Cryptanalysis

Lattices

John Manferdelli
JohnManferdelli@hotmail.com

Lattices

- The set $\Lambda = \mathbb{Z}b_1 + \mathbb{Z}b_2 + \ldots + \mathbb{Z}b_n$, where $b_1, b_2, \ldots, b_n$ are linearly independent is called a lattice.
- $\Lambda^* = \{y \in \mathbb{Z}^n : (x, y) \in \mathbb{Z}, \forall x \in \Lambda\}$
- $vol(\Lambda) = \det(b_1, b_2, \ldots, b_n)$, where $b_1, b_2, \ldots, b_n$ are the generators of Λ. Note that any set of generators will do since they are related by uni-modular transformations.
- Let Λ be a lattice
  - The CVP problem is: Find $v \in \Lambda$: $\left||v|\right| = min_{w \in \Lambda, w \neq 0}(||w||)$
  - The CVP_γ problem is: Find $v \in \Lambda$: $\left||v|\right| \leq \gamma \cdot min_{w \in \Lambda, w \neq 0}(||w||)$
- Volume of n-dimensional sphere: $V_n(r) \approx \frac{1}{\sqrt{n\pi}}(\sqrt{\frac{2\pi e}{n}}r)^n$

Definitions

- Hermite Normal Form (HNF)

$$\begin{bmatrix} >0 & 0 & 0 & \cdots & 0 & 0 & \ldots & 0 \\ \geq 0 & >0 & 0 & 0 & 0 & 0 & \ldots & 0 \\ \geq 0 & \vdots & >0 & \ddots & \vdots & 0 & \ldots & 0 \\ \geq 0 & \geq 0 & \geq 0 & \ldots & 0 & 0 & \ldots & 0 \\ \geq 0 & \geq 0 & \geq 0 & \cdots & >0 & 0 & \ldots & 0 \end{bmatrix}$$

Minkowski's Theorem

- Let Λ be a lattice in $\mathbb{R}^n$ and suppose $S \subseteq \mathbb{R}^n$ is a convex, centrally symmetric region. If $vol(S) > 2^n \det(\Lambda)$ then S has a non-zero lattice point of Λ.

  Suppose first that Λ' is the simple lattice generated by $e_1, e_2, \ldots e_n$. Represent a point $\mathrm{r} \in S$ as $\mathrm{r} = (\alpha_1 + x_1, \alpha_2 + x_2, \ldots, \alpha_n + x_n)$ with $\alpha_i \in \mathbb{Z}$ and $|x_i| \leq 1$, for $1 \leq i \leq n$. Define $T(r) = (x_1, x_2, \ldots, x_n)$. If $S_1 \cap S_2 = \emptyset$, $vol(S_1 \cup S_2) = vol(S_1) + vol(S_2)$. So, if S has the property that $T(t) \neq T(s), \forall s \neq t \in S$, then $vol(S) = vol(T(S))$. Note that $vol(T(S)) \leq 1$. So, if $vol(S) > 1$, there are at least two points $r^{(1)} = \left({\alpha_1}^{(1)} + x_1, {\alpha_2}^{(1)} + x_2, \ldots, {\alpha_n}^{(1)} + x_n\right)$, $r^{(2)} = \left({\alpha_1}^{(2)} + x_1, {\alpha_2}^{(2)} + x_2, \ldots, {\alpha_n}^{(2)} + x_n\right)$, where ${\alpha_i}^{(1)} \neq {\alpha_i}^{(2)}$ for some i. Since S is centrally symmetric, $-r^{(1)}, -r^{(2)} \in S$; finally, note that $0 \neq r^{(1)} - r^{(2)} \in \mathbb{Z}^n$. Similarly, if $vol(S) > 2^n$, there are at least $2^n + 1$ points $r^{(i)}$, $1 \leq i \leq 2^n + 1$ with $0 \neq r^{(i)} - r^{(j)} \in \mathbb{Z}^n, i \neq j$ for at least two, say $r^{(i)}$ and $r^{(j)}$, all corresponding coordinates in $r^{(i)} - T(r^{(i)})$ and $r^{(j)} - T(r^{(j)})$ are equal $(mod\ 2)$. Thus, $0 \neq \frac{r^{(i)} - r^{(j)}}{2} \in \mathbb{Z}^n$. But since S is convex, $\frac{r^{(i)} - r^{(j)}}{2} \in S$. So, the result holds for the simple lattice. Suppose now that Λ is genetated by $a_1, a_2, \ldots a_n$ and put $A = [a_1, a_2, \ldots a_n]$. $e_i = A^{-1}(a_i)$, so $vol(\Lambda') = \frac{vol(\Lambda)}{\det(\Lambda)}$ and the simple lattice result thus implies the general theorem.

q-ary lattices and other definitions

- Definition: If $q \in \mathbb{Z}$, a lattice, Λ, is called q-ary if $q\mathbb{Z}^n \subseteq \Lambda \subseteq \mathbb{Z}^n$.
- Suppose $A \in \mathbb{Z}^{m \times n}$, $\Lambda_q(A) = \{\mathrm{y} \in \mathbb{Z}^n : y = A^T x \ (mod\ q),\ x \in {\mathbb{Z}_q}^m\}$. Note $\Lambda_q(A)$ is q-ary.
- ${\Lambda_q}^{\perp}(A) = \{\mathrm{y} \in \mathbb{Z}^n : Ay = 0 \ (\mathrm{mod\ q})\}$
- $\lambda_1(\Lambda) = \left|min_{v \in \Lambda}\ ||\mathrm{v}|\right||$
- $\lambda_n(\Lambda) = \ min_S(max_{v \in S}\left||v|\right|)$, where $S \subseteq \Lambda$ is a set of linearly independent vectors, $|S| = n$

- Solving CVP in ${\Lambda_q}^{\perp}(A)$ when A is chosen uniformly at random is as hard as worse case CVP.

Some simple results

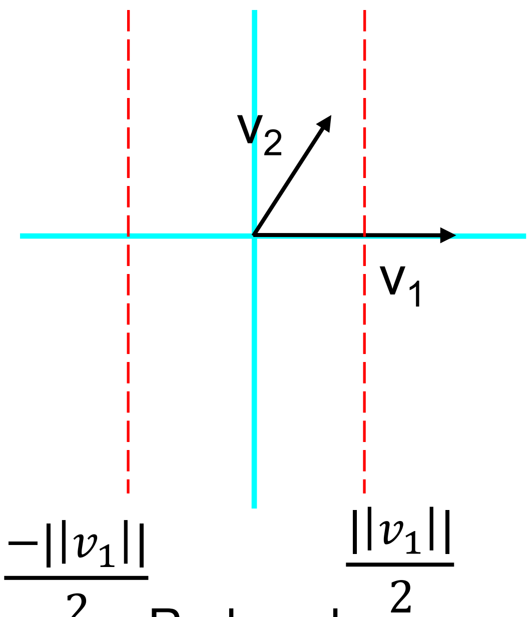
- Remember S is centrally symmetric if $s \in S$ implies $-s \in S$, and S is convex if $s, t \in S$ implies $us + (1-u)t \in S, u \in [0,1]$. We used this in proving Minkowski's Theorem.

- Theorem: $\lambda_1(\Lambda) \leq \sqrt{n}\,\det(\Lambda)^{\frac{1}{n}}$

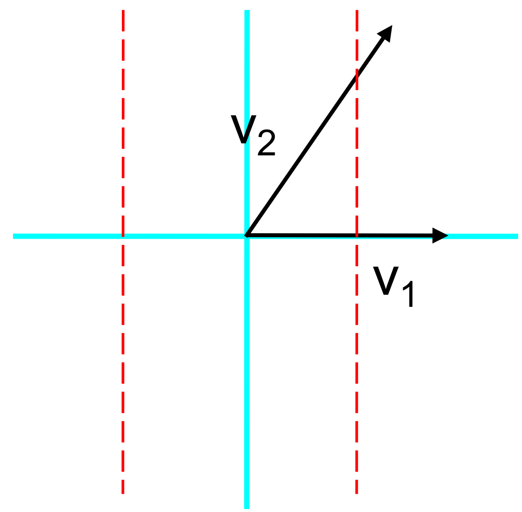
  Let B_r be a ball centered at 0 having radius $r = \sqrt{n}\,\det(\Lambda)^{\frac{1}{n}}$. Let $(x_1, x_2, \dots, x_n)$ be the coordinates of a vector v, with respect to the basis generating the lattice Λ, if $|x_i| \leq 1$ for $1 \leq i \leq n$, $v \in B_r$. So $-\det(\Lambda)^{\frac{1}{n}}\ (1,1, \dots, 1)$ and $\det(\Lambda)^{\frac{1}{n}}\ (1,1, \dots, 1)$ as well as the line joining them are in B_r so $vol(B_r) \geq 2^n \det(\Lambda)$ and the result follows from Minkowski's theorem.

Reduced Basis

- $\langle v_1, v_2 \rangle$ is reduced if
  - $||v_2|| \leqq |v_1||$; and,
  - $-1/2||v_1||^2 \leqq (v_1, v_2) \leqq 1/2||v_1||^2$.



Reduced



Not

Good basis and Gram-Schmidt Orthogonalization

- Good basis for lattices are orthonormal when that is possible. If a basis, $b_1, b_2, \ldots, b_n$ for Λ, is orthonormal, then, for example, $vol(\Lambda) = ||b_1|| \cdot ||b_2|| \cdot \cdots \cdot ||b_n||$
- The orthogonality defect of a basis $b_1, b_2, \ldots, b_n$ is $\frac{||b_1|| \cdot ||b_2|| \cdot \ldots \cdot ||b_n||}{\det(b_1, b_2, \ldots, b_n)}$
- Given a space generated by $b_1, b_2, \ldots, b_n$ can also be generated by a set of vectors, ${b_1}^*, {b_2}^*, \ldots, {b_n}^*$ with the property that $({b_i}^*, {b_j}^*) = 0, i \neq j$. Th Gram-Schmidt orthogonalization procedure computes this.

GSO, given, $b_1, b_2, \ldots, b_n$, compute ${b_1}^*, {b_2}^*, \ldots, {b_n}^*$

1. put ${b_1}^* = b_i$.
2. for $i = 2, i \leq n$

$$ {b_i}^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j \, , \; \mu_{i,j} = \frac{({b_j}^*, b_i)}{({b_j}^*, {b_j}^*)} $$

Size Reduction

- Definition: A basis $b_1, b_2, \ldots, b_n$ is *size reduced* if $\left|\mu_{i,j}\right| \leq \frac{1}{2}$, in the Gram-Schmidt orthogonalization procedure.
- If $b_1, b_2, \ldots, b_n$ is a basis for Λ, in general, $b_1{}^*, b_2{}^*, \ldots, b_n{}^*$ is not also a lattice basis because $\mu_{i,j}$ is generally not an integer. We can find a "nearly" orthogonal set of vectors $b_1', b_2', \ldots, b_n{}'$ in Λ, by rounding the $\mu_{i,j}$. $b_1', b_2', \ldots, b_n{}'$ is also a basis for the lattice and has the same gram Schmidt basis, $b_1{}^*, b_2{}^*, \ldots, b_n{}^*$. When performing GSO on this *reduced* basis, $\left|\mu_{i,j}\right| \leq \frac{1}{2}$.

Size-reduction

for $i = 2$, $i \leq n$

for $j = i - 1$, $j \geq 1$

$b_i \leftarrow b_i - \lceil \mu_{ij} \rfloor b_j$

for $k = 1, k \leq j$

$\mu_{ik} \leftarrow \mu_{ik} - \lceil \mu_{ij} \rfloor \mu_{jk}$

Size reduction and basis reordering

- Let $b_1, b_2, \ldots, b_n$ be a basis for Λ, and $b_1^*, b_2^*, \ldots, b_n^*$ the resulting GSO basis. Let $B_i = ||b_i||^2$. Then $b_1, b_2, \ldots, b_n$ satisfies the *Lovasz condition* with factor δ if it is size reduced and $\left(\delta - {\mu_{i+1,i}}^2\right)B_i \leq B_{i+1}$. The LLL algorithm calculates such a basis.

LLL Algorithm

Given $b_1, b_2, \ldots, b_n$ generating Λ, calculate the LLL reduced basis

1. Reduce the basis $b_1, b_2, \ldots, b_n$ with the size reduction algorithm and calculate $b_1^*, b_2^*, \ldots, b_n^*$ and μ_{ij}
2. Compute $B_i = ||{b_i}^*||^2$, $i = 1, 2, \ldots, n$
3. for $i = 1, i < n$
    4. If ($\left(\delta - {\mu_{i+1,i}}^2\right)B_i > B_{i+1}$)
        5. Swap b_i and b_{i+1}
        6. Go to 1
7. return $b_1, b_2, \ldots, b_n$

Example (LLL including GSO)

- LLL ($\delta = \frac{3}{4}$)
- $b_1 = (2,3,14)^T$, $b_2 = (0,7,11)^T$, $b_3 = (0,0,23)^T$.
  - GSO: $b_1{}^* = b_1$, $b_2{}^* = b_2 - \mu_{21} b_1$, $\mu_{21} = \frac{(b_1{}^*, b_2)}{(b_1{}^*, b_1{}^*)} = \frac{21+154}{4+9+196} = \frac{175}{209}$, $\mu_{31} = \frac{322}{209}$, $\mu_{31} = \frac{3473}{4905}$. $b_2{}^* = (-\frac{350}{209}, \frac{938}{209}, -\frac{151}{209})^T$
  - Size reduction: $b_2 = b_2 - \lceil \mu_{21} \rfloor b_1 = (-2,4,-3)^T$, $\mu_{21} = \mu_{21} - \lceil \mu_{21} \rfloor = -\frac{34}{209}$; $b_3 = b_3 - \lceil \mu_{32} \rfloor b_2 = (-2,4,20)^T$, $\mu_{31} = \mu_{31} - \lceil \mu_{31} \rfloor = -\frac{1432}{4905}$; last change is $b_3 = b_3 - \lceil \mu_{31} \rfloor b_1 = (-4,1,6)^T$, $\mu_{31} = \mu_{31} - \lceil \mu_{31} \rfloor = -\frac{79}{209}$.
  - Now, $b_1 = (2,3,14)^T$, $b_2 = (-2,4,-3)^T$, $b_3 = (-4,1,6)^T$.
  - $B_1 = 209$, $B_2 = \frac{4905}{209}$, $B_3 = \frac{103684}{4905}$. Lovasz condition is not satisfied for $i = 1$: since $(\delta - \mu_{21}{}^2) B_1 > B_2$. So swap b_1 and b_2.
  - Applying GSO we get $\mu_{21} = \frac{-34}{29}$, $\mu_{31} = \frac{-6}{29}$, and $\mu_{32} = \frac{2087}{4905}$.
  - Size reduction produces: $b_2 = b_2 - \lceil \mu_{21} \rfloor b_1 = (0,7,11)^T$ and $\mu_{21} = \frac{-6}{29}$. μ_{31} and μ_{32} don't change. μ_{32}

Example (LLL including GSO) - continued

- Now Lovasz condition is satisfied for $i = 1$ since $(\delta - \mu_{21}{}^2)B_1 < B_2$. but not $i = 2$ since $(\delta - \mu_{32}{}^2)B_2 < B_3$. swap b_2 and b_3.
  - Now, $b_1 = (-2,4,-3)^T$, $b_2 = (-4,1,6)^T$, $b_3 = (0,7,11)^T$. $B_1 = 29$, $B_2 = \frac{1501}{29}$, $B_3 = \frac{103684}{1501}$. GSO coefficients are $\mu_{21} = \frac{-6}{29}$, $\mu_{31} = \frac{-5}{29}$, and $\mu_{32} = \frac{2087}{1501}$. Applying size reduction does not affect b_2 or μ_{21}. $b_3 = b_3 - \lceil \mu_{32} \rfloor b_2 = (4,6,5\,)^T$, $\mu_{31} = \mu_{31} - \lceil \mu_{32} \rfloor \mu_{21} = \frac{1}{29}$, $\mu_{31} = \frac{586}{1501}$. Both Lovasz conditions now hold.
  - LLL basis is thus $b_1 = (-2,4,-3)^T$, $b_2 = (-4,1,6)^T$, $b_3 = (4,6,5)^T$. Notice $||b_1||$ is actually the shortest vector in Λ.

LLL Properties

- Suppose we apply LLL to $b_1, b_2, \dots, b_n$, with ${b_1}^*, {b_2}^*, \dots,\ {b_n}^*$ and $B_1, B_2, \dots, B_n$ defined as above. With $X = min_{v\in\Lambda}(\left||b_i|\right|)$ and $\frac{1}{4} < \delta < 1$, LLL runs in $O(n^6 \ln(x)^3)$.
    1. $B_i \le \left||b_i|\right|^2 \le (\frac{1}{2} + 2^{i-2})B_i$
    2. $||b_i|| \le 2^{\frac{i-1}{2}}||{b_i}^*||$
    3. $\lambda_1(\Lambda) \ge min_i(||{b_i}^*||)$
    4. $||b_1|| \le 2^{\frac{n-1}{2}}\lambda_1(\Lambda)$
    5. $\det(\Lambda) \le \prod_{i=1}^{n}||b_i|| \le 2^{\frac{n(n-1)}{4}}\det(\Lambda)$
    6. $||b_i|| \le 2^{\frac{(n-1)}{4}}\det(\Lambda)^{\frac{1}{n}}$
- If w is a vector in $\mathbb{R}^n$ and the lattice basis for Λ is $b_1, b_2, \dots, b_n$ with $B = [b_1, b_2, \dots, b_n]$, the coefficients for w are $u = B^{-1}(w)$. w is not necessarily in the lattice but if we take each element in u and round it, $B \downharpoonleft B^{-1}(w)$ 1∈ Λ. This is *Babai rounding.*

Attack on RSA using LLL

- Attack applies to messages of the form "M xxx" where only "xxx" varies (e.g.- "The key is xxx") and xxx is small.
- From now on, assume $M(x) = B + x$ where B is fixed
  - $|x|<Y$.
  - Not that $E(M(x)) = c = (B + x)^3 \ (mod\ n)$
  - $f(x) = (B+x)^3 - c = x^3 + a_2 x^2 + a_1 x + a_0$ (mod n).
- We want to find x: $f(x) = 0 \ (mod\ n)$, a solution to this, m, will be the corresponding plaintext.

Attack on RSA using LLL

- To apply LLL, let:
  - $v_1 = (n, 0, 0, 0)$,
  - $v_2 = (0, Yn, 0, 0)$,
  - $v_3 = (0, 0, Y^2 n, 0)$,
  - $v_4 = (a_0 , a_1 Y, a_2 Y^2, a_3 Y^3)$
- When we apply LLL, we get a vector, b_1:
  - $||b_1|| \leqq 2^{3/4} |\det(v_1, v_2, v_3, v_4)| = 2^{3/4} n^{3/4} Y^{3/2}$ …. *Equation 1*.
- Let $b_1 = c_1 v_1 + \ldots + c_4 v_4 = (e_0, Ye_1, Y^2 e_2, Y^3 e_3)$. Then:

Attack on RSA using LLL

- Now set $g(x) = e_3x^3 + e_2x^2 + e_1x + e_0$.
- From the definition of the e_i, $c_4 f(x) = g(x) \ (mod\ n)$, so if m is a solution of $f(x) \ (mod\ n), g(m) = c_4 f(m) = 0 \ (mod\ n)$.
- The trick is to regard g as being defined over the real numbers, then the solution can be calculated using an iterative solver.
- If $Y<2^{(7/6)}n^{(1/6)}$, $|g(x)| \leqq 2||b_1||$.
- So, using the Cauchy-Schwartz inequality, $||b_1|| \leqq 2^{-1}n$.
- Thus $|g(x)| < n$ and $g(x) = 0$ yielding 3 candidates for x.
- Coppersmith extended this to small solutions of polynomials of degree d using a d+1 dimensional lattice by examining the monic polynomial $f(T) = 0 \ (mod\ n)$ of degree d when $|x| \leqq n^{1/d}$.

Example attack on RSA using LLL

- p= 757285757575769, q= 2545724696579693.
- n= 1927841055428697487157594258917.
- B= 200805000114192305180009190000.
- $c = (B+m)3, 0 <= m < 100$.
- $f(x) = (B+x)3 - c = x^3 + a_2x^2 + a_1x + a_0 \ (mod\ n)$.
  - a_2= 602415000342576915540027570000
  - a_1= 1123549124004247469362171467964
  - a_0= 587324114445679876954457927616
  - v_1= (n,0,0,0)
  - v_2= (0,100n,0,0)
  - v_3= (0,0,10^4n,0)
  - v_4= (a_0, $a_1$100, $a_2 10^4$,10^6)

Example attack on RSA using LLL

- Apply LLL, b_1=
  - 308331465484476402v_1 + 589837092377839611v_2 +
  - 316253828707108264v_3 + (-1012071602751202635)v_4 =
  - (246073430665887186108474, -577816087453534232385300, 405848565585194400880000, -1012071602751202635000000)
- g(x)= (-1012071602751202635) t^3 + 40584856558519440088 t^2 + (-57781608745353442323853) t +246073430665887186108474.

- Roots of g(x) are 42.0000000, (-.9496±76.0796i)
- The answer is 42.

GGH Public Key System

- Pick $n, M \in \mathbb{N}$ and σ is “small”, say $\sigma = 4$
- Plaintext: $\mathcal{M} = \{x: -M \ \leq x \leq M\}$, Cipher-space: $\mathcal{C} \in \ \mathbb{Z}^n$.
- Gen:
  1. Choose $B \in \mathbb{Z}^{n \times n}$ with small entries $|B_{ij}| \leq \sigma$
  2. Check B is invertible. B is the secret key.
  3. $H = HNF(B)$
- Enc
  1. For $\vec{m} \in \mathcal{M}^n$, choose $\vec{r} \in (-\sigma, \sigma)^n$ uniformly at random
  2. $\vec{c} = H\vec{m} + \vec{r}$
- Dec
  1. Babai round $\vec{m} = H^{-1}B \lfloor ((B^{-1}(\vec{c})) \rceil$

- Works if $\lfloor B^{-1}\ (r) \rceil = 0$.

GGH Example

- $B = \begin{bmatrix} 2 & -3 & 1 & -4 \\ -2 & 1 & 0 & 4 \\ -1 & 3 & 2 & 1 \\ -1 & -4 & 3 & -2 \end{bmatrix}$, H= $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 44 & 18 & 4 & 49 \end{bmatrix}$
- $B^{-1} = \frac{1}{49}\begin{bmatrix} 61 & 45 & 10 & -27 \\ -10 & -13 & 8 & -2 \\ 29 & 23 & 16 & -4 \\ 33 & 38 & 3 & -13 \end{bmatrix}, H^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{-44}{49} & \frac{-18}{49} & \frac{-4}{49} & \frac{1}{49} \end{bmatrix}$
- $m = (3,-4,1,3)^T$, $r = (-1,1,1,-1)^T$, $c = Hm + r = (2,-3,2,210)^T$
- $B^{-1}c = \frac{1}{7}(-809,-55,-117,-396)^T$, $\lfloor B^{-1}c \rceil = (-116,-8,-17,-57)^T$
- $B \lfloor B^{-1}c \rceil = (3,\ -4,\ 1,\ 211)^T$
- $m = H^{-1}B \lfloor B^{-1}c \rceil = (3,\ -4,\ 1,\ 3)^T$

From Heiko Knopse

Learning with Errors (LWE)

- Based on solving noisy linear equations $mod\ q$. Choose $\overrightarrow{a_i} \in \mathbb{Z}_q{}^n$ uniformly at random. $\vec{s} \in \mathbb{Z}_q{}^n$ is a secret and $m \geq n$ approximate equations $\overrightarrow{a_i} \cdot \vec{s} = b_i\ (mod\ q)$. Errors, $e_1, e_2, \dots, e_n$ are chosen from distribution χ.
- Reduces to LWE:
  – Search LWE problem: Given a_{ij}, $(\vec{b} + \vec{e})$ find $\vec{s}$.
  – Decision LWE: Distinguish, with non-negligible probability, between $\vec{b} = A\vec{s}$+ $\vec{e}$ and $\vec{b} \in \mathbb{Z}_q{}^m$ chosen uniformly at random given $A, \vec{b}$
- Errors chosen from distribution, $p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{x^2}{2\sigma^2})$. Often use $s = \sigma\sqrt{2\pi}$ as parameter specifying distribution.
- Regev's showed it is possible to pick parameters so that solving an LWE cipher is equivalent to solving worst-case LWE .
  – Theorem (Regev): Let $n \in \mathbb{N}$ be a security parameter, $m, q \in \mathbb{N}$, polynomial in n and $\chi = D_{\mathbb{Z},s}$ a discrete Gaussian distribution with $s = \alpha q > 2\sqrt{n}$, $0 < \alpha < 1$. Then solving the LWE decision problem is at least as hard as quantumly solving $SIVP_\Upsilon$ on an arbitrary n-dimensional lattice where $\gamma = \widetilde{O}\ ({}^n/_\alpha)$.

LWE cryptosystem

- Given $(n \geq m, l, t, r, q, \chi)$ where χ is a probability distribution $\mathbb{Z}_q$, message space is ${\mathbb{Z}_2}^l$ and cipher space is ${\mathbb{Z}_q}^n \times {\mathbb{Z}_q}^l$.
- Key Gen
  1. Choose $\mathrm{S} \in {\mathbb{Z}_q}^{n \times l}$, uniformly from the distribution χ.
  2. Choose $A \in {\mathbb{Z}_q}^{m \times n}$, and $\mathrm{E} \in {\mathbb{Z}_q}^{m \times l}$ uniformly from the distribution χ.
  3. Private key is S, public key is $(A, P = AS + E)$
- Enc
  1. For $\vec{v} \in {\mathbb{Z}_2}^l$, choose $\vec{a} \in \{0,1\}^m$, uniformly at random
  2. $\overrightarrow{CT} = (\vec{u} = A^T \vec{a}, \vec{c} = P^T \vec{a} + \upharpoonleft \frac{q}{2} \downharpoonleft \vec{v}))$
- Dec
  1. Compute $\upharpoonleft (\upharpoonleft \frac{q}{2} \downharpoonleft)^{-1} (\vec{c} - S^T \vec{u}) \upharpoonleft (mod\ 2)$
- Decryption may have errors. Suppose χ is a discrete Gaussian $D_{\mathbb{Z},s}$. Then $E^T \vec{a}$ has magnitude $\leq \sqrt{m}s$ with high probability. Error occurs if $E^T \vec{a} \geq \frac{q}{4}$. One can show that for any n, $\exists q, m, s$ such that the error is small and the underlying LWE problem is hard.

LWE example

- $n = 4, q = 23, m = 8, \alpha = \frac{5}{23}, s = 5, \sigma = \frac{s}{\sqrt{2\pi}}, l = 4$
- $A^{m\times n} = A = \begin{bmatrix} 9 & 5 & 11 & 13 \\ 13 & 6 & 6 & 2 \\ 6 & 21 & 17 & 18 \\ 22 & 19 & 20 & 8 \\ 2 & 17 & 10 & 21 \\ 10 & 8 & 17 & 11 \\ 5 & 16 & 12 & 2 \\ 5 & 7 & 11 & 7 \end{bmatrix}, S^{n\times l} = S = \begin{bmatrix} 5 & 2 & 9 & 1 \\ 6 & 8 & 19 & 1 \\ 19 & 18 & 9 & 18 \\ 9 & 2 & 14 & 18 \end{bmatrix}$

From Heiko Knopse

LWE example

- $E^{m\times l} = E = \begin{bmatrix} 0 & 22 & 1 & 21 \\ 0 & 22 & 22 & 22 \\ 6 & 21 & 17 & 18 \\ 22 & 22 & 22 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 1 & 22 & 1 & 22 \\ 22 & 0 & 0 & 1 \end{bmatrix}, P^{m\times l} = \mathrm{P} = \begin{bmatrix} 10 & 5 & 21 & 7 \\ 3 & 1 & 13 & 1 \\ 19 & 15 & 6 & 13 \\ 22 & 22 & 22 & 0 \\ 9 & 20 & 20 & 17 \\ 15 & 21 & 1 & 2 \\ 0 & 12 & 3 & 19 \\ 16 & 2 & 7 & 15 \end{bmatrix},$

From Heiko Knopse

LWE example

- Encrypt $\vec{v} = (1,0,1,1)^T$, using a $= (1,1,0,1,0,0,0,1)^T$
  - $\lfloor \frac{23}{2} \vec{v} \rceil = (12,0,12,12)^T$,
  - $(u, c) = \left(A^T a, P^T a + \lfloor \frac{23}{2} m \rceil \right) = ((3,14,2,7)^T, (4,5,7,5)^T)(mod\ 23)$

- Decrypt:
  - $\vec{v}' = c - S^T u = (11,21,12,10)^T\ (mod\ 23)$,
  - $\lfloor \frac{1}{12} \vec{v}' \rceil\ (mod\ 2) = (1,0,1,1)^T$

LWE example

- Encrypt $m = (1,0,1,1)^T$, using a $= (1,1,0,1,0,0,0,1)^T$
  - $\lfloor \frac{23}{2} m \rceil = (12,0,12,12)^T$,
  - $(u, c) = \left(A^T a, P^T a + \lfloor \frac{23}{2} m \rceil\right) = ((3,14,2,7)^T, (4,5,7,5)^T)(mod\ 23)$
- Decrypt:
  - $m' = c - S^T u = (11,21,12,10)^T\ (mod\ 23)$,
  - $\lfloor \frac{1}{12} m' \rceil\ (mod\ 2) = (1,0,1,1)^T$

From Heiko Knopse

LWE/Ring-LWE parameters

Level	n	q	s	P	P&A	c	Exp
Low	128	4093	8.87	2.9×10^5	7.4×10^5	3.8×10^3	30
High	320	4093	8	4.9×10^5	17.7×10^5	17.4×10^3	42

- Ring-LWE cuts ciphertext by factor of n

Ring-LWE

- Put $R = R_{n,q} = \frac{\mathbb{Z}_q[x]}{x^n+1}, n = 2^k, R \approx {\mathbb{Z}_q}^n$. $a \in R$, generates ideal (a) corresponding to a q-ary ideal lattice.
- Ring LWE: Given $\mathrm{a} \in R$, and $b = as + e$, for $\mathrm{s}, \mathrm{e} \in R$, find s.
- Solving R-LWE is at least as hard as solving CVP_γ on arbitrary ideal lattices

NTRU Public Key System

- NTRU is a ring lattice-based system.
- $R = \frac{\mathbb{Z}[x]}{x^N - 1}, R_p = \frac{\mathbb{Z}_p[x]}{x^N - 1}, R_q = \frac{\mathbb{Z}_q[x]}{x^N - 1}$
- $(c_0 + c_1 x + \cdots + c_{N-1}) = (a_0 + a_1 x + \cdots + a_{N-1}) \otimes (b_0 + b_1 x + \cdots + b_{N-1})$, where $c_k = \sum_{i+j=k \ (mod\ N)} a_i b_j$
- $\mathcal{T}(d_1, d_2)$ is the set of "ternary" polynomials of degree $< N$, having d_1 coefficients equal to 1, having d_2 coefficients equal to -1, and remaining coefficients equal to 0.
- Pick N, p prime and $q, d \in \mathbb{N}$, $(p, q) = (N, q) = 1$, $q > (6d + 1)p$.

NTRU Public Key System

- KeyGen
  1. Pick $f, g \ \in R, f \in \mathcal{T}(d+1, d), g \in \mathcal{T}(d, d)$.
  2. Find f_p, f_q: $f \cdot f_p = 1 \ (mod \ p), \ \ f \cdot f_q = 1 \ (mod \ q), \mathrm{h} = f_q \cdot g \ (mod \ q)$.
  3. Public key is (N, p, q, h), private key is f.
- Plaintext is $m \in R_p$, ciphertext is $c \in R_q$
- Encryption
  1. Chose random $r \in R, r \in \mathcal{T}(d, d)$.
  2. $c = prh + m \ (mod \ q)$.
- Decryption
  1. Compute $a = fc \ (mod \ q)$
  2. Plaintext is f_pa.
  3. Verify that $a = fc = f(prh + m)(mod \ q) = pfrf_q g + fm(mod \ q) = prg + fm(mod \ q)$.

NTRU Example

- $N = 5, p = 3, q = 29, d = 1, f = x^4 + x^3 - 1, g = x^3 - x^2$
- $f_p = -x^3 - x^2 + x - 1, f_q = -5x^4 + 8x^3 + 3x^2 + 11x + 13$
- $h = f_q g = 8x^4 + 2x^3 + 11x^2 + 13x - 5 \ (mod\ 29)$
- $r = x^4 - x$
- $c = prh + m = 8x^4 + 21x^3 + 25x^2 + 20x + 15 \ (mod\ 29)$
- $a = fc = -2x^4 + 2x^3 + 4x^2 - 3x + 1 \ (mod\ 29)$
- We check $a = prg + fm$ in R
- $m = x^3 + x$

From Heiko Knopse

Some NIST Round 3 entries

- Public-Key Encryption/KEMs
  - Classic McEliece
  - CRYSTALS-KYBER
  - NTRU
  - SABER
- Digital Signatures
  - CRYSTALS-DILITHIUM
  - FALCON
  - Rainbow

- Public-Key Encryption/KEMs (Alternates)
  - BIKE;
  - FrodoKEM
  - HQC
  - NTRU Prime
  - SIKE
- Digital Signatures
  - GeMSS
  - Picnic
  - SPHINCS+

Winner: Dilithium (signing), Kyber (key-encapsulation)

Common features of Dilithium and Kyber

- Ring is $\mathbb{Z}_p[x]/(x^{256}+1)$ in both cases
  - $p = 3329$ for Kyber
  - $p = 2^{23} - 2^{13} + 1 = 8380417$ for Dilithium
  - So, the same modular arithmetic we all grew up with.
- For $p = 3329$, there is a primitive (and hence 128 primitive) 256th roots of unity (You are not expected to understand this).
  - As a result, $x^{256} + 1$ factors into coprime 128 quadratics
  - Allows us to perform a "Number Theory Transform" that turns convolution into pointwise multiplication for ring operations giving a nice speedup
- For $p = 8380417$, there is a primitive (and hence 256 primitive) 512th roots of unity
  - As a result, $x^{256} + 1$ factors into coprime 256 linear polys
  - Allows us to perform a "Number Theory Transform" that turns convolution into pointwise multiplication for ring operations giving a nice speedup

Useful definitions

- $r^+ = r\ (mod\ q), q > r^+ \geq 0$
- $r' = r\ mod^{\pm}(m)$ means $r' = r\ (mod\ m)$ and $-\frac{m}{2} \leq r' \leq \frac{m}{2}$, if m is even; $-\frac{m}{2} < r' \leq \frac{m}{2}$, if m is odd

- $decompose(r, \alpha, q)$
  - $r_0 = r^+\ mod^{\pm}(\alpha)$
  - if $r^+ - r_0 == (q - 1)$
    - $r_1 = 0\ , r_0 = q - 1$
  - else
    - $r_1 = \frac{r^+ - r_0}{\alpha}$
  - return (r_1, r_0)

Useful definitions

- $lowbits(x, \alpha, q)$
  - $(r_1, r_0) = decompose(x, \alpha, q)$
  - return r_0

- $highbits(x, \alpha, q)$
  - $(r_1, r_0) = decompose(x, \alpha, q)$
  - return r_1

- $power2round(r, d, q)$
  - $r^+ = r\ mod(q)$
  - $r_0 = \ r^+\ mod^{\pm}(2^d)$
  - return $(\frac{r^+ - r_0}{2^d}, r_0)$

Examples

- $decompose(r, \alpha, q)$ examples (second shows roundoff edge case)

$\boldsymbol{q}$	$\boldsymbol{\alpha}$	$\boldsymbol{r}$	$\boldsymbol{r\ mod^{\pm}(\alpha)}$	$\mathbf{r} - \boldsymbol{r\ mod^{\pm}(\alpha)}$	$\boldsymbol{r_0}$	$\boldsymbol{r_1}$
17	8	5	-3	8	-3	1
17	8	15	-1	16	-2	0
3329	104	50	50	0	50	0
3329	104	100	-4	104	-4	1

SHAKE-256/SHAKE-128

- $H(v, d) = SHAKE256(v, d)$
- $H_{128}(v, d) = SHAKE128(v, d)$

- $RAWSHAKE256(J, d) = KECCAK[512](J||11, d)$
- $SHAKE256(M, d) = RAWSHAKE256(M||11, d)$
- $RAWSHAKE128(J, d) = KECCAK[256](J||11, d)$
- $SHAKE128(M, d) = RAWSHAKE128(M||11, d)$

- Note
  - $SHA3_{256}(M) = KECCAK[512](M||11,256)$
  - $SHA3_{512}(M) = KECCAK[1024](M||11,1024)$

Number Theory Transform (NTT)

- $p = 3329, p - 1 = 2^8 \cdot 13.$
- $\mathbb{Z}_p$ has a primitive $256th$ root of unity ($\zeta = 17$ is a primitive root) but no 512 root of unity, so $x^{256} + 1$ factors into 128 coprime quadratic factors of the form $(x^2 - \xi)$, $17^{128} = -1.$
- $x^{256} + 1 = \prod_{k=0}^{127}(x^2 - \zeta^{2 \cdot bitrev_7(k)+1}).$
- $bitrev_7(k)$ reverses the bit order in a 7-bit byte, $k.$
  - $x^{256} + 1 = (x^2 - 17) \cdot (x^2 - 17^{129}) \cdot \cdots \cdot (x^2 - 17^{255})$
- For $p = 8380417, \zeta = 1753$ is a primitive 512th root of unity,
  - $\mathrm{p} - 1 = 2^{13}(2^{10} - 1) = 2^{13} \cdot 3 \cdot 11 \cdot 31.$
- Because of this, an analog of the Chinese remainder theorem holds in $R_p = \frac{\mathbb{Z}_p(x)}{x^{256}+1}.$

NTT for DIlithium

- NTT: $R_p \rightarrow T_p, f \mapsto \hat{f}, T_q = \prod_{i=0}^{255} \mathbb{Z}_q$
- For $f \in R_p$
  - $\hat{f} = \prod_{i=0}^{511} f \left(mod\ x - \zeta^{2i+1}\right), \zeta = 1753$, so each element in the vector is just an element of $\mathbb{Z}_p$
  - If $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{511} x^{255}$
  - $\hat{a} = (a(r_0), a(-r_0), \ldots, a(r_{127}), a(-r_{127})), r_i = \zeta^{i+128}$
- Multiplication is then pointwise

Dilithium template

- Gen
  - $\boldsymbol{A} \leftarrow R_q^{k\times l}$
  - $(\boldsymbol{s_1}, \boldsymbol{s_2}) \leftarrow S_\eta^l \times S_\eta^k$
  - $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s_1} + \boldsymbol{s_2}$
  - return $(pk = (\boldsymbol{A}, \boldsymbol{t}), sk = (\boldsymbol{A}, \boldsymbol{t}, \boldsymbol{s_1}, \boldsymbol{s_2}))$

- Verify$(pk, M, \sigma = (\boldsymbol{z}, c)$)
  - $\boldsymbol{w_1'} = highbits(\boldsymbol{A}\boldsymbol{z} - c\boldsymbol{t}, 2\gamma_2)$
  - return $||z||_\infty < \gamma_1 - \beta \ \wedge$
    $c == H(M||\boldsymbol{w_1'})$

- Sign(sk, m)
  - z=⊥
  - while $z ==\perp$
    - $\boldsymbol{y} \leftarrow S_{\gamma_1 - 1}^l$
    - $\boldsymbol{w_1} = highbits(Ay, 2\gamma_2)$
    - $c \in B_{60}, c = H(M||w_1)$
    - // view c as polynomial in R_q
    - $\boldsymbol{z} = \boldsymbol{y} + \boldsymbol{c}\boldsymbol{s_1}$
    - **if** $||\boldsymbol{z}||_\infty < \gamma_1 - \beta \ \vee$
      $||lowbits(Ay - cs_2, 2\gamma_2)||_\infty \geq \gamma_2 - \beta$
      - z=⊥
  - return $\sigma = (\boldsymbol{z}, c)$

For real Dilithium, $k = 5, l = 4$

Dilithium security argument -1

$\eta = 5, \gamma_1 = \frac{q-1}{16}, \gamma_2 = \frac{\gamma_1}{2}, R_q = \frac{\mathbb{Z}_q[x]}{x^{256}+1}, q = 2^{23} - 2^{13} + 1,$
$q - 1 = 2^{13} \cdot 3 \cdot 11 \cdot 31.\ k = 5, l = 4$

1. $\boldsymbol{A} \leftarrow R_q^{k \times l}, (\boldsymbol{s_1}, \boldsymbol{s_2}) \leftarrow S_\eta^l \times S_\eta^k, \boldsymbol{t} = \boldsymbol{A}\boldsymbol{s_1} + \boldsymbol{s_2}, pk = (\boldsymbol{A}, \boldsymbol{t}), sk = (\boldsymbol{A}, \boldsymbol{t}, \boldsymbol{s_1}, \boldsymbol{s_2}), S = R_q$
2. $\boldsymbol{y} \leftarrow S_{\gamma_1 - 1}^l, \boldsymbol{w_1} = highbits_{2\gamma_2}(A\boldsymbol{y})$
   - Write coefficients of $\mathbf{w} = A\boldsymbol{y}$, as $\boldsymbol{w}^{[i]} = (2\gamma_1)\boldsymbol{w}_1^{[i]} + \boldsymbol{w}_0^{[i]}$
   - $\boldsymbol{w_1} = highbits_{2\gamma_2}(A\boldsymbol{y})$ then $\boldsymbol{w}_0^{[i]} < \gamma_2$
3. $\boldsymbol{c} \in B_{60}, \boldsymbol{c} = H(M||w_1)$. Set $\beta = max_i((cs_1)^{[i]})$. Then $\beta \leq 60\eta$.
4. Set $\boldsymbol{z} = \boldsymbol{y} + \boldsymbol{c}\boldsymbol{s_1}$, if any coefficient of $z > \gamma_1 - \beta$, reject and start over.
5. If any coefficient of $lowbits_{2\gamma_2}(A\boldsymbol{z} - c\boldsymbol{t}) > \gamma_2 - \beta$, reject and start over.
   - Note: $A\boldsymbol{z} - c\boldsymbol{t} = A\boldsymbol{y} - c\boldsymbol{s_2}$
   - coefficients of $\boldsymbol{z} \leq \gamma_1 - \beta$, coefficients of $lowbits_{2\gamma_2}(A\boldsymbol{z} - c\boldsymbol{t}) \leq \gamma_2 - \beta$
6. Signature is $\sigma = (\boldsymbol{z}, c)$
   - $c \in B_{60}$ is ensured by SampleInBall in the final algorithm.
   - Parameters chosen so that expected rejections in steps 4 and 5 is between 4 and 7.

Dilithium security argument - 2

Verification

- $A\boldsymbol{z} - c\boldsymbol{t} = A\boldsymbol{y} - c\boldsymbol{s_2}$
- To show $highbits_{2\gamma_2}(A\boldsymbol{z} - c\boldsymbol{t}) = highbits_{2\gamma_2}(A\boldsymbol{y})$, we need only show $highbits_{2\gamma_2}(A\boldsymbol{y}) = highbits_{2\gamma_2}(A\boldsymbol{y} - c\boldsymbol{s_2})$.
  - This follows because $\left|lowbits_{2\gamma_2}(A\boldsymbol{y} - c\boldsymbol{s}_2)\right|_\infty < \gamma_2 - \beta$; and,
  - The coefficients of $||c\boldsymbol{s}_2||_\infty < \beta$
  - Adding $c\boldsymbol{s}_2$ never causes a carry of γ_2 from the lowbits and, hence, $highbits_{2\gamma_2}(A\boldsymbol{z} - c\boldsymbol{t}) = highbits_{2\gamma_2}(A\boldsymbol{y})$
  - Now we can compute $highbits_{2\gamma_2}(\boldsymbol{w}_1)$ and hence $H(M||\boldsymbol{w}_1)$

Template → Dilithium

- NTT is used to speed multiplications.
- Produce hint, h to help verifier calculate w_1' in verify.
  - $r_1 \leftarrow Highbits(r), \upsilon \leftarrow Highbits(r+z), h \leftarrow [[r_1 \neq v]]$
  - $h \leftarrow MakeHint(- \ll (ct_0) \gg, w - \ll cs_2 \gg + \ll \mathrm{c}t_0 \gg)$
  - $\tilde{c} \leftarrow H(\mu||w_1, \lambda)$
- Drop $d = 13$, bottom bits in t. $\omega = 75$ is man number of 1's in h.
- In final algorithm, A, is generated from a seed using SHAKE-128.

- Notes:
  - Compute $\boldsymbol{w_1'} = highbits_{2\gamma_2}(\boldsymbol{Az} - c\boldsymbol{t})$ from the compressed public key.
  - SampleinBall, guarantees $c \in B_{60}$ using Fisher-Yates shuffle on $H(M||\boldsymbol{w_1})$

Template → Dilithium

- $\xi \leftarrow H(\{0,1\}^{256}),\ (\rho, \rho', K) \leftarrow H(\xi, 512), \hat{A} \leftarrow Expand(\rho)$
- $tr \leftarrow H(pk,\ 512)$, sign: $\mu \leftarrow H(tr||M, 512), (\hat{s}_1, \hat{s}_2) \leftarrow ExpandS(\rho')$.
- $(t_1, t_0) \leftarrow Power2Round(t, d), sk \leftarrow (\rho, K, tr, s_1, s_2, t_0)$
- $Usehint(h, w'_{appx})$, $\mathrm{r}, \mathrm{z} \in \mathbb{Z}_q$
  - $m \leftarrow \frac{q-1}{2\gamma_2}, (r_1, r_0) \leftarrow decompose(r)$
  - If $(h == 1 \wedge r_0 > 0)$ return $(r_1 + 1) mod\ m$
  - If $(h == 1 \wedge r_0 \leq 0)$ return $(r_1 - 1) mod\ m$
  - return r_1

Algorithm	Public key	Private key size	Signature size
ML-DSA-87	4864	2592	4595

Dilithium, unedited, motivation

- Basic scheme is Fiat-Shamir MSA-DL with aborts.
- Classic version with discrete log is:
  - Prover and verifier know $(g, y = g^x)$. Prover knows x.
  1. Prover generates r, sends commitment g^r.
  2. Verifier sends c.
  3. Prover returns $s = r - cx$.
  4. Verifier can check $g^s \cdot y^c = g^r$
- Non interactive version replaces c with hash of $g^r || M$
- LWE version
  - Publish A, $t = As_1 + s_2$
  - Prover: Pick, y, commit by sending $w_{approx} = Ay + y_2$, y_2 has small coefficients.
  - Verifier: Send challenge, c.
  - Prover: $z = y + cs_1$
  - Verifier: Check z and $Az - tc \approx w_{approx}$

Dilithium (simplified)

- Remember $A^{k \times l}$ is generated randomly from $R = \mathbb{Z}_p[x]/(x^{256} + 1)$.
- s_1 is a vector of dimension l with entries from R has random coefficients $\leq \eta$
- s_2 is a vector of dimension k with entries from R has random coefficients $\leq \eta$
- $t = As_1 + s_2$

Sign

$y \coloneqq S_{\gamma_1 - 1}{}^{l}$
$w_1 \coloneqq \mathrm{highbits}(\mathrm{Ay}, 2\gamma_2)$
$c \coloneqq SH(M||w_1)$
$z \coloneqq y + cs_1$
`return(`z, c`)`

Verify

$w_1{}' \coloneqq \mathrm{highbits}(\mathrm{Az} - \mathrm{ct}, 2\gamma_2)$
$c' \coloneqq SH(M||w_1{}')$
`Check` $c' == c$ `AND` $||z||_\infty <$
$\gamma_1 - \beta$

Dilithium (less simplified)

Parameters: $p = 8380417,\ k = 5, l = 4, \gamma_1 = \frac{p-1}{16}, \gamma_2 = \frac{\gamma_1}{2}, \eta = 5, \beta = 275$

$R_p = \frac{\mathbb{Z}_p[x]}{x^{256}+1}$

- KeyGen
  - $A \in R_p^{k\times l}$, selected from random distribution over R_p
  - $(s_1, s_2) \in {S_\eta}^k \times {S_\eta}^l$, selected at random, ${S_\eta}^k$ consists of elements of ${R_p}^k$ with coefficients $\leq \eta$
  - Set $t = As_1 + s_2$
  - Public key is (A, t), Private key is (s_1, s_2)

For the sake of compression A is generated from a seed and SHAKE-256

Dilithium

- Sign(pk, sk, M) --- simplified
  1. $z = \perp$
  2. while ($z = \perp$) {
  3. $\quad y = S_{\gamma_1}{}^{l} - 1$
  4. $\quad w_1 = highbits(Ay, 2\gamma_2)$
  5. $\quad c = SHAKE - 256(M||w_1)$
  6. $\quad z = y + cs_2$
  7. $\quad$ if ($||z||_\infty \geq \gamma_1 - \beta$) OR $lowbits(Ay - cs_2, 2\gamma_1) \geq \gamma_2 - \beta$) then $z = \perp$
  8. }

  Signature is (z, c)

- Real Dilithium uses a number of functions to generate A from a seed. It also has a hedged version and a deterministic version. The hedged version avoids some possible side channels.

Dilithium

- Verify(pk, M,z, c) --- simplified
  1. $w_1' = highbits(Az - ct, 2\gamma_2)$
  2. Return true if $||z||_\infty \leq \gamma_1 - \beta$ AND $c = SHAKE - 256(M||w_1')$, otherwise return false

Useful definitions

- u$sehint(h, r)$
  - $m = \frac{p-1}{2\gamma_2}$
  - $(r_1, r_0) = decompose(r, 2\gamma_2, p)$
  - If $h == 1$ and $r_0 > 0$ then return $(r_1 + 1)mod(m)$
  - If $h == 1$ and $r_0 \leq 0$ then return $(r_1 - 1)mod(m)$
  - return r_1

- $makehint(z, r)$. // are highbits of $z + r$ different from highbits of r
  - $r_1 = highbits(r)$
  - $v_1 = highbits(r + z)$
  - return $r_1 \neq v_1$

Useful definitions

- $RejNTTPoly(\rho)$ // returns NTT polynomial
  - $c = 0\ ; j = 0$
  - while ($j < 256$)
    - $\hat{a}[j] = coeffFromThreeBytes(H_{128}(\rho||c), H_{128}(\rho||c+1), \ldots, H_{128}(\rho||c+2))$
    - $c \mathrel{+}= 3$
    - If ($\hat{a}[j] \neq \perp$) **then** $j++$
  - return $\hat{a}$

- $RejBoundedPoly(\rho)$
  - $c = 0\ ; j = 0$
  - while ($j < 256$)
    - $z = H(\rho)[c]$
    - $z_0 = CoeffFromHalfByte(z\ mod(16), \eta)$
    - $z_1 = CoeffFromHalfByte(\downharpoonleft z/16 \downharpoonright, \eta)$
    - If ($z_0 \neq \perp$)
      - $a_j = z_0; j++$
    - If ($z_1 \neq \perp$ and $j < 256$)
      - $a_j = z_1; j++$
    - $\boldsymbol{c}++$
  - return a

Useful definitions

- $ExpandA(\rho)$
  - $for\ (r = 0; r < k; k++)$
    $for\ (s = 0; s < l$
    $\hat{A}[r,s] = RejNTTPoly(\rho||IntegerToBits(s,8)||INtegerToBits(r,8))$
  $return\ \hat{A}$

- $ExpandS(\rho)$
  - $for\ (r{=}0; r{<}l; r{+}{+})$
    - $s_1[r] = RejBoundedPoly(\rho||IntegerToBits(r,16))$
  - $for\ (r{=}0; r{<}k; r{+}{+})$
    - $s_2[r] = RejBoundedPoly(\rho||IntegerToBits(r + l16))$
  $return\ (s_1, s_2)$

- $ExpandMask(\rho, \mu)$
  - $c = 1 + bitlen(\gamma_1 - 1)$
  - $for(r = 0; r < l; r++)$
    - $n = IntegerToBits(\mu + r, 16)$
    - $v = (H(\rho||n)[32rc], H(\rho||n)[32rc + 1], \ldots, H(\rho||n)[32rc + 32c - 1])$
    - $s[r] = BitUnpack(v, \gamma_1 - 1, \gamma_1)$
  - $return\ s$

Useful definitions

- // Calculate c(x), coefficients are 1, -1 or 0
- $SampleinBall(\rho, \tau)$
  - $c(x) \coloneqq 0$; k=8;
  - $for(i = 256 - \tau; i < 256; i++)$
    - $while(H(\rho)[[k]] > i)$ // $H(\rho)[[k]]$ is kth byte

      k++

      $j = H(\rho)[k]$

      $c_i = c_j$

      $c_j = (-1)^{H(\rho)[i+\tau-256]}$ // [k] is bit position k

      $k++$
  - *return c*

SampleInBall generates an element of B_{60} pseudorandomly; it is based on the Fisher-Yates shuffe. The first 8 bytes of H(ρ) choose the signs of the nonzero entries of c; subsequent bytes choose the positions of those nonzero entries

Here H is SHAKE256 used as an XOF.

NTT for Dilithium

- $NTT(w)$ --- outputs $\widehat{w_j} = (w(\zeta_0), w(-\zeta_0), w(\zeta_1), w(-\zeta_1), \dots, w(-\zeta_{127}))$
  - $for(j = 0; j < 256; j++)\ \hat{w}[j] = w[j]$
    - $k = 0; len = 128$
    - $while(len \geq 1)$
      - $start = 0$
      - while $(start < 256)$
        - $k++$
        - $zeta = \zeta^{bitrev(k)}\ mod(q)$
        - for$(j = start; j \leq start + len - 1)$
          - $t = zeta \cdot \hat{w}[j + len]$
          - $\hat{w}[j + len] = \hat{w}[j] - t$
          - $\hat{w}[j] = \hat{w}[j] + t$
        - $start += 2 \cdot len$
      - $len = len/2$

NTT for Dilithium

- $NTT^{-1}(\widehat{w})$
  - $for(j = 0; j < 256; j++)\ \ w[j] = \widehat{w}[j]$
  - $k = 256; len = 1$
  - $while(len < 256)$
    - $start = 0$
    - while $(start < 256)$
      - $k--$
      - $zeta = \ \zeta^{bitrev(k)}\ mod(q)$
      - for$(j = start; j \leq start + len - 1)$
        - $t = w[j]$
        - $w[j] = t + w[j + len]$
        - $w[j + len] = t - w[j + len]$
        - $w[j + len] = zeta \cdot w[j + len]$
        - $start \mathrel{+}= 2 \cdot len$
    - $len = len/2$
  - $f = 8347861$
  - $for(j = 0; j < 256; j++)\ \ w[j] = f \cdot w[j]$

Dilithium, unedited, motivation

- Preliminary lattice version is prover generates: $A \in {\mathbb{Z}_q}^{k \times l}$, $S_1 \in {\mathbb{Z}_q}^{l \times n}$, $S_2 \in {\mathbb{Z}_q}^{k \times n}$, with short coefficients and computes $t = AS_1 + S_2$. Public key is (A, t). Private key is (S_1, S_2)
  1. Prover generates $y \in {\mathbb{Z}_q}^l$ with "small coefficients". Sends commitment as Ay
  2. Verifiers sends challenge $c \in {\mathbb{Z}_q}^n$ with small coefficients
  3. Prover returns $z = y + S_1 c$.
  4. Verifier checks coefficients of z are small and that $Az - tc \approx Ay$
- To avoid having z leak S_1, signer applies rejection sampling to z.
- Dilithium
  1. Uses elements of $R_q = \frac{\mathbb{Z}_q[x]}{x^{256}+1}$ rather than $\mathbb{Z}_q$.
  2. Uses a seed, ρ, to generate A, compresses t by dropping low order bits.
  3. Signs a message representative, μ, which is a hash of the public key and the message
  4. Uses a rounded version of $w = Ay$, w_1.
  5. Provides a hint, h, to help reconstruct w_1 from z

Dilithium parameters for security category 5

Parameter	Meaning	Value
q	modulus	8380417
d	# dropped bits from t	13
τ	# ± 1s in $c(x)$	60
λ	Collision strength	256
γ_1	Coefficient range of y	2^{19}
γ_2	Low order rounding range	$\frac{q-1}{32}$
(k,l)	Dimensions of A	$(8,7)$
η	Private key range	2
$\beta = \tau \cdot \eta$		120
ω	Max # of 1's in hint	75

Dilithium, Keygen

- Keygen
  1. $\xi \ \coloneqq \ \mathbb{Z}_2{}^{256}$ (random)
  2. $(\rho, \rho', K) \coloneqq H(\xi, 1024)$, (256, 512, 256) bits respectively
  3. $\hat{A} \coloneqq ExpandA(\rho)$
  4. $(s_1, s_2) \coloneqq ExpandS(\rho')$
  5. $t \ \coloneqq \ NTT^{-1}\left(\hat{A}\, NTT(s_1)\right) + \ s_2$
  6. $(t_1, t_0) \coloneqq Power2Round(t, d)$
  7. $pk \ \coloneqq \ pkEncode(\rho, t_1)$
  8. $tr \ \coloneqq H(BytesToBits(pk), 512)$
  9. $sk \ \coloneqq skEncode(\rho, K, tr, s_1, s_2, t_0)$
  10. return (pk, sk)

Dilithium, Sign

1. $(\rho, K, tr, s_1, s_2, t_0) \coloneqq skdecode(sk)$
2. $\widehat{s_1} \coloneqq NTT(s_1), \widehat{s_2} \coloneqq NTT(s_2), \widehat{s_1} \coloneqq NTT(t_0); \hat{A} \coloneqq ExpandA(\rho)$
3. $\mu \coloneqq H(tr||M, 512); rnd \coloneqq \mathbb{Z}_2{}^{256}$
4. $\rho' \coloneqq H\left(K\middle||rnd\middle||\mu, 512\right)$
5. $\kappa = 0$
6. while(1) {
   a. $y = ExpandMask(\rho', \kappa)$
   b. $w \coloneqq NTT^{-1}\left(\widehat{A}\, NTT(y)\right), \quad w_1 \coloneqq highbits(w, 2\gamma_2)$
   c. $\tilde{c} \coloneqq H(\mu||w1Encode(w_1), 2\lambda)$
   d. $(\hat{c}_1, \hat{c}_2) \coloneqq$ first 256 and last $256 - 2\lambda$ bits
   e. $c \coloneqq SampleBall(\hat{c}_1); \hat{c} = NTT(c)$
   f. $cs_1 \coloneqq NTT^{-1}(\hat{c}\, \hat{s}_1); cs_2 \coloneqq NTT^{-1}(\hat{c}\widehat{s}_2);$
   g. $z \coloneqq y + cs_1$
   h. $r_0 \coloneqq lowbits(w - cs_2)$
   i. If ($||z||_\infty \geq \gamma_1 - \beta$ or $||r_0||_\infty \geq \gamma_2 - \beta$ then **continue**
   j. $ct_0 \coloneqq NTT^{-1}(\tilde{c}t_0); h \coloneqq makehint(-ct_0, w - cs_2 + ct_0)$
   k. If ($\left||ct_0|\right|_\infty < \gamma_2$ and # 1's in $h \leq \omega$) then **break**
   l. $\kappa += l$ }

9. $\sigma \coloneqq sigEncode(\tilde{c}, z\, mod^{\pm}(q), \mathrm{h})$

Dilithium, Verify

- Verify
  1. $(\rho, t_1) \coloneqq pkdecode(pk)$
  2. $(\tilde{c}, z, h) \coloneqq sigdecode(\sigma)$
  3. $\hat{A} \coloneqq ExpandA(\rho)$
  4. $tr \coloneqq H(BytestoBits(pk), 512)$
  5. $\mu \coloneqq H(tr||M, 512)$
  6. $(\tilde{c}_1, \tilde{c}_2) \coloneqq$ first 256 and last $256 - 2\lambda$ bits
  7. $c \coloneqq SampleBall(\tilde{c}_1)$
  8. $w'_{appx} \coloneqq NTT^{-1}(\tilde{A} \cdot NTT(z) - NTT(c)NTT(t_1 2^d))$
  9. $w'_1 \coloneqq usehint(h, w'_{appx})$ // $w_{approx} = (Az - ct_1) \cdot 2^d$
  10. $\tilde{c}' \coloneqq H(\mu||w1Encode(w'_1, 2\lambda))$
  11. return $||z||_\infty < \gamma_1 - \beta$ and $\tilde{c} == \tilde{c}'$ and # 1's in $h \leq \omega$

Kyber

- Kyber is a key encapsulation algorithm that uses a public key encryption algorithm similar to Dilithium in conjunction with an encapsulation mechanism (Fujisaki-Okamoto transform) which converts a conditionally secure encryption into a CCA safe encapsulation. Here are some definitions.
  - $PRF_{\eta}(s, b) = shake256(s||b, 64 \cdot \eta)$
  - $XOF(\rho, i, j) = shake128(\rho||i||j)$
  - $H(s) = sha3_{256}(s),\ J(s) = shake256_{32}(s)$
  - $G(s) = sha3_{512}(s)$
  - NTT and NTT^{-1} are different for Kyber and Dilithium
- Fujisaki-Okamoto transform:
  - $\mathcal{E}_{pk}^{hy}(m) = \mathcal{E}_{pk}^{asym}(\sigma, H(\sigma, m))||\mathcal{E}_{G(\sigma)}^{sym}(m)$
  - σ is random string, G, H are hash functions, $\mathcal{E}_{G(\sigma)}^{sym}$ is symmetric encryption with key $G(\sigma)$ and $\mathcal{E}_{pk}^{asym}$ is original asymmetric encryption algorithm.

Useful definitions

- Parse: $\mathcal{B}^* \rightarrow R_q^n$
- Input: $B = b_0, b_1, \ldots \in \mathcal{B}^*$
- Output: $\hat{a} \in R_q^n$,

  $i = 0; j = 0;$

  while $j < i$

  $\quad d = b_i + 256 \cdot b_{i+1}$

  $\quad$ if $d < 19q$

  $\quad\quad \widehat{a_j} = d$

  $\quad\quad j++$

  $\quad i += 2$

  return $\hat{a}_0 + \hat{a}_1 x + \cdots + \hat{a}_{n-1} x^{n-1}$

Useful definitions

- $SamplePolyCBD(B, \eta))$ --- samples from (Central Binomial) distribution $D_\eta(R_q)$
  Output: $f \in R_q^{256}$
  $b \coloneqq ByteToBits(B)$
  $for(i = 0; i < 256; i++)$
  $x = \sum_{j=0}^{\eta-1} b[2i\eta + j]$; $\mathrm{y} = \sum_{j=0}^{\eta-1} b[2i\eta + \eta + j]$
  $f[i] \coloneqq (x - y)\ mod(q)$
  return f

- $Sample\left(a_1, a_2, \dots, a_\eta, b_1, \dots, b_\eta\right) \leftarrow \{0,1\}^{2\eta}$, output $\sum_{i=1}^{\eta}(a_i - b_i)$

- For central binomial distribution with $N = 10000, p = \frac{1}{2}, \sigma = \sqrt{Np(1-p)}$,
- $P(4900 \le n_1 \le 5100) = \sum_{j=4900}^{5100} \binom{N}{j} p^j (1-p)^{N-j} \approx \Phi(\frac{5100-5000}{50}) - \Phi(\frac{5100-5000}{50})$,
- Φ is CDF for normal distribution

Useful definitions

- $encode_d(x)$, x is an array of length 256, $m = 2^d$, $1 \leq d \leq 12$
  - for (i = 0; i < 256; i++)
    - $a = x[i]$
    - for (j=0; j < d; j++)
      - $b[d \cdot i + j] = a \ (mod\ 2)$
      - $a = \frac{a - b[d \cdot i + j]}{2}$
  - return bits-to-bytes(b)

- $decode_d(x)$, x is a byte array of length $32d$, $m = 2^d$, $1 \leq d \leq 12$
  - $b = bytes - to - bits(x)$
  - for (i=0; i < 256; i++)
    - $out[i] = \sum_{j=0}^{d-1} b[i \cdot d + j] \cdot 2^j$
  - return out

Useful definitions

- $SamplENTT()$ --- samples uniformly from T_q

  $i := 0; j := 0$

  $while\ (j < 256)$

  $d_1 = b[i] + 256(b[i+1]mod(16))$

  $d_2 = b[i+1]/16 + 16(b[i+2]$

  If $(d_1 < q)$

  $\hat{a}[j] = d_1\ ; j++$

  If $(d_2 < q$ and $j < 256$

  $\hat{a}[j] = d_2\ ; j++$

  $i += 3$

  return $\hat{a}$

Useful definitions

- $compress_q(x, d)$
  - $x \to \lceil \frac{2^d}{q} \cdot x \rfloor$
- $decompress_q(y, d)$

  $y \to \lceil \frac{q}{2^d} \cdot y \rfloor$

- $compress_q\left(decompress_q(x, d), d\right) = x$
- $decompress_q\left(compress_q(y, d), d\right) = t, (t - y) mod^{\pm}(q) \leq \lceil \frac{q}{2^{d+1}} \rfloor$

NTT for Kyber

- Note $x^{256} + 1 = \prod_{k=0}^{127}(x^2 - \zeta^{2\cdot bitrev_7(k)+1})$. This allows us to decompose an element in R_p into 127 coprime quadratics and recreate it using the Chinese Remainder Theorem. $\zeta = 17$.
- NTT: $R_p \to T_p, f \mapsto \hat{f}$
- For $f \in R_p$
  - $\hat{f} = (\prod_{i=0}^{127} f \left(mod\ x^2 - \zeta^{2rev_7(i)+1}\right)$
  - $\hat{g} = (\hat{g}_{0,0} + \hat{g}_{01}x, \hat{g}_{1,0} + \hat{g}_{11}x, \dots, \hat{g}_{127,0} + \hat{g}_{127,1}x)$
  - $NTT(g) = \hat{g} = (\hat{g}_{0,0}, \hat{g}_{01}, \hat{g}_{1,0}, \hat{g}_{11}, \dots, \hat{g}_{127,0}, \hat{g}_{127,1})$
  - Operations are performed element-wise
- For $\hat{h} = \hat{f} \cdot \hat{g}$,
  - $\hat{h}_{2i} + \hat{h}_{2i+1}x = (\hat{f}_{2i} + \hat{f}_{2i+1}x) \cdot (\hat{g}_{2i} + \hat{g}_{2i+1}x)(mod\ x^2 - \zeta^{2rev_7(i)+1})$

NTT for Kyber

- $NTT(f)$
  - $\hat{f} = f; k = 1$
  - for($len = 128; len \geq 2; len = len/2$)
    - for($start = 128; start < 256; start += 2len$)
    - $zeta = \zeta^{bitrev(k)}\ mod(q); k + +$
    - for($j = start; j < start + len; j + +$)
      - $t = zeta \cdot \hat{f}[j + len]\ mod(q)$
      - $\hat{f}[j + len] = \hat{f}[j] - t \text{ mod(q)}$
      - $\hat{f}[j] = \hat{f}[j] + t\ mod(q)$
  - return($\hat{f}$)

NTT for Kyber

- $NTT^{-1}(\hat{f})$
  - $f = \hat{f}; k = 127$
  - for($len = 2; len \leq 128 \leq; len = 2 \cdot len$)
    - for($start = 0; start < 256; start += 2len$)
    - $zeta = \zeta^{bitrev(k)}\, mod(q); k -= 1$
    - for($j = start; j < start + len; j + +$)
      - $t = f[j]\, mod(q)$
      - $f[j] = f[j] + f[j + len]\ \mathrm{mod(q)}$
      - $f[j + len] = zeta \cdot (f[j + len] - t)\ \ mod(q)$
  - return($f\ \cdot 3303\, mod(q)$)

NTT for Kyber

- $MultiplyNTT(\hat{f}, \hat{g})$
- For($i = 0; i < 128; i++$)
  - $(\hat{h}_{2i}, \hat{h}_{2i+1}) \leftarrow Basecasemultiply(\hat{f}_{2i}, \hat{f}_{2i+1}, \hat{g}_{2i}, \hat{g}_{2i+1}, \zeta^{2 \cdot bitrev(i)+1})$

- $Basecasemultiply(a_0, a_1, b_0, b_1, \gamma)$
  - $c_0 \leftarrow a_0 \cdot b_0 + a_1 \cdot b_1 \cdot \gamma$
  - $c_1 \leftarrow a_0 \cdot b_1 + a_1 \cdot b_0$
  - return (c_0, c_1)

Kyber (simplified a little)

- **Parameters:** $(p = 3329, R_p = \frac{\mathbb{Z}_p[x]}{x^{256}+1}, k = 4, \eta_1 = \eta_2 = 2)$, $\hat{x} = NTT(x)$

  $KeyGen_{PKE}$ generates a Dilithium-like key (see full version)

  $\hat{t} = \hat{A}\hat{s} + \hat{e}$, A is generated from seed ρ. $A \in R_p^{k\times k}$, $\boldsymbol{s}, \boldsymbol{e} \in R_p^k$
- $Enc_{PKE}(pk, m, r)$ // encrypt m
  1. $\boldsymbol{r} = Sample_{\eta_1}(PRF_{\eta_1}(r, N))$, $\boldsymbol{e_1} = Sample_{\eta_2}(PRF_{\eta_2}(r, N))$, $e_2 = Sample_{\eta_2}(PRF_{\eta_2}(r, N))$
  2. $\boldsymbol{u} = A^T\boldsymbol{r} + \boldsymbol{e_1}$
  3. $\upsilon = (\boldsymbol{t}^T \cdot \boldsymbol{r}) + e_2 + \lceil \frac{q}{2} \rfloor \cdot m$
  4. $c_1 = compress_p(\boldsymbol{u}, d_u))$, $c_2 = compress_p(\upsilon, d_v)$
  5. return $(\boldsymbol{c_1}, c_2)$
- $Dec_{PKE}(\boldsymbol{c_1}, c_2)$ // recover m
  1. $\boldsymbol{u} = decompress_p(\boldsymbol{c_1}, d_u)$, $\nu = decompress_p(c_2, d_v)$
  2. $w = \upsilon - \boldsymbol{s^T} \cdot \boldsymbol{u}$; return $compress_p(w, 1)$

  $compress$ removes the errors

Correctness

- $\boldsymbol{t} = decompress_p\left(compress_p(A\boldsymbol{s} + \boldsymbol{e}, d_u), d_u\right) = A\boldsymbol{s} + \boldsymbol{e} + \boldsymbol{c_t}$
- $\boldsymbol{u} = decompress_p\left(compress_p(A^T\boldsymbol{r} + \boldsymbol{e_1}, d_u), d_u\right) = A\boldsymbol{r} + \boldsymbol{e_1} + \boldsymbol{c_u}$
- $v = decompress_p\left(compress_p\left(\boldsymbol{t}^T \cdot \boldsymbol{r} + e_2 + \lfloor \frac{q}{2} \rceil \cdot m, d_v\right), d_v\right) =$

  $(A\boldsymbol{s} + \boldsymbol{e})^T \cdot \boldsymbol{r} + e_2 + \lfloor \frac{q}{2} \rceil \cdot m + c_v + \boldsymbol{c_t^T} \cdot \boldsymbol{r}$

- $|(A\boldsymbol{s} + \boldsymbol{e})^T \cdot \boldsymbol{r} + e_2 + c_v + \boldsymbol{c_t^T} \cdot \boldsymbol{r} - \boldsymbol{s^T} \cdot \boldsymbol{e_1} - \boldsymbol{s^T} \cdot \boldsymbol{c_u}| < \lfloor \frac{q}{4} \rceil$
- $v - \boldsymbol{s^T} \cdot \boldsymbol{u} = w + m \cdot \lfloor \frac{q}{2} \rceil, ||w||_\infty < \lfloor \frac{q}{4} \rceil$
- So, if $m' = compress_q(v - \boldsymbol{s^T} \cdot \boldsymbol{u}, 1)$,
- $\left|w + \lfloor \frac{q}{2} \rceil \cdot m - \lfloor \frac{q}{2} \rceil \cdot m'\right|_\infty \leq \lfloor \frac{q}{4} \rceil$, hence
- $|| \lfloor \frac{q}{2} \rceil \cdot (m - m') ||_\infty < 2 \cdot \lfloor \frac{q}{4} \rceil$ and so $m = m'$

Kyber simplified a little

- KEMKeygen
  - $z = \mathbb{Z}_2^{256}$(random)
  - $(ek_{PKE}, dk_{PKE}) = KeyGen_{PKE}$()
  - $ek_{KEM} = ek_{PKE}$; $dk_{KEM} = dk_{PKE} || e_{PKE} || \mathrm{H}(e_{PKE}) || \mathrm{z})$
  - return (ek_{KEM}, dk_{KEM})
- $KEMencaps(pk_{KEM})$
  1. m is a random 32-byte value
  2. $(K, r) = SHA3_{512}(m || H(e_{PKE}))$
  3. $c = Enc_{PKE}\ (ek, m, r)$
  4. return (K, c)
- $KEMdecaps(sk_{KEM})$
  1. $m' = Dec_{PKE}(dk, c)$
  2. $(K', r') = \ SHA3_{512}(m' || H(e_{PKE}))$
  3. $\overline{K} = SHAKE256\ (z || c, 32)$
  4. $c' = Enc_{PKE}(e_{PKE}, m', r')$
  5. If $(c == c')$ return K′ else error

Kyber parameters

Alg	n	q	k	η_1	η_2	d_u	d_v	Strength
KEM-512	256	3329	2	3	2	10	768	128
KEM-768	256	3329	3	2	2	10	1088	192
KEM-1024	256	3329	4	2	2	11	1568	256

ML-KEM-1024 is security category 5

Type	Encap-key	Decap-key	Ciphertext	Key
KEM-512	800	1632	768	32
KEM-768	1184	2400	1088	32
KEM-1024	1568	3168	1568	32

Size in bytes

Full Kyber

- $KeyGen_{PKE}$
  1. $d = \mathbb{Z}_2{}^{256}$*, random*
  2. $(\rho, \sigma) = G(d); N = 0$
  3. $for(i = 0; i < k; i++)$
     - $for\ (j = 0; j < k; j++)$
       - $\hat{A}[i, j] = SampleNTT(XOF(\rho, i, j)$
  4. $for(i = 0; i < k; i++)$
       - $s[i] = SamplePolyCBD\left(PRF_{\eta_1}(\sigma, N)\right); N++$
  5. $for(i = 0; i < k; i++)$
       - $e[i] = \ SamplePolyCDB\left(PRF_{\eta_1}(\sigma, N)\right); N++$
  6. $\hat{s} = NTT(s); \hat{e} = NTT(e)$
  7. $\hat{t} = \hat{A}\hat{s} + \hat{e}$
  8. $ek_{PKE} = ByteEncode_{12}(\hat{t})||\rho; dk_{PKE} = ByteEncode_{12}(\hat{s})$
  9. return (e_{PKE}, d_{PKE})

Full Kyber

- $Enc_{PKE}(ek, m, r)$ // r is randomness, $(\boldsymbol{u}, v) = (A^T\boldsymbol{r} + \boldsymbol{e_1}, \boldsymbol{t}^T \cdot \boldsymbol{r} + \boldsymbol{e_2})$,
  // μ added to term 2, ek is $384k + 32$ bytes, m and r are 32 bytes.

$N = 0$; $\hat{t} = ByteDecode_{12}(ek_{PKE}[0{:}\,384k])$; $\rho = ek_{PKE}[384k + 384k + 32]$

$for(i = 0; i < k; i++)$

$for\ (j = 0; j < k; j++)$

$\hat{A}[i, j] = SampleNTT(XOF(\rho, i, j)$

$for(i = 0; i < k; i++)$

$r[i] = SamplePolyCBD_{\eta_1}\left(PRF_{\eta_2}(r, N)\right); N++$

$for(i = 0; i < k; i++)$

$e_1[i] = SamplePolyCBD_{\eta_2}\left(PRF_{\eta_2}(r, N)\right); N++$

$e_2 = SamplePolyCBD_{\eta_2}\left(PRF_{\eta_2}(r, N)\right)$

$\hat{r} = NTT(r)$

$\boldsymbol{u}(x) = NTT^{-1}\left(\hat{A}^T\hat{r}\right) + e_1$

$\mu = decompress_1(decode_1(m)), v = NTT^{-1}(\hat{t}^T \cdot \hat{r}) + e_2 + \mu$

$c_1 = encode_{d_u}(compress_{d_u}(u)), c_2 = encode_{d_v}\left(compress_{d_v}(r)\right)$

return (c_1, c_2)

Full Kyber

- $Dec_{PKE}(dk, c_1, c_2)$ // dk is $384k$ bytes, $(\boldsymbol{c_1}, c_2)$ are $384d_u + d_v$ bytes.
  1. $c_1 = c[0: 32d_u k]; c_2 = c[32(d_u k + d_v)]$
  2. $\boldsymbol{u} = decompress_{d_u}\left(ByteDecode_{d_u}(c_1)\right)$
  3. $v = decompress_{d_v}\left(ByteDecode_{d_v}(c_2)\right)$
  4. $\hat{s} = ByteDecode_{12}(d_{PKE})$
  5. $w = v - NTT^{-1}(\hat{s}^T \cdot NTT(\boldsymbol{u}))$
  6. $m = \ ByteEncode_1(compress_1(w))$
  7. return m // 32 bytes

Full Kyber

- $Keygen_{KEM}()$
  - $z = \mathbb{Z}_2{}^{256}$ (random)
  - $(ek_{PKE}, dk_{PKE}) = KeyGen_{PKE}()$
  - $ek_{KEM} = ek_{PKE}; dk_{KEM} = dk_{PKE}||ek_{PKE}||\mathrm{H}(ek_{PKE})||\mathrm{z})$
  - return (ek_{KEM}, dk_{KEM})
  - $\hat{t} = \hat{A}\hat{s} + \hat{e}$, A is generated from seed ρ
  - return (ek_{PKE}, dk_{PKE}) // ek_{PKE} is $384k + 32$ bytes, dk_{PKE} is $384k +$ 96 bytes.

Full Kyber

- $KEMencaps(ek_{KEM})$. // ek_{KEM} is $384k + 32$ bytes
  1. $m \leftarrow Rand(32)$ // 32-byte
  2. $(K, r) = G(m||H(ek))$ // K is shared key, r is random input to Encaps
  3. $c = Enc_{PKE}\ (ek, m, r)$
  4. return (K, c) // K is 32 bytes

Full Kyber

- $KEMdecaps(\text{c}, dk)$ // c is $32(d_u k + d_v)$ bytes, dk is $768k + 96$ bytes.
  1. $dk_{PKE} = dk[0{:}\, 384k]$
  2. $ek_{PKE} = dk[384k{:}\, 768k + 32]$
  3. $h = dk[768k + 32{:}\, 768k + 64]$
  4. $z = dk[768k + 64{:}\, 768k + 96]$
  5. $m' = Dec_{PKE}(dk, c)$
  6. $(K', r') = G(m' || H(e_k))$
  7. $\bar{K} = J(z || c, 32)$
  8. $c' = Enc_{PKE}(ek, m', r')$
  9. If $(c == c')$ return K′ else return error

Kyber Notes

- Define $\boldsymbol{Adv}_{m,k,\eta}^{mlwe} =$
$|\Pr[\, b' = 1, \boldsymbol{A} \leftarrow R_q^{m\times k}; (\boldsymbol{s}, \boldsymbol{e}) \leftarrow \beta_\eta^k \times \beta_\eta^m; \boldsymbol{b} = \boldsymbol{As} + \boldsymbol{e}; b' = A(\boldsymbol{A}, \boldsymbol{b})] -$
$\Pr[\big(b' = 1, \boldsymbol{A} \leftarrow R_q^{m\times k}; b \leftarrow R_q^m\big)\,; b' = A(\boldsymbol{A}, \boldsymbol{b})]|.$

- **Theorem**: Suppose XOF and G are random oracles. For all adversaries, A, there are adversaries, B, C: $\boldsymbol{Adv}_{kyber,CPAPKE}^{cpa}((A) \leq 2\ \boldsymbol{Adv}_{k+1,k,\eta}^{mlwe}(B) + \boldsymbol{Adv}_{PRF}^{prf}(C)$
- **Theorem**: Suppose XOF and G are random oracles. For any classical adversary, A, that make at most q_{RO} to random oracles XOF, H, G there are adversaries B, C of the same running time: $\boldsymbol{Adv}_{kyber,CCAKEM}^{cca}((A) \leq 2\ \boldsymbol{Adv}_{k+1,k,\eta}^{mlwe}(B) + \boldsymbol{Adv}_{PRF}^{prf}(C) + 4\delta q_{RO}$
- **Theorem**: Suppose XOF and G are random oracles. For any quantum adversary, A, that make at most q_{RO} to random oracles XOF, H, G there are adversaries B, C of the same running time: $\boldsymbol{Adv}_{kyber,CCAKEM}^{cca}((A) \leq 4q_{RO}\sqrt{\boldsymbol{Adv}_{k+1,k,\eta}^{mlwe}(B)} + \boldsymbol{Adv}_{PRF}^{prf}(C) + 8\delta {q_{RO}}^2$

Kyber Parameters

Alg	Failure rate	Alg	Failure rate	Alg	Failure rate
KEM-512	2^{-139}	KEM-768	2^{-164}	KEM-1024	2^{-174}

Failure rates

Attacks

The Blum-Kalai-Wasserman (BKW) algorithm is a combinatorial algorithm used to solve the Learning With Errors (LWE).

The attack typically involves two main phases:

Reduction Phase: This phase progressively reduces the dimension of the LWE/LWR problem, essentially trying to simplify the equations involved. This is achieved by combining samples (vectors with associated 'noise' or errors) in a way that eliminates certain positions in the vectors, albeit at the cost of increasing the noise in the remaining positions.

Solving Phase: Once the problem is reduced to a manageable size, the remaining entries of the secret are recovered. This often involves techniques like hypothesis testing to distinguish the correct guess of the secret sub-vector from incorrect ones.

End

LLL Theorem

- Let L be the n-dimensional lattice generated by <v_1, …, v_n> and l the length of the shortest vector in L. The LLL algorithm produces a reduced basis <b_1, …, b_n> of L.

    1. $||b_1|| \leqq 2^{(n-1)/4} D^{1/n}$.
    2. $||b_1|| \leqq 2^{(n-1)/2} l$.
    3. $||b_1||\ ||b_2|| \ldots ||b_n|| \leqq 2^{n(n-1)/4} D$.

- If $||b_i||^2 \leq C$ algorithm takes $O(n^4 \lg(C))$.

Gauss again

- Let $<v_1, v_2>$ be a basis for a two-dimensional lattice L in R^2. The following algorithm produces a reduced basis.

```
for(;;) {
    if(||v1||>||v2||)
        swap v1 and v2;
    t= [(v1, v2)/(v1, v1)]; // [] is the "closest
      integer" function
    if(t==0)
        return;
    v2 = v2-tv1;
    }
```

- $<v_1, v_2>$ is now a reduced basis and v_1 is a shortest vector in the lattice.