big(\mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) - \mathbf{e}\mathbf{R}^\top, \mathbf{e}\big)$ is in fact the unique vector of Hamming weight w satisfying $\mathbf{x}(\mathbf{Id}_{n-k} \mid \mathbf{R})^\top = \mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt})$ and $\mathbf{x}_{[n-k,n)} = \mathbf{e}$. The original description of Wave [DST19] took $(\mathtt{salt}, \mathbf{x})$ to be the signature. Here, however, we follow the approach of [BDNS21]: $\mathbf{x}$ can be immediately recovered from $\mathbf{m}$, $\mathtt{salt}$, and $\mathbf{e}$ (by definition), so $(\mathtt{salt}, \mathbf{e})$ can serve as the signature, thus reducing the signature size from 2λ bits and n trits to 2λ bits and k trits.

3. Design Rationale: The Wave Trapdoor

The design of Wave follows the "hash and sign" approach *à la* GPV [GPV08] in the same spirit as some lattice-based schemes including Falcon [FHK+17], but in an error-correcting code-based context. While the trapdoors in [GPV08] and [FHK+17] rely on short bases of lattices, the Wave trapdoor is based on permuted generalized $(U|U+V)$-codes (as in Definition 1).

3.1. Weight and the general decoding problem.

Wave security inherits from the hardness of the Decoding Problem (DP): given an $[n,k]_q$-code $\mathcal{C}$ (a subspace of $\mathbb{F}_q^n$ of dimension k), a distance w, and a point $\mathbf{y}$ in the ambient space $\mathbb{F}_q^n$, the aim is to find a $\mathbf{x}$ in $\mathcal{C}$ at Hamming distance w from $\mathbf{y}$, that is, differing from $\mathbf{y}$ on *exactly* w coordinates.

After sixty years of research on DP, the best approach when the $\mathcal{C}$ has no special structure—and in particular, when it is *random*—is to take advantage of linearity: $\mathcal{C}$ is a vector space of dimension k, so one can simply compute $\mathbf{x} \in \mathcal{C}$ by fixing k coordinates. If $\mathbf{y}$ is uniformly distributed over $\mathbb{F}_q^n$, then the resulting $\mathbf{x}$ has $\mathbf{x}(i) - \mathbf{y}(i)$ "controlled" on exactly k coordinates, while the remaining $n-k$ coordinates are uniformly distributed. In particular, there will typically be $(n-k)(q-1)/q$ non-zero coordinates among these $n-k$ coordinates. Depending on the decoding problem to be solved, "close" (small w) or "far away" (large w), the best strategy is to carefully choose $\mathbf{x}(i)$ on the k controlled coordinates i. If one chooses $j \leq k$ coordinates where $\mathbf{x}(i) \neq \mathbf{y}(i)$, then $|\mathbf{x} - \mathbf{y}|$ is typically $j + \frac{q-1}{q}(n-k)$. We can therefore easily compute codewords at distances in $[\frac{q-1}{q}(n-k), k + \frac{q-1}{q}(n-k)]$. Any chosen distance w outside this interval is unlikely to be reached, so the procedure must be repeated a prohibitive number of times for any probability of success.

Figure 1 illustrates the situation. Surprisingly, no known algorithm can solve DP for random codes in polynomial time (in w) outside the "easy" interval in the middle.

Figure 1. Hardness (with respect to w), given a random $[n,k]_q$-code $\mathcal{C} \subseteq \mathbb{F}_q^n$ and $\mathbf{y} \in \mathbb{F}_q^n$, of finding $\mathbf{x} \in \mathcal{C}$ at Hamming distance w from $\mathbf{y}$.

3.2. The Wave trapdoor.

To construct a Wave trapdoor, we sample random codes U and V in $\mathbb{F}_q^{n/2}$ of dimensions k_U and k_V, respectively, a permutation π in $\mathfrak{S}_n$, and vectors $\mathbf{b}$ and $\mathbf{c}$ in $\mathbb{F}_q^{n/2}$ with $\mathbf{c}(i) \neq 0$ for $i \in [0, n/2)$. These are all kept secret, and the permuted generalized $(U|U+V)$-code

$$\mathcal{C} := \left\{ \left((\mathbf{x}_U + \mathbf{b} \star \mathbf{x}_V) \parallel (\mathbf{c} \star \mathbf{x}_U + \mathbf{d} \star \mathbf{x}_V) \right)^{\pi} : \mathbf{x}_U \in U \text{ and } \mathbf{x}_V \in V \right\} \subseteq \mathbb{F}_q^n$$

of length n and dimension $k := k_U + k_V$ is made public (here $\mathbf{d} := \mathbf{1} + \mathbf{b} \star \mathbf{c}$).

Given a uniform random $\mathbf{y}$ in $\mathbb{F}_q^n$, we can find $\mathbf{x} \in \mathcal{C}$ at the chosen distance

$$w := 2k_U + 2\,\frac{q-1}{q}(n/2 - k_U)$$

from $\mathbf{y}$ using the following procedure, *if we know* U, V, π, $\mathbf{b}$, *and* $\mathbf{c}$:

(1) Compute any $\mathbf{x}_V \in V$.

(2) From $\mathbf{y}$ and the knowledge of the secret permutation π, decompose $\mathbf{y}$ as as $(\mathbf{y}_L, \mathbf{y}_R)^\pi$ with $\mathbf{y}_L$ and $\mathbf{y}_R$ in $\mathbb{F}_q^{n/2}$.

(3) *Using the knowledge of* $\mathbf{x}_V$, compute some $\mathbf{x}_U \in U$ such that for k_U of the coordinate indices i, we have

$$\mathbf{x}_U(i) \neq \mathbf{y}_L(i) - \mathbf{b}(i)\mathbf{x}_V(i) \quad \text{and} \quad \mathbf{c}(i)\mathbf{x}_U(i) \neq \mathbf{y}_R(i) - \mathbf{d}(i)\mathbf{x}_V(i) \tag{2}$$

(4) Output $\mathbf{x} := \big((\mathbf{x}_U + \mathbf{b} \star \mathbf{x}_V) \parallel (\mathbf{c} \star \mathbf{x}_U + \mathbf{d} \star \mathbf{x}_V)\big)^\pi \in \mathcal{C}$.

By assumption $\mathbf{c}$ has only non-zero components and $\mathbf{d} = \mathbf{1} + \mathbf{b} \star \mathbf{c}$, so Condition (2) can be satisfied provided $q \geq 3$, which we assume from now on. Furthermore, as $(\mathbf{y}_L, \mathbf{y}_R)$ is uniformly distributed, $\mathbf{y}_L(i) - \mathbf{x}_U(i) - \mathbf{b}(i)\mathbf{x}_V(i)$ and $\mathbf{y}_R(i) - \mathbf{c}(i)\mathbf{x}_U(i) - \mathbf{d}(i)\mathbf{x}_V(i)$ will be uniformly distributed on the other $n/2 - k_U$ coordinates. The Hamming distance is invariant under permutations, so $\big|(\mathbf{y}_L \parallel \mathbf{y}_R)^\pi - \big((\mathbf{x}_U + \mathbf{b} \star \mathbf{x}_V) \parallel (\mathbf{c} \star \mathbf{x}_U + \mathbf{d} \star \mathbf{x}_V)\big)^\pi\big|$ will typically be w, as claimed.

Decoding *without* the secret structure $(U, V, \pi, \mathbf{b}, \mathbf{c})$ of the permuted generalized $(U|U+V)$-code $\mathcal{C}$ is hard, given the discussion above, if the value of w falls outside the "easy" interval $[\frac{q-1}{q}(n-k), k + \frac{q-1}{q}(n-k)]$, where $k = k_U + k_V$. This condition holds when $k_U > k_V$, which is a requirement of Wave.

3.3. **Signing with the trapdoor.** To sign a message $\mathbf{m}$, the signer hashes it (with a random salt `salt`) to a random vector $\mathbf{y} \in \mathbb{F}_q^n$. The trapdoor computes a codeword $\mathbf{x} \in \mathcal{C}$ at a distance of w from $\mathbf{y}$. The signature of $\mathbf{m}$ is represented by $\mathbf{x}$. To verify the signature, we must check that $\mathbf{x}$ belongs to $\mathcal{C}$ and that the Hamming distance between $\mathbf{x}$ and the hash of $\mathbf{m}$ with `salt` is exactly w. If the public code $\mathcal{C}$ has no apparent structure to be exploited—that is, it is indistinguishable from a random code—then any adversary seeking to forge a signature must produce a codeword at distance w from a random target, which has a prohibitive cost because w is outside $[\frac{q-1}{q}(n-k), k + \frac{q-1}{q}(n-k)]$; this ensures the security of the signature scheme. (We give more detailed security analyses in §7 and §8.)

This naive signing algorithm is easily broken: a few signatures $\mathbf{x}$ in the permuted generalized $(U|U+V)$-code are enough to recover the trapdoor and thus break the scheme [DST17, §5.1]. In the same way as for other [GPV08]-like schemes, security requires the distribution of signatures to be independent of the trapdoor, so that signatures cannot leak any information on the secret. In particular, $\mathbf{x}_V$ and $\mathbf{x}_U$ must be carefully sampled.

In [DST19], a safe distribution of signatures was achieved through rejection sampling; but this can be problematic for implementation safety and efficiency, especially when constant-time signing is required. We will see that by carefully choosing some internal distributions we can achieve secure signature distributions, and thus avoid rejection sampling entirely without impacting security.

4. Specifications

The specification of an instance of Wave, for any given security level λ, requires the following parameters:

- The main parameters n, k, w,
- additional parameters k_U, k_V, g (used for key generation and signing),
- internal discrete distributions $\mathcal{D}_V$ on $[0, k_V - g]$ and $\mathcal{D}_U(t)$ on $[0, t]$, for t in $[0, n/2]$,
- a decoder output support $\mathsf{Accept} \subseteq [0, n/2] \times [0, w/2]$, and
- a cryptographic hash function $\mathsf{Hash} : \{0,1\}^* \longrightarrow \mathbb{F}_3^{n-k}$.

Table 4 gives the values of these parameters in the Wave instances proposed for security levels I, III, and V.

Table 4. Wave system parameters, and their values for proposed instances. The corresponding internal distributions $\mathcal{D}_V$ and $\mathcal{D}_U$ and decoder output support Accept are described in §7, and are provided as precomputed values in the reference implementation. The hash function is defined in Appendix A.

Instance		*Main parameters*			*Additional parameters*		
		Code length	Code dimension	Weight	U-dim.	V-dim.	gap
	λ	n	k	w	k_U	k_V	g
Wave822 (Level I)	128	8 576	4 288	7 668	2 966	1 322	40
Wave1249 (Level III)	192	12 544	6 272	11 226	4 335	1 937	40
Wave1644 (Level V)	256	16 512	8 256	14 784	5 704	2 552	40

A full implementation of an instance of Wave requires several algorithmic choices which affect efficiency and security (*e.g.* ensuring the constant-time property). For our reference implementation, they are given in appendix:

- The hash function in §A,
- bit-sliced ternary arithmetic in §B.1,
- sampling ternary vectors in §B.2,
- sampling and applying permutations in §B.3,
- master key for generating and permuting secret matrices §B.4,
- Gaussian elimination variants in §B.5.

The reference implementation prioritises **constant-time key generation** and **constant-time signing**. Other choices are possible, depending on the context and security requirements. The complexity bottleneck of both key generation and signing is (by far) Gaussian elimination; relaxing the constant-time constraint may significantly improve signing time.

4.1. Useful Algorithms: Gaussian Elimination and Variants.

Gaussian elimination on an $r \times n$ full-rank matrix $\mathbf{A}$ produces another matrix $\mathbf{A}'$ whose rows span the same vector space and which contains an $r \times r$ identity sub-matrix, in the columns indexed by an information set $J \subseteq [0, n)$ of size r. This set corresponds to the "pivot" positions.

We will define three Gaussian elimination algorithms: *Systematic*, *Partial*, and *Extended*. Appendix B.5.1 proposes constant-time versions of these algorithms.

Systematic Gaussian elimination. Algorithm 1 defines Gaussian elimination on a matrix $\mathbf{A} \in \mathbb{F}_3^{r\times n}$ permuted with $\sigma \in \mathfrak{S}_n$, using the columns of $\mathbf{A}^\sigma$ as pivot positions and "pushing" failing pivot positions to the right. A constant-time version appears in Appendix B.5.1.

Algorithm 1 SystGaussElim – Systematic Gaussian Elimination

Input: $\mathbf{A} \in \mathbb{F}_3^{r\times n}$ and $\sigma \in \mathfrak{S}_n$.
Output: $\mathbf{A}_{\text{syst}} \in \mathbb{F}_3^{r\times n}$ and $\pi \in \mathfrak{S}_n$.
Requirements:
(1) $\langle \mathbf{A}^\pi \rangle = \langle \mathbf{A}_{\text{syst}} \rangle$,
(2) $\mathbf{A}_{\text{syst}} = (\mathbf{Id}_r \mid \mathbf{B})$,
(3) π is "close" to σ, and

$$\pi(i) = \sigma(\ell_i) \text{ for all } i \in [0,r), \text{ where } \ell_i = \min\left\{ j \in [0,n) \mid \operatorname{rank}\left(\mathbf{A}^\sigma_{[0,j)}\right) = i \right\},$$
$$\pi(i) = \sigma(\ell_i) \text{ for all } i \in [r,n), \text{ where } \ell_i = \min\left([0,n) \setminus \{\sigma^{-1} \circ \pi(j),\ j \in [0,i)\} \right).$$

Remark 1. *Note that, over $\mathbb{F}_3$, the case $\pi = \sigma$ in Algorithm 1 happens 56% of the time.*

Partial Gaussian elimination. Algorithm 2 defines Gaussian elimination $\mathbf{A} \in \mathbb{F}_3^{r\times n}$ but stopping g steps before the end, that is, after $k - g$ pivots. A constant-time version appears in Appendix B.5.1.

Algorithm 2 PartialGaussElim – Partial Gaussian Elimination

Input: $\mathbf{A} \in \mathbb{F}_3^{r\times n}$ and $g \leq r$ be an integer.
Output: $\mathbf{A}_{\text{partSyst}} \in \mathbb{F}_3^{r\times n}$ or $\perp$.
Requirements:
(1) Output is $\perp$ if and only if $\operatorname{rank}(\mathbf{A}_{[0,r-g)}) < r - g$,
(2) $\langle \mathbf{A}_{\text{partSyst}} \rangle = \langle \mathbf{A} \rangle$,
(3) $\mathbf{A}_{\text{partSyst}} = \left(\begin{array}{c|c} \mathbf{Id}_{r-g} & \\ \hline \mathbf{0} & \end{array} \quad \mathbf{R} \quad \right)$.

Extended Gaussian elimination. The Gaussian elimination algorithms above are not sufficient for Wave signing. We also need the "extended" Gaussian elimination defined by Algorithm 3, which outputs a matrix in extended systematic form.

Definition 2 (Extended systematic form)**.** *A matrix $\mathbf{A} \in \mathbb{F}_3^{r\times n}$ is in* extended systematic form *if*

$$\text{for all } i \in [0,r)\,, \begin{cases} \mathbf{A}(i,i) \neq 0 \Longrightarrow \mathbf{A}(i,i) = 1 \text{ and } |\operatorname{col}(\mathbf{A},i)| = 1\,, \\ \mathbf{A}(i,i) = 0 \Longrightarrow |\operatorname{row}(\mathbf{A},i)| = 0\,. \end{cases} \tag{3}$$

We propose a constant-time version of Algorithm 3 in §B.5.2. Basically, Algorithm 3, on input $(\mathbf{A}, g)$ with $\mathbf{A} \in \mathbb{F}_3^{r\times n}$, tries to compute an extended systematic form by first adding g extra (zero) rows and then performing a Gaussian elimination. It succeeds if and only if the first $r + g$ columns of this new matrix have the same rank as $\mathbf{A}$.

Algorithm 3 ExtGaussElim $-$ Extended Gaussian Elimination

Input: $\mathbf{A} \in \mathbb{F}_3^{r\times n}$ and $g \leq (n-r)$ be an integer.
Output: $\mathbf{A}_{\text{extSyst}} \in \mathbb{F}_3^{(r+g)\times n}$ or $\perp$.
Requirements:

(1) Output is $\perp$ if and only if $\text{rank}(\mathbf{A}_{[0,r)}) < r$,
(2) $\langle \mathbf{A}_{\text{extSyst}} \rangle = \langle \mathbf{A} \rangle$,
(3) $\mathbf{A}_{\text{extSyst}}$ is in extended systematic form.

4.2. **Wave Key Generation.** The secret key in Wave is defined to be the underlying structure of a permuted generalized $(U|U+V)$-code while the public key is—up to some transformation, oblivious of the secret—a parity-check matrix $(\mathbf{Id}_{n-k} \mid \mathbf{R})$ for this code. To improve verification speed we follow [BDNS21], taking as the public key a matrix $\mathbf{M}(\mathbf{R}) \in \mathbb{F}_3^{k\times(n-k)}$ that can be publicly computed from $\mathbf{R} \in \mathbb{F}_3^{(n-k)\times k}$ (and reciprocally) as in Definition 3.

Definition 3. *If* $\mathbf{R} \in \mathbb{F}_3^{(n-k)\times k}$*, then we define* $\mathbf{M}(\mathbf{R})$ *to be the matrix in* $\mathbb{F}_3^{k\times(n-k)}$ *such that*

$$\left.\begin{aligned} \text{row}(\mathbf{M}(\mathbf{R}), 2i) &:= \text{col}(\mathbf{R}, 2i) + \text{col}(\mathbf{R}, 2i+1) \\ \text{row}(\mathbf{M}(\mathbf{R}), 2i+1) &:= \text{col}(\mathbf{R}, 2i) - \text{col}(\mathbf{R}, 2i+1) \end{aligned}\right\} \text{ for } 0 \leq i < \frac{k-1}{2}$$

and if k *is odd,*

$$\text{row}(\mathbf{M}(\mathbf{R}), k-1) := -\,\text{col}(\mathbf{R}, k-1)\,.$$

Algorithm 4 Key Generation

Output: $\begin{cases} \mathsf{pk} = \mathbf{M} \in \mathbb{F}_3^{k\times(n-k)}, \\ \mathsf{sk} = (\mathbf{H}_U, \mathbf{G}_V, \pi, \mathbf{b}, \mathbf{c}) \in \mathbb{F}_3^{(n/2-k_U)\times n/2} \times \mathbb{F}_3^{k_V\times n/2} \times \mathfrak{S}_n \times \mathbb{F}_3^{n/2} \times (\mathbb{F}_3\backslash\{0\})^{n/2}. \end{cases}$

$\mathbf{G}_V \xleftarrow{\$} \mathbb{F}_3^{k_V\times n/2}$
$\mathbf{H}_V \leftarrow \mathsf{Orthogonal}(\mathbf{G}_V)$ ▷ $\mathbf{H}_V \in \mathbb{F}_3^{(n/2-k_V)\times n/2}$ of full rank such that $\mathbf{G}_V\mathbf{H}_V^\top = 0$
$\mathbf{H}_U \xleftarrow{\$} \mathbb{F}_3^{(n/2-k_U)\times n/2}$
$\mathbf{b} \xleftarrow{\$} \mathbb{F}_3^{n/2}$
$\mathbf{c} \xleftarrow{\$} (\mathbb{F}_3 \setminus \{0\})^{n/2}$ ▷ Coordinates of $\mathbf{c}$ are non-zero
$\mathbf{d} \leftarrow \mathbf{1} + \mathbf{b} \star \mathbf{c}$
$\mathbf{H} \leftarrow \left(\begin{array}{c|c} \mathbf{d} \star \mathbf{H}_U & -\mathbf{b} \star \mathbf{H}_U \\ \hline -\mathbf{c} \star \mathbf{H}_V & \mathbf{H}_V \end{array}\right)$
$\sigma \xleftarrow{\$} \mathfrak{S}_n$
$((\mathbf{Id}_k \mid \mathbf{R}), \pi) \leftarrow \mathsf{SystGaussElim}(\mathbf{H}, \sigma)$ ▷ Algorithm 1, π may differ from σ
$\mathbf{M} \leftarrow \mathbf{M}(\mathbf{R})$ ▷ As in Definition 3
return $(\mathsf{pk} = \mathbf{M}, \mathsf{sk} = (\mathbf{H}_U, \mathbf{G}_V, \pi, \mathbf{b}, \mathbf{c}))$

About the secret key. The secret key is defined to be $\mathsf{sk} = (\mathbf{H}_U, \mathbf{G}_V, \pi, \mathbf{b}, \mathbf{c})$. In practice, we sample the random matrices $\mathbf{H}_U$ and $\mathbf{G}_V$ using pseudorandom generator seeded with a *master key* mk, so we can store the much more compact $(\mathsf{mk}, \pi, \mathbf{b}, \mathbf{c})$ as the secret key and regenerate $\mathbf{H}_U$ and $\mathbf{G}_V$ as needed. We sample the matrices using an XOF f_{mk} parametrized by mk: for example, $f_{\mathsf{mk}}(\texttt{"U"} \parallel i)$ gives the i-th column of $\mathbf{H}_U$ (including the constant "U" provides domain separation). More precisely, we propose using $\texttt{SHAKE256}(\mathsf{mk} \parallel \cdot)$ for f_{mk}.

4.3. **Wave Signature.** Wave signing is built around a *decoder* algorithm, for which there are multiple equivalent descriptions; here, we describe it in terms of noisy codewords. There are two steps. First, a decoding relative to the code V (Decode_V), producing an error pattern $\mathbf{e}_V$; then, a decoding relative to the code U and dependent on $\mathbf{e}_V$ (Decode_U), producing an error pattern $\mathbf{e}_U$. These error patterns are combined to produce the signature.

Algorithm 5 Wave Signature

Input: a message $\mathbf{m}$, $\mathsf{sk} = (\mathbf{H}_U, \mathbf{G}_V, \pi, \mathbf{b}, \mathbf{c})$
Output: $(\mathbf{s}, \mathtt{salt})$

repeat
- $\mathtt{salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$ ▷ $\lambda \in \{128, 192, 256\}$ denotes the security level
- $\mathbf{x} \leftarrow \mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt})$ ▷ $\mathbf{x} \in \mathbb{F}_3^{n-k}$
- $\mathbf{y} = (\mathbf{y}_L \parallel \mathbf{y}_R) \leftarrow (\mathbf{x} \parallel \mathbf{0}^k)^{\pi^{-1}}$ ▷ $\mathbf{y}_L, \mathbf{y}_R \in \mathbb{F}_3^{n/2}$
- $\mathbf{y}_V \leftarrow \mathbf{y}_R - \mathbf{c} \star \mathbf{y}_L$
- $\mathbf{e}_V \leftarrow \mathsf{Decode}_V(\mathbf{y}_V, \mathbf{G}_V)$ ▷ Algorithm 6
- $\mathbf{y}_U \leftarrow \mathbf{y}_L - \mathbf{b} \star \mathbf{y}_V$ ▷ simplification of $\mathbf{y}_U = (\mathbf{1} + \mathbf{c} \star \mathbf{b}) \star \mathbf{y}_L - \mathbf{b} \star \mathbf{y}_R$
- $\mathbf{e}_U \leftarrow \mathsf{Decode}_U(\mathbf{y}_U, \mathbf{e}_V, \mathbf{b}, \mathbf{c}, \mathbf{H}_U)$ ▷ Algorithm 7
- $\mathbf{e}_L \leftarrow \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$
- $\mathbf{e}_R \leftarrow \mathbf{c} \star \mathbf{e}_L + \mathbf{e}_V$ ▷ simplification of $\mathbf{e}_R = \mathbf{c} \star \mathbf{e}_U + (\mathbf{1} + \mathbf{b} \star \mathbf{c}) \star \mathbf{e}_V$

until $(|\mathbf{e}_V|, n/2 - w + |\mathbf{e}_L \star \mathbf{e}_R|) \in \mathsf{Accept}$
$\mathbf{e} \leftarrow (\mathbf{e}_L \parallel \mathbf{e}_R)^{\pi}$
$\mathbf{s} \leftarrow \mathbf{e}_{[n-k,n)}$
return $(\mathbf{s}, \mathtt{salt})$

Remark 2. *The signatures such that* $(|\mathbf{e}_V|, n/2 - w + |\mathbf{e}_L \star \mathbf{e}_R|) \notin \mathsf{Accept}$ *are rejected and the signature process restarts. The probability of rejection is negligible in practice, less than* 2^{-68} *in our reference implementation. However, this rejection sampling is formally required to guarantee that the Rényi divergence between the output distributions of the actual algorithm and its "ideal" version is small enough (see §7).*

Proposition 1. *Let* $(\mathsf{pk} = \mathbf{M}, \mathsf{sk} = (\mathbf{H}_U, \mathbf{G}_V, \pi, \mathbf{b}, \mathbf{c}))$ *be a valid* Wave *keypair. On input* $\mathbf{m}$ *and* sk*, Algorithm 5 outputs* $(\mathbf{s}, \mathtt{salt})$ *such that*

$$|\mathbf{s}| + \left|\mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) - \mathbf{s}\mathbf{R}^\top\right| = w$$

where $\mathbf{R}$ *is such that* $\mathbf{M} = \mathbf{M}(\mathbf{R})$ *(as in Definition 3).*

Proof. Propositions 2 and 3 show that Algorithms 6 (Decode_V) and 7 (Decode_U) output $\mathbf{e}_V$ and $\mathbf{e}_U$ satisfying

(i) $\mathbf{y}_V - \mathbf{e}_V \in \langle \mathbf{G}_V \rangle$,
(ii) $(\mathbf{y}_U - \mathbf{e}_U)\,\mathbf{H}_U^\top = \mathbf{0}_{n/2-k_U}$, and
(iii) $|\mathbf{e}| = |\mathbf{e}_L| + |\mathbf{e}_R| = w$.

In the notation of Algorithm 4,

$$\mathbf{H} = \left(\begin{array}{c|c} \mathbf{d} \star \mathbf{H}_U & -\mathbf{b} \star \mathbf{H}_U \\ \hline -\mathbf{c} \star \mathbf{H}_V & \mathbf{H}_V \end{array}\right)$$

where $\mathbf{d} = \mathbf{1} + \mathbf{b} \star \mathbf{c}$ and $\mathbf{H}_V$ satisfies

$$\mathbf{G}_V \mathbf{H}_V^\top = \mathbf{0}_{k \times (n-k)}\,.$$

Therefore, according to Condition (i),

$$(\mathbf{y}_V - \mathbf{e}_V)\,\mathbf{H}_V^\top = \mathbf{0}_{n/2-k_V}. \tag{4}$$

Now

$$\begin{aligned} &((\mathbf{e}_L \parallel \mathbf{e}_R) - (\mathbf{y}_L \parallel \mathbf{y}_R))\,\mathbf{H}^\top \\ &= \left(((\mathbf{e}_L - \mathbf{y}_L)\star \mathbf{d} - (\mathbf{e}_R - \mathbf{y}_R)\star \mathbf{b})\,\mathbf{H}_U^\top \parallel (-(\mathbf{e}_L - \mathbf{y}_L)\star \mathbf{c} + (\mathbf{e}_R - \mathbf{y}_R))\,\mathbf{H}_V^\top\right), \end{aligned} \tag{5}$$

but

$$\begin{aligned} (\mathbf{e}_L - \mathbf{y}_L)\star \mathbf{d} - (\mathbf{e}_R - \mathbf{y}_R)\star \mathbf{b} &= (\mathbf{e}_L - \mathbf{y}_L)\star(\mathbf{d} - \mathbf{c}\star\mathbf{b}) - (\mathbf{e}_V - \mathbf{y}_V)\star \mathbf{b} \\ &= \mathbf{e}_L - \mathbf{y}_L - (\mathbf{e}_V - \mathbf{y}_V)\star \mathbf{b} \\ &= \mathbf{e}_U - \mathbf{y}_U \end{aligned} \tag{6}$$

and

$$-(\mathbf{e}_L - \mathbf{y}_L)\star \mathbf{c} + \mathbf{e}_R - \mathbf{y}_R = \mathbf{e}_V - \mathbf{y}_V, \tag{7}$$

so plugging Equations (6) and (7) into Equation (5) gives

$$((\mathbf{e}_L \parallel \mathbf{e}_R) - (\mathbf{y}_L \parallel \mathbf{y}_R))\,\mathbf{H}^\top = \left((\mathbf{e}_U - \mathbf{y}_U)\,\mathbf{H}_U^\top \parallel (\mathbf{e}_V - \mathbf{y}_V)\,\mathbf{H}_V^\top\right).$$

Now using Condition (ii) and Equation (4) yields

$$\begin{aligned} ((\mathbf{e}_L \parallel \mathbf{e}_R)^\pi - (\mathbf{y}_L \parallel \mathbf{y}_R)^\pi)\,(\mathbf{H}^\pi)^\top &= ((\mathbf{e}_L \parallel \mathbf{e}_R) - (\mathbf{y}_L \parallel \mathbf{y}_R))\,\mathbf{H}^\top \\ &= \mathbf{0}_{n-k}. \end{aligned} \tag{8}$$

By the definitions of π and $\mathsf{SystGaussElim}$, there exists a non-singular matrix $\mathbf{S} \in \mathbb{F}_3^{(n-k)\times(n-k)}$ corresponding to the Gaussian elimination such that

$$\mathbf{S}\mathbf{H}^\pi = (\mathbf{Id}_{n-k} \mid \mathbf{R}).$$

Therefore Equation (8) becomes

$$\begin{aligned} \mathbf{0}_{n-k} &= ((\mathbf{e}_L \parallel \mathbf{e}_R)^\pi - (\mathbf{y}_L \parallel \mathbf{y}_R)^\pi)\,(\mathbf{S}\mathbf{H}^\pi)^\top \\ &= ((\mathbf{e}_L \parallel \mathbf{e}_R)^\pi - (\mathbf{y}_L \parallel \mathbf{y}_R)^\pi)\,(\mathbf{Id}_{n-k} \mid \mathbf{R})^\top \end{aligned}$$

but since $\mathbf{e} = (\mathbf{e}_L \parallel \mathbf{e}_R)^\pi$ and $(\mathbf{y}_L \parallel \mathbf{y}_R)^\pi = (\mathbf{x} \parallel \mathbf{0}_k)$ we have

$$\begin{aligned} \mathbf{0}_{n-k} &= (\mathbf{e} - (\mathbf{x} \parallel \mathbf{0}_k))\,(\mathbf{Id}_{n-k} \mid \mathbf{R})^\top \\ &= \mathbf{e}_{[0,n-k)} - \mathbf{x} + \mathbf{e}_{[n-k,n)}\mathbf{R}^\top \\ &= \mathbf{e}_{[0,n-k)} - \mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) + \mathbf{s}\mathbf{R}^\top. \end{aligned}$$

Therefore,

$$\begin{aligned} |\mathbf{s}| + \left|\mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) - \mathbf{s}\mathbf{R}^\top\right| &= |\mathbf{s}| + |\mathbf{e}_{[0,n-k)}| \\ &= |\mathbf{e}_{[n-k,n)}| + |\mathbf{e}_{[0,n-k)}| \\ &= |\mathbf{e}| \end{aligned}$$

and we conclude the proof by applying Condition (iii). □

Leakage-free signatures. One of the key properties of Wave is that the vector $\mathbf{e}$ in Instruction 12 of Algorithm 5 has been proven to be uniformly distributed over words of Hamming weight w (see [DST19, §5]). In particular, its distribution is independent of the secrets involved in the signing process: even with the knowledge of the secret key, signatures are indistinguishable from random words of weight w. This property implies that Wave is a hash-and-sign signature scheme immune to leakage attacks. We stress, however, that this property only holds if Wave is properly implemented. Some implementation mistakes, such as incorrect sampling of random data in Decode_U or Decode_V, may produce valid but biased signatures, opening the door to statistical attacks. Some possible biases are described in [DST17, §5.1].

4.3.1. *Decoder for V.* The input to Decode_V in Instruction 6 of Algorithm 5 is a matrix $\mathbf{G}_V \in \mathbb{F}_3^{k_V \times n/2}$ and a word $\mathbf{y}_V \in \mathbb{F}_3^{n/2}$ and the output is a word $\mathbf{e}_V \in \mathbb{F}_3^{n/2}$ such that $\mathbf{y}_V - \mathbf{e}_V \in \langle \mathbf{G}_V \rangle$. It is also required (to avoiding leakage) that over all possible uniformly distributed inputs $(\mathbf{G}_V, \mathbf{y}_V)$, the weight distribution of the output $\mathbf{e}_V$ is close to the distribution $\mathcal{D}$ of $|\mathbf{e}_L + \mathbf{e}_R|$ when $(\mathbf{e}_L \parallel \mathbf{e}_R)$ is uniformly distributed over words of Hamming weight w in $\mathbb{F}_3^n$.

The algorithm Decode_V first splits $[0, n/2)$ in two by choosing uniformly a set $L \subseteq [0, n/2)$ of $k_V - g$ indices[1]. Without loss of generality, given $\mathbf{x} \in \mathbb{F}_3^{n/2}$, we define $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$ by writing $\mathbf{x}$ (up to a permutation) as follows

$$\mathbf{x} = \Big(\overbrace{\mathbf{x}^{(0)}}^{k_V - g} \;\Big|\; \overbrace{\mathbf{x}^{(1)}}^{n/2 - k_V + g} \Big) \tag{9}$$

with the left part $\mathbf{x}^{(0)}$ indexed by L. Decode_V selects a word $\mathbf{e}_V = (\mathbf{e}^{(0)} \parallel \mathbf{e}^{(1)})$ such that $\mathbf{e}^{(1)}$ is uniformly distributed in $\mathbb{F}_3^{n/2-k_V+g}$ and $\mathbf{e}^{(0)}$ is chosen with a weight t sampled from $\mathcal{D}_V$. The weight of $\mathbf{e}^{(1)}$ follows a binomial distribution with parameters $(n/2 - k_V + g, 2/3)$, say $\mathcal{B}_V$. The distribution $\mathcal{D}_V$ is chosen such that $\mathcal{D}_V + \mathcal{B}_V$[2] is close to $\mathcal{D}$.

Algorithm 6 Decode_V

1: **function** $\mathsf{Decode}_V(\mathbf{y}_V, \mathbf{G}_V)$
2: $\quad t \xleftarrow{\mathcal{D}_V} [0, k_V - g]$
3: $\quad$ **repeat**
4: $\quad\quad \pi \xleftarrow{\$} \mathfrak{S}_{n/2}$
5: $\quad\quad \mathbf{y} \leftarrow \mathbf{y}_V^{\pi}$
6: $\quad\quad \mathbf{G} \leftarrow \mathsf{PartialGaussElim}(\mathbf{G}_V^{\pi}, g)$
7: $\quad$ **until** $\mathbf{G} \neq \perp$ $\qquad \triangleright\ \langle \mathbf{G} \rangle = \langle \mathbf{G}_V^{\pi} \rangle,\ \mathbf{G} = \left(\begin{array}{c|c} \mathbf{Id}_{k_V - g} & \mathbf{R} \\ \hline \mathbf{0} & \end{array} \right)$
8: $\quad \mathbf{x} \xleftarrow{\$} (\mathbb{F}_3 \setminus \{0\})^t \times \{0\}^{k_V - g - t} \times \mathbb{F}_3^g$
9: $\quad \mathbf{e} \leftarrow \mathbf{y} + (\mathbf{x} - \mathbf{y}^{(0)})\mathbf{G}$ $\qquad \triangleright\ \mathbf{y} = (\mathbf{y}^{(0)} \parallel \mathbf{y}^{(1)}) \in \mathbb{F}_3^{n/2}$ with $\mathbf{y}^{(0)} \in \mathbb{F}_3^{k_V - g}$
10: $\quad$ **return** $\mathbf{e}_V = \mathbf{e}^{\pi^{-1}}$

Proposition 2. *On input $(\mathbf{y}_V, \mathbf{G}_V)$, the vector $\mathbf{e}_V \in \mathbb{F}_3^{n/2}$ output by Algorithm 6 satisfies*

$$\mathbf{y}_V - \mathbf{e}_V \in \langle \mathbf{G}_V \rangle.$$

Proof. By definition, $\mathbf{y} - \mathbf{e} \in \langle \mathbf{G} \rangle = \langle \mathbf{G}_V^{\pi} \rangle$ in Instruction 9. The result follows on applying π^{-1}. □

[1]The *gap* g is a system parameter such that the submatrix formed by any $k_V - g$ columns of $\mathbf{G}_V$ has full rank with overwhelming probability.

[2]The distribution of $|\mathbf{x} + \mathbf{y}|$ when $\mathbf{x}$ and $\mathbf{y}$ are independently distributed according to $\mathcal{D}_V$ and $\mathcal{B}_V$.

Remark 3. *As it is described, Algorithm 6 samples a random permutation π and implicitly defines a set $L = \pi([0, k_V - g))$. While L must be distributed uniformly, this is not required for π (which must only have L as its first $k_V - g$ entries). In practice (as in §B.3.2), π is sampled so that $\pi([0, k_V - g))$ and $\pi([0, t))$ are uniformly distributed in $[0, n/2)$ and $\pi([0, k_V - g))$, respectively.*

Remark 4. *Instructions 4 and 5 of* Decode_V *sample a permutation and apply it to a vector; this may be implemented as a single subroutine. Using the procedure suggested in §B.3.2, the call would be*

$$(\pi, \mathbf{y}) \leftarrow \mathsf{RandPerm}_V(n/2, k_V - g, t, \mathbf{y}_V)$$

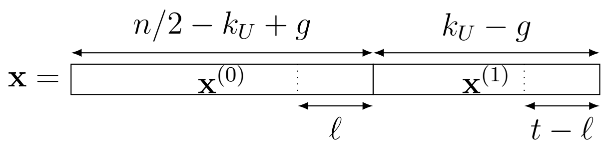
where the arguments $k_V - g$ and t are provided to ensure that both $\pi([0, k_V - g))$ and $\pi([0, t))$ are uniformly distributed (see Algorithm 27).

4.3.2. *Decoder for U.* The input of Decode_U in Instruction 8 of Algorithm 5 is a matrix $\mathbf{H}_U \in \mathbb{F}_3^{(n/2-k_U)\times n/2}$ and a vector $\mathbf{y}_U \in \mathbb{F}_3^{n/2}$, and the output a vector $\mathbf{e}_U \in \mathbb{F}_3^{n/2}$ such that $(\mathbf{y}_U - \mathbf{e}_U)\,\mathbf{H}_U^\top = \mathbf{0}_{n/2-k_U}$. The target distribution for $\mathbf{e}_U$ is conditioned by the result $\mathbf{e}_V$ of the first decoding. It is shown in [DST19, §5] that, conditioned on $t := |\mathbf{e}_V|$, the output $\mathbf{e}_U$ is correctly distributed if and only if the number

$$j = \#\left([0, n/2) \setminus (\mathrm{Supp}(\mathbf{e}_V) \cup \mathrm{Supp}(\mathbf{e}_U))\right)$$

of positions that are simultaneously null in $\mathbf{e}_U$ and $\mathbf{e}_V$ follows a prescribed distribution, dependent on t.

Algorithm 7 (Decode_U) splits $[0, n/2)$ in two by sampling an integer $\ell \in [0, t]$ according to $\mathcal{D}_U(t)$ where $t = |\mathbf{e}_V|$, and then uniformly sampling a set $L \subseteq [0, n/2)$ of $n/2 - k_U + g$ indices containing exactly ℓ elements of $\mathrm{Supp}(\mathbf{e}_V)$. Without loss of generality, given $\mathbf{x} \in \mathbb{F}_3^{n/2}$ we define $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$ by writing $\mathbf{x}$ (up to a permutation) as follows



(10)

with the left part $\mathbf{x}^{(0)}$ being indexed by L. The t positions of $\mathrm{Supp}(\mathbf{e}_V)$ are depicted with ℓ of them being on the left block and the other $t - \ell$ being full right but they could be anywhere within their blocks. The algorithm selects a word $\mathbf{e} = (\mathbf{e}^{(0)} \parallel \mathbf{e}^{(1)})$, which will be equal to $\mathbf{e}_U$ up to some permutation, such that $\mathbf{e}^{(0)}$ is uniformly distributed in $\mathbb{F}_3^{n/2-k_U+g}$ and $\mathbf{e}^{(1)}$ is chosen to reach the final signature weight (see below). The value of j is controlled by ℓ, which is distributed according to $\mathcal{D}_U(t)$. As for V, an appropriate choice of the distribution $\mathcal{D}_U(t)$ allows a distribution of j that is close to the target distribution.

***About the choice of* $\mathbf{e}^{(1)}$.** The choice of $\mathbf{e}^{(1)} \in \mathbb{F}_3^{k_U-g}$ in Instruction 20 of Algorithm 7 is such that both $\mathbf{e}_L = \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$ and $\mathbf{e}_R = \mathbf{c} \star \mathbf{e}_L + \mathbf{e}_V$ (from which a signature is built in Instructions 9-14 of Algorithm 5) have $k_U - g$ non-zero coordinates on $\pi\left([n/2 - k_U + g, n/2)\right)$, as shown in Lemma 3. This will follow from the fact that $\mathbf{e}_U$ and $\mathbf{e}_V$ satisfy

$$\begin{aligned} &\mathbf{e}_U(i) = (\mathbf{c}(i) - \mathbf{b}(i))\,\mathbf{e}_V(i) && \text{if } i \in \pi([n/2 - k_U + g, n/2)) \cap \mathrm{Supp}(\mathbf{e}_V), \\ &\mathbf{e}_U(i) \in \mathbb{F}_3 \setminus \{0\} && \text{if } i \in \pi([n/2 - k_U + g, n/2)) \setminus \mathrm{Supp}(\mathbf{e}_V). \end{aligned} \tag{11}$$

Algorithm 7 Decode_U

function $\mathsf{Decode}_U(\mathbf{y}_U, \mathbf{e}_V, \mathbf{b}, \mathbf{c}, \mathbf{H}_U)$

  $t \leftarrow |\mathbf{e}_V|$

  $\ell \xleftarrow{\mathcal{D}_U^{(t)}} [0, t]$

  repeat

    $\pi \xleftarrow{\$} \{\pi \in \mathfrak{S}_{n/2} \mid \#\, \pi([0, n/2 - k_U + g)) \cap \mathrm{Supp}(\mathbf{e}_V) = \ell\}$

    $\mathbf{y} \leftarrow \mathbf{y}_U^{\pi}$ ▷ $\mathbf{y} \in \mathbb{F}_3^{n/2}$

    $\mathbf{v} \leftarrow ((\mathbf{c} - \mathbf{b}) \star \mathbf{e}_V)^{\pi}$ ▷ $\mathbf{v} \in \mathbb{F}_3^{n/2}$

    $\mathbf{v}^{(0)} \leftarrow \mathbf{v}_{[0, n/2 - k_U + g)}$

    $\mathbf{v}^{(1)} \leftarrow \mathbf{v}_{[n/2 - k_U + g, n/2)}$

    $\mathbf{s} \leftarrow (\mathbf{e}_V \star \mathbf{e}_V)^{\pi}$ ▷ $\mathbf{s} \in \{0, 1\}^{n/2}$

    $\mathbf{s}^{(0)} \leftarrow \mathbf{s}_{[0, n/2 - k_U + g)}$

    $\mathbf{s}^{(1)} \leftarrow \mathbf{s}_{[n/2 - k_U + g, n/2)}$

    $\mathbf{H} \leftarrow \mathsf{ExtGaussElim}(\mathbf{H}_U^{\pi}, g)$ ▷ $\mathbf{H} \in \mathbb{F}_3^{(n/2 - k_U + g) \times n/2}$

  until $\mathbf{H} \neq \perp$

  repeat

    $\mathbf{z}^{(0)} \leftarrow (H_{i,i})_{0 \leq i < n/2 - k_U + g}$

    $\mathbf{e}^{(0)} \xleftarrow{\$} \mathbb{F}_3^{n/2 - k_U + g}$

    $\mathbf{e}^{(0)} \leftarrow (\mathbf{1} - \mathbf{z}^{(0)}) \star \mathbf{e}^{(0)}$

    $\mathbf{e}^{(1)} \xleftarrow{\$} (\mathbb{F}_3 \setminus \{0\})^{k_U - g}$

    $\mathbf{e}^{(1)} \leftarrow \mathbf{v}^{(1)} + (\mathbf{1} - \mathbf{s}^{(1)}) \star \mathbf{e}^{(1)}$

    $\mathbf{e}^{(0)} \leftarrow \mathbf{e}^{(0)} + \left(\mathbf{y} - (\mathbf{e}^{(0)} \parallel \mathbf{e}^{(1)})\right) \mathbf{H}^{\top}$

    $i \leftarrow |\mathbf{s}^{(0)} \star \mathbf{e}^{(0)} - \mathbf{v}^{(0)}|$

    $j \leftarrow n/2 - k_U + g - \ell - |(\mathbf{1} - \mathbf{s}^{(0)}) \star \mathbf{e}^{(0)}|$

  until $(2j + i = n - w)$ ▷ check final weight

  $\mathbf{e} \leftarrow (\mathbf{e}^{(0)} \parallel \mathbf{e}^{(1)})$

  return $\mathbf{e}_U = \mathbf{e}^{\pi^{-1}}$

Proposition 3. *On input* $(\mathbf{y}_U, \mathbf{e}_V, \mathbf{b}, \mathbf{c}, \mathbf{H}_U)$*, Algorithm 7 outputs* $\mathbf{e}_U$ *satisfying*

$$(\mathbf{y}_U - \mathbf{e}_U)\, \mathbf{H}_U^{\top} = \mathbf{0}_{n/2 - k_U} \tag{12}$$

Furthermore, if $\mathbf{e}_L := \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$ *and* $\mathbf{e}_R := \mathbf{c} \star \mathbf{e}_L + \mathbf{e}_V$*, then*

$$|\mathbf{e}_L| + |\mathbf{e}_R| = w. \tag{13}$$

Proof. See Appendix D. □

Remark 5. *As it is described, Algorithm 7 samples an integer* ℓ *according to* $\mathcal{D}_U(t)$*, then, uniformly, a permutation* π *such that* $\# L \cap \mathrm{Supp}(\mathbf{e}_V) = \ell$ *where* $L = \pi\,([0, n/2 - k_U + g))$*. The set* L *must be distributed uniformly with an intersection of size* ℓ *with* $\mathrm{Supp}(\mathbf{e}_V)$*, but uniformity among all permutations is not required for* π *which must only have* L *as its first* $n/2 - k_U + g$ *entries.*

Remark 6. *The $n/2-k_U+g$ leftmost positions, corresponding to L, include an information set (for $\mathbf{H}_U$) plus g additional coordinates*

$$\mathbf{x} = \overbrace{\boxed{\underbrace{\quad\quad}_{g}\ \mathbf{x}^{(0)}\ \underbrace{\quad\quad}_{\ell}}}^{n/2-k_U+g}\overbrace{\boxed{\quad\mathbf{x}^{(1)}\ \underbrace{\quad\quad}_{t-\ell}}}^{k_U-g} \tag{14}$$

The g extra coordinates are depicted on the left here, but they could be anywhere; they are not necessarily disjoint from the ℓ positions of $\mathrm{Supp}(\mathbf{e}_V)$*. Vector operations will use the vectors $\mathbf{z}^{(0)} \in \mathbb{F}_2^{n/2-k_V+g}$ and $\mathbf{s} = (\mathbf{s}^{(0)} \parallel \mathbf{s}^{(1)}) \in \mathbb{F}_2^{n/2}$ as masks to select the various regions.*

Remark 7. *There is a rejection condition $n - 2j - i = w$ to ensure the final weight is correct. This condition is oblivious of private information: its failure doesn't require a new permutation or Gaussian elimination, only fresh values of $\mathbf{e}^{(0)}$ and $\mathbf{e}^{(1)}$.*

Remark 8. *Instructions 5, 6, 7 and 10 sample a permutation and apply it to several vectors. This may be implemented as a single subroutine. Using the procedure suggested in §B.3.3, the call would be*

$$(\pi, \mathbf{x}') \leftarrow \mathsf{RandPerm}_U(n/2, n/2 - k_U + g, \ell, \mathrm{Supp}(\mathbf{e}_V), \mathbf{x})$$

where the vectors $\mathbf{y}_V$, $(\mathbf{c}-\mathbf{b}) \star \mathbf{e}_V$, and $\mathbf{e}_V \star \mathbf{e}_V$ are packed into $\mathbf{x}$ (that is, $\mathbf{x}(i) = (\mathbf{y}_V(i) \parallel (c_i - b_i)\mathbf{e}_V(i) \parallel \mathbf{e}_V(i)^2)$) and the output $\mathbf{x}'$ is unpacked to produce the permuted vectors (see Algorithm 28).

4.4. **Wave Verification.** To verify Wave signatures, we follow the approach of [BDNS21].

Algorithm 8 Verification for Wave signatures.

Input: $\mathbf{m} \in \{0,1\}^*$, $\sigma = (\mathtt{salt}, \mathbf{s}) \in \{0,1\}^{2\lambda} \times \mathbb{F}_3^k$, $\mathbf{M} \in \mathbb{F}_3^{k\times(n-k)}$
Output: True or False

```
x ← Hash(m ∥ salt)
for 0 ≤ i < (k − 1)/2 do                                   ▷ handle entries of s in pairs
    (ŝ(2i), ŝ(2i + 1)) ← (s(2i) + s(2i + 1), s(2i) − s(2i + 1))  ▷ (2i, 2i + 1)-th entries of ŝ
    if ŝ(2i) = 1 then
        x ← x + row(M, 2i)                                 ▷ add 2i-th row of M
    else if ŝ(2i) = 2 then
        x ← x − row(M, 2i)                                 ▷ subtract 2i-th row of M
    if ŝ(2i + 1) = 1 then
        x ← x + row(M, 2i + 1)                             ▷ add (2i + 1)-th row of M
    else if ŝ(2i + 1) = 2 then
        x ← x − row(M, 2i + 1)                             ▷ subtract (2i + 1)-th row of M
if k is odd then                                           ▷ handle last entry if necessary
    if s(k − 1) = 1 then
        x ← x + row(M, k − 1)
    else if s(k − 1) = 2 then
        x ← x − row(M, k − 1)
return |s| + |x| = w
```

Proposition 4. *Given a putative* Wave *signature* $(\mathbf{s}, \mathtt{salt})$ *on a message* $\mathbf{m}$ *under a valid* Wave *public key* $\mathbf{M}$*, Algorithm 8 returns true if and only if*

$$|\mathbf{s}| + \left|\mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) - \mathbf{s}\mathbf{R}^\top\right| = w$$

where $\mathbf{R}$ *is such that* $\mathbf{M} = \mathbf{M}(\mathbf{R})$ *(as in Definition 3).*

Proof. In the notation of Algorithm 8, let $\widehat{\mathbf{s}}$ be defined by

$$\left.\begin{aligned} \widehat{\mathbf{s}}(2i) &:= \mathbf{s}(2i) + \mathbf{s}(2i+1) \\ \widehat{\mathbf{s}}(2i+1) &:= \mathbf{s}(2i) - \mathbf{s}(2i+1) \end{aligned}\right\} \text{ for } 0 \leq i < \frac{k-1}{2}$$

and let $\widehat{\mathbf{s}}(k-1) := \mathbf{s}(k-1)$ if k is odd. At the end of the execution of Algorithm 8, we have

$$\mathbf{x} = \mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) + \widehat{\mathbf{s}}\mathbf{M}. \tag{15}$$

Notice that

$$\mathbf{s}\mathbf{R}^\top = -\widehat{\mathbf{s}}\mathbf{M}. \tag{16}$$

because (from Definition 3)

$$\begin{aligned} &\widehat{\mathbf{s}}(2i) \cdot \mathrm{row}(\mathbf{M}, 2i) + \widehat{\mathbf{s}}(2i+1) \cdot \mathrm{row}(\mathbf{M}, 2i+1) \\ &\qquad = -\left(\mathbf{s}(2i) \cdot \mathrm{col}(\mathbf{R}, 2i) + \mathbf{s}(2i+1) \cdot \mathrm{col}(\mathbf{R}, 2i+1)\right) \end{aligned}$$

for $0 \leq i < (k-1)/2$, and

$$\widehat{\mathbf{s}}(k-1) \cdot \mathrm{row}(\mathbf{M}, k-1) = -\mathbf{s}(k-1) \cdot \mathrm{col}(\mathbf{R}, k-1)$$

if k is odd. Combining Equations (15) and (16) and using Proposition 1 concludes the proof. □

Algorithm 8 implicitly uses Equation (16) to replace the standard Wave verification equation

$$|\mathbf{s}| + \left|\mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) - \mathbf{s}\mathbf{R}^\top\right| = w$$

with the equivalent equation

$$|\mathbf{s}| + |\mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt}) + \widehat{\mathbf{s}}\mathbf{M}| = w\,.$$

The point of verifying with the product $\widehat{\mathbf{s}}\mathbf{M}$ rather than $\mathbf{s}\mathbf{R}^\top$ is explained in [BDNS21]: since the vector $\mathbf{s}$ is supposed to have high weight, the vector $\widehat{\mathbf{s}}$ has many zeroes, and so $\widehat{\mathbf{s}}\mathbf{M}$ is easier to compute than $\mathbf{s}\mathbf{R}^\top$. The coefficients of the vector $\widehat{\mathbf{s}}$ can easily be computed on the fly as the signature vector $\mathbf{s}$ is read (or decompressed).

Indeed, in any given Wave signature verification using Algorithm 8, roughly $|\mathbf{s}|/2$ of the $\widehat{\mathbf{s}}(i)$ are zero, so almost half of the rows of the public key $\mathbf{M}$ are never read, let alone operated on. Given the size of Wave public keys, the latency involved in loading key data from memory is often a bottleneck in practice.

5. Performance Analysis

Our performance analysis is based on the reference implementation, which is written in pure portable C99, and not optimized for any specific architecture; in particular, it does not take advantage of vectorization such as Intel AVX instructions (which were shown in [BDNS21] to significantly improve Wave implementation performance).

Table 5 provides signature and key sizes for each instance for easy reference. Note that the compressed encoding for signatures yields variable-length signatures, and we list the maximum sizes here (see Appendix C.2 for further details).

Table 5. Key and signature sizes for Wave instances.

Instance	Signature (B)	Private key (B)	Public key (B)
Wave822	≤ 822	18 900	3 677 390
Wave1249	$\leq 1\,249$	27 630	7 867 598
Wave1644	$\leq 1\,644$	36 360	13 632 308

Table 6 presents cycle counts for our programs on an Intel i5-1135G7 platform. We provide two counts for signature verification. The first corresponds to the `crypto_sign_open` function from the NIST API, which must first decode the public key from its transport format (i.e., trits packed into a byte array, five trits to a byte, as in Appendix C.1) to an array of bitsliced ternary vectors (as in Appendix B.1). This process is expensive—especially given the size of the public key—and much more expensive than the subsequent testing of the verification equation!

Given the size of the public key, Wave is probably best-suited to applications where the public key is stored on the device in advance of verification (rather than being transmitted with the message). In these cases, we have the option of converting from transport to format at the time of storage, rather than re-converting for each verification. The second Verification column in Table 6 corresponds to this scenario: the bitsliced public key is already loaded, and the cycle count corresponds to pure verification using the `verify` function. While the bitsliced representation imposes a space overhead of 25%, the massive speedup of over two full orders of magnitude may more than justify this approach.

Table 6. Cycle counts for the Wave reference implementation. Programs were executed on an Intel i5-1135G7 with CPU frequency 2.4 GHz and maximum CPU frequency 4.2 GHz running Arch Linux (kernel 6.3.1). We compiled the code using GCC version 13.1.1 with compiler options `-O3` and `-march=native`. Each program was run 100 times with random inputs (using random 100-byte messages). Cycles were counted using the `rdtsc` instruction included in the file `cpucycles.h`.

Instance		Key generation	Signing	Verification (transport PK)	Verification (bitsliced PK)
Wave822	Average	14 468 000 043	1 160 793 621	205 829 565	1 230 598
	Median	13 946 196 718	1 156 182 078	206 097 247	1 197 396
	Lowest	13 672 518 457	1 094 264 195	191 450 426	1 037 092
	Highest	18 921 643 347	1 367 744 577	236 350 926	1 975 376
Wave1249	Average	47 222 134 806	3 507 016 206	464 110 855	2 580 092
	Median	46 285 891 873	3 534 752 493	467 362 945	2 580 640
	Lowest	44 658 551 430	3 233 442 565	420 729 318	2 390 594
	Highest	55 561 328 781	3 695 446 731	491 116 604	2 913 327
Wave1644	Average	108 642 333 507	7 936 541 947	813 301 900	4 328 892
	Median	106 259 626 687	7 851 475 371	806 856 892	4 325 131
	Lowest	104 525 692 173	7 608 672 796	782 171 263	4 054 341
	Highest	117 900 860 252	8 225 419 171	817 790 558	4 611 381

Table 7. Hyperlinks to KATs with SHA2-256 checksums.

Instance	Hyperlink	sha256sum
Wave822	PQCsignKAT_18900	ace94f3e8e1f692632758decb5471e7408bb7669f4d3a59647046e20bb0404b5
Wave1249	PQCsignKAT_27629	75e85be5bdcf30fcbdc898ee7d84a2475cf917af06c1349eb9ded34605798e17
Wave1644	PQCsignKAT_36359	e67a83c9abdf143b4ab01ff5f2692d54534a3de1c3943165d907f077cee07356

6. Known Answer Tests

Wave may be implemented in many different ways, but it is crucial to respect the constraints imposed by the chosen parameters. Further, when utilizing a pseudorandom generator, it is important to ensure the reproducibility of the results.

In order to achieve this reproducibility, we provide Known Answer Test (KAT) values for each instance of Wave on our website. Table 7 provides hyperlinks to the KATs, with SHA2-256 checksums of each KAT file (which may be recomputed with `sha256sum`) to ensure integrity.

7. Provable Security

We now present a security argument for Wave against classical and quantum adversaries. We consider the standard EUF-CMA definition for signature schemes, where adversaries with access to a signing oracle must forge a signature on a new (non-queried) message. Since we are designing a post-quantum signature scheme, we will work in the Quantum Random Oracle Model (QROM), where the main hash function Hash used in our signature scheme is modelled as a truly random function with black-box access only; quantum adversaries have quantum access to this black box.

For quantum adversaries, λ bits of security corresponds to a quantum adversary running in time 2^λ. As per NIST guidelines, adversaries are restricted to $q_S = 2^{64}$ *classical signing queries*, no matter the intended security level.

7.1. **Hard problems.** We begin by defining the problems to which we will reduce the EUF-CMA security of Wave. First, recall the classic decoding problem DP over $\mathbb{F}_3$:

Problem 1. The **Decoding Problem** $\mathsf{DP}(n, k, w)$ is:

- Input: $(\mathbf{H}, \mathbf{s})$ where $\mathbf{H}$ resp. $\mathbf{s}$ are uniformly distributed over $\mathbb{F}_3^{(n-k)\times n}$ resp. $\mathbb{F}_3^{n-k}$.
- Output: a vector $\mathbf{e}$ in $\mathbb{F}_3^n$ of Hamming weight w such that $\mathbf{e}\mathbf{H}^\top = \mathbf{s}$.

Forging a Wave signature on a given message resembles solving a DP instance where $\mathbf{H}$ and $\mathbf{s}$ represent the public key and the hash of that message, respectively. However, our adversary's goal is to forge a signature on a *freely chosen* message, so it is more appropriate to consider a multi-target variant of DP.

Problem 2. The **Decoding One Out of Many Problem** $\mathsf{DOOM}(n, k, w, N)$ is:

- Input: $(\mathbf{H}, \mathbf{s}_1, \ldots, \mathbf{s}_N)$ where $\mathbf{H}$ resp. the $\mathbf{s}_i$ are uniformly distributed over $\mathbb{F}_3^{(n-k)\times n}$ resp. $\mathbb{F}_3^{n-k}$.
- Output: a vector $\mathbf{e} \in \mathbb{F}_3^n$ of Hamming weight w, and $i \in [1, N]$ such that $\mathbf{e}\mathbf{H}^\top = \mathbf{s}_i$.

Forging Wave signatures does not directly imply solving DP or DOOM, because Wave public keys are parity-check matrices not of truly random codes[3], but of permuted generalized $(U|U+V)$-codes (see Definition 1). However, if Wave public keys are indistinguishable from random parity-check matrices, then security reduces to DOOM. (This is similar to McEliece-like cryptosystems [McE78, ABC+20], where security reduces to the decoding problem provided distinguishing Goppa codes from random is sufficiently hard.)

Problem 3 (The **Distinguishing Wave Keys Problem** $\mathsf{DWK}(n, k_U, k_V)$)**.** Given $\mathbf{H} \in \mathbb{F}_3^{(n-(k_U+k_V))\times n}$, decide whether $\mathbf{H}$ has been chosen uniformly at random or among parity-check matrices of permuted generalized $(U|U+V)$-codes where U resp. V has dimension k_U resp. k_V.

7.2. **Signature Distribution and Rényi Divergence.** Wave security reduces to the hardness of Problems 2 and 3. We use the Rényi divergence to achieve a tight reduction, as in [LPSS17, Pre17, BLR+18] for LWE and lattice-based cryptosystems. Recall that if $\mathcal{P}$ and $\mathcal{Q}$ are distributions with $\mathrm{Supp}(\mathcal{P}) \subseteq \mathrm{Supp}(\mathcal{Q})$, then their Rényi divergence of order α (where $\alpha > 0$ and $\alpha \neq 1$) is

$$R_\alpha(\mathcal{P}||\mathcal{Q}) := \Big(\sum_{x\in\mathrm{Supp}(\mathcal{P})} \frac{\mathcal{P}(x)^\alpha}{\mathcal{Q}(x)^{\alpha-1}} \Big)^{1/(\alpha-1)}.$$

[3] A code is *random* if its parity-check matrix is uniformly drawn at random.

Wave signing uses internal distributions $\mathcal{D}_V$ and $\mathcal{D}_U$ to ensure signatures are properly distributed. Algorithm 5 constructs vectors $\mathbf{e}_V$ (using Algorithm 6, governed by $\mathcal{D}_V$) and $\mathbf{e}_U$ (using Algorithm 7, governed by $\mathcal{D}_U$), and then defines $\mathbf{e}_L := \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$ and $\mathbf{e}_R := \mathbf{c}\star\mathbf{e}_U + (\mathbf{1}+\mathbf{b}\star\mathbf{c})\star\mathbf{e}_V$. As shown in [DST19, §5.1], for Algorithm 5 to return signatures distributed as random words of Hamming weight w, it suffices that the distributions Q and Q^{ideal} match, where

- Q is the distribution of $|\mathbf{e}_V|$ and $n/2 - w + |\mathbf{e}_L \star \mathbf{e}_R|$ in Algorithm 5 given $\mathcal{D}_U$ and $\mathcal{D}_V$; and
- Q^{ideal} is the distribution of $|\mathbf{e}_L - \mathbf{c} \star \mathbf{e}_R|$ and $n/2 - w + |\mathbf{e}_L \star \mathbf{e}_R|$ on random words $(\mathbf{e}_L \parallel \mathbf{e}_R)$ of Hamming weight w.

Cutting Q to some interval $\mathsf{Accept} \subseteq [0, n/2] \times [0, w/2]$ (as in Instruction 11 of Algorithm 5) to ensure that its support is included the support of Q^{ideal}, we obtain

$$R_{2\lambda}(Q||Q^{\text{ideal}}) = 1 + \varepsilon_{Renyi}\,. \tag{17}$$

The internal distributions $\mathcal{D}_V$ and $\mathcal{D}_U$ must therefore be chosen to ensure that ε_{Renyi} is sufficiently small. In our reference implementation, the $\mathcal{D}_V$ and $\mathcal{D}_U$ included in $\mathsf{Wave822}$, $\mathsf{Wave1249}$, and $\mathsf{Wave1644}$ all give

$$\varepsilon_{Renyi} \leq 2^{-68}\,.$$

7.3. Statement of the security reduction.

Theorem 1. *Fix any $\lambda \in \mathbb{N}$, and consider a quantum adversary $\mathcal{A}$ running in time 2^λ limited to $q_S = 2^{64}$ classical signing queries that tries to break the $\mathsf{EUF\text{-}CMA}$ security of an instance of Wave. In the QROM, we have*

$$\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{Wave}}(\mathcal{A}) \leq \sqrt{2}(1-\varepsilon_{Renyi})^{q_S}\left(\mathrm{Adv}^{\mathsf{DOOM}}(2^\lambda) + \mathrm{Adv}^{\mathsf{DWK}}(2^\lambda) + \frac{q_S^2}{2^{\lambda_0}}\right)$$

where

- *$\mathrm{Adv}^{\text{EUF-CMA}}_{\mathsf{Wave}}(\mathcal{A})$ is the probability that $\mathcal{A}$ breaks the $\mathsf{EUF\text{-}CMA}$ security of Wave;*
- *ε_{Renyi} is defined in Equation (17);*
- *$\mathrm{Adv}^{\mathsf{DOOM}}(2^\lambda)$ is the maximum probability that a quantum algorithm running in time 2^λ solves DOOM (Problem 2) for the Wave parameters n, k, and w (the parameter N is not restricted, and in the quantum setting we even allow adversaries quantum access to the syndromes: that is, adversaries can efficiently compute the unitary $|i\rangle|0\rangle \to |i\rangle|s_i\rangle$);*
- *$\mathrm{Adv}^{\mathsf{DWK}}(2^\lambda)$ is $|p - \frac{1}{2}|$, where p is the maximum probability that a quantum algorithm running in time 2^λ solves DWK (Problem 3) for the Wave parameters;*
- *λ_0 is the size of the salt. In our case, we always have $\lambda_0 \geq 256$ hence $\frac{q_S^2}{2^{\lambda_0}} \leq 2^{-128}$.*

The above also holds for classical adversaries in the Random Oracle Model (if the best considered attacks against Problems 2 and 3 are also classical).

Theorem 1 is proven by combining two existing results (which hold both in the ROM against classical adversaries and the QROM against quantum adversaries):

(1) The results of [CD20] show that if Wave produces an ideal distribution of signatures (i.e., Q matches Q^{ideal}) then it reduces tightly to Problems 2 and 3.
(2) Then, the results of [Pre17, Section 3.3 of the Eprint version] show that moving from an ideal signing distribution to a real distribution (i.e., replacing Q^{ideal} with Q) induces a multiplicative loss of $\sqrt{2}\,(1-\varepsilon_{Renyi})^{q_S}$ in the EUF-CMA advantage.

The internal distributions in Wave822, Wave1249, and Wave1644 all have $\varepsilon_{Renyi} \leq 2^{-68}$, so $(1+\varepsilon_{Renyi})^{q_S} \leq \sqrt{2}$. Plugging this into Theorem 1 yields

$$\mathrm{Adv}^{\mathrm{EUF\text{-}CMA}}_{\mathsf{Wave}}(\mathcal{A}) \leq 2\left(\mathrm{Adv}^{\mathsf{DOOM}}(2^{\lambda}) + \mathrm{Adv}^{\mathsf{DWK}}(2^{\lambda}) + \frac{q_S^2}{2^{\lambda_0}}\right)$$

for all three instances, so their security tightly reduces to Problems 2 and 3. We discuss the best known attacks on these problems in §8.

8. Best Known Attacks

We now present the best known attacks on DOOM (which we call *message attacks*) and DWK (which we call *key attacks*). Both classes of attacks rely on algorithms to solve DP. While this classic problem has been widely studied [Pra62, Ste88, Dum91, FS09, Ber10, BLP11, MMT11, BJMM12, MO15, BM17, KT17, BM18, Kir18, BBSS20, CDMT22], much less has been done for DP with the code parameters used in Wave (for instance, working over $\mathbb{F}_3$ instead of $\mathbb{F}_2$) [BCDL19, Bri21, CDE21, KL22, Sen23].

8.1. **Classical attacks.** The best known attacks on DP and DOOM use the Information Set Decoding (ISD) framework of [FS09], which is a refinement of Prange's algorithm [Pra62]. The idea is to find a single solution to DP in $\mathbb{F}_3^n$ with weight w by identifying exponentially many solutions to a simplified DP instance in $\mathbb{F}_3^{\ell}$ with weight p, for chosen $\ell \leq n$ and $p \leq w$. For each potential solution, we can efficiently verify if it yields a complete solution to DP. This entire process is repeated until a solution is discovered. Several algorithms exist for listing solutions to this sub-problem: [Ste88] and [Dum91] take advantage of the birthday paradox while [MMT11] and [BJMM12] combine the so-called representation technique with merging techniques.

In the binary case ($q = 2$), the best known ISD ([BM18] with the correction in [CDMT22, Appendix B]) lists solutions to the sub-problem by searching for pairs of *near-neighbours* in some lists. Near-neighbours-based decoders designed for the ternary setting may have better asymptotic complexities than the ISD techniques mentioned above. However, an efficient near-neighbours search algorithm requires *fuzzy hashing functions* that associate close vectors. One of the best known ways to design a good fuzzy hashing function is to use a list decoder of a polar code, resulting in an $O(n\log(n))$ overhead (compared to other ISDs) that we take into account when setting parameters.

8.1.1. *Classical message attacks.* The ISD framework with Wagner's algorithm [BCDL19] is the best known algorithm to find solutions for the sub-DP instance in the Wave context ($w/n \approx 0.89$ and $k/n = 1/2$). This involves creating lists of vectors corresponding to $\mathbf{e}'' \in \mathbb{F}_3^{k+\ell}$ with $|\mathbf{e}''| = p$. In the DOOM solver of [Sen11], one of the lists is filled with some restrictions $\mathbf{s}_i''$s of the $\mathbf{s}_i$ over their last $k+\ell$ coordinates. The lists are then filtered to include only $\mathbf{e}''$ for which $\mathbf{e}''\mathbf{H}''^{\top} = \mathbf{s}_i''$ for some i (the sub-problem), where $\mathbf{H}'' \in \mathbb{F}_3^{(n-k)\times(k+\ell)}$ is some sub-matrix of $\mathbf{H}$. The lists are merged using a binary merging tree, keeping only the pairs of vectors whose first coordinates sum to zero. Ultimately, we obtain a list of vectors from which we can deduce multiple solutions to the DP sub-instance, and thus to the full DP instance. Otherwise, the process is repeated until a solution is found.

8.1.2. *Classical key attacks.* One way to solve DWK, *i.e.* to distinguish the public key $\mathbf{H}$ (a parity-check matrix of some permuted generalized $(U|U+V)$-code) from a randomly uniform matrix in $\mathbb{F}_3^{(n-k)\times n}$ is to exhibit a codeword of form $(\mathbf{u},\mathbf{u})$ of weight t, where t is a parameter that can be chosen. Indeed, these words may be more likely to appear in a $(U|U+V)$-code than in a random code [Sen23]. The best known classical attack uses ISD to find such vectors satisfying $\mathbf{eH}^\top = \mathbf{0}$. Within this range of parameters, Wagner's algorithm [SS81] does not perform better than just having two merging steps and using the representation technique of [MMT11]. This attack computes multiple solutions to the DP sub-instance, and then checks if one of the candidate solutions satisfies the general DP.

8.2. **Quantum attacks.** Code-based problems, and more specifically decoding problems, have resisted quantum attacks very well so far. The best known quantum attacks on these various decoding problems are also based on ISD. Many quantum algorithms have been studied; for simpler examples such as Prange's algorithm, a direct application of quantum amplitude amplification gives a quadratic advantage [Ber10], but we cannot get the same improvement for advanced ISD algorithms which list many solutions to a sub-problem. This has been extensively studied for a binary alphabet in [KT17], and extended in [Kir18, BBSS20]. Quantum algorithms for DP and DOOM have also been studied in the ternary setting of Wave [CDE21, Bri21].

8.2.1. *Quantum message attacks.* We employ the quantum ISD framework, where the sub-problem solver utilizes the quantum Wagner algorithm [CDE21]. In this quantum setting, the list of $\mathbf{s}_i$ is replaced by their quantum superposition, and the number of such elements can exceed the number in the classical setting. The classical lists are merged, and the merging of a classical list with the list in quantum superposition is accomplished using Grover's algorithm [Gro96] to yield a quantum superposition of candidate solutions. If the probability of success is lower than $1 - o(1)$, we apply amplitude amplification before the measurement to obtain a DP solution. We rely on the analysis of [CDE21] as well as the DOOM analysis of [Bri21], which provides the best quantum algorithm in this scenario.

8.2.2. *Quantum key attacks.* To find a codeword $(\mathbf{u}, \mathbf{u})$ of weight t, the best known attack also uses quantum ISD, finding the solutions to the sub-problem using quantum Wagner [CDE21]. As above, amplitude amplification can be applied to get a DP solution with high probability. Again, we combine the analyses of [CDE21] and (the classical) [Sen23] to study the best quantum algorithms here.

8.3. **Claimed security levels.** Table 8 lists the claimed security levels λ for Wave822, Wave1249, and Wave1644 against the attacks listed above. All our best classical and quantum attacks require $poly(n)2^{\alpha n + o(n)}$ elementary operations for some α, where n is the code length. We ignore the $poly(n)$ factor and the $o(n)$ when estimating our security levels. Omitting these large overheads make our parameters very conservative.

We also ignore the cost of memory access, even though these algorithms can use a very large amount of memory: for example, the DOOM attack above requires space equal to its running time. In the quantum setting, these overheads are increased by the fact that the best known quantum attacks use a large amount of QRAM (Quantum Random Access Memory), whose feasibility is still in question [JR23]. This makes our security levels even more conservative in the quantum setting.

TABLE 8. Claimed security levels for Wave instances. In this context, λ bits of security indicate that the most efficient known attacks on DOOM and key distinguishing all require a minimum time of 2^λ to execute.

	NIST Post-Quantum	Classical (bits)	Quantum (bits)
Wave822	Level I	128	77
Wave1249	Level III	192	117
Wave1644	Level V	256	157

9. Advantages and Limitations

9.1. Advantages.

- **Short signature length**: at most 822 bytes for NIST Level I, and scaling linearly with the security parameter.
- **Fast verification**, with scope for further significant speed-ups in dedicated implementations (as in [BDNS21]).
- **Proven secure** under well identified code-based hardness assumptions, in the ROM and the QROM with tight reductions.
- The scheme is **immune to statistical attacks** by design, and the reference implementation is immune to statistical attacks from any adversary limited to 2^{64} queries to a signing oracle.

9.2. Limitations.

- Large public keys: 3.6MB for NIST Level I, and scaling quadratically with the security parameter.
- Signing and key generation rely on inherently slow Gaussian elimination on large matrices. Accelerating these primitives while ensuring implementation safety is a significant challenge.
- We assume the hardness of DWK, distinguishing permuted generalized $(U|U+V)$-codes from random codes. This assumption is fairly new (it was introduced in 2018) but we have reasons to argue for its hardness: for example, these permuted generalized $(U|U+V)$-codes have a lot of inner entropy. The only exploitable structure seems to be the existence of small codewords, which we already fully exploit in our security analysis. We believe our parameters are extremely conservative with respect to these codes, providing a comfortable security margin against possible attacks on a relatively new problem, but further analysis may show that we can reduce parameter sizes, thus improving key sizes and performance.
- The parameters include the internal discrete distributions $\mathcal{D}_V$ and $\mathcal{D}_U(t)$, which condition the decoder output distribution (which must be close to uniform to guarantee security: see §7), Adding rejection sampling if necessary. The internal distributions in our implementation were computed by an ad-hoc procedure (taking a few hours). They produce an output distribution which is close enough to uniform, with a Rényi divergence $\leq 1+2^{-68}$, *without* rejection sampling. At the moment,

the internal distributions are given as tables of numbers. Tools will be provided to guarantee their correctness.

References

[ABC+20] Martin Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Mizoczki, Ruben Niederhagen, Edoardo Persichetti, Kenneth Paterson, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wang Wen. Classic McEliece (merger of Classic McEliece and NTS-KEM). https://classic.mceliece.org, October 2020. Third round finalist of the NIST post-quantum cryptography call.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In *Advances in Cryptology – ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II*, page 633–666, Berlin, Heidelberg, 2020. Springer-Verlag.

[BCDL19] Rémi Bricout, André Chailloux, Thomas Debris-Alazard, and Matthieu Lequesne. Ternary syndrome decoding with large weights. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers*, volume 11959 of *Lecture Notes in Computer Science*, pages 437–466. Springer, 2019.

[BDNS21] Gustavo Banegas, Thomas Debris-Alazard, Milena Nedeljković, and Benjamin Smith. Wavelet: Code-based postquantum signatures with fast verification on microcontrollers. Cryptology ePrint Archive, Report 2021/1432, 2021. https://ia.cr/2021/1432.

[Ber] Daniel J. Bernstein. djbsort. Software library for sorting arrays of integers or floating-point numbers. https://sorting.cr.yp.to/.

[Ber10] Daniel J. Bernstein. Grover vs. McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings*, volume 6061 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2010.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *Advances in Cryptology - EUROCRYPT 2012*, LNCS. Springer, 2012.

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: ball-collision decoding. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *LNCS*, pages 743–760, 2011.

[BLR+18] Shi Bai, Tancrède Lepoint, Adeline Roux-Langlois, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. *J. Cryptology*, 31(2):610–640, 2018.

[BM17] Leif Both and Alexander May. Optimizing BJMM with Nearest Neighbors: Full Decoding in $2^{2/21n}$ and McEliece Security. In *WCC Workshop on Coding and Cryptography*, September 2017.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography 2018*, volume 10786 of *LNCS*, pages 25–46, Fort Lauderdale, FL, USA, April 2018. Springer.

[Bri21] Rémi Bricout. *Comment utiliser des algorithmes quantiques pour remplir son sac à dos et décoder des syndromes*. Theses, Sorbonne Université, March 2021.

[CD20] André Chailloux and Thomas Debris-Alazard. Tight and optimal reductions for signatures based on average trapdoor preimage sampleable functions and applications to code-based signatures. In *Public-Key Cryptography – PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II*, page 453–479, Berlin, Heidelberg, 2020. Springer-Verlag.

[CDE21] André Chailloux, Thomas Debris-Alazard, and Simona Etinski. Classical and quantum algorithms for generic syndrome decoding problems and applications to the Lee metric. In Jung Hee

Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 44–62. Springer, 2021.

[CDMT22] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Statistical decoding 2.0: Reducing decoding to LPN. In *Advances in Cryptology - ASIACRYPT 2022*, LNCS. Springer, 2022.

[DST17] T. Debris-Alazard, N. Sendrier, and J.-P. Tillich. The problem with the surf scheme. preprint, November 2017. arXiv:1706.08065.

[DST19] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 21–51, Kobe, Japan, December 2019. Springer.

[Dum91] I Dumer. On minimum distance decoding of linear codes. *Proc. 5th JointSoviet-Swedish Int. Workshop Inform. Theory, Moscow*, page 50–52, 1991.

[FHK+17] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. First round submission to the NIST post-quantum cryptography call, November 2017.

[FS09] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, 2009.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[Gro96] Lov Grover. A fast quantum mechanical algorithm for database search. *Proc.28th Annual ACM Symposium on the Theory of Computing STOC*, pages 212 – 219, 1996.

[Jou04] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[JR23] Samuel Jaques and Arthur G. Rattew. QRAM: A survey and critique, 2023.

[Kir18] Elena Kirshanova. Improved quantum information set decoding. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 507–527. Springer, 2018.

[KL22] Pierre Karpman and Charlotte Lefevre. Time-memory tradeoffs for large-weight syndrome decoding in ternary codes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 82–111. Springer, 2022.

[KT17] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2017.

[LDK+20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-Dilithium, 2020. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[Leh10] Anja Lehmann. *On the security of hash function combiners*. PhD thesis, Darmstadt University of Technology, 2010.

[Lem19] Daniel Lemire. Fast random integer generation in an interval. *ACM Trans. Model. Comput. Simul.*, 29(1):3:1–3:12, 2019.

[LPSS17] San Ling, Duong Hieu Phan, Damien Stehlé, and Ron Steinfeld. Hardness of k-lwe and applications in traitor tracing. *Algorithmica*, 79(4):1318–1352, 2017.

[McE78] Robert J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $O(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, 2011.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 203–228. Springer, 2015.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[Pre17] Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017*, volume 10624 of *LNCS*, pages 347–374. Springer, 2017.

[Sen11] Nicolas Sendrier. Decoding one out of many. In *Post-Quantum Cryptography 2011*, volume 7071 of *LNCS*, pages 51–67, 2011.

[Sen23] Nicolas Sendrier. Wave parameter selection. *preprint*, 2023.

[SS81] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In G. D. Cohen and J. Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *LNCS*, pages 106–113. Springer, 1988.

[WSN18] Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based Niederreiter cryptosystem using binary Goppa codes. In Tanja Lange and Rainer Steinwandt, editors, *PQCrypto*, volume 10786 of *LNCS*, pages 77–98. Springer, 2018.

Appendix A. Hashing to Ternary Vectors

In order to sign any messages with Wave, we need to define a hash function into $\mathbb{F}_3^{n-k}$. We use the hash function as defined in [BDNS21, §3.1]. We recall here its specifications.

The Hash (Algorithm 9) defines a cryptographic hash function $\{0,1\}^* \to \mathbb{F}_3^{n-k}$ by wrapping the standard `SHA3-512` hash function with the following two functions:

(1) $\mathsf{Ternarize} : \{0,1\}^* \times \mathbb{Z}_{\geq 0} \to \mathbb{F}_3^*$ (Algorithm 10) views its input $((x_0, x_1, \ldots), \tau)$ as the vector of coefficients in the little-endian binary expansion of an integer x, together with a length τ, and returns the ternary vector of length τ representing the little-endian ternary expansion of $x \bmod 3^\tau$,

(2) $\mathsf{Expand} : \{0,1\}^{2\lambda} \to \mathbb{F}_3^\tau$ (Algorithm 11) is a pseudorandom function (λ denotes the security parameter). Expand applies `SHAKE256` to its input to produce a long stream of pseudorandom bytes, which we view as integers in $[0, 256)$. The non-negative integers less than $3^5 = 243$ are in bijection with $\mathbb{F}_3^5$, so if a byte is less than 243 we convert it to an element of $\mathbb{F}_3^5$ with Ternarize and concatenate it to the output; otherwise we skip the byte. We continue processing bytes until we have produced τ elements of $\mathbb{F}_3$ (discarding the last few trits if τ is not a multiple of 5).

The collision- and preimage-resistance of Hash are derived from the properties of `SHA3-512` using standard (conatenation) hash combiner arguments (see e.g. [Jou04, §4] and [Leh10]). Ternarize simply transcodes its input, so the composition of Ternarize and `SHA3-512` preserves the security properties of `SHA3-512`. The composition of Expand and `SHA3-512` has weaker preimage and collision resistance (because bytes in $[243, 256)$ are discarded), but it still has strong pseudorandomness properties, and it is relatively fast to compute. The concatenation of the two has the security of the strong hash, and the good pseudorandomness of both.

Algorithm 9 Hashing from $\{0,1\}^*$ to $\mathbb{F}_3^{n-k}$ in Wave.

function $\mathsf{Hash}(\mathbf{m})$ ▷ $\mathbf{m} \in \{0,1\}^*$
  $h \leftarrow \mathsf{Truncate}(\texttt{SHA3-512}(\mathbf{m}), 2\lambda)$ ▷ Truncate `SHA3-512` output to first 2λ bits
  $\mathbf{t} \leftarrow \mathsf{Ternarize}\,(h, \lfloor 2\lambda / \log_2(3) \rfloor)$ ▷ Algorithm 10
  $\mathbf{p} \leftarrow \mathsf{Expand}\,(h, n-k-\lfloor 2\lambda / \log_2(3) \rfloor)$ ▷ Algorithm 11
  return $(\mathbf{t}, \mathbf{p})$

Algorithm 10 Converting integer values to ternary vectors of a specified length, corresponding to the little-endian ternary expansion of the input.

function $\mathsf{Ternarize}(x, \tau)$ ▷ $x \geq 0$ and $\tau > 0$
  $\mathbf{v} \leftarrow \mathbf{0} \in \mathbb{F}_3^\tau$
  for $0 \leq i < \tau$ **do**
    $(x, \mathbf{v}(i)) \leftarrow (\lfloor x/3 \rfloor, x \bmod 3)$
  return $\mathbf{v}$ ▷ $\mathbf{v} \in \mathbb{F}_3^\tau \cong \{0,1,2\}^\tau$ such that $x = \sum_{i=1}^{\tau} \mathbf{v}(i) 3^{i-1}$

Algorithm 11 Expand a binary seed to a pseudo-random stream of ternary values. The random bytestream may be instantiated with an XOF or a stream cipher. The expected number of bytes drawn from the stream is $(256\tau)/(243 \times 5) \approx 0.21\tau$.

function $\mathsf{Expand}(h, \tau)$ ▷ $h \in \{0,1\}^{2\lambda}$ and $\tau > 0$
  stream = pseudorandom bytestream obtained via `SHAKE256`(h)
  $(\mathbf{p}, r) \leftarrow ((), \tau)$ ▷ $\mathbf{p}$: empty vector over $\mathbb{F}_3$
  while $r > 0$ **do**
    $b \leftarrow$ next byte from **stream**, viewed as an integer in $[0, 255]$
    if $b < 243$ **then**
      $(\mathbf{p}, r) \leftarrow (\mathbf{p}, \mathrm{Ternarize}(b, \min(5, r)), r)$ ▷ Algorithm 10
  return p ▷ $\mathbf{p} \in \mathbb{F}_3^\tau$

Appendix B. Specification toward a Constant Time Implementation

B.1. **Bitsliced Arithmetic over $\mathbb{F}_3$.** The reference implementation uses the following bitsliced $\mathbb{F}_3$-vector arithmetic. An element a of $\mathbb{F}_3$ is represented as a pair of bits (a_h, a_ℓ):

$$0 \longleftrightarrow (0,0), \quad 1 \longleftrightarrow (0,1), \quad 2 \longleftrightarrow (1,1).$$

All the arithmetic can be expressed with binary operations, addition, multiplication, and inclusive or denoted '|'. Addition and subtraction in $\mathbb{F}_3$ requires 7 binary operations, negation in $\mathbb{F}_3$ requires 1 binary operation, and multiplication in $\mathbb{F}_3$ requires 3 binary operations.

$$c = a + b \iff \begin{cases} c_h &= (a_\ell + b_h)(a_h + b_\ell) \\ c_\ell &= (a_\ell + b_\ell) \mid (a_h + b_\ell + b_h) \end{cases}$$

$$c = a - b \iff \begin{cases} c_h &= (a_\ell + b_\ell + b_h)(a_h + b_\ell) \\ c_\ell &= (a_\ell + b_\ell) \mid (a_h + b_h) \end{cases}$$

$$c = -a \iff \begin{cases} c_h &= a_h + a_\ell \\ c_\ell &= a_\ell \end{cases}$$

$$c = a \times b \iff \begin{cases} c_h &= (a_h + b_h) a_\ell b_\ell \\ c_\ell &= a_\ell b_\ell \end{cases}$$

This representation and the corresponding arithmetic fits well with bit slicing techniques and the above operations apply conveniently to vectors. A ternary vector $\mathbf{x} = (x_i)_{0 \le i < n}$ where $x_i \longleftrightarrow (x_{i,h}, x_{i,\ell})$ is represented by $(\mathbf{x}_h, \mathbf{x}_\ell)$ with $\mathbf{x}_h = (x_{i,h})_{0 \le i < n}$ and $\mathbf{x}_\ell = (x_{i,\ell})_{0 \le i < n}$.

The vector operations can be computed for logical operations & and $\oplus$ on binary words:

$$\mathbf{c} = \mathbf{a} + \mathbf{b} \iff \begin{cases} \mathbf{c}_h &= (\mathbf{a}_\ell \oplus \mathbf{b}_h) \,\&\, (\mathbf{a}_h \oplus \mathbf{b}_\ell) \\ \mathbf{c}_\ell &= (\mathbf{a}_\ell \oplus \mathbf{b}_\ell) \mid (\mathbf{a}_h \oplus \mathbf{b}_\ell \oplus \mathbf{b}_h) \end{cases}$$

$$\mathbf{c} = \mathbf{a} - \mathbf{b} \iff \begin{cases} \mathbf{c}_h &= (\mathbf{a}_\ell \oplus \mathbf{b}_\ell \oplus \mathbf{b}_h) \,\&\, (\mathbf{a}_h \oplus \mathbf{b}_\ell) \\ \mathbf{c}_\ell &= (\mathbf{a}_\ell \oplus \mathbf{b}_\ell) \mid (\mathbf{a}_h \oplus \mathbf{b}_h) \end{cases}$$

$$\mathbf{c} = -\mathbf{a} \iff \begin{cases} \mathbf{c}_h &= \mathbf{a}_h \oplus \mathbf{a}_\ell \\ \mathbf{c}_\ell &= \mathbf{a}_\ell \end{cases}$$

$$\mathbf{c} = \mathbf{a} \star \mathbf{b} \iff \begin{cases} \mathbf{c}_h &= (\mathbf{a}_h \oplus \mathbf{b}_h) \,\&\, \mathbf{a}_\ell \,\&\, \mathbf{b}_\ell \\ \mathbf{c}_\ell &= \mathbf{a}_\ell \,\&\, \mathbf{b}_\ell \end{cases}$$

B.2. **Sampling Trits.** The purpose is to efficiently produce—close to—independent and uniform trits from independent and uniform bits. The description assumes a binary pseudorandom generator prng initialized with seed, *e.g.* any XOF with input seed. The instruction $\mathsf{randbits}(a, \mathsf{prng})$ draws the next a bits of prng and returns the corresponding integer in $[0, 2^a)$. It can be used in Algorithm 12 to produce a pseudorandom ternary vector of length n from seed. This algorithm admits as parameters the integers a and b, it will

Algorithm 12 Pseudorandom Sampling of Ternary Vectors

```
function prng_3(n, seed)
    prng ← prng_init(seed)
    i ← 0
    repeat
        x_i, ..., x_{i+b−1} ← randtrits(a, b, prng)
        i ← i + b
    until i ≥ n
    return x_0, ..., x_{n−1}
```

```
function randtrits(a, b, prng)
    repeat
        y ← randbits(a, prng)
    until y < 3^b
    for i = 0, ..., b − 1 do
        x_i = y mod 3
        y = ⌊y/3⌋
    return x_0, ..., x_{b−1}
```

output independent uniform trits if $3^b \geq 2^a$.

Limitation. If Algorithm 12 is used in a cryptographic context it will leak some information about seed as the number of rejection in the calls to $\mathsf{randtrits}()$ will depend of seed. To avoid leakage of possibly meaningful secret information we will favor conversions which are oblivious to their inputs and outputs.

B.2.1. *Oblivious Conversion.* The conversion algorithm can be made oblivious to the prng output by removing the rejection as shown in Algorithm 13.

Algorithm 13 Binary to Ternary Conversion Without Rejection

With modulo

```
function randtrits(a, b, prng)
    y ← randbits(a, prng)
    for i = 0, ..., b − 1 do
        x_i ← y mod 3
        y ← ⌊y/3⌋
    return x_0, ..., x_{b−1}
```

Without modulo

```
function randtrits(a, b, prng)
    y ← randbits(a, prng)
    for i = 0, ..., b − 1 do
        y ← 3 · (y mod 2^a)
        x_i ← ⌊y/2^a⌋
    return x_0, ..., x_{b−1}
```

Limitation. With this modification, no information will leak, however the output won't be uniformly distributed. The variant without modulo has the same distribution overall and avoids the division by 3, see [Lem19]. In both case, the entropy is

$$H_{a,b} = -\frac{r(q+1)}{2^a}\log_2\frac{q+1}{2^a} - \frac{(3^b-r)q}{2^a}\log_2\frac{q}{2^a} = b\cdot(\log_2 3 - \delta_{a,b})$$

where q and r denote the quotient and the remainder of the Euclidean division of 2^a by 3^b, $2^a = q3^b + r$. The quantity $\delta_{a,b}$ measures the loss of entropy per trit, for instance $\delta_{64,32} = 8.5\,10^{-12}$.

B.2.2. *Bitsliced Conversion.* Whenever required, the reference implementation will use Algorithm 14 to draw random bits with the appropriate method and convert them to ternary. The function randtrits() generates 128 random bits, two 64-bit words, and interleaves two calls to convert(), each consuming 64 bits with the property stated in Proposition 5, to return a bitsliced ternary vector of length 64, as defined in §B.1.

Algorithm 14 Bitsliced Ternary Conversion ($a = 64$, $b = 32$)

function randtrits(prng)
- $x \leftarrow \mathsf{randbits}(a, \mathsf{prng})$
- $r \leftarrow \mathsf{convert}(x)$
- $x' \leftarrow \mathsf{randbits}(a, \mathsf{prng})$
- $r' \leftarrow \mathsf{convert}(x')$
- $\mathbf{y}_h \leftarrow ((r \mathbin{\&} \mathtt{Hi}) \gg 1) \oplus (r' \mathbin{\&} \mathtt{Hi})$
- $\mathbf{y}_l \leftarrow (r \mathbin{\&} \mathtt{Lo}) \oplus ((r' \mathbin{\&} \mathtt{Lo}) \ll 1) \oplus \mathbf{y}_h$
- **return** $\mathbf{y}$

function convert(x)
- $r \leftarrow 0$
- **for** $i = 0, \ldots, 7$ **do**
  - $(x, v_i) \leftarrow \mathtt{mul}_{64}(x, 81)$
  - $r \leftarrow v_i \oplus (r \ll 8)$
- $y \leftarrow (((r+\mathtt{Y}_4) \mathbin{\&} (r+2\mathtt{Y}_4) \mathbin{\&} \mathtt{E}_6) \oplus ((r+2\mathtt{Y}_4) \mathbin{\&} \mathtt{E}_7))$
- $r \leftarrow r + (y \gg 6) \times 37$
- $y \leftarrow (((r+\mathtt{Y}_3) \mathbin{\&} (r+2\mathtt{Y}_3) \mathbin{\&} \mathtt{E}_4) \oplus ((r+2\mathtt{Y}_3) \mathbin{\&} \mathtt{E}_5))$
- $r \leftarrow r + (y \gg 4) \times 7$
- $y \leftarrow (((r+\mathtt{Y}_2) \mathbin{\&} (r+2\mathtt{Y}_2) \mathbin{\&} \mathtt{E}_2) \oplus ((r+2\mathtt{Y}_2) \mathbin{\&} \mathtt{E}_3))$
- $r \leftarrow r + (y \gg 2)$
- **return** r

Constants (hexadecimal):

- Lo = 0x5555555555555555
- Hi = 0xAAAAAAAAAAAAAAAA

- $\mathtt{E}_2$ = 0x0404040404040404
- $\mathtt{E}_3$ = 0x0808080808080808
- $\mathtt{E}_4$ = 0x1010101010101010
- $\mathtt{E}_5$ = 0x2020202020202020
- $\mathtt{E}_6$ = 0x4040404040404040
- $\mathtt{E}_7$ = 0x8080808080808080
- $\mathtt{Y}_2$ = 0x0101010101010101
- $\mathtt{Y}_3$ = 0x0707070707070707
- $\mathtt{Y}_4$ = 0x2525252525252525

For $0 \le x, y, u, v < 2^{64}$
$(u, v) \leftarrow \mathtt{mul}_{64}(x, y)$
if $u + 2^{64}v = xy$

Proposition 5. *On input $x = \sum_{i=0}^{63} x_i 2^i$, a 64-bit integer, the function* convert() *of Algorithm 14 returns a 64-bit integer $r = \sum_{i=0}^{31} u_i 4^i$ such that $x/2^{64} = \sum_{i\geq 0} u_i/3^{i+1}$ with $u_i \in \{0,1,2\}$. If x is uniformly distributed in $\{0,1\}^{64}$ then the random variable $(u_i)_{0\leq i<32}$ which takes its value in $\{0,1,2\}^{32}$ has an entropy*

$$H = 32\ (\log_2 3 - \delta) \text{ with } \delta \approx 8.5\,10^{-12}.$$

Note that the uniform distribution over $\{0,1,2\}^{32}$ has entropy $32\ \log_2 3$.

B.3. **Sampling Permutations and Permuting by Sorting.** We give in this section specifications (as implemented in the reference implementation) to sample permutations in constant time. Recall that key generation and signing in Wave involve many permutation samples, sometimes uniform as in Algorithm 4 or sometimes with some constraint as in Algorithms 6 and 7.

In this section, $\mathbb{F}$ is defined as any finite arbitrary alphabet. In practice $\mathbb{F}$ is chosen as $\mathbb{F}_3^\ell$ for some small ℓ, typically it is equal to one or two.

We propose to sample permutations by *sorting*. Notice that it delegates all security related implementation issues to the sorting algorithm. In particular, sorting networks allows the sorting of an array with complexity $O(n(\log n)^2)$ which is oblivious to the data being sorted. Efficient cryptography oriented implementations are available, see `djbsort` [Ber] for instance. Another option is merge sort, as in [WSN18] for instance.

Algorithm 15 RandPerm – Permutation Sampling by Sorting

function $\mathsf{RandPerm}(n, \mathbf{z})$ $\qquad\triangleright\ \mathbf{z} \in \mathbb{F}^n$
    repeat
        for $i = 0, \ldots, n-1$ **do**
            $r_i \xleftarrow{\$} [0, 2^B)$ $\qquad\triangleright$ B an integer, external parameter
        $\mathbf{x} \leftarrow \mathsf{sort}((r_i, i, z_i)_{0\le i<n})$ $\qquad\triangleright$ sort according to r_i
    until $(r_{j-1} \neq r_j, 0 < j < n)$
    for $i = 0, \ldots, n-1$ **do**
        $(*, \sigma(i), y_i) \leftarrow x_i$
    return $(\sigma, \mathbf{y})$ $\qquad\triangleright\ \sigma \in \mathfrak{S}_n$, $\mathbf{y} = \mathbf{z}^\sigma$

B.3.1. *Sampling Permutations for Algorithm 4 (Key Generation).* Algorithm 15 produces a uniform permutation. The sample is rejected if some r_i for $i \in [0, n)$ collide. The rejection probability is upper bounded by a constant smaller than $1/2$ if $B \geq 2\log_2 n$. If the sampler admits a vector as additional input, it returns the vector permuted according to the sampled permutation.

B.3.2. *Sampling Permutations for Algorithm 6* (Decode_V). The call $\mathsf{RandPerm}_V(n, k, t)$ will return a permutation π of $[0, n)$ such that $(t \leq k)$

(*i*) $\pi([0, k))$ is uniformly selected in $[0, n)$,

(*ii*) $\pi([0, t))$ is uniformly selected in $\pi([0, k))$.

If a vector $\mathbf{z} \in \mathbb{F}^n$ is provided as input, the permuted vector $\mathbf{y} = \mathbf{z}^\pi$ is returned in addition to π.

Connection with Decode_V***.*** Using notation of Algorithm 6, Instructions 4 and 5 will be replaced by

$$(\pi, \mathbf{y}) \leftarrow \mathsf{RandPerm}_V(n/2, k_V - g, t, \mathbf{y}_V).$$

The returned permutation does not need to be uniformly distributed, but simply, as specified above, namely $\pi([0, k_V - g))$ and $\pi([0, t))$ are uniformly distributed in $[0, n)$. These conditions are ensured by Instruction 6 of Algorithm 16. Also, no information must leak about t, $\mathbf{y}_V$, or the permutation π.

Algorithm 16 $\mathsf{RandPerm}_V$

1: **function** $\mathsf{RandPerm}_V(n, k, t, \mathbf{z})$ $\triangleright$ $k < n$, $t \leq k$, $\mathbf{z} \in \mathbb{F}^n$
2:     **repeat**
3:         **for** $i = 0, \ldots, n-1$ **do**
4:             $r_i \xleftarrow{\$} [0, 2^B)$ $\triangleright$ B an integer, external parameter
5:         $\mathbf{x} \leftarrow \mathsf{sort}\left((r_i, i, z_i)_{0 \leq i < n}\right)$ $\triangleright$ sort according to r_i
6:     **until** $r_{k-1} \neq r_k$ and $r_{t-1} \neq r_t$
7:     **for** $i = 0, \ldots, n-1$ **do**
8:         $(*, \pi(i), y_i) \leftarrow x_i$
9:     **return** $(\pi, \mathbf{y})$

If the sorting algorithm is oblivious to the data it processes, then this is also the case for Algorithm 16 with the exception of the test $r_{t-1} \neq r_t$ at Instruction 6 which must be implemented without revealing information about t.

B.3.3. *Sampling Permutations for Algorithm 7* (Decode_U). The call $\mathsf{RandPerm}_U(n, k, \ell, J)$ will return a permutation π of $[0, n)$ such that

(*i*) $\pi([0, k-\ell))$ is uniformly selected in $[0, n) \setminus J$,

(*ii*) $\pi([k-\ell, k))$ is uniformly selected in J.

If a vector $\mathbf{z} \in \mathbb{F}^n$ is provided, the permuted vector $\mathbf{y} = \mathbf{z}^\pi$ is returned in addition to π. The algorithm will proceed in two steps

(1) Permute $[0, n)$ randomly with the positions of J coming last,

(2) Swap the last ℓ entries of the following blocks, the first one is given by coordinates $[0, k)$ and the second one by $[k, n)$, to ensure ℓ elements of J in the first k entries.

Instruction 6 will make a check after Step (1) to ensure that the first $k - \ell$ entries of the permutation are uniformly distributed in $[0, n) \setminus J$ (true if and only if $r_{k-\ell-1} \neq r_{k-\ell}$ and the last ℓ entries of the permutation are uniformly distributed in J (true if and only if $r_{n-\ell-1} \neq r_{n-\ell}$).

Connection with Decode_U***.*** Using notation of Algorithm 7, Instructions 5, 6, 7 and 10 will be replaced by

$$(\pi, \mathbf{x}) \leftarrow \mathsf{RandPerm}_U(n/2, n/2 - k_U + g, \ell, \mathrm{Supp}(\mathbf{e}_V), \mathbf{x})$$

where $x_i = (\mathbf{y}_V(i), (c_i - b_i)\mathbf{e}_V(i), \mathbf{e}_V(i)^2)$, $i \in [0, n/2)$. It is required that the first $n/2 - k_V + g$ entries of π consist of ℓ positions uniformly chosen in $\mathrm{Supp}(\mathbf{e}_V)$ and $n/2 - k_U + g - \ell$ positions uniformly chosen in $[0, n) \setminus \mathrm{Supp}(\mathbf{e}_V)$. Algorithm 17 complies to those constraints. Also, no information must leak about ℓ, J, $\mathbf{x}$, or the permutation π.

If the sorting algorithm is oblivious to the data it processes, then this is also the case for Algorithm 17 with the exception of the tests $r_{k-\ell-1} \neq r_{k-\ell}$ and $r_{n-\ell-1} \neq r_{n-\ell}$ at Instruction 6 which must be implemented without revealing information about ℓ.

Algorithm 17 $\mathsf{RandPerm}_U$

Input: : integers n, $k < n$, ℓ, $J \subseteq [0,n)$, $\mathbf{z} \in \mathbb{F}^n$ ▷ assume $\# J \leq n-k$
Output: : $(\pi(i))_{0 \leq i < n}$, $\mathbf{y} = \mathbf{z}^\pi$ with $\pi \in \mathfrak{S}_n$ such that $\# \, \pi([0,k)) \cap J = \ell$

repeat
  for $i = 0, \dots, n-1$ **do**
    $r_i \xleftarrow{\$} [0, 2^B)$ ▷ B an integer, external parameter
    $r_i \leftarrow r_i + (i \in J) \; ? \; 2^B : 0$ ▷ coordinates in J will be last after sorting
  $\mathbf{x} \leftarrow \mathsf{sort}\left((r_i, i, z_i)_{0 \leq i < n}\right)$ ▷ sort according to r_i in increasing order
until $r_{k-\ell-1} \neq r_{k-\ell}$ and $r_{n-\ell-1} \neq r_{n-\ell}$
for $i = 0, \dots, k-1$ **do** ▷ fix size loop for constant time
  $(x_{k-1-i}, x_{n-1-i}) \leftarrow (i < \ell) \; ? \; (x_{n-1-i}, x_{k-1-i}) : (x_{k-1-i}, x_{n-1-i})$ ▷ swap if $i < \ell$
for $i = 0, \dots, n-1$ **do**
  $(*, \pi(i), y_i) \leftarrow x_i$
return $(\pi, \mathbf{y})$

B.3.4. *Permuting Vectors.* It is also possible to permute a vector according to a permutation by sorting. If the input permutation π is given as a sequence of integers $(\pi(i))_{0 \leq i < n}$, sorting this sequence will apply the inverse permutation. Interestingly, applying the inverse of a given permutation is what is needed most of the time in Wave. As an additional (almost free) feature, Algorithm 18 may return the inverse permutation as a sequence of integers.

Algorithm 18 VectInvPerm – Apply (the Inverse of) a Permutation to a Vector

function $\mathsf{VectInvPerm}(\mathbf{z}, \pi)$ ▷ $\mathbf{z} \in \mathbb{F}^n$, $\pi \in \mathfrak{S}_n$
  $\mathbf{x} \leftarrow \mathsf{sort}\left((\pi(i), i, z_i)_{0 \leq i < n}\right)$ ▷ sort according to first coordinate in increasing order
  for $i = 0, \dots, n-1$ **do**
    $(*, \sigma(i), y_i) \leftarrow x_i$
  return $\mathbf{y}, \sigma$ ▷ $\mathbf{y} = \mathbf{z}^{\pi^{-1}}$ $\sigma = \pi^{-1}$

B.3.5. *Permuting the Columns of a Matrix.* As for the coordinates of a vector, permuting the columns of a matrix can be achieved in constant time with an oblivious sorting algorithm. The principle of Algorithm 18 can be applied, but with an alphabet $\mathbb{F}$ large enough to accommodate the columns of a matrix. The reference implementation uses an ad hoc procedure deriving from `djbsort` [Ber] to produce a sorting network and uses it to conditionally swap the columns of a matrix.

B.4. **Master Key to Generate the Secret Matrices.** As the key generation (Algorithm 4) is described, the secret matrices $\mathbf{H}_U$ and $\mathbf{G}_V$ are needed only once per signature in Algorithm 5, when a Gaussian elimination is performed. Columns are permuted just before the elimination. Suppose that a master key mk is used for generating matrices $\mathbf{H}_U$ and $\mathbf{G}_V$, its i-th column is equal to $f_{\mathsf{mk}}(\mathbf{X}, i)$, $\mathbf{X} \in \{\mathbf{H}_U, \mathbf{G}_V\}$ where $f_{\mathsf{mk}}()$ is some, appropriately keyed (here mk), one-way function. The i-th column of $\mathbf{X}^\pi$ is equal to $f_{\mathsf{mk}}(\mathbf{X}, \pi(i))$ and the matrix $\mathbf{X}^\pi$ can be generated without leaking information about π assuming that the execution of $f_{\mathsf{mk}}(\mathbf{X}, i)$ leaks no information on i. For instance assuming we want to generate $\mathbf{G}_V$ and $\mathbf{H}_U$ at run time (using $\mathsf{prng}_3()$ as in Algorithm 12):

- the i-th column of $\mathbf{G}_V$ is the output of $\mathsf{prng}_3(k_V, \mathsf{mk} \parallel 0 \parallel i)$
- the i-th column of $\mathbf{H}_U$ is the output of $\mathsf{prng}_3(n/2 - k_U, \mathsf{mk} \parallel 1 \parallel i)$

where $\parallel$ denotes the concatenation. Concatenating a '0' or a '1' separates the pseudorandom generator domains for cases 'V' and 'U'.

B.5. **Gaussian Elimination and its Variants in Constant Time.** It is assumed that the reader is familiar with the basic concepts. The Gaussian elimination on an $r \times n$ full rank matrix $\mathbf{A}$ will produce another matrix $\mathbf{A}'$ spanning the same vector space and which contains an $r \times r$ identity sub-matrix, indexed by some set J. The pivot positions are the elements of J, ordered from top to bottom. The pivots are selected in order one at at time. A failing pivot is a column (position) that is dependent of the previously selected pivot columns and that could not be included in the identity block. Note that for a given information set J the transformed matrix is essentially unique. Changing the order of the pivots in J will simply change the row order accordingly in the resulting matrix. Recall that, M_i denotes the i-thm row of the matrix $\mathbf{M}$.

B.5.1. *Systematic Gaussian Elimination.* In the key generation routine, Algorithm 4, the purpose is to reach a (strict) systematic form for an input matrix of size $r \times n$ (assuming it has full rank r), that is a matrix $\mathbf{A}_{\text{syst}} = (\mathbf{Id}_r \mid \mathbf{R})$ where $\mathbf{Id}_r$ is the $r \times r$ identity matrix and such that $\mathbf{A}$ and $\mathbf{A}_{\text{syst}}$ span the same row vector space. This systematic form exists if and only if the leftmost $r \times r$ block of $\mathbf{A}$ is non-singular. Else a few columns of $\mathbf{A}$ may have to be permuted to reach the desired form.

In the current context the columns of a matrix $\mathbf{A}$ are permuted with $\sigma \in \mathfrak{S}_n$ before the Gaussian elimination and the call in Instruction 9 of Algorithm 4 is

$$(\mathbf{A}_{\text{syst}} = (\mathbf{Id}_r \mid \mathbf{R}), \pi) \leftarrow \mathsf{SystGaussElim}(\mathbf{A}, \sigma)$$

where $\mathbf{A}^\pi$ and $\mathbf{A}_{\text{syst}}$ span the same row-space and $\pi \in \mathfrak{S}_n$ is "close" to σ; the Gaussian elimination uses the columns of $\mathbf{A}^\sigma$ as pivots from left to right and "push" the failing pivot positions right, yielding

$$\forall i \in [0, r),\ \pi(i) = \sigma(\ell) \text{ where } \ell = \min\left\{ j \in [0, n) \mid \operatorname{rank}\left(\mathbf{A}^\sigma_{[0,j)}\right) = i \right\}$$

and

$$\forall i \in [0, r)n,\ \pi(i) = \sigma(\ell) \text{ where } \ell = \min\left\{ [0, n) \setminus \{\sigma^{-1} \circ \pi(j), j \in [0, i)\} \right\}.$$

Key security. If $(\mathbf{A}, \pi)$ is private and $\mathbf{A}_{\text{syst}}$ is public and obtained as above, it should be noted that π is not uniformly distributed in $\mathfrak{S}_n$. However, no security penalty incurs as $\mathbf{A}_{\text{syst}}$ would be the result of the Gaussian elimination on $\mathbf{S}\mathbf{A}^\sigma$ for any non-singular $r \times r$ matrix $\mathbf{S}$. Revealing $\mathbf{S}\mathbf{A}^\sigma$ for random uniform σ and $\mathbf{S}$ is innocuous (within the security assumptions) and publishing $\mathbf{A}_{\text{syst}}$ rather than $\mathbf{S}\mathbf{H}^\sigma$ provides an information that anyone could have computed easily from public data.

Implementation security. Still the implementation of the Gaussian elimination could give rise to timing or cache attacks. For instance, if $\mathbf{A} = \mathbf{H}$ as in Instruction 7 of Algorithm 4, the special form and content of $\mathbf{H}$ could induce secret dependent timing variations, typically because some columns of $\mathbf{d} \star \mathbf{H}_U$ or $\mathbf{b} \star \mathbf{H}_U$ can be null depending on secret information. This could be easily avoided by multiplying $\mathbf{H}$ on the left by a random non singular matrix $\mathbf{S}$ before the Gaussian elimination, or, probably less expensive, by a careful implementation of the Gaussian elimination, making sure that the timing and memory access pattern of each pivot elimination is independent of the coordinates of this particular column. Note that timing or memory access variations due to pivot failures do not need to be masked as they would also happen if the Gaussian elimination was applied to $\mathbf{SH}^{\sigma}$, and this latter matrix could be safely revealed.

Basic Gaussian elimination in constant time. Algorithm 19 uses the constant time routine $\mathsf{Reduce}_j()$ defined in §B.5.2. This algorithm will be used in a context (Instruction 9 in Algorithm 4) where it can be not oblivious to pivot failure, but oblivious to everything else: an adversary can only learn which column indices led to a failing pivot. Indeed, this algorithm will be used to output the public key, when revealing pivot failures it only reveals positions where a Gaussian elimination may fail on the public matrix.

Algorithm 19 Systematic Gaussian Elimination in constant time

```
function CTSystGaussElim(M, π)                                  ▷ M ∈ F_3^{k×n}
    j ← 0
    ℓ ← n − 1
    while j < k and j < ℓ do
        for i = j + 1, ..., k − 1 do
            M_j, M_i ← Reduce_j(M_j, M_i)                       ▷ defined in §B.5.2
        if M_{j,j} ≠ 0 then
            M_j ← M_{j,j}^{-1} M_j
            for i = 0, ..., j − 1 do
                M_i ← M_i − M_{i,j} M_j
            j ← j + 1
        else
            M ← SwapColumns(M, j, ℓ)
            ℓ ← ℓ − 1
            π ← π ∘ (j ℓ)                                       ▷ π(j), π(ℓ) ← π(ℓ), π(j)
    return (M, π)
```

Gaussian elimination with abort in constant time. Algorithm 20 is a variant of Algorithm 19 but with abort. Algorithm 20 will be used in a context, the generation of $\mathbf{H}_V$ in Instruction 2 of Algorithm 4, where rejection is not allowed unless to weaken the security reduction. Basically, $\mathbf{H}_V$ is computed by performing a Gaussian elimination of $\mathbf{G}_V$ and revealing the failing pivot indices may reveal information on the particular permutation drawn for a secret matrix generation. It is why the algorithm always compute pivots with Instructions 5 and 6 (defined in §B.5.2). The computation of these pivots may fail. It is why the algorithm also outputs some integer p. Checking that p is equal to k or not enables to verify the success or not of the Gaussian elimination.

Partial Gaussian elimination in constant time. The constant time partial Gaussian elimination on $\mathbf{M} \in \mathbb{F}_3^{k\times n}$ will stop the process g steps before the end, that is after $k-g$ pivots in constant time (Instructions 5 and 6). In the current context Algorithm 21 will be called with a value of g such the first $k-g$ pivots may fail (with a probability $\approx 1/2^{64}$) but the algorithm will stop in any case after $k-g$ iterations. The algorithm also outputs an integer p which gives the number of successful pivots. Checking that p is equal or not to $k-g$ enables to verify the success or not of the partial Gaussian elimination.

Algorithm 20 Gaussian Elimination “with Abort” in constant time

function $\mathsf{CTGE}_{\mathsf{abort}}(\mathbf{M})$ ▷ $\mathbf{M} \in \mathbb{F}_3^{k\times n}$
  $p \leftarrow 0$
  for $\ell = 0, \ldots, k-1$ **do**
    for $i = \ell+1, \ldots, k-1$ **do**
      $M_\ell, M_i \leftarrow \mathsf{Reduce}_\ell(M_\ell, M_i)$ ▷ defined in §B.5.2
    $M_\ell \leftarrow \mathsf{Normalize}_\ell(M_\ell)$
    $p \leftarrow p + M_{\ell,\ell}$ ▷ Addition in $\mathbb{Z}$
    for $i = 0, \ldots, \ell-1$ **do**
      $M_i \leftarrow M_i - M_{i,\ell}\, M_\ell$
  return $(\mathbf{M}, p)$

Algorithm 21 Partial Gaussian Elimination in constant time

function $\mathsf{CTPartialGaussElim}(\mathbf{M}, g)$ ▷ $\mathbf{M} \in \mathbb{F}_3^{k\times n}$
  $p \leftarrow 0$
  for $\ell = 0, \ldots, k-g-1$ **do**
    for $i = \ell+1, \ldots, k-1$ **do**
      $M_\ell, M_i \leftarrow \mathsf{Reduce}_\ell(M_\ell, M_i)$ ▷ defined in §B.5.2
    $M_\ell \leftarrow \mathsf{Normalize}_\ell(M_\ell)$
    $p \leftarrow p + M_{\ell,\ell}$ ▷ Addition in $\mathbb{Z}$
    for $i = 0, \ldots, \ell-1$ **do**
      $M_i \leftarrow M_i - M_{i,\ell} \cdot M_\ell$
  return $(\mathbf{M}, p)$

Application to Algorithm 6 ($\mathsf{CTDecode}_V$)**.** As Decode_V was specified in Algorithm 6, the matrix $\mathbf{G}_V$ is permuted, then a g-partial systematic form is computed. For a constant time implementation failure and rejection is not allowed. To this end, Instruction 6 will be replaced by

$$(\mathbf{G}, p) \leftarrow \mathsf{CTPartialGaussElim}(\mathbf{G}_V^\pi, g)$$

which tries to compute the permuted matrix $\mathbf{G}_V^\pi$, possibly using a master key mk, in a g-partial systematic form. Checking that p is equal to $k_V - g$ will enable to test if the algorithm has been successful. It will lead to a constant time specification of Decode_V as $\mathsf{CTDecode}_V$ in Algorithm 27.

B.5.2. *Extended Gaussian Elimination in Constant Time.* The Gaussian elimination algorithms in constant time proposed above are not sufficient for the signing algorithm, in particular Algorithm 7 which needs to compute the g-extended systematic form (see Definition 2) of some matrix. In this context, rejection is also not allowed without weakening the security reduction, and revealing the failing pivot indices may reveal information on the particular permutation drawn for a particular signature generation. We must produce an algorithm which is completely oblivious to its input. This is achieved by adding zero rows at the bottom of the matrix (according to Lemma 2 in Appendix §D, a matrix $\mathbf{A} \in \mathbb{F}_3^{(r+g)\times n}$ in extended systematic form has exactly g zero rows) and then performing a Gaussian elimination oblivious to the failing pivots.

Algorithm 22, on input $(\mathbf{A}', g)$ with $\mathbf{A}' \in \mathbb{F}_3^{r\times n}$, tries to compute an extended systematic form of $\mathbf{A}'$ by first adding g extra (zero) rows to form a matrix $\mathbf{A} \in \mathbb{F}_3^{(r+d)\times n}$. It succeeds if and only if the first $r+g$ columns of $\mathbf{A}$ have the same rank as $\mathbf{A}'$. If it fails, the output $\mathbf{A}$ will be such that $\langle \mathbf{A}' \rangle = \langle \mathbf{A} \rangle$ but it won't be in extended systematic form; the weight

$$\#\{i \in [0, r+g),\ A_{i,i} \neq 0\}$$

of the main diagonal will be equal to $\operatorname{rank}(\mathbf{A}'_{[0,r+g)}) < r$.

Algorithm 22 Extended Gaussian Elimination in constant time

1: **function** $\mathsf{CTExtGaussElim}(\mathbf{A}', g)$ $\qquad\triangleright\ \mathbf{A}' \in \mathbb{F}_3^{r\times n},\ g \leq n-r$

2: $\quad\mathbf{A} \leftarrow \mathsf{stack}(\mathbf{A}', \mathbf{0}^{g\times n})$ $\qquad\triangleright\ \mathbf{A} = \begin{array}{|c|}\hline \mathbf{A}' \\ \hline \mathbf{0} \\ \hline \end{array}\begin{array}{l} r \text{ rows} \\ g \text{ rows} \end{array}$

3: $\quad p \leftarrow 0$

4: $\quad$**for** $\ell = 0, \ldots, r+g-1$ **do**

5: $\quad\quad$**for** $i = \ell+1, \ldots, r+g-1$ **do**

6: $\quad\quad\quad A_\ell, A_i \leftarrow \mathsf{Reduce}_\ell(A_\ell, A_i)$

7: $\quad\quad A_\ell \leftarrow \mathsf{Normalize}_\ell(A_\ell)$

8: $\quad\quad p \leftarrow p + A_{\ell,\ell}$ $\qquad\triangleright$ Addition in $\mathbb{Z}$

9: $\quad\quad$**for** $i = 0, \ldots, \ell-1$ **do**

10: $\quad\quad\quad A_i \leftarrow A_i - A_{i,\ell}\cdot A_\ell$

11: $\quad$**return** $(\mathbf{A}, p)$ $\qquad\triangleright\ \mathbf{A} \in \mathbb{F}_3^{(r+g)\times n}$

The functions $\mathsf{Normalize}_\ell$ and Reduce_ℓ are defined as

$$\begin{aligned}
\mathsf{Normalize}_\ell(\mathbf{x}) &:= \begin{cases} \mathbf{x} & \text{if } x_\ell = 0 \\ x_\ell^{-1}\cdot\mathbf{x} & \text{if } x_\ell \neq 0 \end{cases} \\
\mathsf{Reduce}_\ell(\mathbf{x}, \mathbf{y}) &:= \begin{cases} (\mathbf{y}, \mathbf{x}) & \text{if } x_\ell = 0 \\ (\mathbf{x}, \mathbf{y} - (y_\ell x_\ell^{-1})\cdot\mathbf{x}) & \text{if } x_\ell \neq 0 \end{cases}
\end{aligned}$$

Both functions enjoy simple secure implementations, for instance, using properties of $\mathbb{F}_3$:

$$\begin{aligned}
\mathsf{Normalize}_\ell(\mathbf{x}) &= (1 + x_\ell - x_\ell^2)\cdot\mathbf{x} \\
\mathsf{Reduce}_\ell(\mathbf{x}, \mathbf{y}) &= (x_\ell^2\cdot\mathbf{x} + (1 - x_\ell^2)\cdot\mathbf{y}, (1 - x_\ell^2 - x_\ell y_\ell)\cdot\mathbf{x} + x_\ell^2\cdot\mathbf{y})
\end{aligned}$$

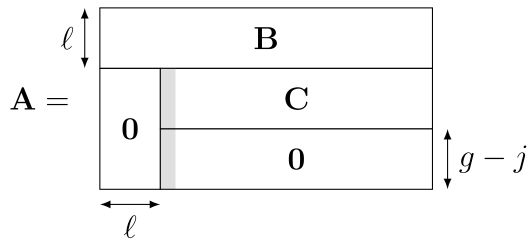
and thus the whole procedure can be securely implemented.

Proposition 6. *If $\mathbf{A}' \in \mathbb{F}_3^{r\times n}$ is such that $r = \operatorname{rank}(\mathbf{A}') = \operatorname{rank}\left(\mathbf{A}'_{[0,r+g)}\right)$, then the output $\mathbf{A}$ of Algorithm 22 on input $(\mathbf{A}', g)$ is an extended systematic form of $\mathbf{A}'$.*

Proof.

(1) Let us first remark that if $(\mathbf{x}', \mathbf{y}') = \mathsf{Reduce}_\ell(\mathbf{x}, \mathbf{y})$ then $(\mathbf{x}', \mathbf{y}')$ spans the same space as $(\mathbf{x}, \mathbf{y})$ and $y'_\ell = 0$. The matrix $\mathbf{A}$ is modified in place. Initially it spans the same vector space as $\mathbf{A}'$, and this property remains true during all the computation, as Instruction 6, 7, and 10 will not change the spanned space. So we have $\langle \mathbf{A} \rangle = \langle \mathbf{A}' \rangle$ at all times. It remains to prove that $\mathbf{A}$ is in extended systematic form.

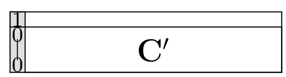
(2) Let us prove by induction on ℓ that at the beginning of the ℓ-th iteration the matrix $\mathbf{A}$ has the following form



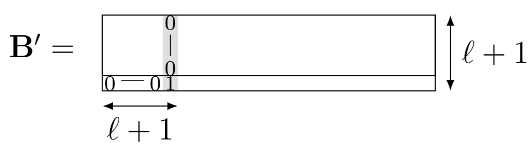
(18)

with $\mathbf{B} \in \mathbb{F}_3^{\ell \times n}$ in extended systematic form and $j = \ell - \mathrm{rank}(\mathbf{B})$. It is true for $\ell = 0$, with $\mathbf{C} = \mathbf{A}'$ and $\mathbf{B}$ has vanished. During the ℓ-th iteration, the algorithm will explore the ℓ-th column of $\mathbf{A}$ below $\mathbf{B}$ (in grey above).

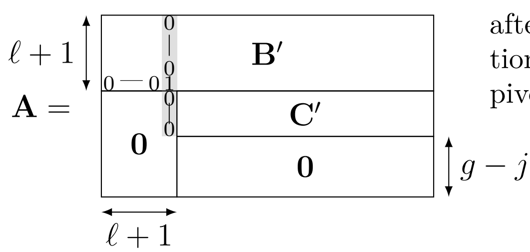
- First case: the leftmost column of $\mathbf{C}$ has a non-zero coefficient. Instructions 4, 5 and 6 will transform $\mathbf{C}$ into a matrix of same size, spanning the same space, of the following form:



The remainder of $\mathbf{A}$ is unchanged by Instructions 5, 6 and 7. Instructions 8 and 9 will eliminate the ℓ-th column of $\mathbf{B}$ by removing a multiple of A_ℓ, the ℓ-th row of $\mathbf{A}$ (also the top row of $\mathbf{C}$) is added at the bottom of $\mathbf{B}$ leading, after elimination to a matrix $\mathbf{B}' \in \mathbb{F}_3^{(\ell+1)\times n}$ of the form



If $\mathbf{B}$ is in extended systematic form, so is $\mathbf{B}'$. Finally, at the end of the ℓ-th iteration, the matrix $\mathbf{A}$ becomes



after the ℓ-th iteration with successful pivot

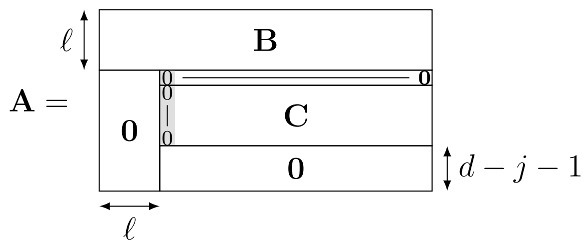
which is compliant with the induction condition at the start of the $(\ell+1)$-th iteration.

- Second case: the leftmost column of $\mathbf{C}$ is null. The following lemma is needed.

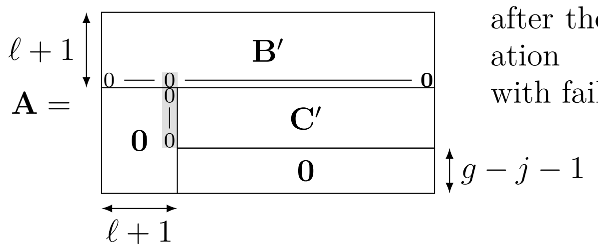
Lemma 1. *In (18), if the first column of* $\mathbf{C}$ *is null then* $g - j > 0$.

Proof. The first $r+g$ column of $\mathbf{B}$ have rank $\ell - j = \operatorname{rank}(\mathbf{B})$. To comply with the rank condition of the statement, the first $r+g$ columns of $\mathbf{A}$ must have rank $r = \operatorname{rank}(\mathbf{A})$, thus the first $r+g-\ell$ columns of $\mathbf{C}$ must have rank $r - \operatorname{rank}(\mathbf{B}) = r + j - \ell = \operatorname{rank}(\mathbf{C})$, and in particular $r+j-\ell \geq r+g-\ell$, that is $g \geq j$. If $g = j$ then the first $r+g-\ell$ columns of $\mathbf{C}$ would have rank $r+g-\ell$. This cannot happen if the first column of $\mathbf{C}$ is null, hence $g > j$. $\square$

Instructions 5, 6 and 7 will move $\mathbf{C}$ down by one row and, because $g - j > 0$, insert one of the $g - j$ all-zero bottom rows as the new ℓ-th row



With A_ℓ null now, Instructions 8 and 9 do nothing and at the end of the ℓ-th iteration the matrix $\mathbf{A}$ becomes



after the ℓ-th iteration with failing pivot

Since $\mathbf{B}'$ is obtained by adding an all-zero row at the bottom of $\mathbf{B}$, it remains in extended systematic form. The rank default of $\mathbf{B}'$ is increased by one and becomes $j + 1 = \ell + 1 - \operatorname{rank}(\mathbf{B}')$.

When $\ell = r - g$, everything vanishes except $\mathbf{B}$ in (18), and thus $\mathbf{A}$ is in extended systematic form.

$\square$

Application to Algorithm 7 ($\mathsf{CTDecode}_U$)**.** As Decode_U was specified in Algorithm 7, the matrix $\mathbf{H}_U$ is permuted, then a g-extended systematic form is computed. For a constant time implementation failure and rejection is not allowed. To this end, Instruction 13 will be replaced by

$$(\mathbf{H}, p) \leftarrow \mathsf{CTExtGaussElim}(\mathbf{H}_U^\pi, g)$$

which tries to compute the permuted matrix $\mathbf{H}_U^\pi$, possibly using a master key mk, in a g-extended systematic form. Checking that p is equal to $n/2 - k_U$ will enable to test if the algorithm has been successful. It will lead to a constant time specification of Decode_U as $\mathsf{CTDecode}_U$ in Algorithm 28.

B.6. **Wave Key Generation and Signing in Constant Time.** Algorithms 23 and 26 are a specification of Wave Key Generation and Signing in constant time. Contrary to the specifications of the main part of the document, it is specified how to produce matrices $\mathbf{G}_V$ and $\mathbf{H}_U$ thanks to a master key (see §B.4), how to sample permutations (see §B.3), how to perform Gaussian elimination and its variants (see §B.5) as well as sampling random elements in their domain (see §B.2). Let us stress that all algorithms are implementation choices, there are a particular instantiation of algorithms presented in the main part of the document and they were specified to reach a constant time implementation.

Algorithm 23 Key Generation in constant time

Output: $\begin{cases} \mathsf{pk} & = & \mathbf{M} \in \mathbb{F}_3^{k\times(n-k)} \\ \mathsf{sk} & = & (\mathsf{mk}, \mathbf{b}, \mathbf{c}, \pi) \in \mathbb{F}_3^{\lambda} \times \mathbb{F}_3^{n/2} \times \mathbb{F}_2^{n/2} \times \mathfrak{S}_n \end{cases}$

repeat
- $\mathsf{mk} \xleftarrow{\$} \{0,1\}^{\lambda}$ ▷ π_{id} the identity permutation
- $\mathbf{G} \leftarrow \mathsf{RandMatPerm}(k_V, n/2, \mathsf{mk} \parallel 0, \pi_{id})$ ▷ Algorithm 24
- $(\mathbf{G}_V, p) \leftarrow \mathsf{CTGE}_{\mathsf{abort}}(\mathbf{G})$ ▷ Algorithm 21

until $p = k_V$

$(\mathbf{Id}_{k_V} \mid \mathbf{R}_V) \leftarrow \mathbf{G}_V$

$\mathbf{H}_V \leftarrow (-\mathbf{R}_V^{\top} \mid \mathbf{Id}_{n/2-k_V})$

$\mathbf{b} \xleftarrow{\$} \mathbb{F}_3^{n/2}$

$\mathbf{c} \xleftarrow{\$} (\mathbb{F}_3 \setminus \{0\})^{n/2}$

$\mathbf{d} \leftarrow \mathbf{1} + \mathbf{b} \star \mathbf{c}$

$\pi, \mathbf{x} \leftarrow \mathsf{RandPerm}(n, (\mathbf{d}, -\mathbf{b}))$ ▷ Algorithm 15, $\mathbf{x} = (\mathbf{d}, -\mathbf{b})^{\pi}$

$(\mathbf{M}, \pi) \leftarrow \mathsf{WavePublicKey}(\mathsf{mk}, \mathbf{H}_V, \pi, \mathbf{x}, \mathbf{c})$ ▷ Algorithm 25, π may change

return $(\mathsf{pk} = \mathbf{R}, \mathsf{sk} = (\mathsf{mk}, \mathbf{b}, \mathbf{c}, \pi))$

Algorithm 24 Pseudorandom (Permuted) Matrix

function $\mathsf{RandMatPerm}(n, k, \mathsf{seed}, \pi)$
- **for** $i = 0, \dots, n-1$ **do**
  - $\mathbf{x}_i \leftarrow \mathsf{prng}_3(k, \mathsf{seed} \parallel \pi(i))$ ▷ Algorithm 12 (p. 29)
- **return** $\mathsf{MatrixFromColumns}(\mathbf{x}_0, \dots, \mathbf{x}_{n-1})$

Algorithm 25 Public Key Computation

function WavePublicKey(mk, $\mathbf{G}_V, \pi, \mathbf{x}, \mathbf{c}$)
  for $i = 0, \ldots, n-1$ **do**
    $j \leftarrow \pi(i) \mod (n/2)$
    $\mathbf{y}_i \leftarrow \mathsf{prng}_3(n/2 - k_U, \mathsf{mk} \parallel 1 \parallel j)$ ▷ Algorithm 12 (p. 29)
    $\mathbf{y}_i \leftarrow x_i \cdot \mathbf{y}_i$
  $\mathbf{H}_{\mathsf{up}} \leftarrow \mathsf{MatrixFromColumns}(\mathbf{y}_0, \ldots, \mathbf{y}_{n-1})$
  $\mathbf{H}_{\mathsf{down}} \leftarrow \mathsf{MatPerm}((-\mathbf{c} \star \mathbf{H}_V \mid \mathbf{H}_V), \pi)$ ▷ Oblivious permutation, §B.3.5
  $\mathbf{H}_{\mathsf{sec}} \leftarrow \begin{pmatrix} \mathbf{H}_{\mathsf{up}} \\ \hline \mathbf{H}_{\mathsf{down}} \end{pmatrix}$
  $((\mathbf{Id}_{n-k} \mid \mathbf{R}), \pi) \leftarrow \mathsf{SystGaussElim}(\mathbf{H}_{\mathsf{sec}}, \pi)$ ▷ Algorithm 19, π may change
  $\mathbf{M} \leftarrow \mathbf{M}(\mathbf{R})$ ▷ As in Definition 3
  return $(\mathbf{M}, \pi)$

Algorithm 26 Wave Signature

Input: : a message $\mathbf{m}$, $\mathsf{sk} = (\mathsf{mk}, \mathbf{b}, \mathbf{c}, \pi)$
Output: : $(\mathbf{s}, \mathtt{salt})$
repeat
  $\mathtt{salt} \xleftarrow{\$} \mathbb{F}_2^{2\lambda}$ ▷ $\lambda \in \{128, 192, 256\}$ denotes the security levels
  $\mathbf{x} \leftarrow \mathsf{Hash}(\mathbf{m} \parallel \mathtt{salt})$ ▷ $\mathbf{x} \in \mathbb{F}_3^{n-k}$
  $\mathbf{y} = (\mathbf{y}_L \parallel \mathbf{y}_R), \sigma \leftarrow \mathsf{VectInvPerm}((\mathbf{x} \parallel \mathbf{0}^k), \pi)$ ▷ Algorithm 18, $\mathbf{y} = (\mathbf{x} \parallel \mathbf{0}^k)^\sigma$, $\sigma = \pi^{-1}$
  $\mathbf{y}_V \leftarrow \mathbf{y}_R - \mathbf{c} \star \mathbf{y}_L$
  $\mathbf{e}_V \leftarrow \mathsf{CTDecode}_V(\mathbf{y}_V, \mathsf{mk})$ ▷ Algorithm 27
  $\mathbf{y}_U \leftarrow \mathbf{y}_L - \mathbf{b} \star \mathbf{y}_V$ ▷ simplification of $\mathbf{y}_U = (\mathbf{1} + \mathbf{c} \star \mathbf{b}) \star \mathbf{y}_L - \mathbf{b} \star \mathbf{y}_R$
  $\mathbf{e}_U \leftarrow \mathsf{CTDecode}_U(\mathbf{y}_U, \mathbf{e}_V, \mathbf{b}, \mathbf{c}, \mathsf{mk})$ ▷ Algorithm 28
  $\mathbf{e}_L \leftarrow \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$
  $\mathbf{e}_R \leftarrow \mathbf{c} \star \mathbf{e}_L + \mathbf{e}_V$ ▷ simplification of $\mathbf{e}_R = \mathbf{c} \star \mathbf{e}_U + (\mathbf{1} + \mathbf{b} \star \mathbf{c}) \star \mathbf{e}_V$
until $(|\mathbf{e}_V|, n/2 - w + |\mathbf{e}_L \star \mathbf{e}_R|) \in \mathsf{Accept}$
$\mathbf{e}, * \leftarrow \mathsf{VectInvPerm}((\mathbf{e}_L \parallel \mathbf{e}_R), \sigma)$ ▷ Algorithm 18, $\sigma = \pi^{-1}$
$\mathbf{s} \leftarrow \mathbf{e}_{[n-k,n)}$
return $(\mathbf{s}, \mathtt{salt})$

Appendix C. Key Representation and Compressing Signatures

C.1. **Key Representation.** To store public and private keys as byte arrays, we need to convert trits into bytes efficiently. We do this using a simple arithmetic encoding, packing five trits $v_0, \ldots, v_4$ in $\{0, 1, 2\}$ into the one-byte integer $v_0 + 3v_1 + 9v_2 + 27v_3 + 81v_4$. This approach uses 242 of the 256 values that one byte can take, giving a storage efficiency of $242/256 = 94.5\%$. We could achieve a higher efficiency by packing more trits together in larger words, at the cost of more computation (and the specification of a byte order). On balance, we decided that five-trits-per-byte provides a good trade-off between storage efficiency, computational cost, and specification complexity.

C.2. **Compressing Signatures.** As with keys, Wave signatures can be stored and transmitted using the five-trits-per-byte encoding above. This would result in a signature size of, e.g., $\lceil 4288/5 \rceil = 845$ bytes (plus 32 salt bytes) for Wave822 (at NIST security Level I).

Algorithm 27 Decode$_V$ in constant time

function $\mathsf{CTDecode}_V(\mathbf{y}_V, \mathsf{mk})$
  $t \xleftarrow{\mathcal{D}_V} [0, k_V - g]$
  repeat
    $(\pi, \mathbf{y}) \leftarrow \mathsf{RandPerm}_V(n/2, k_V - g, t, \mathbf{y}_V)$ ▷ Algorithm 16
    $\mathbf{G}_V \leftarrow \mathsf{RandMatPerm}(n/2, k_V, \mathsf{mk} \parallel 0, \pi)$ ▷ Algorithm 24
    $(\mathbf{G}, p) \leftarrow \mathsf{CTPartialGaussElim}(\mathbf{G}_V, g)$ ▷ Algorithm 21
  until $p = k_V - g$
  $\mathbf{x} \xleftarrow{\$} (\mathbb{F}_3 \setminus \{0\})^t \times \{0\}^{k_V - g - t} \times \mathbb{F}_3^g$
  $\mathbf{e} \leftarrow \mathbf{y} + (\mathbf{x} - \mathbf{y}^{(0)})\mathbf{G}$ ▷ $\mathbf{y} = (\mathbf{y}^{(0)}, \mathbf{y}^{(1)}) \in \mathbb{F}_3^{n/2}$ with $\mathbf{y}^{(0)} \in \mathbb{F}_3^{k_V - g}$
  $\mathbf{e}_V, * \leftarrow \mathsf{VectInvPerm}(\mathbf{e}, \pi)$ ▷ Algorithm 18 (p. 33), $\mathbf{e}_V = \mathbf{e}^{\pi^{-1}}$
  return $\mathbf{e}_V$

Algorithm 28 Decode$_U$ in constant time

function $\mathsf{CTDecode}_U(\mathbf{y}_U, \mathbf{e}_V, \mathbf{b}, \mathbf{c}, \mathsf{mk})$
  $t \leftarrow |\mathbf{e}_V|$
  $\ell \xleftarrow{\mathcal{D}_U(t)} [0, t]$
  repeat
    $\mathbf{x} = \mathsf{pack}(\mathbf{y}_V, (\mathbf{c} - \mathbf{b}) \star \mathbf{e}_V, \mathbf{e}_V \star \mathbf{e}_V)$
    $(\pi, \mathbf{x}) \leftarrow \mathsf{RandPerm}_U(n/2, n/2 - k_U + g, \ell, \mathrm{Supp}(\mathbf{e}_V), \mathbf{x})$ ▷ Algorithm 17
    $\mathbf{y}, \mathbf{v}, \mathbf{s} \leftarrow \mathsf{unpack}(\mathbf{x})$ ▷ $\mathbf{y} = \mathbf{y}_V^{\pi}$; $\mathbf{v} = ((\mathbf{c} - \mathbf{b}) \star \mathbf{e}_V)^{\pi}$; $\mathbf{s} = (\mathbf{e}_V \star \mathbf{e}_V)^{\pi}$
    $\mathbf{H}_U \leftarrow \mathsf{RandMatPerm}(n/2, n/2 - k_U, \mathsf{mk} \parallel 1, \pi)$ ▷ Algorithm 24
    $(\mathbf{H}, p) \leftarrow \mathsf{CTExtGaussElim}(H_U, d)$ ▷ Algorithm 22
  until $p = n/2 - k_U$
  repeat
    $\mathbf{z}^{(0)} \leftarrow (H_{i,i})_{0 \le i < n/2 - k_U + g}$
    $\mathbf{e}^{(0)} \xleftarrow{\$} \mathbb{F}_3^{n/2 - k_U + g}$
    $\mathbf{e}^{(0)} \leftarrow (\mathbf{1} - \mathbf{z}^{(0)}) \star \mathbf{e}^{(0)}$
    $\mathbf{e}^{(1)} \xleftarrow{\$} (\mathbb{F}_3 \setminus \{0\})^{k_U - g}$
    $\mathbf{e}^{(1)} \leftarrow \mathbf{v}^{(1)} + (\mathbf{1} - \mathbf{s}^{(1)}) \star \mathbf{e}^{(1)}$
    $\mathbf{e}^{(0)} \leftarrow \mathbf{e}^{(0)} + (\mathbf{y} - (\mathbf{e}^{(0)} \parallel \mathbf{e}^{(1)}))\mathbf{H}^{\top}$
    $i \leftarrow |\mathbf{s}^{(0)} \star \mathbf{e}^{(0)} - \mathbf{v}^{(0)}|$
    $j \leftarrow n/2 - k_U + g - \ell - |(\mathbf{1} - \mathbf{s}^{(0)}) \star \mathbf{e}^{(0)}|$
  until $(2j + i = n - w)$ ▷ check final weight
  $\mathbf{e}_U, * \leftarrow \mathsf{VectInvPerm}(\mathbf{e}, \pi)$ ▷ Algorithm 18, $\mathbf{e}_U = \mathbf{e}^{\pi^{-1}}$
  return $\mathbf{e}_U$

▷ $\left\{ \begin{array}{l} \text{all vectors } \mathbf{x} \in \{\mathbf{y}, \mathbf{v}, \mathbf{s}, \mathbf{e}\} \text{ split as } \mathbf{x} = (\mathbf{x}^{(0)} \parallel \mathbf{x}^{(1)}) \text{ with} \\ \mathbf{x}^{(0)} \text{ of length } n/2 - k_U + g \text{ and } \mathbf{x}^{(1)} \text{ of length } k_U - g, \\ \text{the vector } \mathbf{z}^{(0)} \text{ has length } n/2 - k_U + g. \end{array} \right.$

But Wave signature vectors have high weight by definition, so their true entropy is lower than that of a random ternary vector. This makes signature compression a viable option. We considered several standard compression techniques, including arithmetic coding and combinadics, before settling on Huffman coding.

We use a static Huffman encoding to encode three trits at a time, with codewords based on the expected average weight of a signature. Therefore, no information about the encoding needs to be transmitted with a compressed signature: hard-coded encoding and decoding routines can be used (see `util/compress.c` in the reference implementation).

The lossless compression of the signature involves only public information, so has no impact on the security of the overall signature scheme. This encoding is purely a trade-off between computational effort and resulting signature length. Better compression may be achieved by tweaking the parameters of the compression routine, or by using a different compression algorithm. The theoretical upper limit of the signature size for any signature encoding is the length of a non-compressed signature, the lower limit is defined by the entropy of an individual signature.

C.3. **Bounding signature lengths.** The use of Huffman coding (and variations in Wave signature vector weights) results in a varying signature length. We limit the maximum signature size by defining `CRYPTO_BYTES` in such a way that at fewer than one in 2^{61} signatures are expected to exceed the maximum signature length. In this case, re-signing with a new salt gives a short compressed signature with high probability, without leaking any information on the private key.

To model perfect compression, we define the parameter $p = \frac{w}{n}$, where w is the weight and n is the length. This gives trit probabilities $\mathbb{P}(0) = 1 - p$, $\mathbb{P}(1) = \frac{p}{2}$, and $\mathbb{P}(2) = \frac{p}{2}$, so the average encoding for a word of length $\frac{n}{2}$ and weight t is

$$\left(\frac{n}{2} - t\right) \log_2 \left(\frac{1}{1-p}\right) + t \log_2 \left(\frac{2}{p}\right) .$$

On the other hand, the probability that a signature vector has weight exactly t is

$$\mathbb{P}(\text{signature weight} = t) = \binom{\frac{n}{2}}{t} p^t (1-p)^{\frac{n}{2}-t}.$$

To define the maximum signature size for Wave822 (NIST Level I), we first computed the probability that a random Wave822 signature would compress perfectly to a given length. We then compressed many random Wave822 signature vectors to determine how close our Huffman compression is to a perfect compression, used this difference as an offset from the theoretical signature length with probability 2^{-61}, and took the resulting signature length of 790 bytes (plus salt) as our maximum signature length for Wave822. We scaled the maximum length for the other security levels accordingly.

Table 9 lists some candidate bounds on signature lengths, togther with the corresponding probabilities of rejection and re-signing.

Table 9. Probability of rejection and re-signing for various signature length bounds.

Instance	Security	Signature length bound (B) Salt	Vector	Total	P(Re-signing)
Wave822	Level I	32	741	773	$2^{-1.096}$
		32	790	822	$2^{-61.9}$
Wave1249	Level III	64	1085	1149	$2^{-1.66}$
		64	1185	1249	2^{-130}
Wave1644	Level V	64	1424	1488	$2^{-1.113}$
		64	1580	1644	2^{-128}

Appendix D. Proof of Proposition 3

We use in this section notations of Algorithm 7.

Proposition 3. *On input* $(\mathbf{y}_U, \mathbf{e}_V, \mathbf{b}, \mathbf{c}, \mathbf{H}_U)$, *Algorithm 7 outputs* $\mathbf{e}_U$ *satisfying*

$$(\mathbf{y}_U - \mathbf{e}_U)\,\mathbf{H}_U^\top = \mathbf{0}_{n/2-k_U} \tag{12}$$

Furthermore, if $\mathbf{e}_L := \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$ *and* $\mathbf{e}_R := \mathbf{c} \star \mathbf{e}_L + \mathbf{e}_V$, *then*

$$|\mathbf{e}_L| + |\mathbf{e}_R| = w. \tag{13}$$

The proof of this proposition will rely on the following lemmas.

Lemma 2. *Let* $\mathbf{A}_{\mathrm{extSyst}} \in \mathbb{F}_3^{(r+g)\times n}$ *be the output of Algorithm 3 given as input* $\mathbf{A} \in \mathbb{F}_3^{r\times n}$ *with* $r = \mathrm{rank}(\mathbf{A}) < n$. *Let,*

$$J = \{i \in [0, r+g), \quad \mathbf{A}_{\mathrm{extSyst}}(i,i) \neq 0\}\,.$$

We have,

$$\#J = r$$

and

$$\forall \mathbf{s} \in \mathbb{F}_3^{r+g}, \quad \mathrm{Supp}(\mathbf{s}) \subseteq J \Longrightarrow (\mathbf{s}, \mathbf{0}_{n-r-g})\mathbf{A}_{\mathrm{extSyst}}^\top = \mathbf{s}.$$

Proof. First, $\langle \mathbf{A}_{\mathrm{extSyst}} \rangle = \langle \mathbf{A} \rangle$ and $r = \mathrm{rank}(\mathbf{A}) < n$. We deduce that $\mathrm{rank}(\mathbf{A}_{\mathrm{partSyst}})$ is equal to $r < r+g$. Therefore, as the matrix $\mathrm{rank}(\mathbf{A}_{\mathrm{partSyst}})$ is in extended systematic form (see Definition 2), it has exactly r non-zero rows, and g zero rows. Furthermore, the set J is an information set for $\mathbf{A}_{\mathrm{extSyst}}$. We deduce that it has cardinal $\mathrm{rank}(\mathbf{A}_{\mathrm{partSyst}}) = r$ and as the columns of $\mathbf{A}_{\mathrm{extSyst}}$ restricted to it form an identity matrix we have

$$\forall \mathbf{s} \in \mathbb{F}_3^{r+g}, \quad \mathrm{Supp}(\mathbf{s}) \subseteq J \Rightarrow (\mathbf{s}, \mathbf{0}_{n-r-g})\mathbf{A}_{\mathrm{extSyst}}^\top = \mathbf{s}$$

which concludes the proof. □

In what follows we use notation of Algorithm 7 (Decode_U).

Lemma 3. *Let* $\mathbf{e}_L = \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$ *and* $\mathbf{e}_R = \mathbf{c} \star \mathbf{e}_L + \mathbf{d} \star \mathbf{e}_V$ *where* $c_a \neq 0$ *for all* a. *We have,*

$$\forall i \in \pi([n/2 - k_U + g, n/2)), \quad \mathbf{e}_L(i)\mathbf{e}_R(i) \neq 0.$$

Proof. First,

$$\forall i \in [n/2 - k_U + g, n/2), \quad \mathbf{e}_U(\pi(i)) = \mathbf{e}(\pi^{-1}(\pi(i))) = \mathbf{e}(i) = \mathbf{e}^{(1)}(i - n/2 + k_U - g).$$

Now by definition of $\mathbf{e}^{(1)}$ it exists $\mathbf{x} \in (\mathbb{F}_3 \backslash \{0\})^{k_U - g}$ such that

$$\mathbf{e}^{(1)} = \mathbf{v}^{(1)} + (\mathbf{1} - \mathbf{s}^{(1)}) \star \mathbf{x}.$$

But,

$$\mathbf{v}^{(1)}(i - n/2 + k_U - g) = \mathbf{v}(i) = (\mathbf{c} - \mathbf{b}) \star \mathbf{e}_V(\pi(i))$$

and

$$\mathbf{s}^{(1)}(i - n/2 + k_U - g) = \mathbf{s}(i) = \mathbf{e}_V(\pi(i))^2$$

Combining these four equations shows Equation (11) which concludes the proof. $\square$

Lemma 4. *Let $\mathbf{e}_L = \mathbf{e}_U + \mathbf{b} \star \mathbf{e}_V$ and $\mathbf{e}_R = \mathbf{c} \star \mathbf{e}_L + \mathbf{e}_V$ where $c_a \neq 0$ for all a. In Instruction 22 and 23, i and j verify*

$$j = \# \{a \in [0, n/2) \ : \ \mathbf{e}_L(a) = \mathbf{e}_R(a) = 0\} \tag{19}$$

and

$$i = \# \{a \in [0, n/2) \ : \ \mathbf{e}_L(a) \neq \mathbf{e}_R(a) \ \textit{and} \ \mathbf{e}_L(a)\mathbf{e}_R(a) = 0\}. \tag{20}$$

Proof. Let us first prove Equation (19). Let,

$$m := \# \{a \in [0, n/2) \ : \ \mathbf{e}_L(a) = \mathbf{e}_R(a) = 0\}.$$

Our aim is to show that $m = j$. First, according to Lemma 3,

$$m = \# \{a \in \pi ([0, n/2 - k_U + g)) \ : \ \mathbf{e}_L(a) = \mathbf{e}_R(a) = 0\}. \tag{21}$$

We have for all $b \in [0, n/2 - k_U + g)$,

$$\begin{aligned} \mathbf{e}_L(\pi(b)) &= \mathbf{e}_U(\pi(b)) + (\mathbf{b} \star \mathbf{e}_V)(\pi(b)) \\ &= \mathbf{e}^{(0)}(b) + (\mathbf{b} \star \mathbf{e}_V)(\pi(b)) \end{aligned} \tag{22}$$

and

$$\begin{aligned} \mathbf{e}_R(\pi(b)) &= (\mathbf{c} \star \mathbf{e}_L)(\pi(b)) + \mathbf{e}_V(\pi(b)) \\ &= \mathbf{c}(\pi(b))\mathbf{e}^{(0)}(b) + \Big(1 + \mathbf{b}(\pi(b))\mathbf{c}(\pi(b))\Big)\mathbf{e}_V(\pi(b)). \end{aligned} \tag{23}$$

Combining these two equations with Equation (21) and the fact that $c_a \neq 0$ for all a shows

$$m = \# \left\{b \in [0, n/2 - k_U + g) \ : \ \mathbf{e}_V(\pi(b)) = 0 \text{ and } \ \mathbf{e}^{(0)}(b) = 0\right\}.$$

Therefore,

$$\begin{aligned} m = n/2 - k_U + g - &\underbrace{\# \{b \in [0, n/2 - k_U + g) \ : \ \mathbf{e}_V(\pi(b)) \neq 0\}}_{=\ell \ \text{(see Instruction 5)}} \\ &- \underbrace{\# \left\{b \in [0, n/2 - k_U + g) \ : \ \mathbf{e}_V(\pi(b)) = 0 \text{ and } \mathbf{e}^{(0)}(b) \neq 0\right\}}_{=|(\mathbf{1} - \mathbf{s}^{(0)}) \star \mathbf{e}^{(0)}|} \end{aligned}$$

showing that $m = j$ as defined in Instruction 23.

Let us prove now Equation (20). Let,

$$q := \#\left\{a \in [0, n/2) \ : \ \mathbf{e}_L(a) \neq \mathbf{e}_R(a) \text{ and } \mathbf{e}_L(a)\mathbf{e}_R(a) = 0\right\}.$$

Our aim is to show that $q = i$. First, according to Lemma 3,

$$\begin{aligned} q &= \#\left\{a \in \pi\left([0, n/2 - k_U + g)\right) \ : \ \mathbf{e}_L(a) \neq \mathbf{e}_R(a) \text{ and } \mathbf{e}_L(a)\mathbf{e}_R(a) = 0\right\} \\ &= q_1 + q_2 \end{aligned}$$

where,

$$\begin{cases} q_1 := \#\left\{a \in \pi\left([0, n/2 - k_U + g)\right) \ : \ \mathbf{e}_L(a) = 0 \text{ and } \mathbf{e}_R(a) \neq 0\right\}, \\ q_2 := \#\left\{a \in \pi\left([0, n/2 - k_U + g)\right) \ : \ \mathbf{e}_L(a) \neq 0 \text{ and } \mathbf{e}_R(a) = 0\right\}. \end{cases}$$

Using Equations (22) and (23) and the fact that $c_a \neq 0$ for all a, we obtain,

$$q_1 = \#\left\{b \in [0, n/2 - k_U + g) : \ \mathbf{e}_V(\pi(b)) \neq 0 \text{ and } \mathbf{e}^0(b) = -\Big(\mathbf{b} \star \mathbf{e}_V\Big)(\pi(b))\right\}.$$

$$q_2 = \#\left\{b \in [0, n/2 - k_U + g) : \ \mathbf{e}_V(\pi(b)) \neq 0 \text{ and } \mathbf{e}^0(b) = -\Big(\left(\mathbf{c} + \mathbf{b}\right) \star \mathbf{e}_V\Big)(\pi(b))\right\}.$$

Notice now that when $\mathbf{e}_V(\pi(b)) \neq 0$, as $c_a \neq 0$ for all a, we necessarily have

$$-\Big(\mathbf{b} \star \mathbf{e}_V\Big)(\pi(b)) \neq (\mathbf{c} - \mathbf{b}) \star \mathbf{e}_V(\pi(b))$$

and,

$$-\Big(\left(\mathbf{c} + \mathbf{b}\right) \star \mathbf{e}_V\Big)(\pi(b)) \neq \Big(\left(\mathbf{c} - \mathbf{b}\right) \star \mathbf{e}_V\Big)(\pi(b)).$$

Therefore,

$$\begin{aligned} q &= q_1 + q_2 \\ &= \#\left\{b \in [0, n/2 - k_U + g) \ : \ \mathbf{e}_V(\pi(b)) \neq 0 \text{ and } \mathbf{e}^0(b) \neq \Big((\mathbf{c} - \mathbf{b}) \star \mathbf{e}_V\Big)(\pi(b))\right\}. \end{aligned}$$

which is equal $\left|\mathbf{s}^{(0)} \star \mathbf{e}^{(0)} - \mathbf{v}^{(0)}\right| = i$ and it concludes the proof. □

Lemma 5. *In Instruction 18, $\mathbf{e}^{(0)}$ verifies,*

$$\mathbf{e}^{(0)}\left(\mathbf{H}_{[0,n/2-k_U+g)}\right)^{\top} = \mathbf{0}_{n/2-k_U+g}.$$

Proof. First, in Instruction 18, $\mathbf{e}^{(0)}$ is equal to

$$(\mathbf{1} - \mathbf{z}^{(0)}) \star \mathbf{f}$$

for some $\mathbf{f} \in \mathbb{F}_3^{n/2-k_U+g}$. The matrix $\mathbf{H}$ has been obtained in Instruction 13 via a ExtGaussElim (Algorithm 3), therefore it is in extended systematic form (see Definition 2). In particular, $\mathbf{H}_{[0,n/2-k_U+g)}$ is full of zero except on the diagonal where there are potentially some 1's. Therefore, by definition of $\mathbf{z}^{(0)}$ in Instruction 16,

$$(\mathbf{1} - \mathbf{z}^{(0)}) \star \mathbf{f} = \mathbf{f}\left(\mathbf{Id}_{n/2-k_U+g} - \mathbf{H}_{[0,n/2-k_U+g)}\right)^{\top}$$

Furthermore, using once again that $\mathbf{H}_{[0,n/2-k_U+g)}$ has only 0 or 1 on its diagonal, and otherwise is 0, we obtain

$$(1 - \mathbf{z}^{(0)}) \star \mathbf{f}\ \left(\mathbf{H}_{[0,n/2-k_U+g)}\right)^{\top} = \mathbf{0}_{n/2-k_U+g}$$

which concludes the proof of the lemma. □

Equipped with Lemmas 3, 4 and 5 we are now ready to prove Proposition 3.

Proof of Proposition 3. Let us first show Equation (13). By Lemma 4 and Instruction 24 we have

$$n - |(\mathbf{e}_L, \mathbf{e}_R)| = 2j + i \quad \text{and} \quad 2j + i = n - w$$

showing that $|\mathbf{e}_L| + |\mathbf{e}_R| = w$.

Let us now prove Equation (12) given in the proposition. Notice that $\mathbf{y}_U = \mathbf{y}^{\pi^{-1}}$ and $\mathbf{e}_U = \mathbf{e}^{\pi^{-1}}$ where $\mathbf{e}$ is given in Instruction 25. Therefore,

$$(\mathbf{y}_U - \mathbf{e}_U)\,\mathbf{H}_U^\top = (\mathbf{y} - \mathbf{e})\,(\mathbf{H}_U^\pi)^\top . \tag{24}$$

Recall now that $\mathbf{H}$ has been computed in Instruction 13 via ExtGaussElim (Algorithm 3), and therefore is in extended systematic form (see Definition 2). Let,

$$J = \{i \in [0, n/2 - k_U + g), \quad H_{i,i} \neq 0\} .$$

By definition, we have

$$\begin{cases} \forall i \in J: \quad H_{i,i} = 1 \text{ and } |\operatorname{col}(\mathbf{H}, i)| = 1, \\ \forall i \in [0, n/2 - k_U + g) \backslash J: \quad |\operatorname{row}(\mathbf{H}, i)| = 0. \end{cases}$$

where $\operatorname{col}(\mathbf{H}, i)$ (*resp.* $\operatorname{row}(\mathbf{H}, i)$) is defined as the i-th column (*resp.* row) of $\mathbf{H}$. But as $\langle \mathbf{H} \rangle = \langle \mathbf{H}_U^\pi \rangle$, the set J is an information set of $\mathbf{H}_U^\pi$. We deduce that it exists a non-singular $\mathbf{S} \in \mathbb{F}_3^{(n/2-k_U)\times(n/2-k_U)}$ such that $\mathbf{S}\mathbf{H}_U^\pi \in \mathbb{F}_3^{(n/2-k_U)\times n/2}$ verifies

$$\forall j \in J, \quad \operatorname{row}(\mathbf{S}\mathbf{H}_U^\pi, j) = \operatorname{row}(\mathbf{H}, j)$$

Therefore,

$$\forall j \in J, \quad (\mathbf{y} - \mathbf{e})\mathbf{H}^\top(j) = 0 \implies (\mathbf{y} - \mathbf{e})\,(\mathbf{H}_U^\pi)^\top = \mathbf{0}_{n/2-k_U} \tag{25}$$

Let $\mathbf{e}_{new}^{(0)}$ as in Instruction 21 and $\mathbf{e}_{old}^{(0)}$ as in Instruction 18. According to Lemma 5,

$$\mathbf{e}_{new}^{(0)} = \mathbf{e}_{old}^{(0)} + \mathbf{y}\mathbf{H}^\top - \mathbf{e}^{(1)}\big(\mathbf{H}_{[n/2-k_U+g, n/2)}\big)^\top \tag{26}$$

where $\mathbf{e}^{(1)}$ is given in Instruction 20. Notice now that ($\mathbf{e}$ is given in Instruction 25)

$$\mathbf{e}\mathbf{H}^\top = \mathbf{e}_{new}^{(0)}\big(\mathbf{H}_{[0,n/2-k_U+g)}\big)^\top + \mathbf{e}^{(1)}\big(\mathbf{H}_{[n/2-k_U+g,n/2)}\big)^\top$$

Using Equation (26) and Lemma 5, we get

$$\mathbf{e}_{new}^{(0)}\big(\mathbf{H}_{[0,n/2-k_U+g)}\big)^\top = \mathbf{y}\mathbf{H}^\top\big(\mathbf{H}_{[0,n/2-k_U+g)}\big)^\top - \mathbf{e}^{(1)}\big(\mathbf{H}_{[n/2-k_U+g,n/2)}\big)^\top\big(\mathbf{H}_{[0,n/2-k_U+g)}\big)^\top$$

Therefore,

$$\begin{aligned}(\mathbf{y} - \mathbf{e})\,\mathbf{H}^\top = \mathbf{y}\mathbf{H}^\top &- \mathbf{y}\mathbf{H}^\top\big(\mathbf{H}_{[0,n/2-k_U+g)}\big)^\top \\ &+ \mathbf{e}^{(1)}\big(\mathbf{H}_{[n/2-k_U+g,n/2)}\big)^\top - \mathbf{e}^{(1)}\big(\mathbf{H}_{[n/2-k_U+g,n/2)}\big)^\top\big(\mathbf{H}_{[0,n/2-k_U+g)}\big)^\top \end{aligned} \tag{27}$$

Recall now that,

$$\forall i \in J: \quad H_{i,i} = 1 \text{ and } |\operatorname{col}(\mathbf{H}, i)| = 1$$

We deduce that for any $\mathbf{x} \in \mathbb{F}_3^{n/2-k_U+g}$,

$$\forall j \in J, \quad \mathbf{x}\big(\mathbf{H}_{[0,n/2-k_U+g)}\big)^\top(j) = \mathbf{x}(j).$$

Plugging this in Equation (27) shows that

$$\forall j \in J, \quad (\mathbf{y} - \mathbf{e})\mathbf{H}^\top(j) = 0.$$

Therefore, according to (25),

$$(\mathbf{y} - \mathbf{e}) \left(\mathbf{H}_U^\pi\right)^\top = \mathbf{0}_{n/2-k_U}$$

which concludes the proof by using Equation (24). $\square$