

Algorithm 14 [CoeffFromThreeBytes](#)(ρ, b_0, b_1, b_2)

Generates an element of $\{0, 1, 2, \dots, a - 1\} \cup \{\perp\}$.

Input: Bytes b_0, b_1, b_2 .

Output: An integer modulo

1: $b'_0 \leftarrow b_0$

Algorithm 20 [HintBitPack\(h\)](#)

Encodes a polynomial A into a hint bit.

Algorithm 35 [Power2Ro](#)

Decomposes r into (r_1, r_0)

Input: $r \in \mathbb{Z}_q$.

Output: Integers (r_1, r_0)

1: $r^+ \leftarrow r \bmod q$
 2: $r_0 \leftarrow r^+ \bmod 2^d$
 3: **return** $((r^+ - r_0) / 2^d)$

Algorithm 41 [NTT\(w\)](#)

Computes the NTT.

Input: Polynomial $w(X)$

Output: $\hat{w} = (\hat{w}[0], \dots, \hat{w}[255])$

1: **for** j **from** 0 **to** 255
 2: $\hat{w}[j] \leftarrow w_j$
 3: **end for**
 4: $m \leftarrow 0$
 5: $len \leftarrow 128$
 6: **while** $len \geq 1$ **do**
 7: $start \leftarrow 0$
 8: **while** $start < 2$
 9: $m \leftarrow m + 1$
 10: $z \leftarrow \text{zetas}[m]$
 11: **for** j **from** $start$ **to** $start + len - 1$ **do**
 12: $t \leftarrow (z \cdot \hat{w}[j + len]) \bmod q$
 13: $\hat{w}[j + len] \leftarrow (\hat{w}[j] - t) \bmod q$
 14: $\hat{w}[j] \leftarrow (\hat{w}[j] + t) \bmod q$
 15: **end for**
 16: $start \leftarrow start + 2 \cdot len$
 17: **end while**
 18: $len \leftarrow \lfloor len / 2 \rfloor$
 19: **end while**
 20: **return** $\hat{w}$

20: **return** $\hat{h}$

Algorithm 47 [ScalarVectorNTT](#)($\hat{c}, \hat{v}$)

Computes the product $\hat{c} \circ \hat{v}$ of a scalar $\hat{c}$ and a vector $\hat{v}$ over T_q .

Input: $\hat{c} \in T_q, \ell \in \mathbb{N}, \hat{v} \in T_q^\ell$

Output: $\hat{w} \in T_q^\ell$

1: **for** i **from** 0 **to** $\ell - 1$ **do**
 2: $\hat{w}[i] \leftarrow \text{MultiplyNTT}(\hat{c}, \hat{v}[i])$
 3: **end for**
 4: **return** $\hat{w}$

Algorithm 48 [MatrixVectorNTT](#)($\hat{M}, \hat{v}$)

Computes the product $\hat{M} \circ \hat{v}$ of a matrix $\hat{M}$ and a vector $\hat{v}$ over T_q .

Input: $k, \ell \in \mathbb{N}, \hat{M} \in T_q^{k \times \ell}, \hat{v} \in T_q^\ell$

Output: $\hat{w} \in T_q^k$

1: $\hat{w} \leftarrow 0^k$
 2: **for** i **from** 0 **to** $k - 1$ **do**
 3: **for** j **from** 0 **to** $\ell - 1$ **do**
 4: $\hat{w}[i] \leftarrow \text{AddNTT}(\hat{w}[i], \text{MultiplyNTT}(\hat{M}[i, j], \hat{v}[j]))$
 5: **end for**
 6: **end for**
 7: **return** $\hat{w}$

Algorithm 22 [pkEncode](#)(ρ, t_1)

Encodes a public key for ML-DSA into a byte string.

efficient in $[0, 2^{\text{bitlen}(q) - k \cdot (\text{bitlen}(q) - d)}]$.

it of [Decompose](#) (r).

Algorithm 49 [MontgomeryReduce](#)(a)

Computes $a \cdot 2^{-32} \bmod q$.

Input: Integer a with $-2^{31}q \leq a \leq 2^{31}q$.

Output: $r \equiv a \cdot 2^{-32} \bmod q$.

1: $\text{QINV} \leftarrow 58728449$
 2: $t \leftarrow ((a \bmod 2^{32}) \cdot \text{QINV}) \bmod 2^{32}$
 3: $r \leftarrow (a - t \cdot q) / 2^{32}$
 4: **return** r

two elements $\hat{a}, \hat{b} \in T_q$.

Algorithm 46 [AddVectorNTT](#)($\hat{v}, \hat{w}$)

Computes the sum $\hat{v} + \hat{w}$ of two vectors $\hat{v}, \hat{w}$ over T_q .

Input: $\ell \in \mathbb{N}, \hat{v} \in T_q^\ell, \hat{w} \in T_q^\ell$

Output: $\hat{u} \in T_q^\ell$

1: **for** i **from** 0 **to** $\ell - 1$ **do**
 2: $\hat{u}[i] \leftarrow \text{AddNTT}(\hat{v}[i], \hat{w}[i])$
 3: **end for**
 4: **return** $\hat{u}$

Algorithm 29 [SampleInBall](#)(ρ)

Sample

Input:

Output:

Input: $\hat{w} = (\hat{w}[0], \dots, \hat{w}[255]) \in T_q$

Output: Polynomial $w(X) = \sum_{j=0}^{255} w_j X^j \in R_q$.

1: **for** j **from** 0 **to** 255 **do**

2: $w_j \leftarrow \hat{w}[j]$

3: **end for**

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

Input:

Output:

is on s and r

efficiently are

depends on r

ds on $r + \ell$

power of 2

ds on $\mu + r$

M

$\gamma_1 + 1$ and

f bits in its 8-bit binary expansion.

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

$\gamma_1 + 1$ and

- Efficient, Open-Source HW Implementation of ML-DSA
- Reduce SigGen() latency by 13 - 36 %



GitHub
RTL Source Code

- Starting point for **secure & efficient** implementations
- Come talk to us @ Poster Session!



IACR ePrint
2025/2337

- Post-Quantum, Lattice-based Digital Signature Algorithm
- Open challenges: implementing securely & efficiently
 - Variety of platforms: MCU, HW, ...
- Inherently complex
 - Rejection sampling, polynomial arithmetic, ...