Opis działania kryptosystemu GGH		
Projekt	**Temat:** Omówienie kryptosystemu GGH	**Przedmiot:** Kryptografia
Nr indeksu		**423442, 413685**
Autorzy:	Piotr Skoczylas, Mieszko Makowski	

Wstep teoretyczny

Kryptosystem Goldreich-Goldwasser-Halevi (GGH) to asymetryczny system kryptograficzny, który opiera sie na problemach z zakresu teorii krat (ang. *lattice theory*), bedacej jedna z kluczowych dziedzin matematyki stosowanej w kryptografii. Kraty to struktury geometryczne definiowane jako zbiory punktów w przestrzeni wielowymiarowej, wyrażane w postaci kombinacji liniowych pewnego zestawu liniowo niezależnych wektorów zwanych baza kraty.

GGH wykorzystuje problem najbliższego wektora (ang. *Shortest Vector Problem*, SVP), który jest matematycznie trudnym zagadnieniem i stanowi podstawe bezpieczeństwa tego systemu. Trudność tego problemu jest zwiazana z redukcja krat, czyli poszukiwaniem możliwie najkrótszego (lub najbliższego) wektora w strukturze geometrycznej, co w praktyce wymaga znacznej mocy obliczeniowej.

Kryptosystem GGH został zaprezentowany w 1997 roku przez Odeda Goldreicha, Shafiego Goldwassera i Shaia Haleviego. System ten wykorzystuje jednokierunkowa funkcje zapadni (ang. *trapdoor one-way function*), której trudność polega na tym, że przy znajomości bazy kraty łatwo można przekształcać punkty i wektory w przestrzeni w określony sposób (np. dodajac mały wektor błedu). Jednak aby odwrócić ten proces i odnaleźć oryginalny punkt, konieczna jest znajomość specjalnej bazy – tzw. zapadni.

Przykładowo, w schemacie GGH, szyfrowanie polega na dodaniu niewielkiego wektora błedu do punktu w kratce. Dzieki zapadni, odbiorca (znajacy odpowiednia baze kraty) jest w stanie odwrócić proces szyfrowania i odzyskać oryginalna wiadomość. Natomiast dla osoby nieposiadajacej zapadni, problem sprowadza sie do trudnego SVP, co czyni system bezpiecznym w założeniu.

Historia i bezpieczeństwo

Schemat szyfrowania GGH poczatkowo wydawał sie obiecujacy, jednak jego bezpieczeństwo zostało podważone w 1999 roku, kiedy Phong Q. Nguyen przeprowadził skuteczna kryptoanalize, łamiac system za pomoca algorytmów redukcji bazy kraty, takich jak LLL (ang. Lenstra–Lenstra–Lovász).

Podobne problemy napotkano w schemacie podpisu GGH, który opierał sie na podobnych zasadach. W 2006 roku Phong Q. Nguyen, wspólnie z Odedem Regevem, złamał również schemat podpisu, co podważyło praktyczne zastosowanie kryptosystemu w rzeczywistych implementacjach.

Mimo to, teoria stojaca za systemem GGH oraz trudność zwiazana z problemami krat, takimi jak SVP, wciaż stanowi podstawe wielu nowoczesnych systemów kryptograficznych, w tym kryptografii odpornej na komputery kwantowe (ang. *post-quantum cryptography*). Kraty pozostaja jednym z najaktywniej badanych obszarów kryptografii, a wyniki badań zwiazanych z GGH miały kluczowy wpływ na rozwój tego pola.

Schemat działania kryptosystemu GGH

1. Generowanie klucza

Wybór macierzy:

- Macierz bazowa B: odwracalna i dobrze uwarunkowana.
- Macierz unimodularna U: macierz całkowitoliczbowa o wyznaczniku $|\det(U)| = 1$.

Obliczenie klucza publicznego:

$$B' = U \cdot B$$

- Klucz publiczny: B'.
- Klucz prywatny: B, U.

2. Szyfrowanie wiadomości

Reprezentacja wiadomości: Wiadomość m reprezentowana jest jako wektor liczb całkowitych.

Dodanie błedu: Wygeneruj mały losowy wektor błedu e.

Obliczenie szyfrogramu:

$$c = m \cdot B' + e$$

3. Deszyfrowanie wiadomości

Otrzymanie szyfrogramu: c.

Przekształcenie w przestrzeń krat: Oblicz:

$$v = c \cdot (B')^{-1}$$

Zastosowanie algorytmu Babai'ego: Przybliżenie v do najbliższego punktu na kracie:

$$v_{\text{rounded}} = \text{round}(v)$$

Odzyskanie wiadomości w pierwotnej przestrzeni:

$$m = v_{\text{rounded}} \cdot U^{-1}$$

Podsumowanie

- Klucz publiczny: Macierz B', która maskuje strukture krat.
- Klucz prywatny: Macierze B i U, które umożliwiaja efektywne deszyfrowanie.
- Bezpieczeństwo: Oparte na trudności problemu najkrótszego wektora (Shortest Vector Problem, SVP), który jest matematycznie złożonym zagadnieniem.

Implementacja kryptosystemu GGH

W tej sekcji przedstawiono implementacje kryptosystemu GGH z podziałem na trzy główne elementy: generowanie klucza, szyfrowanie wiadomości oraz deszyfrowanie jej.

1. Generowanie klucza

```
import numpy as np
import random

def random_invertible_matrix(n, min_val=-5, max_val=5):
    while True:
        M = np.random.randint(min_val, max_val+1, size=(n, n))
        try:
            _ = np.linalg.inv(M.astype(np.float64))
            return M
        except np.linalg.LinAlgError:
            pass

def random_unimodular_matrix(n, min_val=-2, max_val=2):
    U = np.eye(n, dtype=np.int64)
    ops = 2 * n
    for _ in range(ops):
        i = random.randint(0, n-1)
        j = random.randint(0, n-1)
        if i != j:
            k = random.randint(min_val, max_val)
            U[i, :] = U[i, :] + k * U[j, :]
    detU = round(np.linalg.det(U))
    if detU == 1 or detU == -1:
        return U
    else:
        return random_unimodular_matrix(n, min_val, max_val)

def matrix_to_list_of_lists(M):
    return M.tolist()

def main():
    print("Podaj wymiar macierzy (n): ", end="")
    n_str = input()
    n = int(n_str)
```

```python
    print(f"===␣Generowanie␣klucza␣GGH␣w␣wymiarze␣n␣=␣{n}␣===")

    B = random_invertible_matrix(n, -5, 5)
    U = random_unimodular_matrix(n, -2, 2)

    B_inv = np.linalg.inv(B.astype(np.float64))
    U_inv = np.linalg.inv(U.astype(np.float64))

    B_prime = U @ B

    B_list       = matrix_to_list_of_lists(B)
    B_inv_list   = matrix_to_list_of_lists(B_inv)
    U_list       = matrix_to_list_of_lists(U)
    U_inv_list   = matrix_to_list_of_lists(U_inv)
    B_prime_list = matrix_to_list_of_lists(B_prime)

    print("\n===␣Macierze␣do␣skopiowania:␣===")
    print("B␣=", B_list)
    print("B_inv␣=", B_inv_list)
    print("U␣=", U_list)
    print("U_inv␣=", U_inv_list)
    print("B'␣=", B_prime_list)

    print("\nSkopiuj␣powy sze␣dane␣i␣u yj␣w␣encrypt.py␣i␣
       decrypt.py")

if __name__ == "__main__":
    main()
```

Listing 1: Kod generowania klucza dla kryptosystemu GGH

Opis:

- Funkcja `random_invertible_matrix(n)` generuje losowa macierz B, która jest odwracalna.
- Funkcja `random_unimodular_matrix(n)` generuje macierz unimodularna U, która ma wyznacznik ± 1.
- Funkcja `main()` tworzy klucz publiczny $B' = U \cdot B$ oraz jego składowe macierze, a także ich odwrotności.

2. Szyfrowanie wiadomości

```python
import numpy as np
import ast
import random

MAX_ERROR_NORM = 5.0

def main():
```

```
    print("Podaj wymiar macierzy (n): ", end="")
    n_str = input()
    n = int(n_str)

    print(f"=== Szyfrowanie GGH w wymiarze (n={n}) ===")
    print(f"Podaj macierz B' (klucz publiczny) w formacie np. [[1,2],[3,4], ...] (dla n={n}]:")
    B_prime_str = input("> ")
    B_prime_list = ast.literal_eval(B_prime_str)
    B_prime = np.array(B_prime_list, dtype=np.float64)

    if B_prime.shape != (n, n):
        print("[B  D] Wczytana macierz B' ma niepoprawny rozmiar!")
        return

    print(f"Teraz wprowad  liczby ca kowite wiadomo ci m w ilo ci r wnej {n}.")
    m_values = []
    for i in range(n):
        print(f"m[{i}] = ", end="")
        val_str = input()
        val_int = int(val_str)
        m_values.append(val_int)

    m = np.array(m_values, dtype=np.float64)
    print("\nPodano wiadomo   m =", m_values)

    while True:
        e_rand = np.random.uniform(-3, 3, size=n)
        if np.linalg.norm(e_rand, ord=2) <= MAX_ERROR_NORM:
            e = e_rand
            break

    e = e.astype(np.int64)
    c = m @ B_prime + e
    c_list = c.astype(np.int64).tolist()

    print("\n=== Szyfrogram ===")
    print("c =", c_list)
    print("\n(Wektor b   du e =", e.tolist(), ")")

if __name__ == "__main__":
    main()
```

Listing 2: Kod szyfrowania dla kryptosystemu GGH

Opis:

- Klucz publiczny B' jest używany do szyfrowania wiadomości m.
- Wiadomość m to wektor liczb całkowitych wprowadzony przez użytkownika.

- Wektor błedu e jest generowany losowo, a nastepnie dodawany do $m \cdot B'$.
- Wynik to szyfrogram c, który zostaje wypisany w konsoli.

3. Deszyfrowanie wiadomości

```
import numpy as np
import ast

def babai_round(vec):
    return np.round(vec).astype(np.int64)

def main():
    print("Podaj wymiar macierzy (n): ", end="")
    n_str = input()
    n = int(n_str)

    print(f"=== Deszyfrowanie GGH w wymiarze n={n} ===")

    print("Podaj macierz B (w formacie list-of-lists), np. 
      [[1,2],[3,4], ...]:")
    B_str = input("> ")
    B_list = ast.literal_eval(B_str)
    B = np.array(B_list, dtype=np.float64)
    if B.shape != (n, n):
        print("[B  D] Rozmiar macierzy B niezgodny z n!")
        return

    print("Podaj macierz B_inv (w formacie list-of-lists), np. 
      [[1,2],[3,4], ...]:")
    B_inv_str = input("> ")
    B_inv_list = ast.literal_eval(B_inv_str)
    B_inv = np.array(B_inv_list, dtype=np.float64)
    if B_inv.shape != (n, n):
        print("[B  D] Rozmiar macierzy B_inv niezgodny z n!")
        return

    print("Podaj macierz U_inv (w formacie list-of-lists), np. 
      [[1,2],[3,4], ...]:")
    U_inv_str = input("> ")
    U_inv_list = ast.literal_eval(U_inv_str)
    U_inv = np.array(U_inv_list, dtype=np.float64)
    if U_inv.shape != (n, n):
        print("[B  D] Rozmiar macierzy U_inv niezgodny z n!")
        return

    print(f"Podaj szyfrogram c (jako list  {n} liczb), np. [123,
      56, 0, -22, ...]:")
    c_str = input("> ")
    c_list = ast.literal_eval(c_str)
    c = np.array(c_list, dtype=np.float64)
```

```
    if c.shape != (n,):
        print("[B   D]␣Rozmiar␣szyfrogramu␣c␣niezgodny␣z␣n!")
        return

    v = c @ B_inv
    v_rounded = babai_round(v)
    m_float = v_rounded @ U_inv
    m_decrypted = babai_round(m_float)

    print("\n===␣Odszyfrowana␣wiadomo   ␣===")
    print("m␣=", m_decrypted.tolist())

if __name__ == "__main__":
    main()
```
Listing 3: Kod deszyfrowania dla kryptosystemu GGH

Opis:

- `babai_round(vec)`: Zaokragla wektor do najbliższej kratki.
- `main()`: Implementuje proces deszyfrowania:
  - Przyjmuje klucz prywatny (B, B^{-1}, U^{-1}) i szyfrogram c.
  - Odtwarza wiadomość m za pomoca zaokraglenia Babai'ego.

Działanie kryptosystemu GGH - szczegółowe omówienie

1. Teoria krat

Kryptosystem GGH opiera sie na zaawansowanych zagadnieniach z zakresu teorii krat (ang. lattice theory), która zajmuje sie badaniem struktur geometrycznych definiowanych jako zbioru punktów w przestrzeni wielowymiarowej. Krata jest zbiorem punktów, które moga być zapisane w postaci:

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} \alpha_i \mathbf{b}_i \mid \alpha_i \in \mathbb{Z} \right\}$$

gdzie $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ to liniowo niezależne wektory nazywane baza kraty, a n to ranga kraty. Kraty sa używane w kryptografii ze wzgledu na ich własności geometryczne i algebraiczne.

2. Problem najbliższego wektora (Nearest Vector Problem - NVP)

Jednym z kluczowych problemów teorii krat jest problem najbliższego wektora (ang. Nearest Vector Problem, NVP). Polega on na znalezieniu wektora $\mathbf{v} \in \mathcal{L}$, który jest najbliższy danemu punktowi $\mathbf{p} \in \mathbb{R}^n$. Problem ten jest matematycznie trudny i znany z bycia NP-trudnym dla ogólnych przypadków. Bezpieczeństwo kryptosystemu GGH wynika właśnie z trudności rozwiazania tego problemu bez znajomości specjalnej struktury kraty.

3. Redukcja krat

Redukcja krat odnosi sie do procesu znajdowania bardziej "dobrze uwarunkowanej" bazy dla danej kraty. Baza ta charakteryzuje sie krótszymi i bardziej ortogonalnymi wektorami. W kryptosystemie GGH wykorzystywana jest baza dobrze uwarunkowana (prywatna) oraz baza "gesta" (publiczna), która utrudnia znalezienie najbliższego punktu bez znajomości bazy prywatnej.

4. Algorytm LLL

Jednym z popularnych algorytmów redukcji krat jest algorytm Lenstra-Lenstra-Lovásza (LLL). Algorytm ten znajduje baze kraty, która jest "wystarczajaco" dobrze zredukowana. W kontekście kryptosystemu GGH, algorytm LLL może być wykorzystywany do ataków, dlatego ważne jest staranne dobieranie parametrów systemu.

5. Funkcje zapadniowe (Trapdoor Functions)

Kryptosystem GGH wykorzystuje pojecie funkcji zapadniowych (ang. trapdoor functions), które sa łatwe do obliczenia w jedna strone, ale trudne do odwrócenia bez znajomości dodatkowych informacji (tzw. zapadni). W przypadku GGH, funkcje zapadniowa realizuje przekształcenie przestrzeni kraty za pomoca bazy publicznej, podczas gdy baza prywatna stanowi "zapadnie" umożliwiajaca odwrócenie przekształcenia.

6. Problem najkrótszego wektora (Shortest Vector Problem - SVP)

Podobnie jak NVP, problem najkrótszego wektora (ang. Shortest Vector Problem, SVP) jest fundamentalnym problemem teorii krat. Polega on na znalezieniu najkrótszego, niezerowego wektora w kracie. Problem ten również jest NP-trudny, a jego trudność zapewnia dodatkowy poziom bezpieczeństwa dla kryptosystemów opartych na kratach, takich jak GGH.

7. Algebra liniowa i macierze

System GGH intensywnie wykorzystuje narzedzia algebry liniowej, takie jak mnożenie macierzy, odwracanie macierzy i przekształcenia liniowe. Klucz prywatny składa sie z macierzy bazy dobrze uwarunkowanej oraz macierzy unimodularnej U, gdzie $\mathbf{B}$ jest macierza bazy dobrze uwarunkowanej, a U jest macierza unimodularna. Klucz publiczny obliczany jest jako iloczyn $\mathbf{A} = U \cdot \mathbf{B}$. Właśnie ta złożona struktura matematyczna sprawia, że odzyskanie klucza prywatnego z publicznego jest praktycznie niemożliwe.

8. Algorytm Babai'ego

Deszyfrowanie w kryptosystemie GGH wykorzystuje algorytm Babai'ego, który pozwala na przybliżenie danego punktu do najbliższego punktu kraty. Algorytm ten bazuje na zaokraglaniu współrzednych punktu w przestrzeni współrzednych kanonicznych bazy prywatnej, co pozwala na efektywne odzyskanie zaszyfrowanej wiadomości.

9. Generowanie macierzy unimodularnej

Macierz unimodularna to macierz kwadratowa o wyznaczniku równym ± 1. Generowanie takich macierzy jest kluczowe dla systemu GGH, ponieważ pozwala na przekształcenie bazy prywatnej w baze publiczna bez zmiany właściwości kraty. Macierze te sa generowane losowo, z zachowaniem własności odwracalności i struktur geometrycznych.

Atak na kryptosystem GGH

W tej sekcji przedstawiono główne podatności kryptosystemu GGH, narzedzia którymi można je wykorzystać, oraz realizacje takiego procesu.

1. Podatności kryptosystemu GGH

Kryptosystem GGH, oparty na teorii krat, wykazuje kilka podatności, które można wykorzystać w atakach krypto analitycznych. Kluczowe słabości wynikaja z faktu, że klucz publiczny jest baza nieortogonalna, czyli taka, której wektory bazowe nie sa wzajemnie prostopadłe i maja silne zależności liniowe , co umożliwia zastosowanie algorytmów redukcji bazy krat.

2. GSO (ang. Gram-Schmidt Orthogonalization)

Algorytm ortogonalizacji Gram-Schmidta (GSO) pozwala rozłożyć wektory bazy na cześci ortogonalne (czyli wektory składowe prostopadłe wzgledem siebie, co eliminuje zależności liniowe miedzy nimi), co stanowi podstawe wielu metod redukcji bazy krat. GSO wykorzystuje klasyczna metod ortogonalizacji wektorów w przestrzeni wektorowej:

- Każdy wektor bazy jest korygowany poprzez usuwanie składowych równoległych do wcześniejszych wektorów ortogonalnych.
- Proces ten tworzy zbiór wektorów , które sa ortogonalne, lecz moga mieć różne długości.

Proces ten tworzy zbiór wektorów , które sa ortogonalne, lecz moga mieć różne długości.Ortogonalizacja Gram-Schmidta umożliwia efektywna analize bazy i jest fundamentem dla algorytmu LLL.

3. Warunek Lovásza

Warunek Lovásza jest kluczowym elementem algorytmu LLL, który zapewnia jakość redukcji bazy, umożliwiajac uzyskanie krótszych i lepiej ułożonych wektorów. Jego matematyczna definicja brzmi:

$$\delta \cdot \|b_{i-1}^*\|^2 \leq \|b_i^* + \mu_{i-1,i} b_{i-1}^*\|^2$$

gdzie:

- δ to parametr kontrolujacy stopień redukcji,
- b_i^* i b_{i-1}^* to wektory ortogonalne uzyskane w procesie ortogonalizacji,
- $\mu_{i-1,i}$ to współczynnik korekcji, odpowiadajacy za projekcje wektora b_i^* na b_{i-1}^*

Interpretacja matematyczna:

Nierówność Lovásza wymaga, aby nowy wektor bazy był odpowiednio skrócony w stosunku do poprzednich wektorów. W praktyce oznacza to, że jeśli długość składowej ortogonalnej b_i^* jest zbyt duża, algorytm dokonuje zamiany wektorów w bazie, aby poprawić jej jakość.

4. Algorytm redukcji bazy krat LLL (ang. Lenstra-Lenstra-Lovász)

Algorytm LLL redukuje baze krat, przekształcajac ja w taki sposób, by wektory były krótsze i bardziej zbliżone do siebie ortogonalnie. Oto kluczowe kroki algorytmu:

- Baza jest najpierw przekształcana na cześci ortogonalne za pomoca GSO, aby zidentyfikować wektory składowe.
- Każdy wektor bazy jest korygowany poprzez zmniejszanie zależności od wcześniejszych wektorów.
- Jeśli długości wektorów nie spełniaja warunku Lovásza, nastepuje zamiana wektorów w bazie.
- Kroki sa powtarzane, aż wszystkie warunki zostana spełnione.

Algorytm LLL zapewnia redukcje w czasie wielomianowym, co czyni go wydajnym narzedziem do analizy baz krat.

5. Przebieg ataku przy pomocy algorytmu LLL

1. Cel ataku na kryptosystemu GGH

Celem ataku jest odzyskanie wiadomości m reprezentowany jako wektor liczb całkowitych, znajac klucz publiczny B i szyfrogram c. Proces ten można sprowadzić do problemu najbliższego wektora (NVP).

2. Redukcja klucza publicznego za pomoca algorytmy LLL

Klucz publiczny jest baza skomplikowane, co oznacza, że wektory bazy sa długie i skośne. Użycie algorytmu LLL umożliwia uzyskanie zredukowanej bazy L, w której:

- Wektory sa krótsze.
- Wektory sa bardziej quasi-ortogonalne

Algorytm LLL działa w nastepujacych krokach:

a) GSO

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

gdzie

- b_i^* to ortogonalny odpowiednik b_i,
- $\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}$ to współczynnik projekcji.

b) Sprawdzenie warunku Lovásza:

$$\delta \cdot \|b_{i-1}^*\|^2 \leq \|b_i^* + \mu_{i-1,i} b_{i-1}^*\|^2,$$

Nierówność musi być spełniona dla stałej $\delta = \frac{3}{4}$.. Jeśli nie jest, wektory sa zamieniane.

c) Powtarzanie redukcji

Proces powtarza sie, aż baza zostanie zredukowana do L.

3. Rozwiazanie problemu najbliższego wektora (NVP)

a) Projekcja szyfrogramu na baze ortogonalna

- Współczynniki r_i sa obliczane przez projekcje szyfrogramu c na baze ortogonalna b_i^*:

$$r_i = \frac{\langle c, b_i^* \rangle}{\langle b_i^*, b_i^* \rangle}.$$

- r_i jest zaokraglane do najbliższej liczby całkowitej:

b) Rekonstrukcja najbliższego wektora kraty

Zaokraglone współczynniki r_i sa używane do rekonstrukcji najbliższego wektora kraty:

$$v = \sum_{i=1}^{n} \text{round}(r_i) b_i.$$

4. Rekonstrukcja wiadomości Na podstawie najbliższego wektora v, wiadomość m jest odzyskiwana jako:

$$m = v \cdot L^{-1}.$$

Implementacja ataku na kryptosystem GGH

W tej sekcji przedstawiono implementacje łamania klucza publicznego kryptosystemu GGH z podziałem na trzy główne elementy: redukcja bazy krat, rozwiazanie problemu najbliższego wektora (NVP) oraz rekonstrukcja wiadomośc.

1. Redukcja bazy

Rozszerzamy macierz klucza publicznego o dodatkowy wymiar, w którym uwzgledniony jest szyfrogram , a nastepnie redukujemy baze za pomoca algorytmu LLL:

```
import numpy as np
from fpylll import IntegerMatrix, LLL

def extend_lattice(B, c, alpha):
    """
    Funkcja rozszerza krate, dodaj c wymiar dla szyfrogramu c.
    B: macierz klucza publicznego
    c: szyfrogram
    alpha: duza liczba wzmacniaj ca projekcje szyfrogramu
    """
    n = B.shape[0]
    extended_B = np.zeros((n + 1, n + 1))
    extended_B[:n, :n] = B
    extended_B[-1, :-1] = c
    extended_B[-1, -1] = alpha
    return extended_B

# Przykladowa macierz klucza publicznego
B = np.array([
    [7, 1, 1],
    [1, 5, 1],
    [1, 1, 3]
])

# Szyfrogram
c = np.array([12, 7, 5])

# Wartosc alpha
alpha = 1000

# Rozszerzenie kraty
B_ext = extend_lattice(B, c, alpha)
print("Rozszerzona krata:")
print(B_ext)

# Konwersja do formatu fpylll
B_ext_fpylll = IntegerMatrix.from_matrix(B_ext)

# Redukcja bazy
LLL.reduction(B_ext_fpylll)
print("Zredukowana baza (LLL):")
print(np.array(B_ext_fpylll))

# Najkrotszy wektor
shortest_vector = np.array(B_ext_fpylll[0])
print("Najkr tszy wektor:", shortest_vector)
```

Listing 4: Algorytm LLL

2. Rozwiazanie problemu najbliższego wektora (NVP)

Po redukcji bazy , najkrótszy wektor jest najbliższy szyfrogramowi w przestrzeni kraty. Rozwiazujemy problem najbliższego wektora za pomoca projekcji i zaokragleń:

```
# Wyodrebnienie oryginalnego wektora kratowego
v = shortest_vector[:-1]
print("Najblizszy␣wektor␣kraty:", v)
```

Listing 5: Rozwiazanie problemu najblizszego wektora (NVP)

3. Rekonstrukcja wiadomości

Odzyskanie wiadomości na podstawie zredukowanego najkrótszego wektora :

```
# Rekonstrukcja wiadomo ci
B_inv = np.linalg.inv(B)
m = np.dot(B_inv, v)
print("Odzyskana␣wiadomo  :", m)S
```

Listing 6: Rekonstrukcja wiadomości

Podsumowanie

Podstawy matematyczne kryptosystemu GGH obejmuja teorie krat, problemy NP-trudne (SVP i NVP), redukcje krat oraz narzedzia algebry liniowej, takie jak mnożenie macierzy i generowanie macierzy unimodularnych. Bezpieczeństwo systemu wynika z połaczenia tych zaawansowanych technik, co czyni go fascynujacym przykładem zastosowania matematyki w kryptografii.