V3: The Dilithium signature scheme

Kyber and Dilithium

August 2024

Dilithium

- Dilithium is a quantum-safe digital signature scheme.
- Standardized by NIST in FIPS 204, where it is called ML-DSA (Module-Lattice-based Digital Signature Algorithm).

V3 outline

- V3a: Dilithium (toy version)
- V3b: Dilithium
  (without t compression)
- V3c: t compression
- V3d: Dilithium (full scheme)

V3a: Dilithium (toy version)

1. Schnorr signature signature
2. Dilithium: first attempt
3. Dilithium: toy version
4. Problems and solutions
5. Rejection sampling

Schnorr signature scheme

Domain parameters:

A cyclic group of order n and generator g, and a hash function H.

Key generation: Alice does:

1. Select $a \in_R [1, n-1]$ and compute g^a.
2. Alice's verification (public) key is g^a; her signing (private) key is a.

Note: Computing a from g^a is an instance of the discrete logarithm problem.

Schnorr signature scheme (2)

Signature generation: To sign a message $M \in \{0,1\}^*$, Alice does:

1. Select $y \in_R [1, n-1]$ and compute $w = g^y$.
2. Compute $c = H(M \| w)$ and $z = y + ca \bmod n$.
3. Output the signature $\sigma = (c, z)$.

Signature verification: To verify Alice's signature $\sigma = (c, z)$ on M, Bob does:

1. Obtain an authentic copy of Alice's public key g^a.
2. Compute $w' = g^z (g^a)^{-c}$.
3. Accept iff $c = H(M \| w')$.

Note: Signature verification works because $z = y + ca \bmod n \iff g^z = g^{y+ca}$ $\iff g^y = g^z (g^a)^{-c} \iff w = g^z (g^a)^{-c}$.

Terminology: w commitment, c challenge, z response.

Notation

- $\mathbb{Z}_q$, mod q, mods q.
- $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.
- S_η = set of polynomials in R_q with (mods q) coefficients in $[-\eta, \eta]$.
- $\tilde{S}_{\gamma_1}$ = set of polynomials in R_q with (mods q) coefficients in $(-\gamma_1, \gamma_1]$.
- (k, ℓ) with $k > \ell$.
- B_τ = polynomials in S_1, exactly τ of whose coefficients are ± 1.
- $\beta = \tau\eta$.

ML-DSA-87 parameters

- $q = 2^{23} - 2^{13} + 1$
  $= 8{,}380{,}417$
  $\approx 2^{23}$
- $n = 256$
- $(k, \ell) = (8,7)$
- $\eta = 2$
- $\gamma_1 = 2^{19}$
- $\tau = 60,\ \beta = 120$

Dilithium: first attempt (1)

For concreteness, consider the ML-DSA-87 domain parameters:

- $q = 2^{23} - 2^{13} + 1$
- $n = 256$
- $(k, \ell) = (8,7)$
- $\eta = 2$
- $\gamma_1 = 2^{19}$
- $\tau = 60, \quad \beta = \tau\eta = 120.$
- $H : \{0,1\}^* \longrightarrow B_\tau.$

Key generation: Alice does:

1. Select $A \in_R R_q^{k \times \ell}$, $s_1 \in_R S_\eta^\ell$, and $s_2 \in_R S_\eta^k$.
2. Compute $t = As_1 + s_2$.
3. Alice's verification (public) key is (A, t); her signing (private) key is (s_1, s_2).

Note: Computing s_1 from (A, t) is an instance of MLWE.

Dilithium: first attempt (2)

Signature generation: To sign a message $M \in \{0,1\}^*$, Alice does:

1. Select $y \in_R \tilde{S}^{\ell}_{\gamma_1}$.
2. Compute $w = Ay$ (commitment).
3. Compute $c = H(M \| w)$ (challenge).
4. Compute $z = y + cs_1$ (response).
5. Output $\sigma = (c, z)$ (signature).

   Note: $w \in R_q^k$, $c \in B_\tau$, $z \in R_q^\ell$.

Signature verification: To verify Alice's signature $\sigma = (c, z)$ on M, Bob does:

1. Obtain an authentic copy of Alice's public key (A, t).
2. Compute the commitment w', and verify that $c = H(M \| w')$.

Note: A "mods q" operator is not needed when computing $z = y + cs_1$ provided that $\gamma_1 + \beta < q/2$ where $\beta = \tau\eta$.

Computing the commitment w'

- Question: How can the verifier compute w'?
  - Since $z = y + cs_1$, we have $Az = Ay + c(As_1) = w + c(t - s_2)$, whence $Az - ct = w - cs_2$. *So the verifier is only able to compute* $w - cs_2$.
- To circumvent this problem, note that the polynomials in cs_2 have "small" coefficients (since c and s_2 have small coefficients).
  - Main idea: The signer uses the "HighBits" (of the coefficients) of w to compose the challenge: $c = H(M \| w_1)$ where $w_1 = \text{HighBits}(w)$.
- If the "LowBits" of $w - cs_2$ are sufficiently small then, since cs_2 is also small, we have that $\text{HighBits}(w - cs_2) = \text{HighBits}(w)$.
- In this event, $w_1' = \text{HighBits}(Az - ct) = \text{HighBits}(w - cs_2) = \text{HighBits}(w) = w_1$, so $c = H(M \| w_1')$.
- Answer: The signer ensures that $\text{LowBits}(w - cs_2)$ are sufficiently small, and the verifier computes $w_1' = \text{HighBits}(Az - ct)$.

Dilithium: toy version

Signature generation: To sign a message $M \in \{0,1\}^*$, Alice does:

1. Found $\leftarrow$ false.
2. While Found = false do
   a) Select $y \in_R \tilde{S}^{\ell}_{\gamma_1}$.
   b) Compute $w = Ay$ and $w_1 = \text{HighBits}(w)$.
   c) Compute $c = H(M \| w_1)$.
   d) Compute $z = y + cs_1$.
   e) If LowBits$(w - cs_2)$ are "sufficiently small" then
      - Found $\leftarrow$ true .
3. Output $\sigma = (c, z)$.

Signature verification: To verify Alice's signature $\sigma = (c, z)$ on M, Bob does:

1. Obtain an authentic copy of Alice's verification key (A, t).
2. Compute $w_1' = \text{HighBits}(Az - ct)$.
3. Accept iff $c = H(M \| w_1')$.

- Alice's verification key is (A, t), where $A \in_R R_q^{k \times \ell}$ and $t = As_1 + s_2$ where $s_1 \in_R S_\eta^{\ell}$ and $s_2 \in_R S_\eta^{k}$.
- Alice's signing key is (s_1, s_2).

Problem #1: Large public and private keys

- Problem: Large public key $(A, t) \in R_q^{k \times \ell} \times R_q^k$.
  - For $q = 2^{23} - 2^{13} + 1$, $n = 256$, and $(k, \ell) = (8,7)$, the size of the matrix A is $56 \times 256 \times 23$ bits = 41,216 bytes, and the size of t is $8 \times 256 \times 23$ bits = 5,888 bytes.
  - Solution: Generate A from a 256-bit public seed ρ; the public key is now (ρ, t).
  - Solution: Use only the "high order bits" of the coefficients of polynomials in t. (This will be done in V3c.)
- Problem: Large secret key $(s_1, s_2) \in S_\eta^\ell \times S_\eta^k$.
  - For $q = 2^{23} - 2^{13} + 1$, $n = 256$, $(k, \ell) = (8,7)$, $\eta = 2$, the size of (s_1, s_2) is $(8 + 7) \times 256 \times 3$ bits = 1,440 bytes.
  - Solution: Generate s_1 and s_2 from a 512-bit secret seed ρ'.

Problem #2: Repeated hashing of M

- Problem: The message M is hashed once during each iteration of the signing algorithm:
  - Step 2(c): Compute $c = H(M \| w_1)$.

  This is slow if M is relatively large.

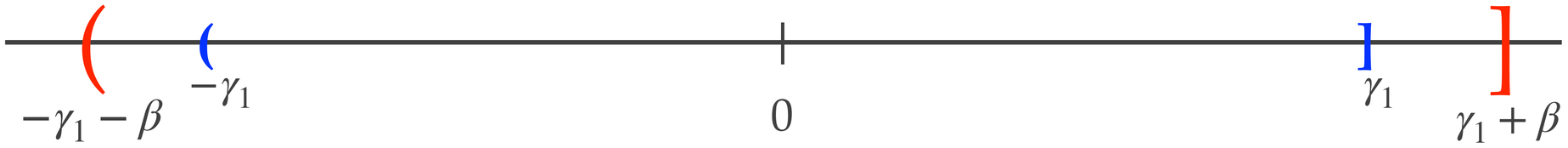
- Solution: Compute $\mu = H(M)$ and use μ instead of M.
  - Step 2(c): Compute $c = H(\mu \| w_1)$.

Problem #3: Random bits required for signing

- Problem: A large number of random bits are needed for each iteration of signature generation:
  - Step 2(a): Select $y \in_R \tilde{S}^{\ell}_{\gamma_1}$.

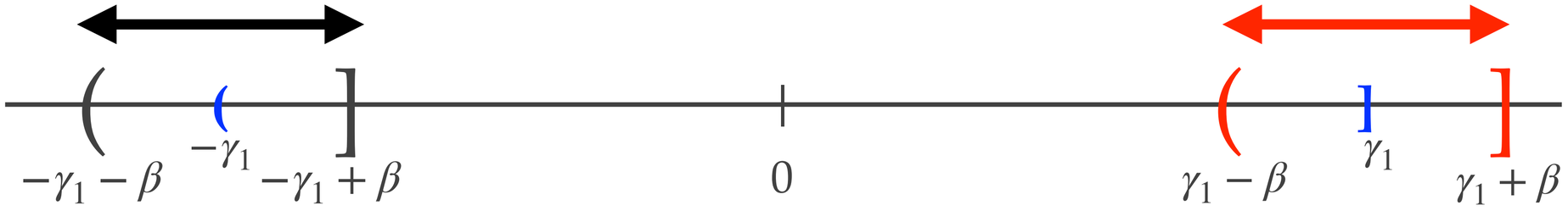
- Solution: Generate y from a secret seed ρ'' and counter κ, where ρ'' itself is obtained by hashing a secret seed K and the message hash μ.
  - This makes the signature generation algorithm deterministic.

Problem #4: A signature (c, z) leaks information about s_1



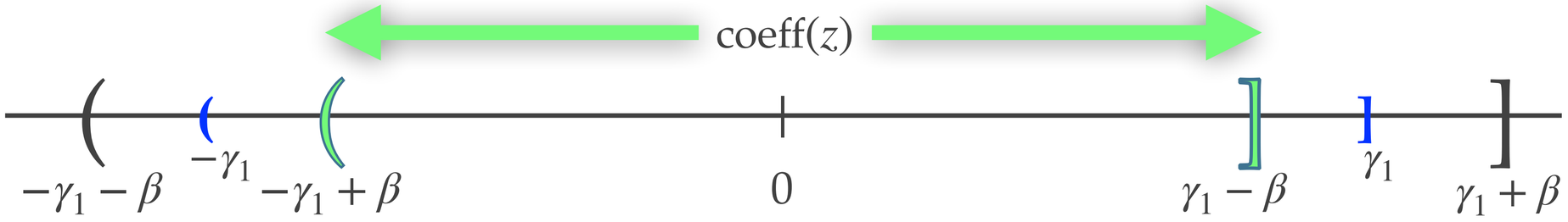
- Since $y \in \tilde{S}^{\ell}_{\gamma_1}$, we have $-\gamma_1 < \text{coeff}(y) \leq \gamma_1$ (where coeff(y) denotes the mods q representation of an arbitrary coefficient of an arbitrary polynomial in y).
- And, since $c \in B_\tau$ and $s_1 \in S^{\ell}_{\eta}$, we have $\|cs_1\|_\infty \leq \beta$ (where $\beta = \tau\eta$).
- Thus, since $z = y + cs_1$ and $\gamma_1 + \beta < q/2$, we have $-\gamma_1 - \beta < \text{coeff}(z) \leq \gamma_1 + \beta$.
- Suppose now that some coefficient of a polynomial in z is $\gamma_1 + \beta$. Then the corresponding coefficient of cs_1 must be β, which leaks *some* information about s_1.
- Similarly, if a coefficient of a polynomial in z is $\gamma_1 + \beta - 1$, then the corresponding coefficient of cs_1 must be either $\beta - 1$ or β, again leaking *some* information about s_1.

Problem #4: A signature (c, z) leaks information about s_1 (2)



- In general, if a coefficient of a polynomial in z is $\gamma_1 + a$ for some $-\beta + 1 \leq a \leq \beta$, then the corresponding coefficient of cs_1 must be in the interval $[a, \beta]$.
- Similarly, if a coefficient of a polynomial in z is $-\gamma_1 + a$ for some $-\beta < a \leq \beta$, then the corresponding coefficient of cs_1 must be in the interval $[-\beta, a - 1]$.

Problem #4: A signature (c, z) leaks information about s_1 (3)



- Solution: The signer repeatedly selects $y \in_R \tilde{S}^{\ell}_{\gamma_1}$ and computes $z = y + cs_1$ until $-\gamma_1 + \beta < \text{coeff}(z) \leq \gamma_1 - \beta$.
  - This ensures that the coefficients of z don't leak any information about the coefficients of cs_1 (and thus about s_1).
  - Dilithium imposes a slightly stricter constraint: $\|z\|_\infty < \gamma_1 - \beta$.
  - The signer also checks that LowBits$(w - cs_2)$ are sufficiently small.
  - This is an example of rejection sampling.

Problem #5: Computing HighBits($Az - ct$)

- Problem: In signature verification, the verifier needs to compute HighBits($Az - ct$). However, he only knows the "high order bits" of t.

- Solution: Use "hint bits" h so that HighBits($Az - ct$) can be computed using only the "high order bits" of t.
    - The details will be presented in V3c.

V3b: Dilithium (without *t* compression)

1. HighBits and LowBits (simplified)
2. Hash functions
3. Domain parameters
4. Key generation
5. Signature generation
6. Signature verification
7. Example
8. Security

HighBits and LowBits (simplified)

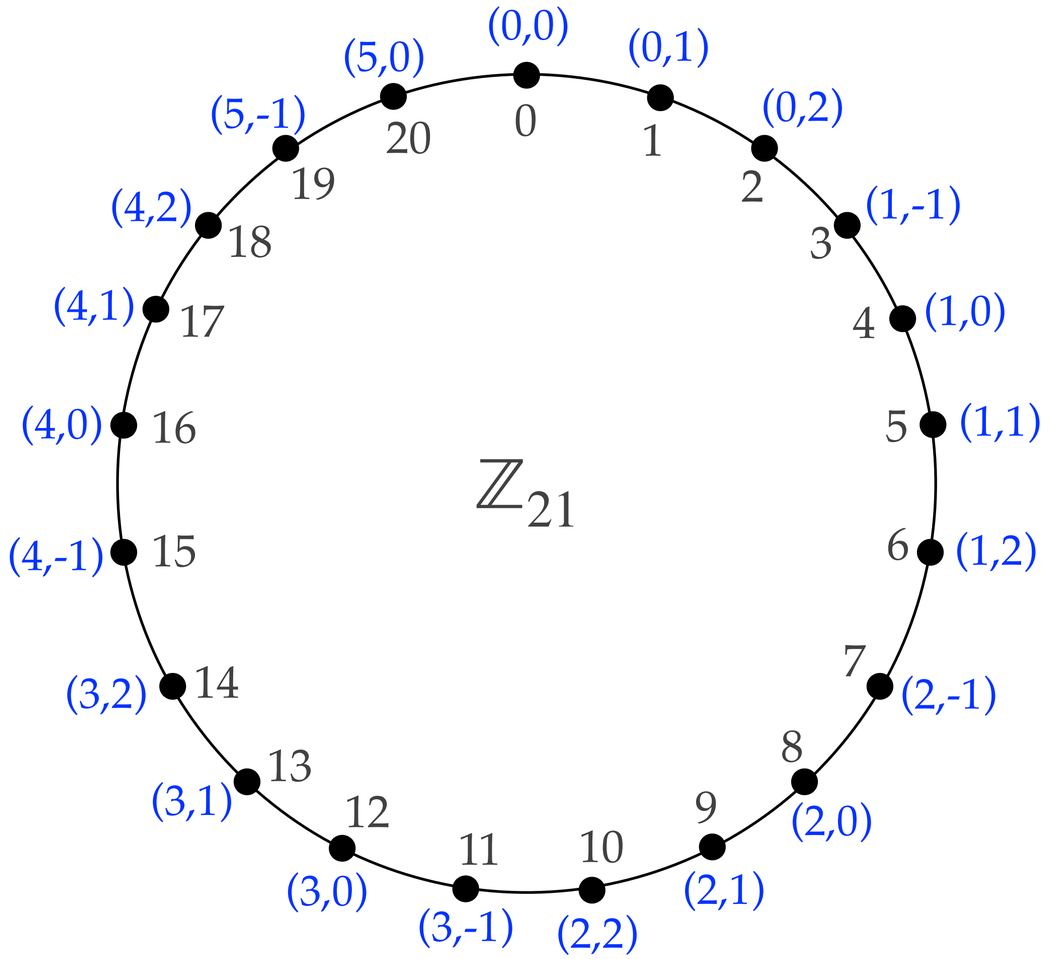
Decompose(r, α) [simplified]
Input: $r \in [0, q-1]$,
α even with $q - 1 = m\alpha$.
Output: (r_1, r_0) such that $r = r_1\alpha + r_0$,
with $-\alpha/2 < r_0 \leq \alpha/2$ and $0 \leq r_1 \leq m$.

1. Set $r_0 = r \bmod \alpha$ and
   $r_1 = (r - r_0)/\alpha$.

2. Return$((r_1, r_0))$

Notation:
$r_1 = \text{HighBits}(r, \alpha)$, $r_0 = \text{LowBits}(r, \alpha)$.

Example: Consider $q = 21$, $\alpha = 4$, $m = (q-1)/\alpha = 5$.



Note: $r_1\alpha$ is the nearest multiple of α to r,
with ties broken by choosing the smaller multiple.

Hashing

FIPS PUB 202

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

- SHAKE256 is a variable-length hash function specified in FIPS 202.
  - For $M \in \{0,1\}^*$ and $d \geq 1$, SHAKE256(M, d) is the d-bit hash of M.
  - SHAKE256 is an eXtendable-Output Function (XOF), which means that if $d' \leq d$ then SHAKE256(M, d') is exactly equal to the first d' bits of SHAKE256(M, d).
- Dilithium uses SHAKE256 (and also SHAKE128).
  - For $M \in \{0,1\}^*$ and $d \geq 1$, define $H(M, d) =$ SHAKE256(M, d).

Notation

- $\mathbb{Z}_q$, $\bmod q$, $\text{mods } q$, $\|r\|_\infty$
- $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.
- S_η = set of polynomials in R_q with (mods q) coefficients in $[-\eta, \eta]$.
- $\tilde{S}_{\gamma_1}$ = set of polynomials in R_q with (mods q) coefficients in $(-\gamma_1, \gamma_1]$.
- (k, ℓ) with $k > \ell$.
- B_τ = polynomials in S_1, exactly τ of whose coefficients are ± 1.
- $\beta = \tau\eta$.
- γ_2 and $\alpha = 2\gamma_2$ (for Decompose).
- λ (half the bitlength of signature component $\tilde{c}$).

ML-DSA-87 parameters

- $q = 2^{23} - 2^{13} + 1 = 8{,}380{,}417 \approx 2^{23}$
- $n = 256$
- $(k, \ell) = (8,7)$
- $\eta = 2$
- $\gamma_1 = 2^{19}$
- $\tau = 60$, $\beta = 120$
- $\gamma_2 = (q-1)/32 = 262{,}144 \approx 2^{18}$
- $\lambda = 256$

Dilithium (without t compression) (1)

For concreteness, we'll use the ML-DSA-87 domain parameters:

- $q = 2^{23} - 2^{13} + 1$
- $n = 256$
- $(k, \ell) = (8,7)$
- $\eta = 2$
- $\gamma_1 = 2^{19}$
- $\tau = 60, \quad \beta = \tau\eta = 120$
- $\lambda = 256$
- SampleInBall : $\{0,1\}^{2\lambda} \rightarrow B_\tau$
- $\gamma_2 = 262{,}144 \approx 2^{18}$

Key generation: Alice does:

1. Select $\xi \in_R \{0,1\}^{256}$.
2. Compute $(\rho, \rho', K) = H(\xi, 1024)$, where $\rho \in \{0,1\}^{256}$, $\rho' \in \{0,1\}^{512}$ and $K \in \{0,1\}^{256}$.
3. Compute $A = \text{ExpandA}(\rho)$ $(A \in R_q^{k \times \ell})$.
4. Compute $(s_1, s_2) = \text{ExpandS}(\rho')$ $((s_1, s_2) \in S_\eta^\ell \times S_\eta^k)$.
5. Compute $t = As_1 + s_2$ $(t \in R_q^k)$.
6. Compute $tr = H(\rho \| t, 2\lambda)$.
7. Alice's verification key is $PK = (\rho, t)$; her signature key is $SK = (\rho, K, tr, s_1, s_2)$.

Dilithium (without t compression) (2)

Signature generation: To sign a message $M \in \{0,1\}^*$, Alice does:

1. Compute $A = \text{ExpandA}(\rho)$.
2. Compute $\mu = H(tr \| M, 512)$.
3. Compute $\rho'' = H(K \| \text{rnd} \| \mu, 512)$, where either $\text{rnd} = 0^{256}$ (deterministic) or $\text{rnd} \in_R \{0,1\}^{256}$ (hedged).
4. $\kappa \leftarrow 0$.
5. Found $\leftarrow$ false.
6. While Found = false do:
   (a) Compute $y = \text{ExpandMask}(\rho'', \kappa)$. $(y \in \tilde{S}_{\gamma_1}^{\ell})$
   (b) Compute $w = Ay$ and $w_1 = \text{HighBits}(w, 2\gamma_2)$.
   (c) Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$.
   (d) Compute $c = \text{SampleInBall}(\tilde{c})$ and $z = y + cs_1$.
   (e) Compute $r_0 = \text{LowBits}(w - cs_2, 2\gamma_2)$.
   (f) If $\|z\|_\infty < \gamma_1 - \beta$ and $\|r_0\|_\infty < \gamma_2 - \beta$ then Found $\leftarrow$ true.
   (g) $\kappa \leftarrow \kappa + \ell$.
7. Return$(\sigma = (\tilde{c}, z))$.

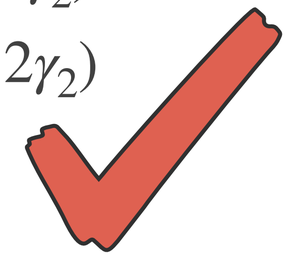
Dilithium (without t compression) (3)

Signature verification: To verify Alice's signature $\sigma = (\tilde{c}, z)$ on M, Bob does:

1. Obtain an authentic copy of Alice's public key $PK = (\rho, t)$.
2. Check that $\|z\|_\infty < \gamma_1 - \beta$; if not then reject.
3. Compute $A = \text{ExpandA}(\rho)$.
4. Compute $tr = H(\rho \| t, 512)$ and $\mu = H(tr \| M, 512)$.
5. Compute $c = \text{SampleInBall}(\tilde{c})$.
6. Compute $w_1' = \text{HighBits}(Az - ct, 2\gamma_2)$.
7. Check that $\tilde{c} = H(\mu \| w_1', 2\lambda)$; if not then reject.
8. Accept the signature.

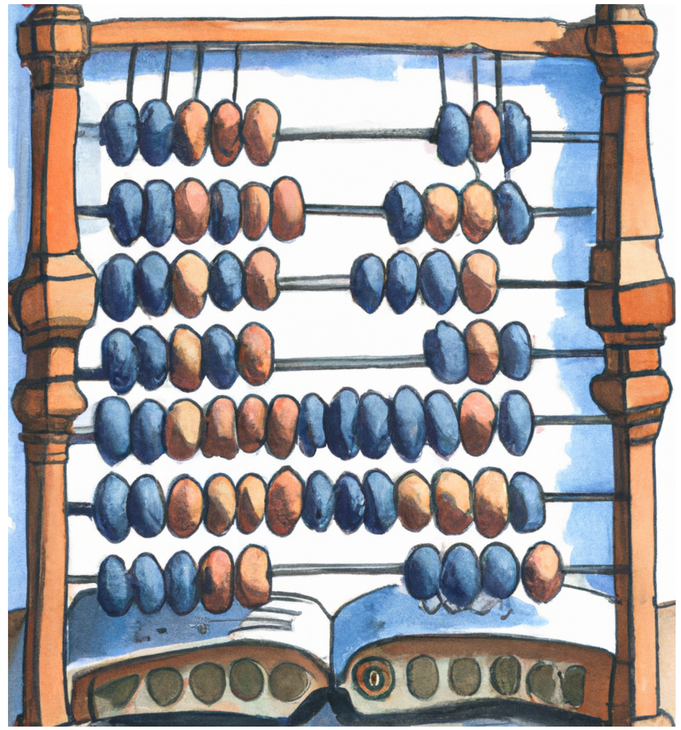
Correctness (signature verification works)

- We have
$$\begin{aligned} Az - ct &= A(y + cs_1) - c(As_1 + s_2) \\ &= Ay - cs_2 \\ &= w - cs_2 \,. \end{aligned}$$
- Since $\|\text{LowBits}(w - cs_2, 2\gamma_2)\|_\infty < \gamma_2 - \beta$ and $\|cs_2\|_\infty \le \beta$, we have
$$\begin{aligned} w_1' &= \text{HighBits}(Az - ct, 2\gamma_2) \\ &= \text{HighBits}(w - cs_2, 2\gamma_2) \\ &= \text{HighBits}(w, 2\gamma_2) \\ &= w_1 \,. \end{aligned}$$



Toy example: Parameters

- $q = 16{,}417 \approx 2^{14}, \quad (q-1)/2 = 8{,}208.$
- $n = 4, \quad \text{so } R_q = \mathbb{Z}_{16417}[x]/(x^4+1).$
- $(k, \ell) = (3,2).$
- $\eta = 10.$
- $\gamma_1 = 2^{10} = 1{,}024.$
- $\tau = 4, \quad \beta = \tau\eta = 40.$
- $\gamma_2 = (q-1)/32 = 513, \quad \alpha = 2\gamma_2 = 1{,}026, \quad m = 16.$
- $\gamma_1 - \beta = 984.$
- $\gamma_2 - \beta = 473.$



Toy example: Key generation

Alice selects

$$A = \begin{bmatrix} 15196 + 7926x + 8057x^2 + 13612x^3 & 14303 + 5x + 12257x^2 + 2347x^3 \\ 9765 + 10436x + 10983x^2 + 5860x^3 & 5393 + 311x + 5144x^2 + 10841x^3 \\ 11868 + 7995x + 10716x^2 + 3121x^3 & 9390 + 2055x + 6505x^2 + 2440x^3 \end{bmatrix},$$

$$s_1 \bmod q = \begin{bmatrix} 2 + 5x - 10x^2 - 5x^3 \\ 2 + 3x - 4x^2 - 4x^3 \end{bmatrix}, \quad s_2 \bmod q = \begin{bmatrix} -8 + 3x + 4x^2 + 4x^3 \\ 8 + 10x + x^2 + 8x^3 \\ -3 - 8x + 6x^2 + 6x^3 \end{bmatrix},$$

and computes $t = As_1 + s_2 = \begin{bmatrix} 5384 + 8401x + 14221x^2 + 11425x^3 \\ 4578 + 1291x + 5976x^2 + 9841x^3 \\ 3959 + 14751x + 15381x^2 + 14072x^3 \end{bmatrix}$.

Alice's verification key is (A, t); her signing key is (s_1, s_2).

Toy example: Signature generation (1)

To sign M, Alice selects y mods $q = \begin{bmatrix} 707 - 155x + 357x^2 - 822x^3 \\ 474 - 566x - 869x^2 - 542x^3 \end{bmatrix}$ and

$\gamma_1 = 1024$

computes $w = Ay = \begin{bmatrix} 7023 + 3184x + 4074x^2 + 5566x^3 \\ 9495 + 7254x + 7431x^2 + 13483x^3 \\ 4189 + 7420x + 13635x^2 + 4161x^3 \end{bmatrix}$, and

$w_1 = \text{HighBits}(w, 2\gamma_2) = \begin{bmatrix} 7 + 3x + 4x^2 + 5x^3 \\ 9 + 7x + 7x^2 + 13x^3 \\ 4 + 7x + 13x^2 + 4x^3 \end{bmatrix}$.

$2\gamma_2 = 1026$

Example:

$\text{LowBits}_q(5566, 1026) = 436.$

$\text{HighBits}_q(5566, 1026) = 5.$

Toy example: Signature generation (2)

Alice computes $\tilde{c} = H(\mu \| w_1)$ and $c = \text{SampleInBall}(\tilde{c})$, say c mods $q = -1 - x + x^2 - x^3$, and

$$z = y + cs_1 = \begin{bmatrix} 715 - 167x + 359x^2 - 804x^3 \\ 475 - 571x - 870x^2 - 533x^3 \end{bmatrix}.$$

Note that $\|z\|_\infty < \gamma_1 - \beta = 984$.

Alice also computes $w - cs_2 = \begin{bmatrix} 7012 + 3179x + 4085x^2 + 5563x^3 \\ 9486 + 7279x + 7426x^2 + 13490x^3 \\ 4194 + 7409x + 13630x^2 + 4178x^3 \end{bmatrix}$, and

$r_0 = \text{LowBits}(w - cs_2, 2\gamma_2) = \begin{bmatrix} -170 + 101x - 19x^2 + 433x^3 \\ 252 + 97x + 244x^2 + 152x^3 \\ 90 + 227x + 292x^2 + 74x^3 \end{bmatrix}$. Note that $\|r_0\|_\infty < \gamma_2 - \beta = 473$.

Alice's signature on M is $\sigma = (\tilde{c}, z)$.

Toy example: Signature verification

To verify Alice's signature $\sigma = (\tilde{c}, z)$ on M, Bob computes c mods $q = -1 - x + x^2 - x^3$,

$$Az - ct = \begin{bmatrix} 7012 + 3179x + 4085x^2 + 5563x^3 \\ 9486 + 7279x + 7426x^2 + 13490x^3 \\ 4194 + 7409x + 13630x^2 + 4178x^3 \end{bmatrix}, \text{ and}$$

$$w_1' = \text{HighBits}(Az - ct, 2\gamma_2) = \begin{bmatrix} 7 + 3x + 4x^2 + 5x^3 \\ 9 + 7x + 7x^2 + 13x^3 \\ 4 + 7x + 13x^2 + 4x^3 \end{bmatrix}.$$

Since $w_1' = w_1$, $H(\mu \| w_1', 2\lambda) = H(\mu \| w_1, 2\lambda)$ so the signature is accepted.

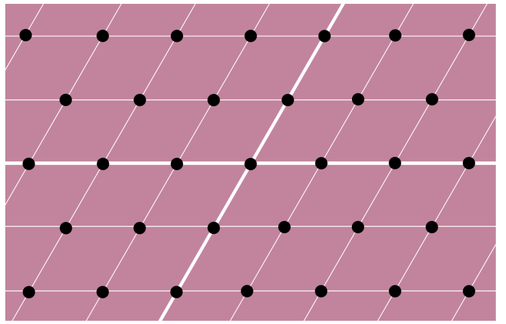
Security (1)

- Key generation: If D-MLWE is intractable, then an adversary is unable to learn anything about the secret key component s_1 from the public key (A, t).
- Signature forgery: The adversary's task is the following:
  Given (A, t), find $(M, \tilde{c}, z)$ such that $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = H(\mu \| w_1)$, where $\mu = H(M)$, $w_1 = \text{HighBits}(Az - ct, 2\gamma_2)$, and $c = \text{SampleInBall}(\tilde{c})$.
  - Note that c depends on $\tilde{c}$, $\tilde{c}$ depends on w_1, and w_1 depends on c.
  - Assuming that H has no weaknesses, it would appear that the adversary's best strategy would be to randomly select M and $w_1 \in R_q^k$ (where each coefficient is in $[0, (q-1)/2\gamma_2])$, and then compute $\mu = H(M)$, $\tilde{c} = H(\mu \| w_1)$, and $c = \text{SampleInBall}(\tilde{c})$.
  - The adversary's remaining task is to find a suitable z.

Security (2)

- Finding z:
  - Since $w_1 = \text{HighBits}(Az - ct, 2\gamma_2)$, we can write $Az - ct = 2\gamma_2 w_1 + w_0$, where $w_0 \in R_q^k$ with $-\gamma_2 < \text{coeff}(w_0) \leq \gamma_2$.
  - Rearranging terms gives $Az - w_0 = ct + 2\gamma_2 w_1$.
  - So, the adversary's task is to find a solution (z, w_0) to the linear system of equations $\left[A \mid I_k\right] \begin{bmatrix} z \\ -w_0 \end{bmatrix} = ct + 2\gamma_2 w_1$, where $z \in R_q^\ell$ and $w_0 \in R_q^k$ are relatively small.
    (More precisely, $\|z\|_\infty < \gamma_1 - \beta$ and $-\gamma_2 < \text{coeff}(w_0) \leq \gamma_2$.)
  - This is an instance of the (inhomogeneous) MSIS problem.

I-MSIS and MSIS



- I-MSIS: The Inhomogeneous Module Short Integer Solutions (I-MSIS) problem is the following:
  - Given $A \in_R R_q^{k \times \ell}$ and $b \in_R R_q^k$, and find a small solution $y \in R_q^{k+\ell}$ to the linear system of equations $[A \mid I_k]\, y = b$.

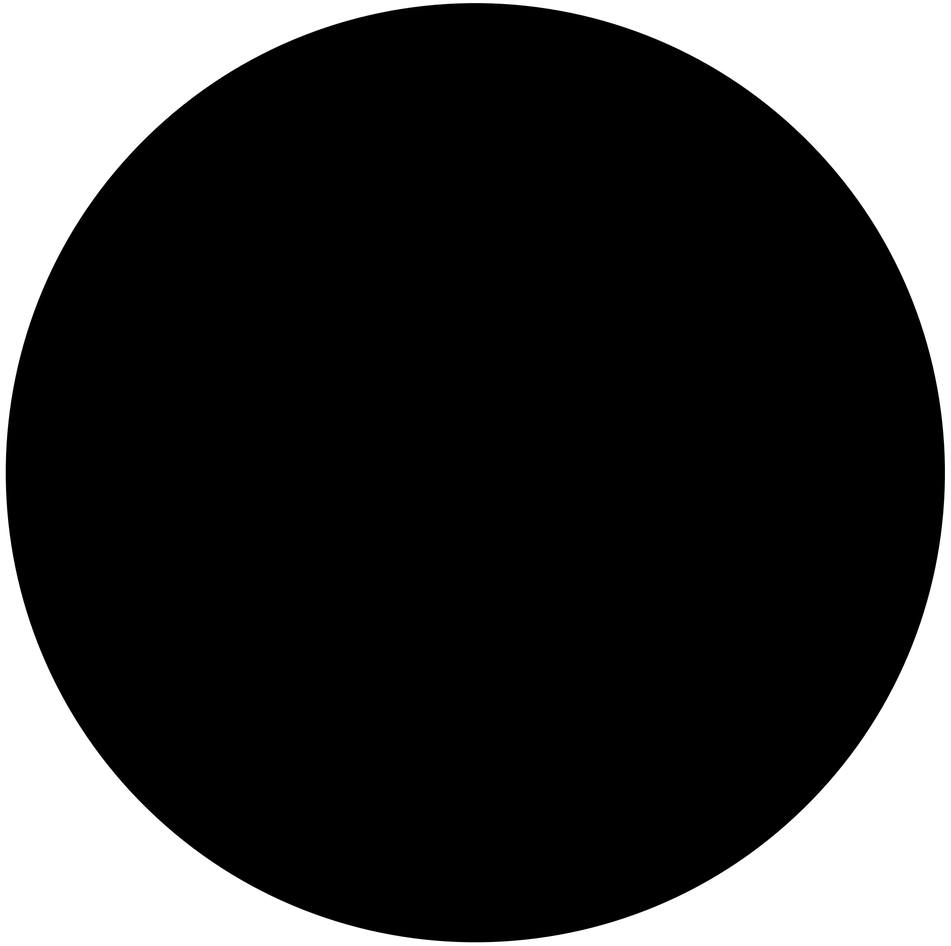
- MSIS: The Module Short Integer Solutions (MSIS) problem is the following:
  - Given $A \in_R R_q^{k \times \ell}$, find a (nonzero) small solution $y \in R_q^{k+\ell}$ to the linear system of equations $[A \mid I_k]\, y = 0$.

Security (3)

- Claim: Dilithium (without t compression) is existentially unforgeable against chosen-message attack assuming that D-MLWE and MSIS are intractable, and H is modelled as a random function.

V3c: *t* compression

1. Rounding
2. Compression
3. Hint bits
4. Modifying Decompose
5. MakeHint and UseHint



Rounding: Power2Round

Power2Round(r, d)

Input: $r \in [0, q-1]$, $d \in [1, \log_2 q]$.

Output: (r_1, r_0) such that $r = r_1 2^d + r_0$, with $-2^{d-1} < r_0 \leq 2^{d-1}$ and $0 \leq r_1 \leq \lceil (q-1)/2^d \rceil$.

1. Set $r_0 = r \bmod 2^d$ and $r_1 = (r - r_0)/2^d$.

2. Return$((r_1, r_0))$.

Example: Consider $q = 23$, $d = 3$.

r	(r_1, r_0)	r	(r_1, r_0)
0	(0,0)	12	(1,4)
1	(0,1)	13	(2,-3)
2	(0,2)	14	(2,-2)
3	(0,3)	15	(2,-1)
4	(0,4)	16	(2,0)
5	(1,-3)	17	(2,1)
6	(1,-2)	18	(2,2)
7	(1,-1)	19	(2,3)
8	(1,0)	20	(2,4)
9	(1,1)	21	(3,-3)
10	(1,2)	22	(3,-2)
11	(1,3)		

Power2Round (2)

- Power2Round can be extended to polynomials by applying it to the coefficients.
- Example:
  - Consider $R_q = \mathbb{Z}_{23}[x]/(x^4 + 1)$ and $d = 3$.
  - Let $t = 2 + 11x + 5x^2 + 19x^3 \in R_q$.
  - Then Power2Round$(t, d) = (t_1, t_0)$, where $t_1 = x + x^2 + 2x^3$ and $t_0 = 2 + 3x - 3x^2 + 3x^3$.
  - Note that $t = t_1 2^d + t_0$.
- Power2Round can be extended to vectors of polynomials in R_q^k by applying it to each polynomial.

$q = 23,\ d = 3.$

r	(r_1, r_0)	r	(r_1, r_0)
0	(0,0)	12	(1,4)
1	(0,1)	13	(2,-3)
2	(0,2)	14	(2,-2)
3	(0,3)	15	(2,-1)
4	(0,4)	16	(2,0)
5	(1,-3)	17	(2,1)
6	(1,-2)	18	(2,2)
7	(1,-1)	19	(2,3)
8	(1,0)	20	(2,4)
9	(1,1)	21	(3,-3)
10	(1,2)	22	(3,-2)
11	(1,3)		

t compression

- Recall that the Dilithium verification key includes a vector of polynomials $t \in R_q^k$.
- t compression: Drop the d low-order bits of each coefficient of each polynomial in t.
  - Use Power2Round to write $t = t_1 2^d + t_0$.
  - Include t_1 (instead of t) in the verification key, and t_0 is retained in the signing key.
- For ML-DSA-87, we have $k = 8$ and $d = 13$. So, the size of t is $8 \times 256 \times 23$ bits = 5,888 bytes, whereas the size of t_1 is $8 \times 256 \times 10$ bits = 2,560 bytes.
- This introduces some complications in signature generation and verification.

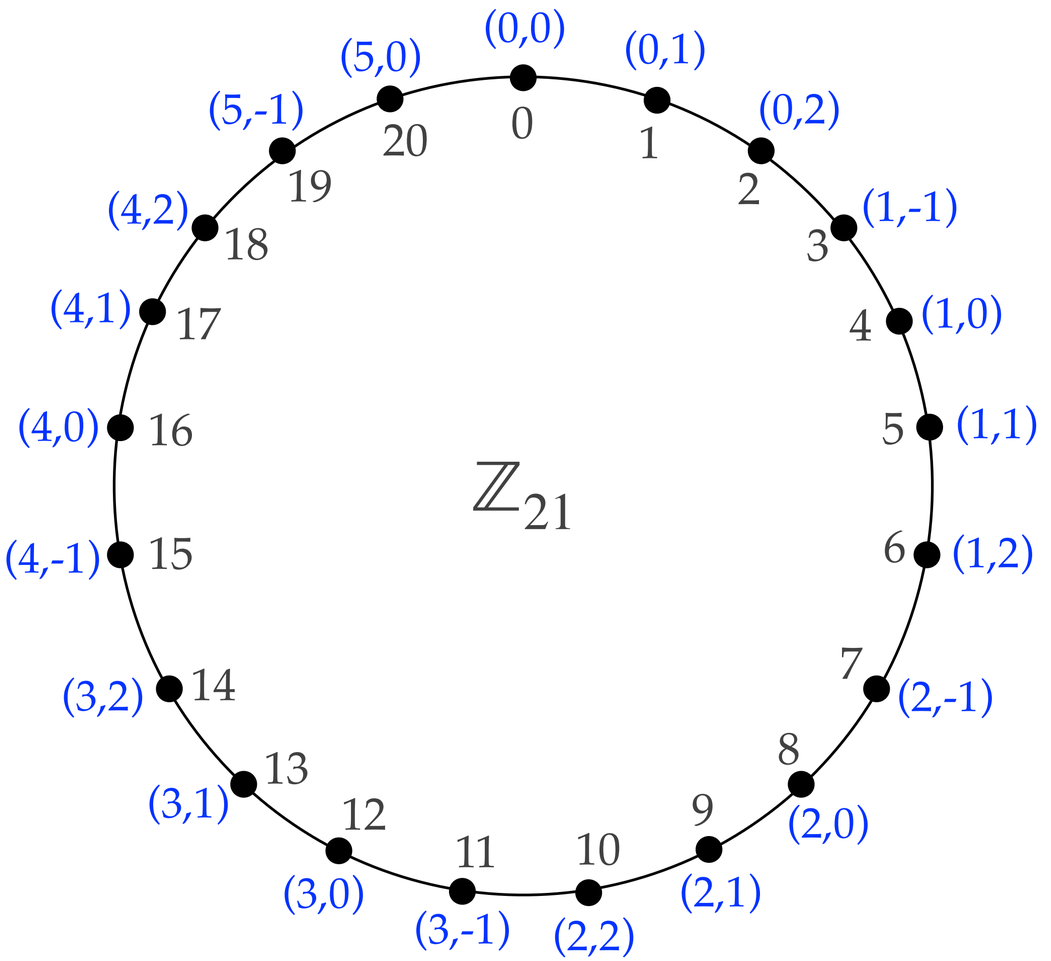
Complications due to t compression

- Recall that the verifier needs to compute $Az - ct$ in order to recover HighBits$(w - cs_2, 2\gamma_2)$.
  - However, the verifier only knows t_1 and not t_0.
- So, instead of computing $Az - ct$, the verifier computes
$$Az - ct_1 2^d = Az - c(t - t_0) = Az - ct + ct_0 = w - cs_2 + ct_0.$$
- Now, the coefficients of the polynomials in $-ct_0$ are expected to be relatively small. More precisely, we can expect that $\| -ct_0 \|_\infty \leq \gamma_2$ with high probability.
  - e.g., when $d = 13,\ \tau = 60,$ and $\gamma_2 = 262144$, the probability is 1.
- If indeed $\| -ct_0 \|_\infty \leq \gamma_2$, then adding $-ct_0$ to $w - cs_2 + ct_0$ will affect the HighBits of $w - cs_2 + ct_0$ by -1, 0 or $+1$.
- To enable the verifier to compute HighBits$(w - cs_2, 2\gamma_2)$ from $w - cs_2 + ct_0$ (without knowledge of t_0), the signer includes some "hint bits" in the signature. These hint bits are essentially the "carry digits" when $-ct_0$ is added to $w - cs_2 + ct_0$.

Hint bits

- Let $r \in [0, q-1]$, and let $r_1 = \text{HighBits}(r, \alpha)$ and $r_0 = \text{LowBits}(r, \alpha)$ (recall: $\alpha = 2\gamma_2$, $q - 1 = m\alpha$, $r_0 = r \bmod \alpha$, and $r = r_1\alpha + r_0$).
- Let $z \in [-\gamma_2, \gamma_2]$.
- A hint bit $h \in \{0,1\}$ should allow one to compute $\text{HighBits}(r + z, \alpha)$ from r and h without knowledge of z.
- Problem: The multiples $m\alpha$ and 0α are side-by-side.

Example: Consider $q = 21$, $\alpha = 4$, $\gamma_2 = 2$, $m = 5$.



HighBits and LowBits (modified)

Decompose(r, α) [modified]
Input: $r \in [0, q-1]$,
α even with $q - 1 = m\alpha$.
Output: (r_1, r_0) such that $r = r_1\alpha + r_0 \bmod q$,
with $-\alpha/2 \le r_0 \le \alpha/2$ and $0 \le r_1 < m$.

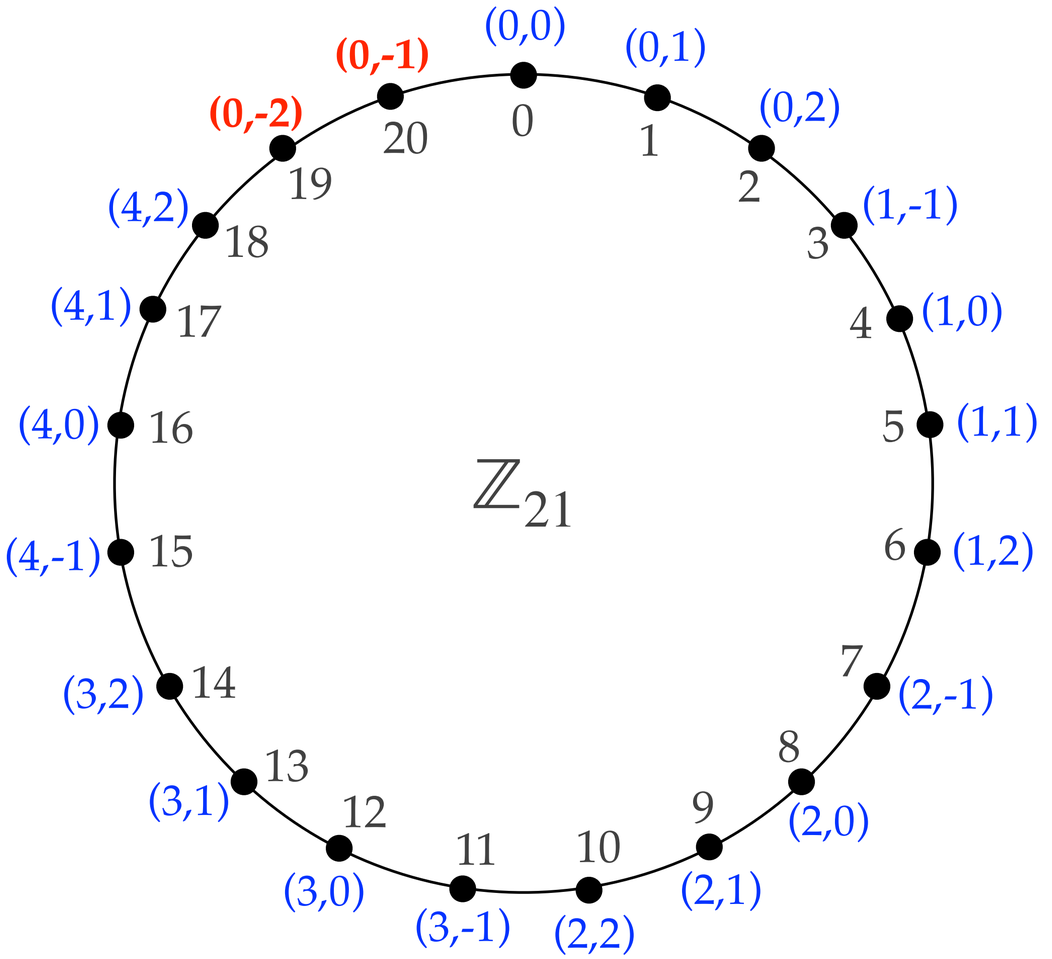
1. Compute $r_0 = r$ mods α.

2. If $r - r_0 = q - 1$ then
   Set $r_1 = 0,\ r_0 = r_0 - 1$
   Else
   Set $r_1 = (r - r_0)/\alpha$.

3. Return$((r_1, r_0))$

Notation:
$r_1 = \text{HighBits}(r, \alpha),\ r_0 = \text{LowBits}(r, \alpha)$.

Example: Consider $q = 21,\ \alpha = 4,\ m = (q-1)/\alpha = 5$.



MakeHint and UseHint

MakeHint(z, r, α)
Input: $r \in [0, q-1]$, $-\alpha/2 \le z \le \alpha/2$.
Output: A hint bit $h \in \{0,1\}$.

1. Set
$$h = \begin{cases} 1, & \text{if HighBits}(r+z, \alpha) \neq \text{HighBits}(r, \alpha), \\ 0, & \text{otherwise}. \end{cases}$$
2. Return(h).

UseHint(h, r, α)
Input: $h \in \{0,1\}$, $r \in [0, q-1]$.
Output: HighBits$(r+z, \alpha)$.

1. Compute $(r_1, r_0) = \text{Decompose}(r, \alpha)$.
2. If $h = 1$ and $r_0 > 0$ then $r_1 \leftarrow r_1 + 1 \bmod m$.
3. If $h = 1$ and $r_0 \le 0$ then $r_1 \leftarrow r_1 - 1 \bmod m$.
4. Return(r_1).

Claim: Suppose that $r \in [0, q-1]$ and $-\alpha/2 \le z \le \alpha/2$.
Then UseHint(MakeHint$(z, r, \alpha), r, \alpha) = \text{HighBits}(r+z, \alpha)$.

Using MakeHint and UseHint

- MakeHint and UseHint extend naturally to polynomials in R_q and vectors of polynomials in R_q^ℓ.
- Signature generation: The signer verifies that $\| -ct_0 \|_\infty \leq \gamma_2$ and appends a hint vector $h = \text{MakeHint}(-ct_0, w - cs_2 + ct_0, 2\gamma_2)$ to the signature.
  - There is also the requirement that the number of 1's in h be $\leq \omega$.
- Signature verification: The verifier uses h to compute $w_1' = \text{UseHint}(h, Az - ct_1 2^d, 2\gamma_2)$.
  - Since $Az - ct_1 2^d = w - cs_2 + ct_0$, we have $w_1' = \text{HighBits}(w - cs_2, 2\gamma_2)$.

V3d: Dilithium (full scheme)

1. Domain parameters
2. Key generation
3. Signature generation
4. Signature verification
5. Security
6. Parameters
7. Key sizes and signature sizes
8. Expected number of iterations
9. Omitted details

Notation

- $\mathbb{Z}_q$, $\bmod q$, $\text{mods } q$, $\|r\|_\infty$
- $R_q = \mathbb{Z}_q[x]/(x^n + 1)$
- S_η = set of polynomials in R_q with (mods q) coefficients in $[-\eta, \eta]$
- $\tilde{S}_{\gamma_1}$ = set of polynomials in R_q with (mods q) coefficients in $(-\gamma_1, \gamma_1]$
- (k, ℓ) with $k > \ell$
- B_τ = polynomials in S_1, exactly τ of whose coefficients are ± 1
- $\beta = \tau\eta$
- γ_2 and $\alpha = 2\gamma_2$
- λ (half the bitlength of signature component $\tilde{c}$)
- d = number of bits dropped from t
- ω = max number of 1's in h

ML-DSA-87 parameters

- $q = 2^{23} - 2^{13} + 1 = 8{,}380{,}417 \approx 2^{23}$
- $n = 256$
- $(k, \ell) = (8,7)$
- $\eta = 2$
- $d = 13$
- $\gamma_1 = 2^{19}$
- $\tau = 60$, $\beta = 120$
- $\gamma_2 = (q-1)/32 = 262{,}144 \approx 2^{18}$
- $\alpha = (q-1)/16 = 523{,}776 \approx 2^{19}$
- $\lambda = 256$
- $\omega = 75$

Dilithium (full scheme) (1)

For concreteness, we'll use the ML-DSA-87 domain parameters:

- $q = 2^{23} - 2^{13} + 1, \quad n = 256$
- $(k, \ell) = (8,7)$
- $\eta = 2, \quad d = 13, \quad \gamma_1 = 2^{19}$
- $\tau = 60, \quad \beta = \tau\eta = 120$
- SampleInBall : $\{0,1\}^{2\lambda} \rightarrow B_\tau$
- $\gamma_2 = 262{,}144 \approx 2^{18}$,
  $\alpha = 2\gamma_2 \approx 2^{19}$
- $\omega = 75$

Key generation: Alice does:

1. Select $\xi \in_R \{0,1\}^{256}$.
2. Compute $(\rho, \rho', K) = H(\xi, 1024)$, where $\rho \in \{0,1\}^{256}$, $\rho' \in \{0,1\}^{512}$ and $K \in \{0,1\}^{256}$.
3. Compute $A = \text{ExpandA}(\rho)$ $(A \in R_q^{k \times \ell})$.
4. Compute $(s_1, s_2) = \text{ExpandS}(\rho')$ $((s_1, s_2) \in S_\eta^\ell \times S_\eta^k)$.
5. Compute $t = As_1 + s_2$ $(t \in R_q^k)$.
6. Compute $(t_1, t_0) = \text{Power2Round}(t, d)$.
7. Compute $tr = H(\rho \| t_1, 512)$.
8. Alice's verification key is $PK = (\rho, t_1)$;
   her signature key is $SK = (\rho, K, tr, s_1, s_2, t_0)$.

Dilithium (full scheme) (2)

Signature generation: To sign a message $M \in \{0,1\}^*$, Alice does:

1. Compute $A = \text{ExpandA}(\rho)$.
2. Compute $\mu = H(tr \| M, 512)$.
3. Compute $\rho'' = H(K \| \text{rnd} \| \mu, 512)$, where either $\text{rnd} = 0^{256}$ (deterministic) or $\text{rnd} \in_R \{0,1\}^{256}$ (hedged).
4. $\kappa \leftarrow 0$.
5. Found $\leftarrow$ false.
6. While Found = false do:
   (a) Compute $y = \text{ExpandMask}(\rho'', \kappa)$. $(y \in \tilde{S}_{\gamma_1}^{\ell})$
   (b) Compute $w = Ay$ and $w_1 = \text{HighBits}(w, 2\gamma_2)$.
   (c) Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$ and $c = \text{SampleInBall}(\tilde{c})$. $(c \in B_\tau)$
   (d) Compute $z = y + cs_1$.
   (e) Compute $r_0 = \text{LowBits}(w - cs_2, 2\gamma_2)$.
   (f) If $\|z\|_\infty < \gamma_1 - \beta$ and $\|r_0\|_\infty < \gamma_2 - \beta$ then
      i) If $\| - ct_0\|_\infty < \gamma_2$ then
         - Compute $h = \text{MakeHint}(-ct_0, w - cs_2 + ct_0, 2\gamma_2)$.
         - If the number of 1's in h is $\leq \omega$ then: Found $\leftarrow$ true.
   (g) $\kappa \leftarrow \kappa + \ell$.
7. Return$(\sigma = (\tilde{c}, z, h))$.

Dilithium (full scheme) (3)

Signature verification: To verify Alice's signature $\sigma = (\tilde{c}, z, h)$ on M, Bob does:

1. Obtain an authentic copy of Alice's public key $PK = (\rho, t_1)$.
2. Check that $\|z\|_\infty < \gamma_1 - \beta$ and that the number of 1's in h is $\leq \omega$; if not then reject.
3. Compute $A = \text{ExpandA}(\rho)$.
4. Compute $tr = H(\rho \| t_1, 512)$ and $\mu = H(tr \| M, 512)$.
5. Compute $c = \text{SampleInBall}(\tilde{c})$.
6. Compute $w_1' = \text{UseHint}(h, Az - ct_1 2^d, 2\gamma_2)$.
7. Check that $\tilde{c} = H(\mu \| w_1', 2\lambda)$; if not then reject.
8. Accept the signature.

Correctness

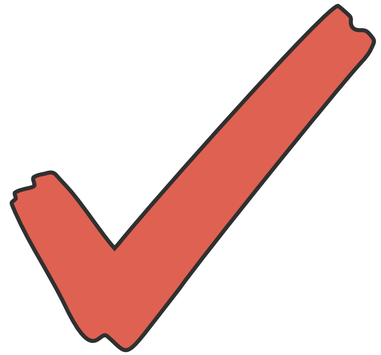
Correctness (signature verification works)

- We have

$$Az - ct_1 2^d = A(y + cs_1) - c(t - t_0) = Ay + cAs_1 - c(As_1 + s_2) + ct_0$$
$$= w - cs_2 + ct_0 \,.$$

- Since $\| -ct_0 \|_\infty < \gamma_2$, we have

$$w_1' = \text{UseHint}(h, Az - ct_1 2^d, 2\gamma_2)$$
$$= \text{HighBits}(Az - ct_1 2^d - ct_0, 2\gamma_2)$$
$$= \text{HighBits}(Az - ct, 2\gamma_2)$$
$$= \text{HighBits}(w - cs_2, 2\gamma_2) \,.$$



- And, since $\|\text{LowBits}(w - cs_2, 2\gamma_2)\|_\infty < \gamma_2 - \beta$ and $\|cs_2\|_\beta \le \beta$, we have

$$\text{HighBits}(w - cs_2, 2\gamma_2) = \text{HighBits}(w, 2\gamma_2) = w_1.$$

- Thus, $w_1' = w_1$. □

Dilithium parameter sets

	Security category	q	n	(k,ℓ)	η	d	γ_1	τ	β	γ_2	λ	ω
ML-DSA-44	2	$2^{23}-2^{13}+1$	256	(4,4)	2	13	2^{17}	39	78	$(q-1)/88$	128	80
ML-DSA-65	3	$2^{23}-2^{13}+1$	256	(6,5)	4	13	2^{19}	49	196	$(q-1)/32$	192	55
ML-DSA-87	5	$2^{23}-2^{13}+1$	256	(8,7)	2	13	2^{19}	60	120	$(q-1)/32$	256	75

Categories 2, 3, 5: Fastest known attacks require at least as much resources as needed for exhaustive key search on, respectively, a 256-bit hash function, a 192-bit block cipher , and a 256-bit block cipher.

Key sizes and signature sizes

- Verification key (ρ, t_1)
  ρ: 256 bits, t_1: $8 \times 256 \times 10$ bits
  Total size: 2,592 bytes
- Signing key $(\rho, tr, s_1, s_2, K, t_0)$
  ρ: 256 bits, tr: 512 bits, s_1: $7 \times 256 \times 3$ bits,
  s_2: $8 \times 256 \times 3$ bits, K: 256 bits, t_0: $8 \times 256 \times 13$ bits
  Total size: 4,896 bytes
- Signature $\sigma = (\tilde{c}, z, h)$
  $\tilde{c}$: 512 bits, z: $7 \times 256 \times 20$ bits, h: 75+8 bytes
  Total size: 4,627 bytes

ML-DSA-87 parameters

- $q = 2^{23} - 2^{13} + 1 = 8{,}380{,}417 \approx 2^{23}$
- $n = 256$
- $(k, \ell) = (8,7)$
- $\eta = 2$
- $d = 13$
- $\gamma_1 = 2^{19}$
- $\tau = 60,\ \beta = 120$
- $\gamma_2 = (q-1)/32 = 262{,}144 \approx 2^{18}$
- $\alpha = (q-1)/16 = 523{,}776 \approx 2^{19}$
- $\lambda = 256$
- $\omega = 75$

Hint vector representation

- Input: $h_1, h_2, \ldots, h_k \in \{0,1\}^{256}$, $u \leq \omega$ of whose bits are 1.
- Output: Byte array y of length $\omega + k$.
  - Initialize the elements of y to 0.
  - Let $y[\omega + 1]$ = number of 1 bits in h_1.
  - Let $y[\omega + 2]$ = number of 1 bits in h_1 and h_2.
  - ...
  - Let $y[\omega + k]$ = number of 1 bits in $h_1, h_2, \ldots, h_k$.
  - Let $y[1], y[2], \ldots, y[u]$ be the bit positions of the 1's in $h_1, h_2, \ldots, h_k$.
- Exercise: Describe how one can reconstruct the $h_1, h_2, \ldots, h_k$ from the array y.

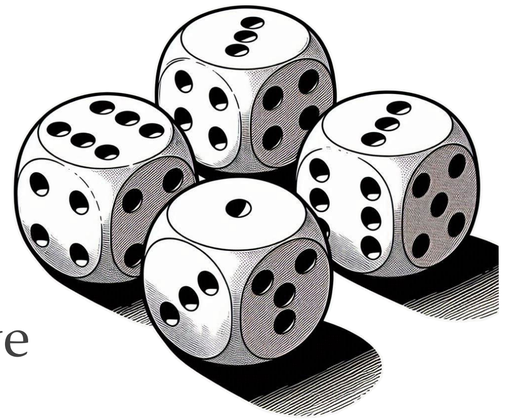
Key sizes and signature sizes (2)

	Security category	Signing key (bytes)	Verification key (bytes)	Signature (bytes)
ML-DSA-44	2	2560	1312	2420
ML-DSA-65	3	4032	1952	3309
ML-DSA-87	5	4896	2592	4627

Expected number of iterations (signing)

1. Let $p_1 = \text{Prob}(\|z\|_\infty < \gamma_1 - \beta)$.
   - Recall that $z = y + cs_1$ where $y \in \tilde{S}^{\ell}_{\gamma_1}$. Suppose that a coefficient of cs_1 is $u \in [-\beta, \beta]$. Then the corresponding coefficient of z is in $(-\gamma_1 + \beta, \gamma_1 - \beta)$ provided that the corresponding coefficient of y is in $(-\gamma_1 + \beta - u, \gamma_1 - \beta - u)$. This interval for the y coefficient has length $2(\gamma_1 - \beta) - 1$, which is independent of u. Hence,

$$p_1 = \left(\frac{2(\gamma_1 - \beta) - 1}{2\gamma_1}\right)^{256\ell} = \left(1 - \frac{\beta + 1/2}{\gamma_1}\right)^{256\ell} \approx e^{-256\ell\beta/\gamma_1}.$$

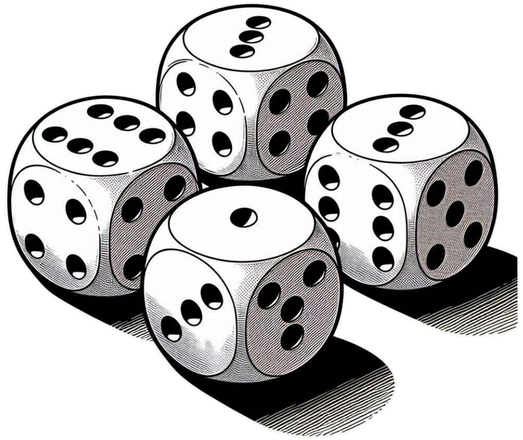


2. Let $p_2 = \text{Prob}(\|r_0\|_\infty < \gamma_2 - \beta)$.
   - Assuming that $\text{LowBits}(w - cs_2, 2\gamma_2)$ are uniformly distributed, we have

$$p_2 = \left(\frac{2(\gamma_2 - \beta) - 1}{2\gamma_2 + 1}\right)^{256k} \approx \left(1 - \frac{\beta + 1/2}{\gamma_2}\right)^{256k} \approx e^{-256k\beta/\gamma_2}.$$

3. Let $p_3 = \text{Prob}(\| - ct_0\|_\infty < \gamma_2$ and the number of 1's in h is $\leq \omega)$.
   - Heuristically, $p_3 > 0.98$.

Expected number of iterations (signing) (2)



- The probability that all four requirements are met is $p_1 p_2 p_3$.
- Thus, the expected number of iterations in signing is
$$1/p_1 p_2 p_3 \approx 1/p_1 p_2 \approx e^{256\beta(\ell/\gamma_1 + k/\gamma_2)}.$$
- The expected number of iterations for ML-DSA-44, ML-DSA-65, and ML-DSA-87 signing is 4.25, 5.1, and 3.85, respectively.

Security

- Compression of the public key component t doesn't affect the security claim for Dilithium.
- Claim: Dilithium is existentially unforgeable against chosen-message attack assuming that D-MLWE and MSIS are intractable, and H is modelled as a random function.

Omitted details

FIPS 204

Federal Information Processing Standards Publication

Module-Lattice-Based Digital Signature Standard

- ExpandA(ρ) uses SHAKE128.
- ExpandS(ρ') uses SHAKE256.
- ExpandMask(ρ'', κ) uses SHAKE256.
- Description of SampleInBall : $\{0,1\}^{2\lambda} \rightarrow B_\tau$.
- Formatting for bit strings and byte strings.
- An optional upper bound on number of iterations in signature generation.
- Number-Theoretic Transform
  - (NTT) for fast polynomial multiplication in $R_q = \mathbb{Z}_{2^{23}-2^{13}+1}[x]/(x^{256}+1)$ (see V4).

References

CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme

Léo Ducas[1], Eike Kiltz[2], Tancrède Lepoint[3], Vadim Lyubashevsky[4], Peter Schwabe[5], Gregor Seiler[6] and Damien Stehlé[7]

IACR Transactions on Cryptographic Hardware and Embedded Systems, Vol. 2018, No. 1, pp. 238-268.

CRYSTALS-Dilithium

Algorithm Specifications and Supporting Documentation (Version 3.1)

Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler and Damien Stehlé

pq-crystals.org/dilithium

csrc.nist.gov/pubs/fips/204/final

NIST Information Technology Laboratory COMPUTER SECURITY RESOURCE CENTER

FIPS 204

Federal Information Processing Standards Publication

Module-Lattice-Based Digital Signature Standard

