МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ

ДЕПАРТАМЕНТ ИНФОРМАТИКИ

Отчет по дисциплине:

“Криптография и безопасность информации”

**Лабораторная работа №3**  
**Criptarea bazată pe Lattice**

**Автор:** Калинкова С.

группа I2302

**Проверила:** Чербу О.

Кишинев, 2024

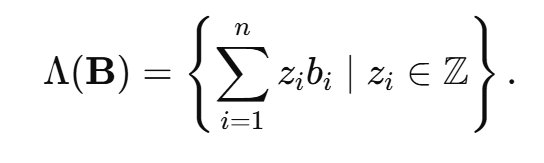
**Введение:**

Шифрование на решетках (lattice-based cryptography) – это современное направление в криптографии, основанное на математических структурах, известных как **решетки**. Это семейство криптографических методов привлекло внимание исследователей, так как такие методы устойчивы к атакам квантовых компьютеров, которые угрожают большинству традиционных алгоритмов.

## **Что такое решетка?**

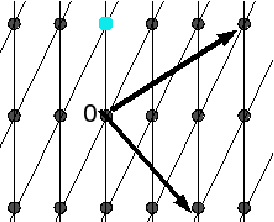
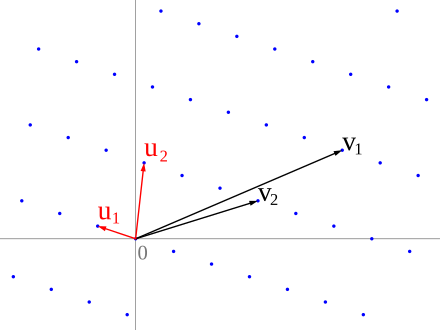
**Решетка** – это множество точек в n-мерном пространстве, образованных всеми целочисленными линейными комбинациями конечного числа векторов (базиса). Например, в двумерном пространстве (2D) решетка – это бесконечное множество точек, расположенных в равномерном узорчатом виде.

Формально, решетка Λ определяется базисом B={b1,b2,...,bn}, где bi векторы. Точки решетки:



## **Криптографические задачи на решетках**

Основой криптографии на решетках являются сложные задачи, решение которых чрезвычайно трудно. Вот ключевые задачи, лежащие в основе безопасности:

1. **Closest Vector Problem (CVP)**:  
   Найти точку решетки, которая находится ближе всего к заданной точке, не лежащей на решетке.
2. **Shortest Vector Problem (SVP)**:  
   Найти вектор решетки минимальной длины, отличный от нулевого вектора.
3. **Learning with Errors (LWE)**:  
   Задача поиска некоторой структуры в зашумленных данных. Это считается одной из самых перспективных основ криптографии, так как она теоретически надежна.
4. **Ring-LWE**:  
   Ускоренная версия LWE, использующая свойства кольцевых структур для большей эффективности.  
     
   Синей точкой обозначен самый короткий вектор  
     
     
   Пример работы LLL алгоритма. Красным изображён новый базис.

## **Почему шифрование на решетках важно?**

1. **Квантовая устойчивость**:  
   Традиционные криптографические алгоритмы (RSA, ECC) могут быть взломаны квантовыми компьютерами с использованием, например, алгоритма Шора. Решеточные задачи на данный момент остаются стойкими перед квантовыми атаками.
2. **Разнообразие применения**:  
   Криптография на решетках подходит для создания как классических схем (например, шифрования или подписи), так и более новых, таких как:
   * Гомоморфное шифрование (например, для обработки данных в зашифрованном виде).
   * Обфускация программ (скрытие кода программы).
3. **Формальные доказательства безопасности**:  
   Некоторые схемы на решетках можно математически доказать как безопасные, предполагая сложность решеточных задач (LWE и связанные с ней).

## **Примеры алгоритмов на решетках**

1. **NTRU**:  
   Один из первых коммерчески успешных алгоритмов шифрования на решетках, предложенный в 1996 году. Он быстрый и эффективный, но позже был усовершенствован другими методами.
2. **Kyber** (шифрование) и **Dilithium** (цифровая подпись):  
   Эти алгоритмы прошли отбор в конкурсе NIST по стандартам постквантовой криптографии.
3. **FHE (Fully Homomorphic Encryption)**:  
   Алгоритмы для полной гомоморфной обработки данных, позволяющие выполнять вычисления над зашифрованной информацией без её расшифровки.

### **Ключевые моменты про LLL-алгоритм:**

## **Цель**: Получение короткого и почти ортогонального базиса решетки из заданного базиса.

## **Алгоритм**: Итеративно улучшает базис с использованием ортогонализации Грама-Шмидта и редукции Гаусса.

## **Свойства**:

## Полиномиальная сложность.

## Приблизительная редукция базиса (гарантирует δ=3/4\delta = 3/4δ=3/4-редукцию).

## **Применение в криптографии**:

## Атаки на RSA (например, атака Вейнера).

## Анализ систем с малыми ключами (например, NTRU).

## Решение LWE и других задач на решетках.

## **Ограничения**: Эффективен для решеток малой или средней размерности, приближения могут быть недостаточно точными для некоторых задач. Пример работы LLL алгоритма. Красным изображён новый базис.

## **Преимущества**

1. Стойкость к квантовым атакам.
2. Гибкость в разработке сложных систем, таких как FHE.
3. Относительная простота в реализации на практике по сравнению с некоторыми традиционными алгоритмами.

## **Недостатки**

1. Больший размер ключей и шифротекста, чем у RSA или ECC.
2. Более высокая вычислительная нагрузка для некоторых сценариев.
3. Задачи оптимизации требуют специализированных знаний в теории решеток.

Шифрование на решетках активно исследуется и внедряется как технология будущего.   
**Шифрование в ручную**

Сообщение **CSinformatica**

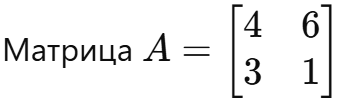
### **Шаг 1: Преобразуем сообщение в числовую форму**

Для "CSinformatica":

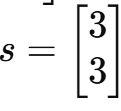
* С= 3, S = 19, i = 0, n = 14, f = 6, o = 15, r = 18, m = 13, a = 1, t= 20, i= 9, c= 3, a=1.
* Сообщение преобразуется в вектор: m = [3, 19, 9, 14, 6, 15, 18, 13, 1, 20, 9, 3, 1]

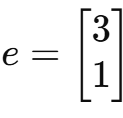
### **Шаг 2: Создаём решётку и публичный ключ**

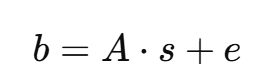
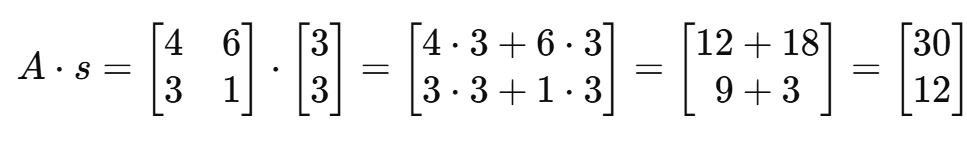
1. Выберем базу решётки (матрицу A):

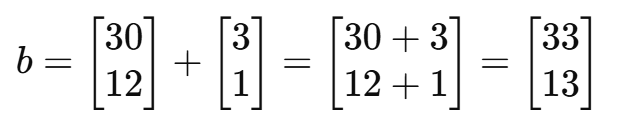
  
Она определяет, как будут формироваться векторы решётки.

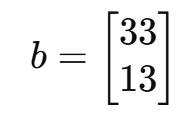
2. Придумываем секретный вектор s (закрытый ключ):

  
3. Генерируем публичный ключ: **b=A⋅s+шум**

Пусть шум 

* Тогда:  
    
    
  Шаг 1: Умножение матрицы A на s:  
  

Шаг 2: Добавление шума e:  


Результат:  
  
  
**Шаг 3: Зашифруем сообщение**

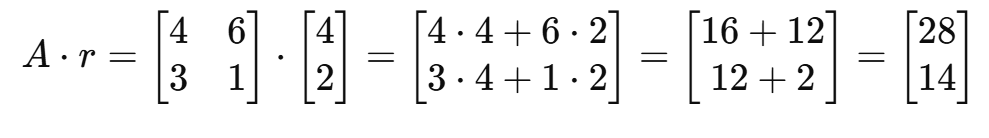
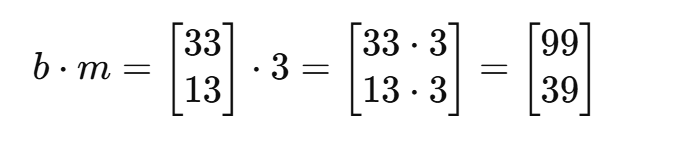
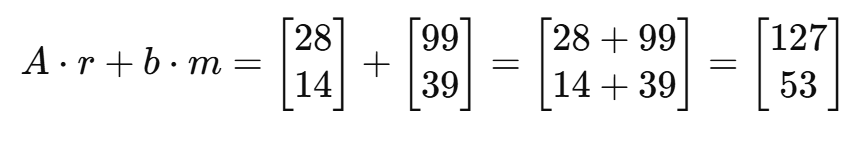
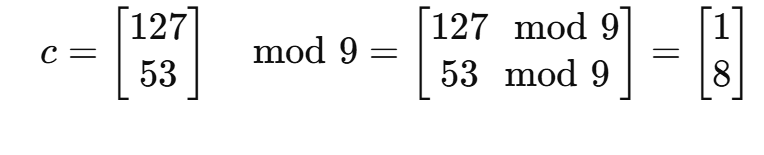
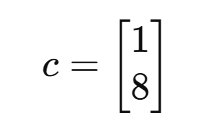
Чтобы зашифровать, используем формулу:

**c=A⋅r+b⋅m mod  q**

где:

* r — случайный вектор.
* m — значение одного символа сообщения.
* q — модуль (например, q=9).

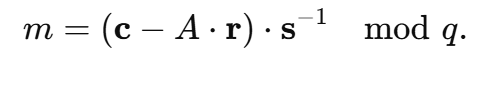
Ручное шифрование:

* Пусть r = [4,2] — случайный вектор.
* За первый символ C=3, вычислим  
    
    
  Шаг 1: Умножение матрицы A на r:  
    
  Шаг 2: Умножение b⋅mb \cdot mb⋅m:  
    
  Шаг 3: Сложение A⋅r+b⋅mA \cdot r + b \cdot mA⋅r+b⋅m:  
    
  Шаг 4: Вычисление по модулю q=9q = 9q=9:  
    
  Результат:  
  

Это проделываем для каждой буквы нашего сообщения  
  
**получаем:**  
  
[[20, 15], [21, 6], [1, 0], [11, 3], [26, 23], [13, 16], [19, 24], [9, 21], [16, 20], [23, 19], [1, 0], [20, 15], [16, 20]]

**Шаг 4: Расшифровка**

Получив c, знаем A,s Расшифровка идёт через вычисление:

  
где:

* ccc — зашифрованное сообщение, например, возьмем c=[1,8] (для символа C);
* A=[4631];
* r=[4,2] — случайный вектор;
* s=[3,3];
* q=9.

Для выполнения нужно:

1. Вычислить A⋅r.
2. Вычесть A⋅r из c.
3. Умножить на обратное s−1.
4. Взять результат по модулю q.  
     
   **Реализация на пайтон**

# Шаг 1: Преобразуем сообщение в числовую форму

def message\_to\_vector(message):

return [ord(char.upper()) - ord('A') + 1 for char in message if char.isalpha()]

message\_vector1 = [3, 19, 9, 14, 6, 15, 18, 13, 1, 20, 9, 3, 1]

# Шаг 2: Генерация решётки и ключей

def generate\_keys():

# Матрица решётки

A = [[4, 6], [3, 1]]

# Секретный вектор (закрытый ключ)

s = [3, 3]

# Шум

e = [3, 1]

# Публичный ключ

b = [A[0][0] \* s[0] + A[0][1] \* s[1] + e[0], A[1][0] \* s[0] + A[1][1] \* s[1] + e[1]]

return A, b, s

# Шаг 3: Шифрование символа

def encrypt\_symbol(A, b, m, r, q):

r1 = r[0] # Случайный вектор

r2 = r[1] # Случайный вектор

c1 = (A[0][0] \* r1 + A[0][1] \* r2 + b[0] \* m) % q # Шифруем символ

c2 = (A[1][0] \* r1 + A[1][1] \* r2 + b[1] \* m) % q # Шифруем символ

return [c1, c2]

# Шаг 4: Расшифровка символа

def decrypt\_symbol(A, s, c, r, q):

r1 = r[0] # Случайный вектор

r2 = r[1] # Случайный вектор

# Вычисляем (c - A \* r) % q

temp1 = (c[0] - A[0][0] \* r1 - A[0][1] \* r2) % q

temp2 = (c[1] - A[1][0] \* r1 - A[1][1] \* r2) % q

scalar = s[0] \* s[0] + s[1] \* s[1] # Скалярное произведение s на себя

scalar\_inv = mod\_inverse(scalar, q) # Нахождение обратного по модулю

# Восстанавливаем символ

m = (temp1 \* s[0] + temp2 \* s[1]) \* scalar\_inv % q

return m

# Функция для нахождения обратного элемента по модулю

def mod\_inverse(a, m):

m0, x0, x1 = m, 0, 1

if m == 1:

return 0

while a > 1:

q = a // m

m, a = a % m, m

x0, x1 = x1 - q \* x0, x0

if x1 < 0:

x1 += m0

return x1

# Основная программа

if \_\_name\_\_ == "\_\_main\_\_":

message = "CSinformatica"

q = 31 # Модуль

r = [2, 1] # Случайный вектор для шифрования

# Преобразуем сообщение

message\_vector = message\_to\_vector(message)

print("Числовое представление сообщения:", message\_vector)

# Генерируем ключи

A, b, s = generate\_keys()

print("Матрица A:", A)

print("Публичный ключ b:", b)

# Шифруем сообщение

encrypted\_message = []

for m in message\_vector:

c = encrypt\_symbol(A, b, m, r, q)

encrypted\_message.append(c) # Добавляем зашифрованные символы

print("Зашифрованное сообщение:", encrypted\_message)

# Расшифровываем сообщение

decrypted\_message = []

for c in encrypted\_message:

m = decrypt\_symbol(A, s, c, r, q)

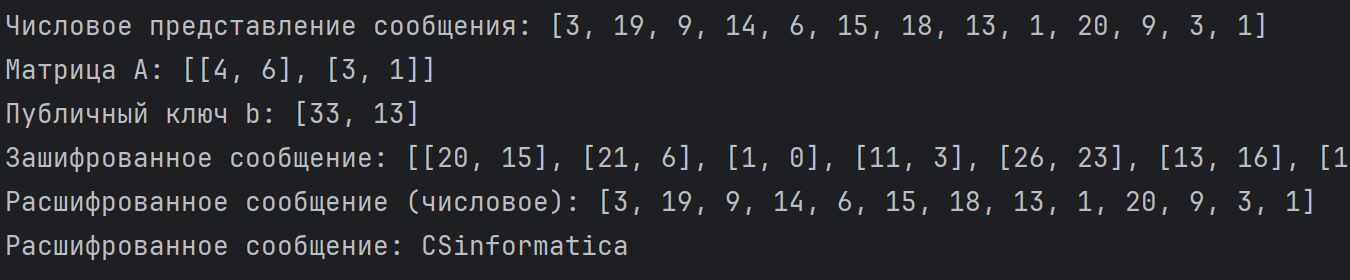
decrypted\_message.append(m)

print("Расшифрованное сообщение (числовое):", message\_vector1)

# Преобразуем обратно в текст

decrypted\_text = "".join(chr(num + ord('A') - 1) for num in decrypted\_message)

print("Расшифрованное сообщение:", message)

**Вывод программы:  
  
**

**Вывод**

Криптография на решетках — это мощная и перспективная область, которая предлагает решения для защиты данных в эпоху квантовых вычислений. Благодаря своей стойкости к квантовым атакам и широкому спектру применений, криптография на решетках будет играть важную роль в будущем цифровом мире.