# Shortest Vector Problem 해결을 위한 기술 조사

**김현지**, 장경배, 오유진, 서화정
한성대학교

# Lattice and Basis

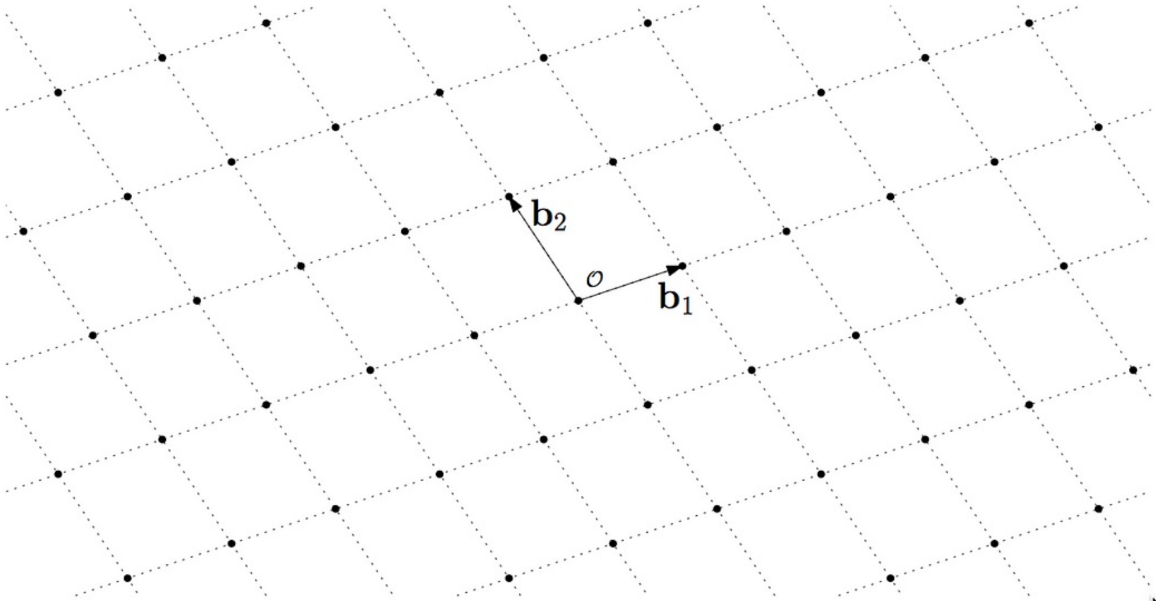
- **Lattice** $(L)$ : 기저 벡터 $(B)$의 **선형 결합으로 만들어진 점들의 집합**
- $\boldsymbol{x}$ : 정수, $(\boldsymbol{b}_1, \ldots, \boldsymbol{bn})$ :기저 벡터

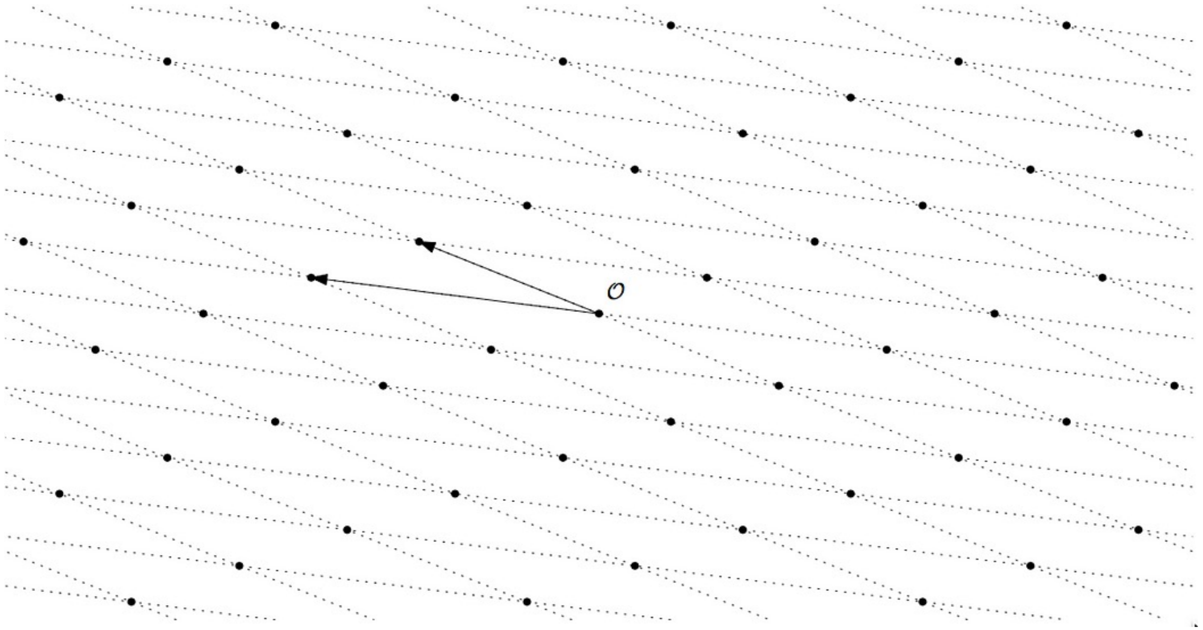
$$L(b_1, \ldots, b_n) = \Sigma_{i=1}^{n}(x_i \cdot b_i, x_i \in Z)$$

- **격자는 Rank $(n)$와 Dimension $(m)$이라는 두 요소를 가짐:**
  - **Rank** : 격자 벡터를 이루는 요소의 수
    - Rank= $3$ → $(00_{(2)}, 10_{(2)}, 11_{(2)})$
  - **Dimension** : 각 벡터의 길이
    - Dimension = $4$ → $(0000_{(2)}, 0100_{(2)}, 1100_{(2)})$
  - 일반적으로, a full-rank lattice 가 사용됨 $(m = n)$.
  - **격자 기반 암호시스템에서는** $m, n \geq 500$

# Lattice and Basis

- **하나의 격자를 표현하는 기저는 여러 개가 존재함**
  - 다시 말하면, 하나의 격자에는 여러 기저 벡터가 존재



# Good and bad basis

**Good basis** vs **bad basis**

일반적으로 **짧은 벡터**로 구성

Bad basis로부터 **Good basis**를 찾는 것은 **매우 어려운 문제**

Good basis에 unimodular matrix를 여러 번 곱함으로써 **good basis**를 만들 수 있음

Good basis로 부터 **bad basis**를 찾는 것은 상대적으로 쉬운 문제
→ **몇 번의 행렬 곱**만 필요하기 때문

- **격자 기반 암호에서는:**
  - **a bad basis**는 **the public key**로 사용
  - **a good basis**는 **the private key**로 사용
- **두 기저는 동일한 격자를 생성**
- **이러한 방식으로 공개키 및 개인키를 만드는 것은 격자 기반에서 메시지 복호화를 어렵게 만든다**

*https://en.wikipedia.org/wiki/Unimodular_matrix

# Shortest Vector Problems (SVP)

- **영벡터가 아닌 격자 상의 가장 짧은 벡터를 찾는 문제**

- Miklo's Ajtai 에서 SVP가 **NP-hard problem**라는 것이 밝혀짐

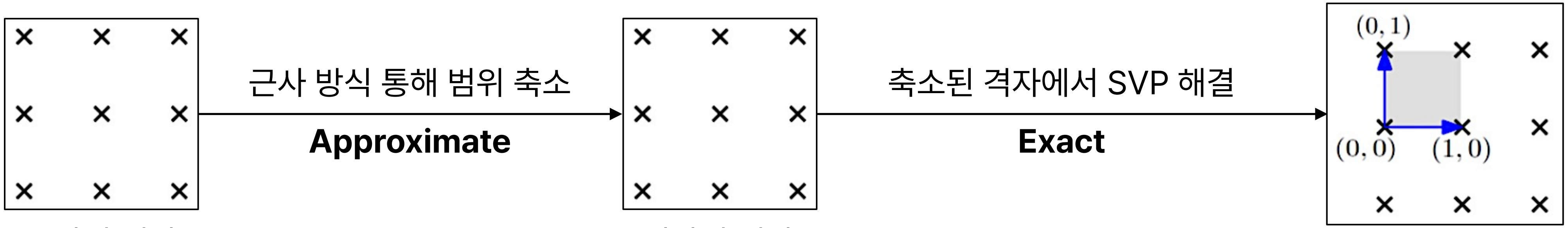
- **격자 기반 암호는 격자 문제 (SVP, CVP, etc.) 에 기반함**
  - 격자 기반 문제의 보안 강도는 격자 문제 해결의 어려움에 기반

- **즉 SVP를 해결하는 것 → 격자 기반 암호 시스템 (e.g. LWE)를 위협하는 것**

- **어려움**
  - **만약 bad basis가 입력으로 사용된다면, SVP 해결의 어려움이 증가 → Hard**
  - **만약 good basis가 입력으로 사용된다면, SVP 해결의 어려움은 감소 → Easy**
    - 높은 확률로 가장 짧은 벡터가 이미 good basis vectors에 포함되어 있을 수 있으므로
  - 또한, **rank ($n$) 와 dimension ($m$)이 증가할수록, SVP 해결은 어려워짐**

# Overview

**고차원에서 Approximate 통해 범위 축소 후, 해당 벡터들을 입력으로 Exact 방식 수행하여 SVP 해결**




고차원 격자 (Rank $\geq 100$)

저차원 격자 (Rank $\leq 60$)

- SVP를 위한 알고리즘 실행이 아닌 호출로 인한 **시간 오버헤드가 크게 발생**
- SV 찾기가 아닌 범위 축소를 위해 근사 방식 사용 즉, **고차원 격자 상의 벡터를 걸러내는 역할** (고차원에서는 **근사 방식만 효율적**)

- Exact 알고리즘을 집중적으로 사용
- **정확하게 가장 짧은 벡터** 찾아냄

- **Lattice crytanalysis를 위해서는 SVP를 해결해야 함**
  - 고차원에서는 **범위 축소**를 위한 **Approximate** 방식 사용
  - **정확한 가장 짧은 벡터**를 찾는 방식은 **Exact** 알고리즘
  - 따라서, 최상의 **실용적/이론적 SVP 솔루션은 저차원에서 정확하며 효율적**이어야 함
  - **낮은 차원에서의 SVP를 정확하게 해결**한 후, **해결 가능한 가장 높은 차원을 결정**하는 것이 중요

# Survey on the approximate algorithms for SVP

- **Approximate 방식**
  - 높은 차원의 격자를 낮은 차원의 격자로 감소
  - 대표적인 알고리즘으로는 LLL과 BKZ
  - 해당 방법들은 가장 짧은 벡터가 아닌 그보다 다소 긴 벡터들을 얻는 것
- **LLL (Lenstra–Lenstra–Lovász)**
  - 가장 초기의 알고리즘 (1982년 제안)
  - 주어진 격자에 대해 축소 기저를 계산
  - 다항 시간 안에 실행되지만, 가장 짧은 벡터를 직접 찾는 것이 아니므로 약간 긴 벡터를 얻게 됨
- **BKZ (Block Korkine-Zolotarev)**
  - LLL 이후 개발된 알고리즘
  - 격자의 퀄리티와 시간 간의 트레이드 오프 존재
  - Block 단위로 격자 축소를 진행 → 새로 계산된 블록은 이전 기저 블록들의 조합 $\mathbf{b}_{\text{new}}$ → $\mathbf{b}_1, \ldots, \mathbf{b}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_{i+\beta-1}$
  - 알고리즘의 성능은 block 크기에 의존

# Survey on the exact algorithms for SVP

- Exact 알고리즘으로 잘 알려진 알고리즘 : **AKS and NV Sieve**

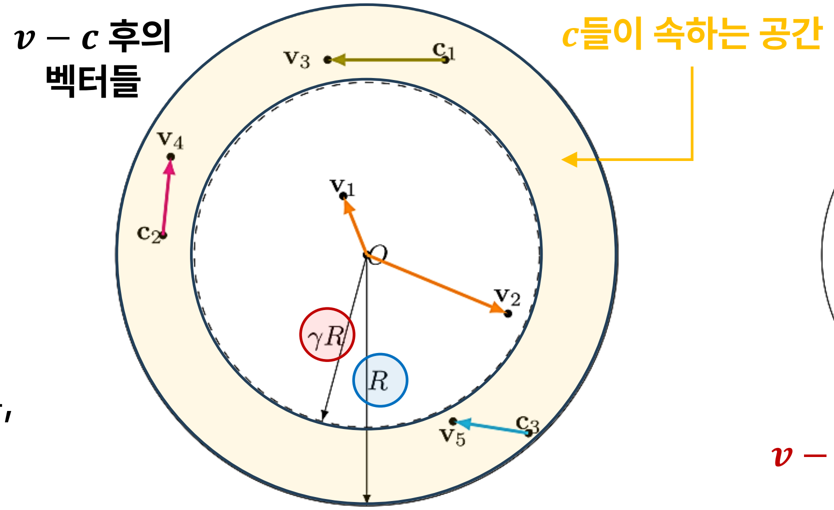
- **AKS**
  - 가장 초기의 알고리즘
  - 그러나 **많은 파라미터와 시간 및 공간 복잡도를 가짐**
  - 또한, 이에 대한 실질적인 구현과 최적 파라미터의 부재로 인해 **impractical algorithm**으로 여겨짐

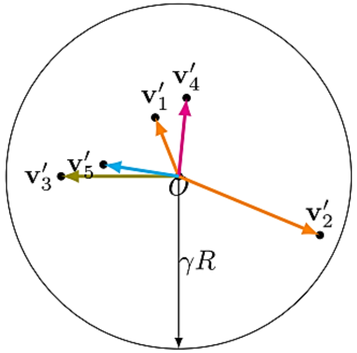
- **NV Sieve**
  - **AKS의 단점을 보완**하기 위해 제안
  - **시간 및 공간 복잡도 감소 및 실용적인 알고리즘**
  - **실제 구현 및 평가를 제시**
  - NV Sieve가 제안된 후로, AKS 및 NV Sieve를 기반으로 List Sieve and Gaussian Sieve 등의 많은 변형 알고리즘들이 제안됨

- 또한, 이론적 계산이긴 하나 양자 컴퓨터 상에서의 복잡도가 제시됨. (구현 아님)

# NV Sieve

- **목적** : 짧은 벡터에 대한 손실이 없게 하기 위해 **$c$를 랜덤으로 선택**하여 **범위를 줄여 나가며 $\gamma R$ 보다 짧은 벡터 얻기**
- **입력** : 최대 길이가 $R$인 격자 상의 벡터
- **출력** : $v$보다 짧은 격자 상의 벡터
- **동작 과정**
  1. $S$ 에 속한 벡터들 중 최대 길이 → $R$
     $C, S'$ 초기화
  2. $\gamma R$보다 길이가 짧은 벡터들은 $S'$에 저장
  3. $\gamma R$ 보다 길이가 긴 벡터들은 $c$라는 포인트와 뺄셈 한 후,
     그 길이가 $\gamma R$보다 짧으면 $S'$에 저장 / 길면 $C$에 저장
  4. $S'$ 반환 ($\gamma R$보다 길이가 짧은 벡터들)




**$v - c$ 들을 원점으로 이동해보면?**
**범위 안에 속함**

- **이번 발표에서 초점을 두는 복잡도 관련 부분은 다음과 같음**
- 이 과정에서 **조건을 만족하는 짧은 벡터를 찾게 되면 $S'$에 저장 후 다음 입력으로 이동**

  → $c$를 찾는 해당 부분이 NV Sieve의 **복잡도에 큰 영향을 미침**

  → **$c$가 될 수 있는 충분히 많은 포인트들을 알 수 없으며, 그 중에서도 조건을 만족하는 점이 있는지 모름**

# NV Sieve 이후의 Sieve 알고리즘 동향

- 대표적인 알고리즘으로 List Sieve, Gaussian Sieve 등이 존재

- **List Sieve**
  - 2009년에는 AKS Sieve 알고리즘을 기반으로 제안
  - $N$개의 $(p_i, e_i)$벡터 쌍을 입력으로 하여 각 sieve iteration에서 list 상의 모든 벡터들에 의해 $p_i$를 줄여나감
  - 이후, $v_i = p_i - e_i$를 통해 구해지는 최종 $v_i$ 를 list에 더함으로써 계속해서 벡터 집합을 축소시켜 나가는 방식
  - 이후, Sieve-Birthday Sieve, Gaussian Sieve 등이 제시

- **이러한 변형 알고리즘 외에도 향상된 NV Sieve 알고리즘도 제안**
  - 해당 연구에서는 2-level NV Sieve 를 제안
  - 계산 및 공간 복잡도 $(\log_2(time, space))$를 각각 $time = 0.3836$, $time = 0.2557$로 감소시킴

- **현재는 Quantum 환경에서 구현한 Quantum NV Sieve에 대한 연구도 진행 중**

# 결론

- **SVP를 해결하는 것은 격자 기반 암호의 안정성을 위협하는 것**
- 그러나, 실제 격자 기반 암호는 **500 이상의 rank와 dimension**을 가짐
- 이를 해결하기 위한 방법론은 **approximate + exact 알고리즘을 함께 사용하는 것**
  - Approximate algorithm : 매우 큰 차원에서 적당히 짧은 벡터들을 걸러내는 것
  - Exact algorithm : 정확한 짧은 벡터를 찾는 알고리즘
- Approximate 알고리즘에는 대표적으로 **LLL, BKZ** 알고리즘이 존재
- **Exact 알고리즘에는 대표적으로 AKS와 NV Sieve 알고리즘이 있으며, 이를 기반으로 여러 변형들이 제안됨**
  - 시간 및 공간 복잡도들이 줄어든 케이스가 다수 존재
- **양자 컴퓨터에서 SVP를 해결하기 위한 시도들이 진행 중**
  - 이론적으로 계산 및 공간 복잡도를 계산한 연구
  - NV Sieve를 실제 구현으로 옮긴 연구

# Q & A