RIKKYO UNIVERSITY

九州大学IMI研究集会
「耐量子計算機暗号と量子情報の数理」
（ハイブリッド開催）

格子基底簡約と LWE/NTRU問題に対する格子攻撃

2022年8月3日（水）
9:30～10:30

安田雅哉（立教大学）

Basics on Lattices

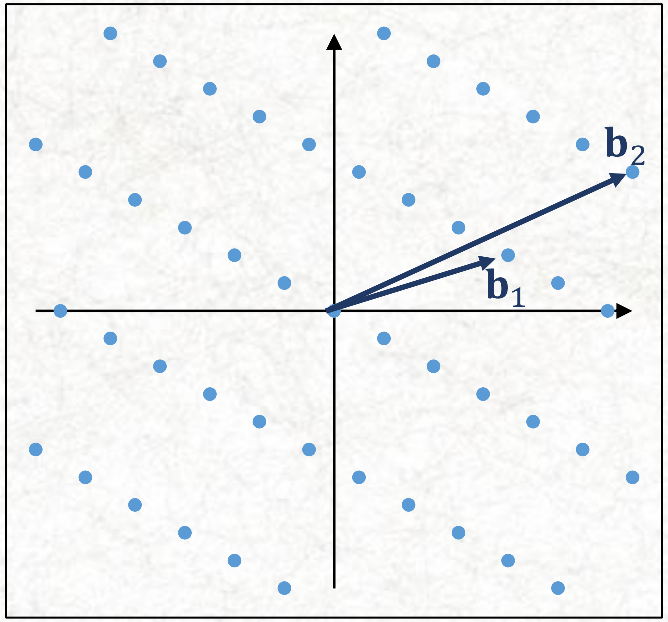
- For linearly independent $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{Z}^n$,

Integral combination

$$L = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) := \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \right\}$$

is a (full-rank) **lattice** of dimension n

- $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$: a **basis** of L
  - Regard it as the $n \times n$ matrix
- Infinitely many bases if $n \geq 2$
  - If $\mathbf{B}_1$ and $\mathbf{B}_2$ span the same lattice,
  - then $\exists \mathbf{V} \in \mathrm{GL}_n(\mathbb{Z})$ such that $\mathbf{B}_1 = \mathbf{B}_2 \mathbf{V}$
- $\mathrm{vol}(L) = |\det(\mathbf{B})|$: the **volume** of L
  - Independent of the choice of bases
- $\lambda_1(L)$: the **first successive minimum** of L
  - The length of a shortest non-zero vector in L



A lattice of dimension $n = 2$

Lattices in Cryptography

RIKKYO UNIVERSITY

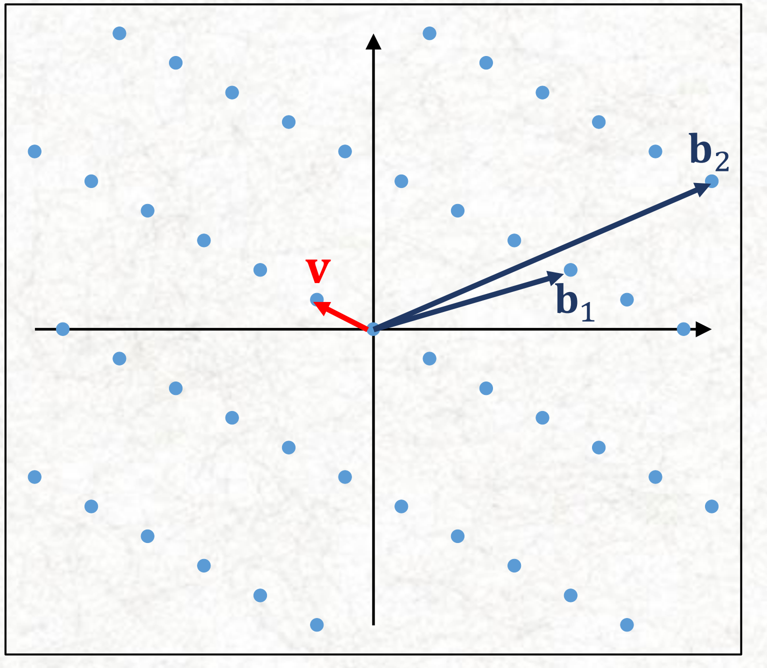
- **Post-Quantum Cryptography (PQC) Standardization**
  - Since 2015, National Institute of Standards and Technology (NIST) has proceeded a standardization project for PQC
  - In July 2020, NIST selected **7 Finalists** and **8 Alternates**
    - **7 lattice-based schemes** had been evaluated at the 3rd round
      - **5 Finalists (Kyber, NTRU, SABER, Dilithium, Falcon)**
      - **2 Alternates (FrodoKEM, NTRUprime)**
  - **In July 2022, NIST has selected the first algorithms to be standardized**
    - NISTIR 8413: https://csrc.nist.gov/publications/detail/nistir/8413/final

	Finalists	Alternates
KEMs/Encryption	$\boxed{\text{Kyber}}$ NTRU SABER Classic McEliece	~~Bike~~ FrodoKEM HQC NTRUprime SIKE
Signatures	$\boxed{\text{Dilithium}}$ $\boxed{\text{Falcon}}$ Rainbow	~~GeMSS~~ ~~Picnic~~ $\boxed{\text{SPHINCS+}}$

Lattice Problems

RIKKYO UNIVERSITY

- **Algorithmic problems for lattices**
  - **SVP** (Shortest Vector Problem)
    - Given a basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L
    - Find a non-zero shortest vector in L
  - **CVP** (Closest Vector Problem)
  - **LWE** (Learning with Errors)
  - **NTRU**, etc.
- **Relationship with cryptography**
  - The security of lattice-based cryptography is based on the hardness of lattice problems
  - Most lattice problems can be reduced to (approximate) SVP and CVP



<u>**SVP in a two-dimensional lattice**</u>

- Given linearly independent $\mathbf{b}_1, \mathbf{b}_2$
- Find a non-zero shortest vector

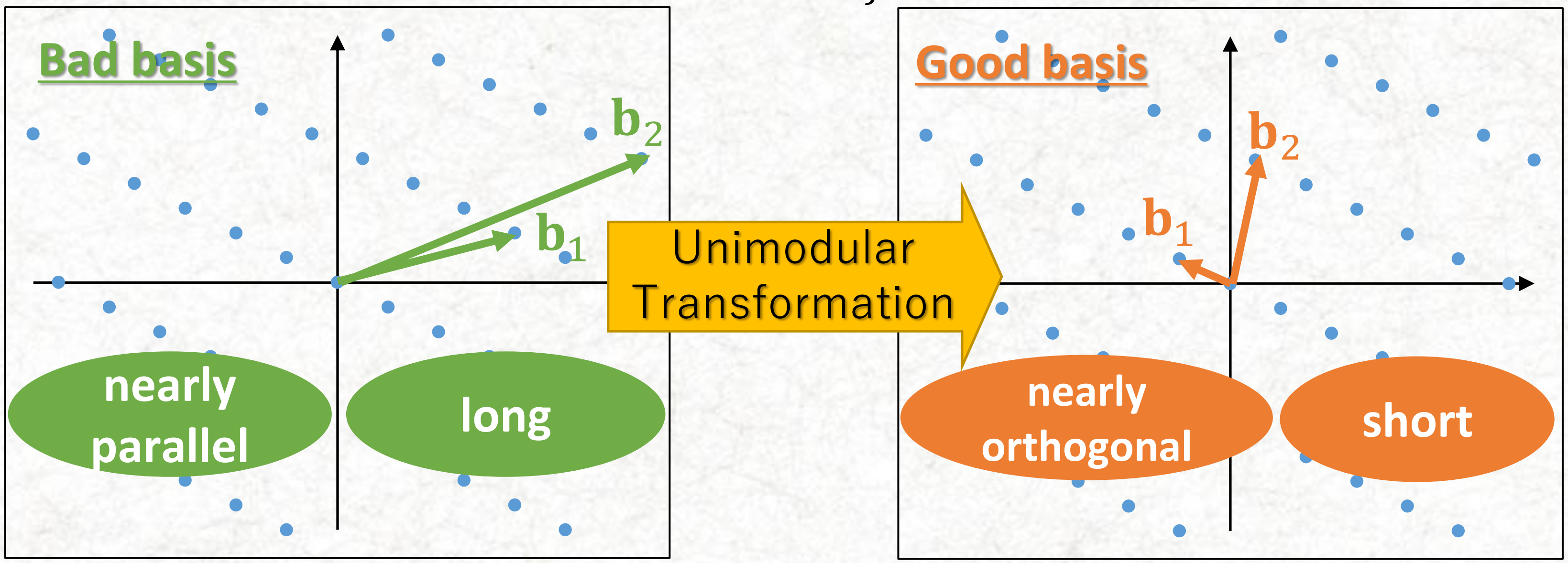
  $\mathbf{v} = a_1\mathbf{b}_1 + a_2\mathbf{b}_2$ for some $a_1, a_2 \in \mathbb{Z}$

Lattice Basis Reduction

- **Strong tool for solving lattice problems**
  - Find a basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ with short and nearly orthogonal vectors
    - Such a basis is called "**good**" or "**reduced**"
    - Some basis vectors $\mathbf{b}_i$'s are very short
  - Consist of basic unimodular transformations
    - ① Multiply by (-1): $\mathbf{b}_i \leftarrow -\mathbf{b}_i$
    - ② Swap $\mathbf{b}_i$ and $\mathbf{b}_j$
    - ③ Multiply (by integer)-Add: $\mathbf{b}_i \leftarrow \mathbf{b}_i$+a$\mathbf{b}_j$ $(a \in \mathbb{Z})$

$$\begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_n \end{pmatrix} \xrightarrow{\text{Basic unimodular transformations}} \cdots \rightarrow \cdots \rightarrow \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_n \end{pmatrix}$$

Input basis (Bad basis) — Output basis (Good basis)



LLL (1/3): Definition and Properties

RIKKYO UNIVERSITY

- **Lenstra-Lenstra-Lovász (LLL)-reduction** [LLL82]
  - $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is **δ-LLL-reduced** if it satisfies two conditions
    - ① **Size-reduced**: $|\mu_{ij}| \leq \frac{1}{2}$ for all $1 \leq j < i \leq n$
    - ② **Lovász' condition**: $\|\mathbf{b}_k^*\|^2 \geq \left(\delta - \mu_{k,k-1}^2\right)\|\mathbf{b}_{k-1}^*\|^2$
      - $\frac{1}{4} < \delta < 1$: reduction parameter (e.g., $\delta = 0.99$ for practice)
      - $\mathbf{B}^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$, $\mu = (\mu_{ij})$: Gram-Schmidt information of $\mathbf{B}$:

$$\mathbf{b}_1^* = \mathbf{b}_1,\ \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij}\mathbf{b}_j^*,\ \mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\left\|\mathbf{b}_j^*\right\|^2}$$

  - Every LLL-reduced basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L satisfies
    - $\|\mathbf{b}_1\| \leq \alpha^{\frac{n-1}{2}} \lambda_1(L)$, where $\alpha = \frac{4}{4\delta - 1} > \frac{4}{3}$
    - $\|\mathbf{b}_1\| \leq \alpha^{\frac{n-1}{4}} \mathrm{vol}(L)^{\frac{1}{n}}$

[LLL82] A.K. Lenstra, H.W. Lenstra and L. Lovász, "Factoring polynomials with rational coefficients", Mathematische Annalen 261 (4): 515—534 (1982).

LLL (2/3): Basic Algorithm

RIKKYO UNIVERSITY

- **It consists of two procedures to find an LLL-reduced basis**
  - ① **Size-reduction**: $\mathbf{b}_k \leftarrow \mathbf{b}_k - q\mathbf{b}_j$ with $q = \lfloor \mu_{k,j} \rceil$
  - ② **Swap adjacent vectors**: $\mathbf{b}_{k-1} \leftrightarrow \mathbf{b}_k$ if they do not satisfy Lovász' condition

Algorithm: The basic LLL Lenstra et al. (1982)

Input: A basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L, and a reduction parameter $\frac{1}{4} < \delta < 1$
Output: A δ-LLL-reduced basis $\mathbf{B}$ of L

Compute Gram–Schmidt information $\mu_{i,j}$ and $\|\mathbf{b}_i^*\|^2$ of the input basis $\mathbf{B}$
$k \leftarrow 2$
while $k \leq n$ **do**
① Size-reduce $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ // At each k, we recursively change $\mathbf{b}_k \leftarrow \mathbf{b}_k - \lfloor \mu_{k,j} \rceil \mathbf{b}_j$ for $1 \leq j \leq k-1$ (e.g., see Galbraith 2012, Algorithm 24)
  if $(\mathbf{b}_{k-1}, \mathbf{b}_k)$ satisfies Lovász' condition **then**
    $k \leftarrow k + 1$
  else
② Swap $\mathbf{b}_k$ with $\mathbf{b}_{k-1}$, and update Gram–Schmidt information of $\mathbf{B}$
    $k \leftarrow \max(k-1, 2)$
  end if
end while

A Survey of Solving SVP Algorithms and Recent Strategies for Solving the SVP Challenge | SpringerLink

LLL (3/3): Sage Code

RIKKYO UNIVERSITY

```
def GSO(B, n):
    GS = Matrix(QQ, n)
    mu = Matrix(QQ, n)
    for i in range(n):
        GS[i] = B[i]
        mu[i, i] = 1
        for j in range(i):
            mu[i,j] = B[i].inner_product(GS[j])/GS[j].norm()^2
            GS[i] -= mu[i,j]*GS[j]
    return GS, mu

def LLL(B, n, delta):
    GS, mu = GSO(B, n)
    BB = vector(QQ, n)
    for i in range(n):
        BB[i] = GS[i].norm()^2
    k=1
    while k<=n-1:
        for j in range(k)[::-1]:
            if abs(mu[k,j])> 0.50:
                q=round(mu[k, j])
                B[k] -= q*B[j]
                for l in range(j+1):
                    mu[k,l] -= q*mu[j,l]
        if BB[k] >= (delta - mu[k, k-1]^2)*BB[k-1]:
            k+=1
        else:
            v = B[k-1]; B[k-1]=B[k]; B[k]=v;
            GS, mu=GSO(B, n)
            for i in range(n):
                BB[i] = GS[i].norm()^2
            k=max(k-1, 1)
    return true
```

```

n = 10; d = 100000
B = Matrix(ZZ, n)
for i in range(0, n):
    B[i, i] = 1
    B[i, 0] = randint(-d, d)
print("Input basis")
show(B)
LLL(B, n, 0.99)
print("¥nOutput basis")
show(B)
```

Please use
[Sage Cell Server (sagemath.org)]

Enumeration (1/3): Basic Idea

RIKKYO UNIVERSITY

- **Enumerate all vectors $\boldsymbol{s} = \sum v_i \mathbf{b}_i \in \mathcal{L}(\mathrm{B})$ such that $\|\boldsymbol{s}\| \leq R$**
  - $R > 0$: search radius (e.g., $R = 1.05\mathrm{GH}(L)$)
  - With Gram-Schmidt information, write

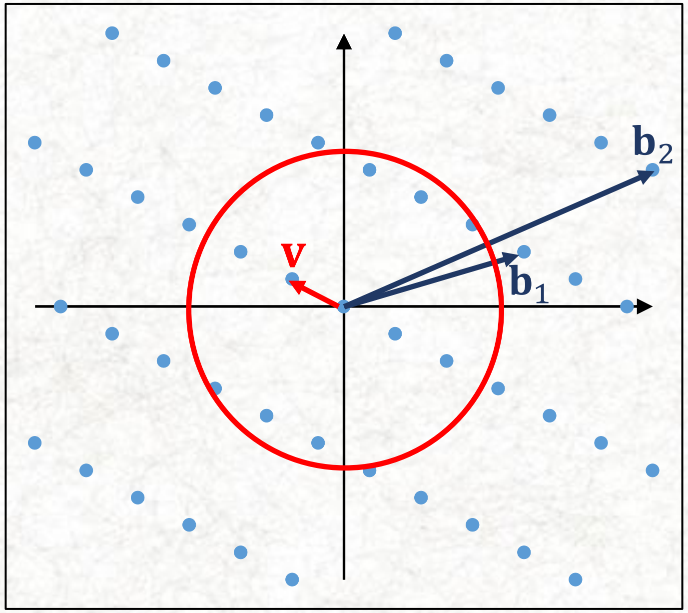
$$\mathbf{s} = \sum_{j=1}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{ij} v_i \right) \mathbf{b}_j^*$$

  - By the orthogonality of Gram-Schmidt vectors,

$$\|\pi_k(\mathbf{s})\|^2 = \sum_{j=k}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{ij} v_i \right)^2 \|\mathbf{b}_j^*\|^2$$

    for $1 \leq k \leq n$, where π_k denotes the projection map to $\langle \mathbf{b}_k^*, \dots, \mathbf{b}_n^* \rangle_{\mathbb{R}}$
  - Consider n inequalities $\|\pi_k(\mathbf{s})\|^2 \leq R^2$ for $1 \leq k \leq n$:

$$\begin{cases} v_n^2 \leq R^2 / \|\mathbf{b}_n^*\|^2 \\ \left(v_{n-1} + \mu_{n,n-1} v_n\right)^2 \leq \left(R^2 - v_n^2 \|\mathbf{b}_n^*\|^2\right) / \|\mathbf{b}_{n-1}^*\|^2 \\ \vdots \end{cases}$$



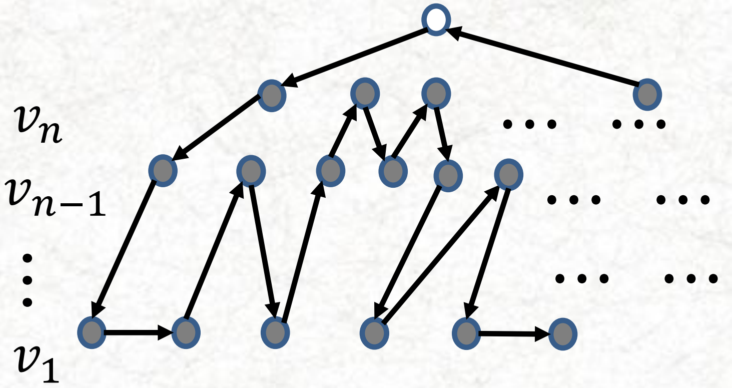
Enumeration (2/3): Basic Algorithm

Algorithm: The basic Schnorr–Euchner enumeration Schnorr and Euchner (1994)

Input: A basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L and a radius R with $\lambda_1(L) \leq R$
Output: The shortest non-zero vector $\mathbf{s} = \sum_{i=1}^{n} v_i \mathbf{b}_i$ in L

Compute Gram–Schmidt information $\mu_{i,j}$ and $\|\mathbf{b}_i^*\|^2$ of $\mathbf{B}$
$(\rho_1, \ldots, \rho_{n+1}) = \mathbf{0}$, $(v_1, \ldots, v_n) = (1, 0, \ldots, 0)$, $(c_1, \ldots, c_n) = \mathbf{0}$, $(w_1, \ldots, w_n) = \mathbf{0}$
$k = 1$, $\texttt{last\_nonzero} = 1$ // largest i for which $v_i \neq 0$
while $\texttt{true}$ **do**
  $\rho_k \leftarrow \rho_{k+1} + (v_k - c_k)^2 \cdot \|\mathbf{b}_k^*\|^2$ // $\rho_k = \|\pi_k(\mathbf{s})\|^2$
  if $\rho_k \leq R^2$ **then**
    if $k = 1$ **then** $R^2 \leftarrow \rho_k$, $\mathbf{s} \leftarrow \sum_{i=1}^{n} v_i \mathbf{b}_i$; // update the squared radius
    else $k \leftarrow k - 1$, $c_k \leftarrow -\sum_{i=k+1}^{n} \mu_{i,k} v_i$, $v_k \leftarrow \lfloor c_k \rceil$, $w_k \leftarrow 1$;
  else
    $k \leftarrow k + 1$ // going up the tree
    if $k = n + 1$ **then return** $\mathbf{s}$;
    if $k \geq \texttt{last\_nonzero}$ **then** $\texttt{last\_nonzero} \leftarrow k$, $v_k \leftarrow v_k + 1$;
    else
      if $v_k > c_k$ **then** $v_k \leftarrow v_k - w_k$; **else** $v_k \leftarrow v_k + w_k$; // zig-zag search
      $w_k \leftarrow w_k + 1$
    end if
  end if
end while

- Enumerate lattice vectors
  $$\mathbf{s} = \sum v_i \mathbf{b}_i \in L$$
  such that $\|\mathbf{s}\| \leq R$
- Built an enumeration tree to find integral combinations $(v_1, \ldots, v_n)$



[A Survey of Solving SVP Algorithms and Recent Strategies for Solving the SVP Challenge | SpringerLink]

Enumeration (3/3): Sage Code

RIKKYO UNIVERSITY

```
def GSO(B, n):
    GS = Matrix(QQ, n)
    mu = Matrix(QQ, n)
    for i in range(n):
        GS[i] = B[i]
        mu[i, i] = 1
        for j in range(i):
            mu[i,j] = B[i].inner_product(GS[j])/GS[j].norm()^2
            GS[i] -= mu[i,j]*GS[j]
    return GS, mu

def ENUM(B, n, R):
    GS, mu = GSO(B, n)
    BB = vector(QQ, n)
    for i in range(n):
        BB[i] = GS[i].norm()^2
    sigma = Matrix(QQ, n+1, n)
    r = vector(ZZ, n+1)
    rho = vector(QQ, n+1)
    v = vector(ZZ, n)
    c = vector(QQ, n)
    w = vector(ZZ, n)
    for i in range(n+1):
        r[i] = i
    v[0] = 1
    last_nonzero = 1
    k = 1
    while (1):
        rho[k-1] = rho[k] + (v[k-1] - c[k-1])^2*BB[k-1]
        if RR(rho[k-1]) <= RR(R):
            if k==1:
                print("Solution found"); return v
            k = k-1
            r[k-1] = max(r[k-1], r[k])
            for i in range(k+1, r[k]+1)[::-1]:
                sigma[i-1, k-1] = sigma[i, k-1] + mu[i-1, k-1]*v[i-1]
            c[k-1] = -sigma[k, k-1]
            v[k-1] = round(c[k-1])
            w[k-1] = 1
        else:
            k = k+1
            if k==n+1:
                print("No solution"); return false
            r[k-1] = k
            if k>=last_nonzero:
                last_nonzero = k
                v[k-1] = v[k-1] + 1
            else:
                if RR(v[k-1]) > RR(c[k-1]):
                    v[k-1] = v[k-1] - w[k-1]
                else:
                    v[k-1] = v[k-1] + w[k-1]
                w[k-1] = w[k-1] + 1
```

```
#Main
n = 20
B = random_matrix(ZZ, n, x=0, y = 30)
B.LLL()
print("LLL-reduced basis =¥n", B)
R = 0.99*RR(B[0].norm()^2)
while (1):
    v = vector(ZZ, n)
    v = ENUM(B, n, R)
    if v != false:
        vec = v[0]*B[0]
        for i in range(1, n):
            vec += v[i]*B[i]
        R = 0.99*RR(vec.norm()^2)
        print("Norm=", RR(vec.norm()), ", Vector=", vec)
    else:
        break
print("End")
```

BKZ (1/3): Definition and Properties

RIKKYO UNIVERSITY

- **Block Korkine-Zolotarev (BKZ)-reduction**
  - A blockwise generalization of LLL with blocksize β
  - $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is **β-BKZ-reduced** if it satisfies two conditions
    - ① It is size-reduced (same as LLL)
    - ② The k-th Gram-Schmidt vector $\mathbf{b}_k^*$ is shortest in $L_{[k,\,\ell]}$ with $\ell = \min(k + \beta - 1,\ n)$ for all $1 \leq k < n$

$$
\begin{array}{lcccccccc}
L_{[1,\,\beta]}: & \mathbf{b}_1 & \cdots & \cdots & & \mathbf{b}_\beta & & & \\
L_{[2,\,\beta+1]}: & & \pi_2(\mathbf{b}_2) & \cdots & & \cdots & \pi_2(\mathbf{b}_{\beta+1}) & & \\
\vdots & & & \ddots & & & & \ddots & \\
L_{[n-\beta+1,\,n]}: & & & & \pi_{n-\beta+1}(\mathbf{b}_{n-\beta+1}) & & \cdots & \cdots & \pi_{n-\beta+1}(\mathbf{b}_n)
\end{array}
$$

  - Every β-BKZ-reduced basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L satisfies

$$\|\mathbf{b}_1\| \leq \gamma_\beta^{\frac{n-1}{\beta-1}} \lambda_1(L)$$

    - γ_β: Hermite's constant of dimension β, i.e., $\gamma_\beta = \sup_L \lambda_1(L)^2 / \mathrm{vol}(L)^{2/n}$
    - As β increases, $\gamma_\beta^{1/(\beta-1)}$ decreases and thus $\mathbf{b}_1$ can be shorter

BKZ (2/3): Basic Algorithm

RIKKYO UNIVERSITY

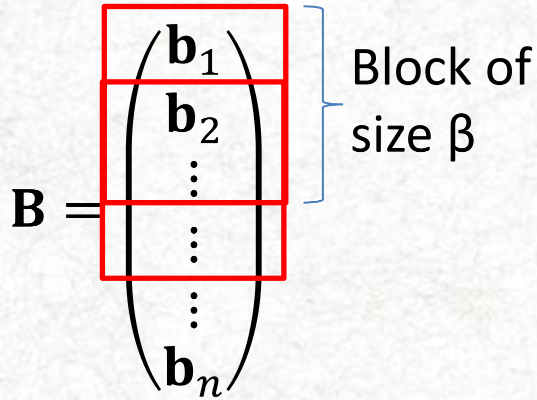
- **It consists of LLL and ENUM:**
  - Call ENUM to find a non-zero shortest vector in $L_{[k,\ell]}$
  - Call LLL to reduce a projected block basis of $L_{[k,\ell]}$

Algorithm: The basic BKZ Schnorr and Euchner (1994)

Input: A basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L, a blocksize $2 \le \beta \le n$, and a reduction parameter $\frac{1}{4} < \delta < 1$ of LLL

Output: A β-DeepBKZ-reduced basis $\mathbf{B}$ of L

```
B ← LLL(B, δ)  // Compute μ_{i,j} and ‖b*_j‖² of the new basis B together
z ← 0, j ← 0
while z < n − 1 do
  j ← (j mod (n − 1)) + 1, k ← min(j + β − 1, n), h ← min(k + 1, n)
  Find v ∈ L such that ‖π_j(v)‖ = λ_1(L_[j,k]) by enumeration or sieve
  if ‖π_j(v)‖² < ‖b*_j‖² then
    z ← 0 and call LLL((b_1, ..., b_{j−1}, v, b_j, ..., b_h), δ)  // Insert v ∈ L and
    remove the linear dependency to obtain a new basis
  else
    z ← z + 1 and call LLL((b_1, ..., b_h), δ)
  end if
end while
```



※ As reference, please look at BKZ-60 – YouTube by Martin Albrecht

A Survey of Solving SVP Algorithms and Recent Strategies for Solving the SVP Challenge | SpringerLink

BKZ (3/3): Sage Code

RIKKYO UNIVERSITY

```
def ENUM(B, n, R, g, h):
    BB, U = GSO(B, n)
    Bnn = vector(QQ, n)
    for i in range(n):
        Bnn[i] = BB[i].norm()^2
    BB, U = GSO(B, n)
    sigma = Matrix(QQ, n+1, n)
    r = vector(ZZ, n+1)
    rho = vector(QQ, n+1)
    v = vector(ZZ, n)
    c = vector(QQ, n)
    w = vector(ZZ, n)
    for i in range(n+1):
        r[i] = i
    v[g] = 1
    last_nonzero = 1
    k = g + 1
    flag = 0
    v1 = vector(ZZ, n)
    while (1):
        rho[k-1] = rho[k] + (v[k-1] - c[k-1])^2*Bnn[k-1]
        if rho[k-1] <= R:
            if k==g+1:
                R = 0.99*rho[k-1]
                flag += 1
                for i in range(n):
                    v1[i] = v[i]
            k = k-1
            r[k-1] = max(r[k-1], r[k])
            for i in range(k+1, r[k]+1)[::-1]:
                sigma[i-1, k-1] = sigma[i, k-1] + U[i-1, k-1]*v[i-1]
            c[k-1] = -sigma[k, k-1]
            v[k-1] = round(c[k-1])
            w[k-1] = 1
        else:
            k = k+1
            if k==h+1:
                if flag == 0:
                    return False
                else:
                    vv = v1[g]*B[g]
                    for i in range(g+1, h+1):
                        vv += v1[i]*B[i]
                    return vv
            r[k-1] = k
            if k>=last_nonzero:
                last_nonzero = k
                v[k-1] = v[k-1] + 1
            else:
                if v[k-1] > c[k-1]:
                    v[k-1] = v[k-1] - w[k-1]
                else:
                    v[k-1] = v[k-1] + w[k-1]
                w[k-1] = w[k-1] + 1
```

```
def BKZ(B, n, block):
    B.LLL()
    BB, U = GSO(B, n)
    Bnn = vector(QQ, n)
    for i in range(n):
        Bnn[i] = BB[i].norm()^2
    z = 0
    k = -1
    while z < n-1:
        k = lift(mod(k+1, n-2))
        l = min(k+block-1, n-1)
        h = min(l+1, n-1)
        print("(k, l, h) = ", k, l, h)

        R = 0.99*Bnn[k]
        v = 0
        v = ENUM(B, n, R, k, l)
        if v != 0:
            z = 0
            C = Matrix(ZZ, h+1, n)
            for i in range(k):
                C[i] = B[i]
            C[k] = v
            for i in range(k+1, h+1):
                C[i] = B[i-1]
            C = C.LLL()
            for i in range(1, h+1):
                B[i-1] = C[i]
            BB, U = GSO(B, n)
            Bnn = vector(QQ, n)
            for i in range(n):
                Bnn[i] = BB[i].norm()^2
        else:
            z += 1
            B = B.LLL()
            BB, U = GSO(B, n)
            Bnn = vector(QQ, n)
            for i in range(n):
                Bnn[i] = BB[i].norm()^2

n = 20; d = 1000000
B = Matrix(ZZ, n)
for i in range(0, n):
    B[i, i] = 1
    B[i, 0] = randint(-d, d)
show(B)
B = B.LLL()
BKZ(B, n, 10)
show(B)
```

Log-Lengths of Gram-Schmidt Vectors of Reduced Bases

RIKKYO UNIVERSITY

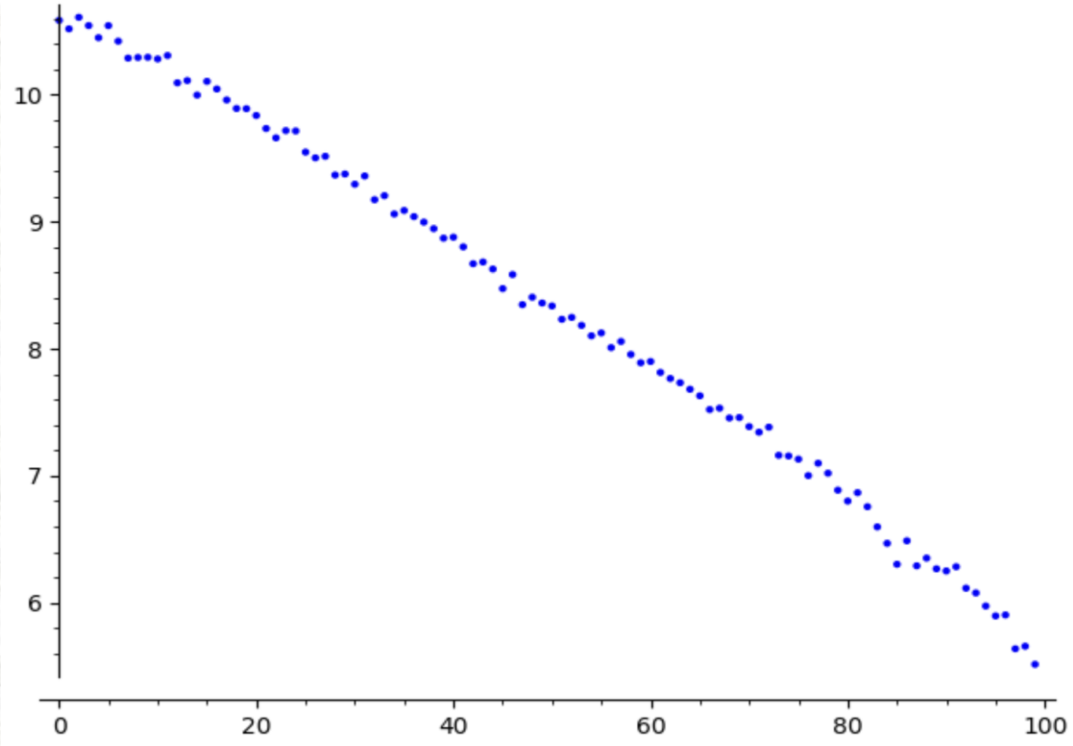
```
from sage.modules.free_module_integer import IntegerLattice
from fpylll import *

def MGSO(B, n):
    a = Matrix(RR, n); qq = Matrix(RR, n)
    r = Matrix(RR, n); mu = Matrix(RR, n)
    BB = vector(RR, n)
    for k in range(n):
        qq[k] = B[k]
        for j in range(k):
            r[j, k] = qq[j].inner_product(qq[k])
            qq[k] -= r[j, k]*qq[j]
        r[k, k] = qq[k].norm()
        qq[k] /= r[k, k]
    for i in range(n):
        mu[i, i] = 1.0
        BB[i] = r[i, i]**2
        for j in range(i):
            mu[i, j] = r[j, i]/r[j, j]
    a = r.transpose()
    r = a
    return BB, mu

d = 100
BB = 2**(6*d)
L = sage.crypto.gen_lattice(type='random', n=1, m=d, q=BB, lattice=True)
A = L.LLL()

B = IntegerMatrix(d, d)
for i in range(d):
    for j in range(d):
        B[i, j] = A[i, j]
par = BKZ.Param(50, strategies=BKZ.DEFAULT_STRATEGY, max_loops = 2)
B = BKZ.reduction(B, par)
C = Matrix(ZZ, d, d)
for i in range(d):
    for j in range(d):
        C[i, j] = B[i, j]
BB, mu = MGSO(C, d)
list = []
for i in range(d):
    list.append(RR(log(BB[i])))
show(list_plot(list))
```

- **Geometric Series Assumption (GSA)**
  - Log-lengths $\log\|\mathbf{b}_i^*\|^2$ of Gram-Schmidt vectors of a reduced basis $(\mathbf{b}_1, \dots, \mathbf{b}_n)$ for a "random" lattice are roughly on a straight line



The LWE Problem and Its Reduction (1/2)

RIKKYO UNIVERSITY

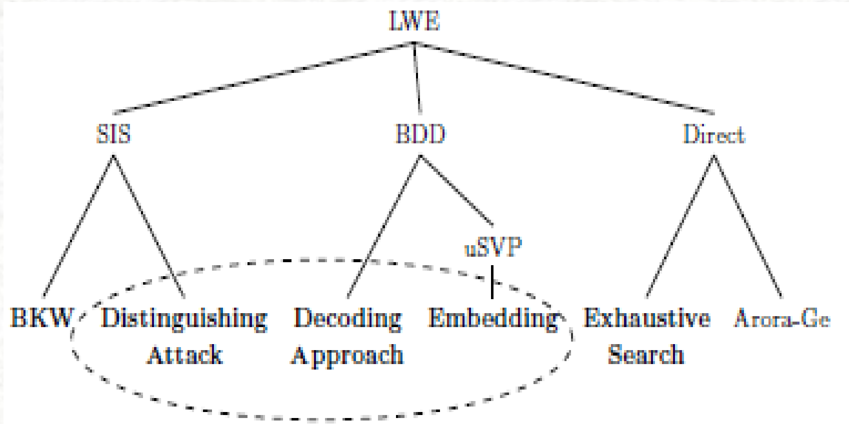
- **Search-LWE problem with (n, q, σ, m)**
  - A kind of solving a system of linear **approximate** equations
  - Given $(\mathbf{A}, \mathbf{b})$ with $\mathbf{b} \equiv \mathbf{A}^T \mathbf{s} + \mathbf{e} \bmod q$, find $\mathbf{s}$
    - $\mathbf{A} = (a_{ij}), \mathbf{s} = (s_i)$: uniform over $\mathbb{Z}_q$
    - $\mathbf{e} = (e_i)$: Gaussian distributed with σ (small error vector)

$$\boxed{\begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \equiv \begin{pmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{pmatrix} \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} e_1 \\ \vdots \\ e_m \end{pmatrix} \bmod q}$$

$$\begin{cases} 14s_1 + 15s_2 + 5s_3 + 2s_4 \approx 8 & \pmod{17} \\ 13s_1 + 14s_2 + 14s_3 + 6s_4 \approx 16 & \pmod{17} \\ 6s_1 + 10s_2 + 13s_3 + s_4 \approx 12 & \pmod{17} \\ 10s_1 + 4s_2 + 12s_3 + 16s_4 \approx 12 & \pmod{17} \\ 9s_1 + 5s_2 + 9s_3 + 6s_4 \approx 9 & \pmod{17} \\ 3s_1 + 6s_2 + 4s_3 + 5s_4 \approx 16 & \pmod{17} \\ \vdots & \\ 6s_1 + 7s_2 + 16s_3 + 2s_4 \approx 3 & \pmod{17} \end{cases}$$

$\mathbf{s} = (0, 13, 9, 11) \in \mathbb{F}_{17}$

- **Approaches for solving LWE[BBG+17]**
  - We shall describe reduction of LWE to BDD in the next slide



[BBG+17] N. Bindel, J. Buchmann, F. Gopfert and M. Schmidt, “Estimation of the hardness of the learning with errors problem with a restricted number of samples," IACR ePrint 2017/140, available at https://eprint.iacr.org/2017/140.

The LWE Problem and Its Reduction (2/2)

RIKKYO UNIVERSITY

Search-LWE

$\mathbf{b} \equiv \mathbf{A}^T\mathbf{s} + \mathbf{e} \bmod q$

- $(\mathbf{A}, \mathbf{b})$: public
- $(\mathbf{s}, \mathbf{e})$: secret

- **Reduction to BDD**
  - BDD = Bounded Distance Decoding
    - A particular case of CVP
  - Find a vector $\mathbf{A}^T\mathbf{s} \in \Lambda$ close to the target $\mathbf{b}$
    - $\Lambda = \{\mathbf{y} \in \mathbb{Z}^d \,:\, \exists \mathbf{s} \in \mathbb{Z}^n$ s.t $\mathbf{y} \equiv \mathbf{A}^T\mathbf{s}$ (mod q)$\}$: **q-ary lattice** of dimension d
    - Distance $\|\mathbf{b} - \mathbf{A}^T\mathbf{s}\| = \|\mathbf{e}\|$ is guaranteed to be small (e.g., $\|\mathbf{e}\| < 3\sigma\sqrt{d}$)
- **Transformation of BDD to (unique-)SVP**
  - E.g., Kannan's embedding technique[Kan87]

    ① From a basis $\mathbf{B}$ of Λ, generate a matrix $\overline{\mathbf{B}} = \begin{pmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{b} & 1 \end{pmatrix}$ to define a lattice $\bar{L} = \mathcal{L}(\overline{\mathbf{B}})$, spanned by rows of $\overline{\mathbf{B}}$

    ② Find a short vector $\mathbf{v} = (\mathbf{e}, 1) \in \bar{L}$
    - If d is large enough (e.g., $d > 2n$), then $\mathbf{v}$ is the shortest in $\bar{L}$
    - It is extremely short for most LWE instances

[Kan87] R. Kannan. Minkowski's convex body theorem and integer programming. Mathmatics of operations research,12(3):415-440,1987.

Solving the LWE problem
Sage Code

RIKKYO UNIVERSITY

```
from sage.crypto.lwe import LWE
from sage.stats.distributions.discrete_gaussian_integer import DiscreteGaussianDistributionIntegerSampler

n = 30; q = next_prime(500)
D = DiscreteGaussianDistributionIntegerSampler(2.0)
lwe = LWE(n, q, D = D)
print(lwe)

d = 80
A = Matrix(ZZ, d, n); b = vector(ZZ, d)
for i in range(d):
    sample = lwe()
    for j in range(n):
        A[i, j] = (sample[0])[j]
    b[i] = sample[1]

C = Matrix(ZZ, n+d, d)
AT = A.transpose()
for i in range(n):
    for j in range(d):
        C[i, j] = AT[i, j]
for i in range(d):
    C[i+n, i] = q
C = C.LLL()

BB = Matrix(ZZ, d+1, d+1)
for i in range(d):
    for j in range(d):
        BB[i, j] = C[i+n, j]
for j in range(d):
    BB[d, j] = b[j]
BB[d, d] = 1
print(); print(BB)

BB = BB.LLL()
print(); print(BB[0])
```

Search-LWE

$$\boxed{\mathbf{b} \equiv \mathbf{A}^T\mathbf{s} + \mathbf{e} \bmod q}$$

- $(\mathbf{A}, \mathbf{b})$: public
- $(\mathbf{s}, \mathbf{e})$: secret

$$\boxed{\bar{\mathbf{B}} = \begin{pmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{b} & 1 \end{pmatrix}}$$

Applying LLL/BKZ

$$\boxed{\mathbf{v} = (\mathbf{e}, 1) \in \mathcal{L}(\bar{\mathbf{B}})}$$

Extension of Embedding for Ring-Based LWE (1/5)

RIKKYO UNIVERSITY

- **Ring-based LWE[CIV16]**
  - A general framework containing Ring-LWE and Poly-LWE
    - Given ring-based samples $(a_i(x), t_i(x))$ over $R_q = \mathbb{Z}_q[x]/(x^n + 1)$
    - Find a secret $s(x) \in R_q$ (or equivalently, small errors $e_i(x)$)

Given samples $(a_i(x), t_i(x))$ with

$$\begin{cases} t_1(x) = s(x)a_1(x) + e_1(x), \\ \vdots \\ t_m(x) = s(x)a_m(x) + e_m(x), \end{cases}$$

$\Rightarrow$ Find a secret $s(x)$ (or errors $e_i(x)$)

- **Coefficient representation and rotations**
  - Coefficient representation: $f(x) = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1} \mapsto \mathbf{f} = (f_0, f_1, \dots, f_{n-1})$
    - This representation can reduce ring-based LWE to standard LWE
  - **Rotation**: $\mathrm{rot}(\boldsymbol{f}) \coloneqq (-f_{n-1}, f_0, f_1, \dots, f_{n-2})$
    - It is the coefficient vector of $xf(x)$ for any $f(x) \in R$ since $x^n = -1$

[CIV16] W. Castryck, I. Iliashenko, and F. Vercauteren, On error distributions in ring-based LWE, LMS Journal of Computation and Mathematics19(A), 130–145 (2016)

Extension of embedding for Ring-Based LWE (2/5)

RIKKYO UNIVERSITY

- **Extended Kannan's embedding[NY21]**
  - Add rotated targets $\mathrm{rot}^{i-1}(\tilde{\mathbf{t}})$ for $1 \le i \le k$ to Kannan's lattice
    - The case k=1 is the same as original Kannan's embedding
  - It includes ***k* short lattice vectors** with norm $\sqrt{\|\tilde{\mathbf{e}}\|^2 + \eta^2}$
    - Remark that $\mathrm{rot}^i(\tilde{\mathbf{e}}) \equiv \mathrm{rot}^i(\tilde{\mathbf{t}}) - \mathrm{rot}^i(\tilde{\mathbf{s}})\tilde{\mathbf{A}}$ for $1 \le i \le k$
    - However, the dimension increases: $\dim L_k = d + k$

      ($L_k = \mathcal{L}(\mathbf{B})$: the extended lattice)

$$
\mathbf{B} = \left(\begin{array}{c|cccc}
\mathbf{C} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\
\hline
\tilde{\mathbf{t}} & \eta & 0 & \cdots & 0 \\
\mathrm{rot}(\tilde{\mathbf{t}}) & 0 & \eta & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\mathrm{rot}^{k-1}(\tilde{\mathbf{t}}) & 0 & 0 & \cdots & \eta
\end{array}\right)
$$

Add k rotated targets

$$
\begin{cases}
\bar{\mathbf{e}} = (\tilde{\mathbf{e}} \mid \eta, 0, \ldots, 0), \\
\mathrm{rot}(\bar{\mathbf{e}}) = (\mathrm{rot}(\tilde{\mathbf{e}}) \mid 0, \eta, \ldots, 0), \\
\qquad \vdots \\
\mathrm{rot}^{k-1}(\bar{\mathbf{e}}) = \left(\mathrm{rot}^{k-1}(\tilde{\mathbf{e}}) \mid 0, \ldots, 0, \eta\right).
\end{cases}
$$

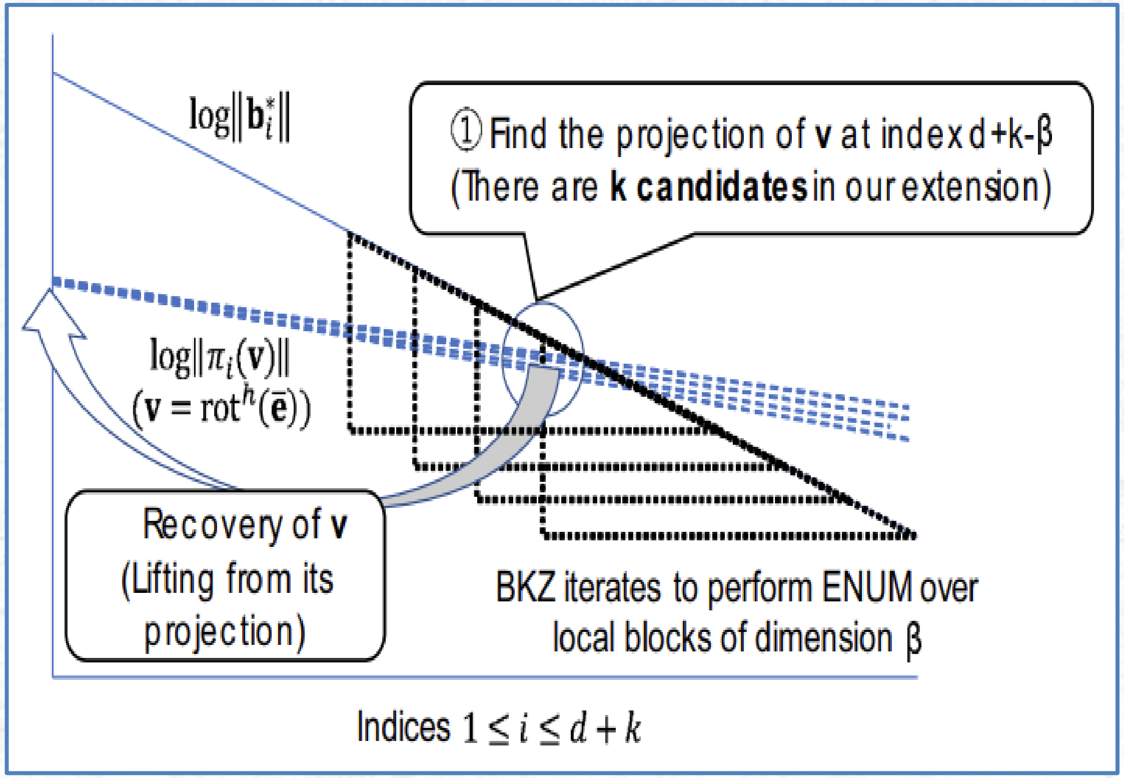
k short vectors in $L_k = \mathcal{L}(\mathbf{B})$
(with the same norm)

[NY21] S. Nakamura and M. Yasuda, "An extension of Kannan's embedding for solving ring-based LWE problems," IMA Cryptography and Coding (IMACC2021)

Extension of embedding for Ring-Based LWE (3/5)

RIKKYO UNIVERSITY

- **Recovering rotated targets $\mathbf{v} = \mathbf{rot}^h(\bar{\mathbf{e}}) \in \boldsymbol{L_k}$ by BKZ**
  ① Find its projection $\pi_i(\mathbf{v})$ by enumeration over the projected lattice $\mathcal{L}(\mathbf{B}_{[i:d+k]})$ in the procedure of BKZ
  ② Lift to the whole vector $\mathbf{v}$ by enumeration over other projected lattices

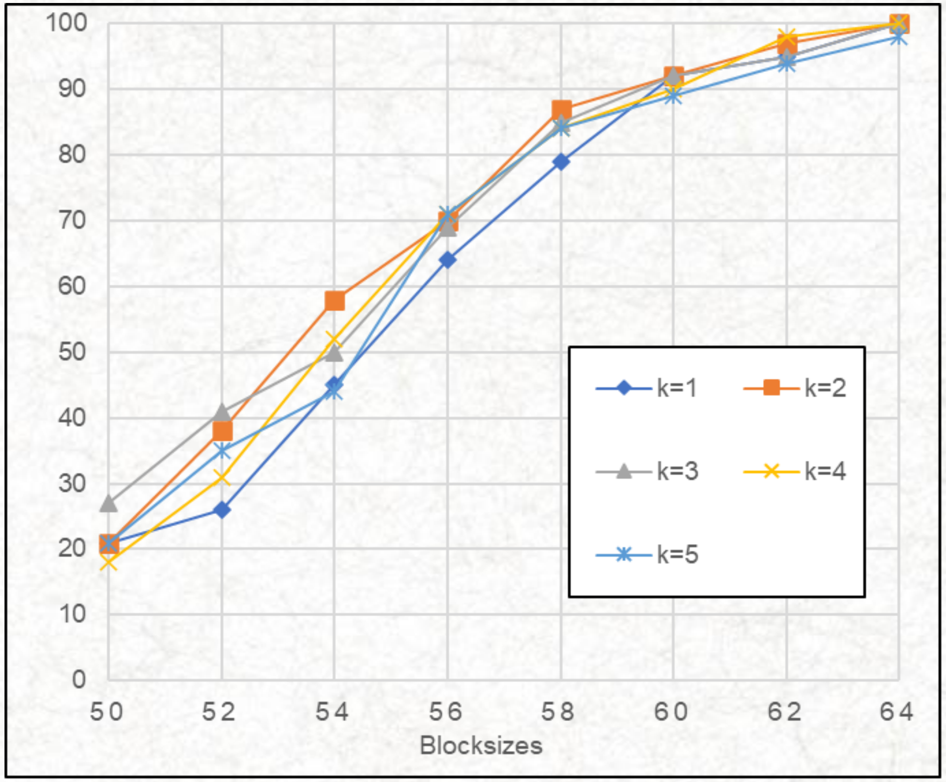


- **Trade-offs**
  - It could **increase the probability to recover rotated targets**
    - Since there are k short targets
  - It could also **increase the running time of BKZ**
    - Since the dimension increases

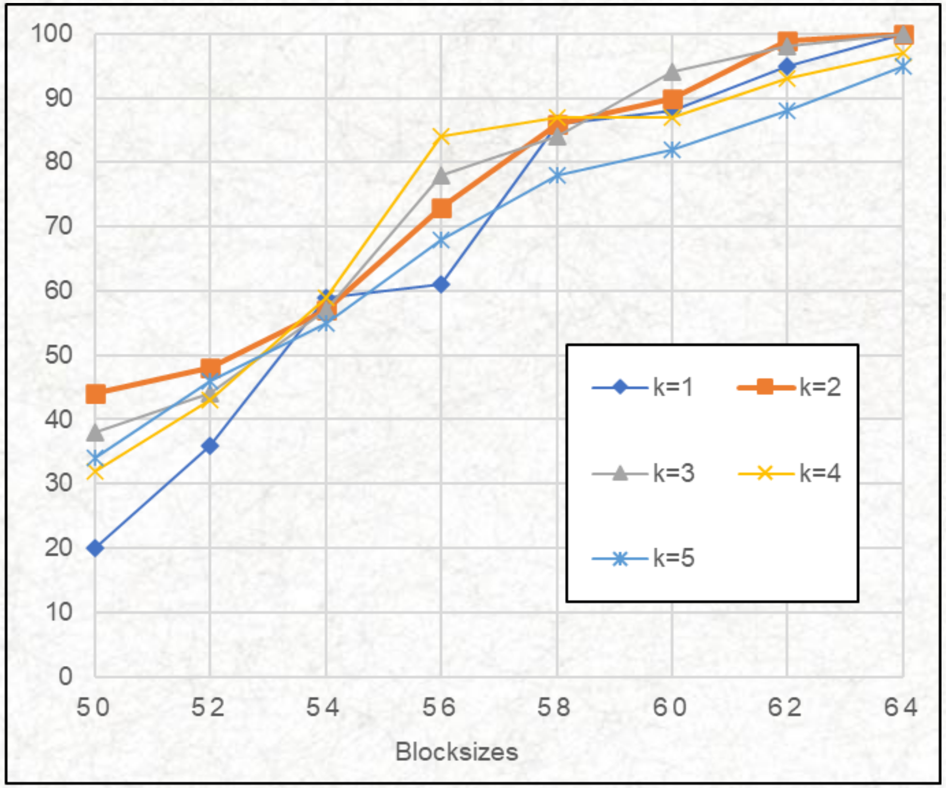
Extension of embedding for Ring-Based LWE (4/5)

RIKKYO UNIVERSITY

- **Experimental results**
  - Transition of success probabilities by blocksizes of BKZ
  - **k=2 or 3** gives the highest success probability for most β
    - Cf., the running time of BKZ increases slightly for k = 2 and 3



(a) n=32, q = 257, σ=6.0, d = 96
η=6.0, loop_max = 2



(b) n=64, q = 257, σ=1.7, d = 128
η=2.0, loop_max = 4

Extension of embedding for Ring-Based LWE (5/5)

RIKKYO UNIVERSITY

```
from sage.crypto.lwe import RingLWE
from sage.crypto.lwe import DiscreteGaussianDistributionPolynomialSampler, RingLWE, RingLWEConverter
from sage.stats.distributions.discrete_gaussian_polynomial import DiscreteGaussianDistributionPolynomialSampler
from fpylll import *

# Rotation
def rot(v, l):
    w = copy(v)
    for i in range(1, l):
        w[i] = v[i-1]
    w[0] = -v[l-1]
    return w
#=======================
# Setting of parameters
#=======================
n = 64; N = 2*n         # security parameter
q = 1153                # modulus parameter
sigma = 4.0             # standard deviation of the discrete Gaussian distribution
m = 2                   # number of ring-LWE samples
d = m*n                 # number of LWE samples
k = 5                   # extension parameter for Kannan's embedding
# t = 1
# t = round(sigma)
t = 2*round(sigma)

success = 0
for s in range(100):
    #===============================
    # Generation of ring-LWE samples
    #===============================
    D = DiscreteGaussianDistributionPolynomialSampler(ZZ['x'], euler_phi(N), sigma)
    ringlwe = RingLWE(N, q, D, secret_dist='uniform')
    a = Matrix(m, n)
    b = Matrix(m, n)
    for i in range(m):
        Sample = ringlwe()
        a[i] = copy(Sample[0])
        b[i] = copy(Sample[1])

    #================================
    # Contruction of a q-ary lattice
    #================================
    A = Matrix(n, d)
    for i in range(m):
        v = copy(a[i])
        for j in range(n):
            for l in range(n):
                A[j, n*i + l] = v[l]
            v = rot(v, n)

    C = Matrix(n+d, d)
    for i in range(n):
        C[i] = copy(A[i])
    for i in range(d):
        C[i+n, i] = q
    C = C.LLL()

    #=============================
    # Extended Kannan's embedding
    #=============================
    B = Matrix(ZZ, d+k, d+k)
    for i in range(d):
        for j in range(d):
            B[i, j] = C[i+n, j]
    for i in range(k):
        # B[d+i, d+i] = 1
        B[d+i, d+i] = t
        for j in range(m):
            v = copy(b[j])
            for l in range(n):
                B[d+i, n*j + l] = v[l]
            b[j] = rot(b[j], n)
    # print("B = ", B)
    # print("b = ", b)

    #============================
    # Lattice basis reduction
    #============================
    # B = B.LLL()
    # print("B[0] = ", B[0])
    # BB = B.BKZ(block_size=30, prune=10, fp='fp')
    flags = BKZ.AUTO_ABORT|BKZ.MAX_LOOPS|BKZ.GH_BND
    par = BKZ.Param(55, strategies=BKZ.DEFAULT_STRATEGY, max_loops=4, flags=flags)

    A = IntegerMatrix(d+k, d+k)
    for i in range(d+k):
        for j in range(d+k):
            A[i, j] = B[i, j]
    # print("A = ", A)
    BB = BKZ.reduction(A, par)

    tmp = 0
    if BB[0].norm() >= 1.2*sigma*sqrt(d):
        tmp = 1
    else:
        v = BB[0]
        for i in range(d):
            if abs(v[i]) > 4*sigma:
                tmp = 1
    if tmp == 0:
        print("Success: ", BB[0])
        success = success + 1
    else:
        print("Failure")

print("k = ", k)
print("The number of success = ", success)
```

The NTRU Problem and Its Extension (1/3)

RIKKYO UNIVERSITY

- **NTRU problem**
  - Given $h = g \cdot f^{-1} \in R_q$, find f or $g \in R_q$
    - $R = \mathbb{Z}/q\mathbb{Z}[x]/(\phi)$ with $\phi = x^N \pm 1$
    - $f, g \in R_q$ have small coefficients (e.g., ± 1) s.t. f is invertible in R_q
- **NTRU lattice $\boldsymbol{L} = \boldsymbol{\mathcal{L}}(\mathbf{B})$**
  - $h = h_0 + h_1 x + \cdots + h_{N-1} x^{N-1} \mapsto \boldsymbol{h} = (h_0, h_1, \dots, h_{N-1})$: public
  - $\mathbf{B} = \begin{pmatrix} q\mathbf{I}_{N\times N} & \mathbf{0}_{N\times N} \\ \mathbf{H} & \mathbf{I}_{N\times N} \end{pmatrix}, \mathbf{H} = \begin{pmatrix} \boldsymbol{h} \\ \mathrm{rot}(\boldsymbol{h}) \\ \vdots \\ \mathrm{rot}^{N-1}(\boldsymbol{h}) \end{pmatrix}$
  - N short lattice vectors $\left(\mathrm{rot}^i(\boldsymbol{g}) \,\middle|\, \mathrm{rot}^i(\boldsymbol{f})\right) \in L$ for $0 \le i \le N-1$
    - Write $g(x) = f(x)h(x) + q \cdot r(x)$, $\exists r(x) \in R(x)$
    - $(\boldsymbol{g} \mid \boldsymbol{f}) = (\boldsymbol{f}\mathbf{H} - q\boldsymbol{r} \mid \boldsymbol{f}) = (-\boldsymbol{r} \mid \boldsymbol{f}) \begin{pmatrix} q\mathbf{I}_{N\times N} & \mathbf{0}_{N\times N} \\ \mathbf{H} & \mathbf{I}_{N\times N} \end{pmatrix} \in L$

The NTRU Problem and Its Extension (2/3)

RIKKYO UNIVERSITY

- **Extended NTRU lattice $L_k = \mathcal{L}(\mathbf{B}_k)$**
  - Add k rotated vectors $\mathrm{rot}^i(\boldsymbol{h})$

$$\mathbf{B}_k = \begin{pmatrix} q\mathbf{I}_{N\times N} & \mathbf{0}_{N\times N+k} \\ \mathbf{H_k} & \mathbf{I}_{N+k\times N+k} \end{pmatrix}, \mathbf{H}_k = \begin{pmatrix} \mathbf{H} \\ \boldsymbol{h} \\ \mathrm{rot}(\boldsymbol{h}) \\ \vdots \\ \mathrm{rot}^{k-1}(\boldsymbol{h}) \end{pmatrix}$$

  - $(k+1)N$ short vectors in L_k of form $\left(\mathrm{rot}^i(\boldsymbol{g}) \middle| \mathbf{0}_i \mid \boldsymbol{f} \mid \mathbf{0}_{k-i}\right)$ and its rotations
- **Experimental results**
  - The success probability for recovering a secret vector $\boldsymbol{f}, \boldsymbol{g}$, or its rotations
  - We used BKZ with $\beta = 60$
  - $\boldsymbol{k} = \mathbf{1}$ gives the highest success probability for most instances

  (cf., k=0: the original NTRU lattice)

表 1: 拡張 NTRU 格子 L_k に対する格子攻撃の成功確率
($\beta = 60$ の BKZ 2.0 を利用, $k = 0$ は元の NTRU 格子)

NTRU パラメータ $(N, q, d)^*$	拡張パラメータ $k=0$	$k=1$	$k=2$	$k=3$
(64, 31, 18)	31%	36%	32%	31%
(64, 41, 23)	46%	52%	38%	42%
(64, 53, 28)	65%	71%	78%	67%
(72, 31, 14)	71%	78%	68%	74%
(72, 41, 19)	52%	58%	48%	51%
(72, 53, 27)	18%	15%	13%	21%
(80, 67, 25)	41%	48%	42%	45%
(80, 89, 31)	69%	80%	75%	70%
(80, 101, 36)	66%	74%	62%	69%

The NTRU Problem and Its Extension (3/3)

```
from fpylll import *

N = 64; q = 31; d = 18; k = 0
R.<x> = PolynomialRing(ZZ)
Rq.<x> = PolynomialRing(GF(q))
I = R.ideal([x^N-1])
Iq = Rq.ideal([x^N-1])
S = R.quotient_ring(I, 'x')
Sq = Rq.quotient_ring(Iq, 'x')

def invertible_sample(N, o, mo):
    v = [0]*(N+1)
    v[0] = -1; v[N] = 1
    F = Rq(v)
    while(1):
        s=[1]*o+[-1]*mo+[0]*(N-o-mo)
        shuffle(s); res = Rq(s).gcd(F)
        if res == 1:
            break
    return S(s), Sq(s)

def sample(N, o, mo):
    s = [1]*o + [-1]*mo + [0]*(N-o-mo)
    shuffle(s)
    return S(s), Sq(s)

total = 0
for l in range(100):
    f, fq = invertible_sample(N, d+1, d)
    g, gq = sample(N, d, d)
    hq = gq*(fq)^-1
    H = Matrix(ZZ, N+k, N+k); F = hq
    for i in range(N+k):
        for j in range(N):
            H[i, j] = F[j]
        F *= x

    B = Matrix(ZZ, 2*N+2*k, 2*N+k)
    for i in range(N+k):
        B[i, i] = 1
        for j in range(N):
            B[i, j+N+k] = H[i, j]
    for i in range(N):
        B[i+N+k, i+N+k] = q
    for i in range(k):
        B[i+2*N+k, i] = 1
        B[i+2*N+k, N+i] = -1
    B = B.LLL()

    C = IntegerMatrix(2*N+k, 2*N+k)
    for i in range(2*N+k):
        for j in range(2*N+k):
            C[i, j] = B[i+k, j]
    flags = BKZ.AUTO_ABORT|BKZ.MAX_LOOPS|BKZ.GH_BND
    par = BKZ.Param(60, strategies=BKZ.DEFAULT_STRATEGY, max_loops=2, flags=flags)
    C = BKZ.reduction(C, par)

    ff = 0
    for i in range(N):
        G = [0]*(N); h = 0; flag = 0
        for j in range(N):
            G[j] = C[i, j+N+k]
            if abs(G[j]) <=1:
                h += abs(G[j])
            else:
                flag = 1
        if flag == 0 and h == 2*d:
            F = [0]*(N+k)
            for j in range(N+k):
                F[j] = C[i, j]
            F = Sq(F); G = Sq(G)
            if F*hq == G:
                print("Success")
                print("G = ", G)
                ff = 1
                total += 1
                break
    if ff == 0:
        print("Failure")

print("k = ", k)
print("total = ", total)
```