УДК 621.391

М.О. Поліщук
І.А. Клєопа

МАТРИЧНИЙ ШИФР ГІЛЛА: МАТЕМАТИЧНА МОДЕЛЬ ТА РЕАЛІЗАЦІЯ В C++

Вінницький національний технічний університет

Аннотація

Звіт присвячено програмній реалізації алгоритму симетричного шифрування Гілла мовою C++ з використанням матриці ключа розміром 3 X 3 . У роботі детально проаналізовано математичний апарат методу, зокрема операції лінійної алгебри (матричне множення, обчислення детермінанта та побудова союзної матриці) та їх адаптацію для роботи в кільці лишків Z_{26}. Окрему увагу приділено вирішенню проблеми оборотності ключа шляхом реалізації алгоритму перевірки детермінанта на взаємну простоту з модулем 26, що гарантує коректність процесів шифрування та дешифрування даних.

Ключові слова: шифр Гілла, криптографія, лінійна алгебра, матричне множення, модульна арифметика, обернена матриця, детермінант, симетричне шифрування, мова програмування C++.

Abstract

The report focuses on the software implementation of the symmetric Hill cipher algorithm in C++ using a 3 X 3 key matrix. The work provides a detailed analysis of the mathematical apparatus of the method, specifically linear algebra operations (matrix multiplication, determinant calculation, and adjugate matrix construction) and their adaptation for the residue ring Z_{26}. Special attention is paid to solving the key invertibility problem by implementing an algorithm to check the determinant for coprimality with the modulus 26, ensuring the correctness of data encryption and decryption processes.

Keywords: Hill Cipher, cryptography, linear algebra, matrix multiplication, modular arithmetic, inverse matrix, determinant, symmetric encryption, C++ programming language.

Вступ

Шифр Гілла — це класичний метод поліграфічного шифрування, що базується на лінійній алгебрі та модульній арифметиці. На відміну від простих шифрів підстановки, він оперує блоками символів через множення на матрицю-ключ, що забезпечує вищу стійкість до злому.

Метою роботи є програмна реалізація алгоритму Гілла мовою **C++** для матриці розміром 3 X 3. Ключовими завданнями є організація обчислень у кільці лишків Z_{26} та вирішення проблеми генерації валідної матриці-ключа. Зокрема, реалізовано алгоритм перевірки детермінанта на взаємну простоту з модулем 26, що гарантує існування оберненої матриці для коректного дешифрування повідомлень.

Результати дослідження

1. Підключення бібліотек

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <cstdio>
#include <cmath>
```

Цей блок підключає заголовні файли стандартної бібліотеки C++, необхідні для роботи програми:

iostream: Забезпечує базове введення-виведення даних (потоки cin, cout, хоча в коді переважно використовується cstdio).

ctime: Містить функцію time(), яка використовується для ініціалізації генератора випадкових чисел поточним часом.

cstdlib: Надає функції rand() та srand() для генерації псевдовипадкових чисел, а також керування пам'яттю.

cstdio: Підключає функції в стилі C (printf, scanf_s) для форматованого виведення матриць та зчитування даних.

cmath: Містить математичні функції (наприклад, pow або abs), які можуть знадобитися для складних обчислень.

2. Допоміжна функція модуля **(mod26)**

```
int mod26(int a) {
	return (a % 26 + 26) % 26;
}
```

Стандартний оператор залишку від ділення % у C++ працює некоректно для від'ємних чисел у контексті криптографії (наприклад, -5 % 26 поверне -5). Ця функція реалізує математично правильну операцію модуля для кільця лишків Z_{26}. Вираз (a % 26 + 26) гарантує, що проміжний результат буде додатним, а фінальне % 26 залишає число в діапазоні [0, 25]. Це критично важливо при обчисленні оберненої матриці, де часто виникають від'ємні значення.

3. Пошук оберненого елемента **(modInverse)**

```
int modInverse(int n) {
	n = mod26(n);
	for (int x = 1; x < 26; x++)
		if (((n % 26) * (x % 26)) % 26 == 1)
			return x;
	return -1;
}
```

Функція шукає мультиплікативне обернене число для заданого числа n за модулем 26. Це таке число x, при множенні якого на n остача від ділення на 26 дорівнює 1. Алгоритм перебирає всі числа від 1 до 25. Якщо обернене число не знайдено (що означає, що вхідне число і 26 мають спільні дільники), функція повертає -1. Це сигнал, що поточна матриця не може бути використана для шифрування.

4. Генерація випадкових чисел та матриці

```
int randomI()
{
	int randomI = (rand() % 101) + 0;
	return randomI;
}

void MatrixGener(int** matr)
{
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			matr[i][j] = randomI();
		}
	}
}
```

Функція **randomI** генерує випадкове число від 0 до 100. Функція **MatrixGener** приймає вказівник на матрицю розміром 3 на 3 і заповнює кожну її клітинку цими випадковими значеннями. Це створює первинний варіант ключа шифрування.

5. Обчислення детермінанта

```
int CalculateDeterminant(int** keyMatr) {
    int det = keyMatr[0][0] * ((keyMatr[1][1] * keyMatr[2][2]) - (keyMatr[2][1] * keyMatr[1][2])) -
              keyMatr[0][1] * ((keyMatr[1][0] * keyMatr[2][2]) - (keyMatr[2][0] * keyMatr[1][2])) +
              keyMatr[0][2] * ((keyMatr[1][0] * keyMatr[2][1]) - (keyMatr[2][0] * keyMatr[1][1]));
    return mod26(det);
}
```

Ця функція розраховує визначник (детермінант) матриці 3 на 3 за стандартною математичною формулою розкладання за першим рядком. Отримане значення одразу приводиться до діапазону 0–25 за допомогою функції **mod26**. Значення детермінанта є критично важливим для перевірки того, чи можна створити обернену матрицю для розшифрування.

6. Функції виведення **(PrintMatrix)**

```
void PrintMatrix33(int** matr)
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            printf("%4d  ", matr[i][j]);
        }
        printf("\n\n\n");
    }
}
```

Це допоміжні функції, які виводять вміст матриці 3 на 3 або вектора (масиву з 3 елементів) у консоль у зручному табличному вигляді. Вони використовуються для візуалізації процесу шифрування на кожному етапі.

7. Шифрування (Кодування)

```
int* MatrixCode(int** matr, int* matrh, int* matrc)
{
    int asum = 0, bsum = 0, csum = 0;
    for (int i = 0; i < 3; i++) {
        asum += matr[0][i] * matrh[i];
    }
    for (int i = 0; i < 3; i++)
    {
        bsum += matr[1][i] * matrh[i];
    }
    for (int i = 0; i < 3; i++)
    {
        csum += matr[2][i] * matrh[i];
    }
    matrc[0] = asum;
    matrc[1] = bsum;
    matrc[2] = csum;
    return matrc;
```

```
}
```

8. Дешифрування (Декодування)

```
int* MatrixDecode(int** keyMatr, int* cipherVec, int* decodedVec)
{
	int det = CalculateDeterminant(keyMatr);
	int detInv = modInverse(det);

	if (detInv == -1) {
		printf("ERROR: Matrix determinant (%d) has no inverse mod 26.\n", det);
		return decodedVec;
	}

	int inv[3][3];
	inv[0][0] = mod26((keyMatr[1][1] * keyMatr[2][2] - keyMatr[2][1] * keyMatr[1][2]) * detInv);
	inv[0][1] = mod26((keyMatr[0][2] * keyMatr[2][1] - keyMatr[0][1] * keyMatr[2][2]) * detInv);
	inv[0][2] = mod26((keyMatr[0][1] * keyMatr[1][2] - keyMatr[0][2] * keyMatr[1][1]) * detInv);

	inv[1][0] = mod26((keyMatr[1][2] * keyMatr[2][0] - keyMatr[1][0] * keyMatr[2][2]) * detInv);
	inv[1][1] = mod26((keyMatr[0][0] * keyMatr[2][2] - keyMatr[0][2] * keyMatr[2][0]) * detInv);
	inv[1][2] = mod26((keyMatr[1][0] * keyMatr[0][2] - keyMatr[0][0] * keyMatr[1][2]) * detInv);

	inv[2][0] = mod26((keyMatr[1][0] * keyMatr[2][1] - keyMatr[2][0] * keyMatr[1][1]) * detInv);
	inv[2][1] = mod26((keyMatr[2][0] * keyMatr[0][1] - keyMatr[0][0] * keyMatr[2][1]) * detInv);
	inv[2][2] = mod26((keyMatr[0][0] * keyMatr[1][1] - keyMatr[1][0] * keyMatr[0][1]) * detInv);

	for (int i = 0; i < 3; i++) {
		int sum = 0;
		for (int j = 0; j < 3; j++) {
			sum += inv[i][j] * mod26(cipherVec[j]);
		}
		decodedVec[i] = mod26(sum);
	}
	return decodedVec;
}
```

Це найскладніша частина алгоритму, яка відновлює початковий текст. Процес складається з таких кроків:

1. Обчислюється детермінант матриці ключа.
2. Знаходиться обернене число до детермінанта. Якщо його немає — виводиться помилка.
3. Будується так звана "союзна матриця", що складається з алгебраїчних доповнень.
4. Кожен елемент цієї матриці множиться на обернений детермінант за модулем 26. Так утворюється обернена матриця.
5. Отримана обернена матриця множиться на зашифрований вектор, що дає початкові індекси літер.

Головна функція

```
int main()
{
	char a, b, c;
	int af, bf, cf;
	srand(time(NULL));
	int** Matr33 = new int* [3];
```

```
	for (int i = 0; i < 3; i++)
	{
		Matr33[i] = new int[3];
	}
	int det, detInv;
	int attempts = 0;
	do {
		MatrixGener(Matr33);
		det = CalculateDeterminant(Matr33);
		detInv = modInverse(det);
		attempts++;
	} while (detInv == -1);
	PrintMatrix33(Matr33);

	printf("Enter word ->");
	scanf_s("%c%c%c", &a, 1, &b, 1, &c, 1);
	printf("\n\n");
	af = a - 97;
	bf = b - 97;
	cf = c - 97;
	int* Matr13 = new int[3];
	Matr13[0] = af;
	Matr13[1] = bf;
	Matr13[2] = cf;
	PrintMatrix13(Matr13);
	int* Matr13c = new int[3];
	Matr13c = MatrixCode(Matr33, Matr13, Matr13c);
	printf("Coded word:\n");
	PrintMatrix13(Matr13c);
	printf("Decoded word (Hill Logic):\n");
	int* Matr13Decoded = new int[3];
	Matr13Decoded = MatrixDecode(Matr33, Matr13c, Matr13Decoded);
	PrintMatrix13(Matr13Decoded);
	printf("Decoded text: %c%c%c\n", Matr13Decoded[0] + 97, Matr13Decoded[1] + 97, Matr13Decoded[2]
+ 97);
	delete[] Matr13Decoded;
	for (int i = 0; i < 3; i++) delete[] Matr33[i];
	delete[] Matr33;
	system("pause");
}
```

9. Генерація валідного ключа (Цикл перевірки)

```
do {
	MatrixGener(Matr33);
	det = CalculateDeterminant(Matr33);
	detInv = modInverse(det);
	attempts++;
} while (detInv == -1);
```

Висновок

У ході виконання роботи було розроблено та протестовано програмну реалізацію поліграфічного шифру Гілла мовою C++ з використанням матриці ключа розміром 3 на 3.

Під час написання програми було вирішено кілька важливих завдань:

Математична адаптація: Реалізовано функції лінійної алгебри (множення матриць, обчислення детермінанта, знаходження союзної матриці) з урахуванням специфіки модульної арифметики для кільця цілих чисел за модулем 26.

Генерація валідного ключа: Найважливішим досягненням стала реалізація алгоритму автоматичної перевірки матриці на оборотність. Завдяки циклічній перевірці детермінанта на наявність оберненого числа, програма гарантовано створює ключ, який дозволяє успішно розшифрувати повідомлення, відсіюючи некоректні варіанти.

Робота з пам'яттю: Закріплено навички роботи з динамічним виділенням пам'яті для двовимірних масивів, що дозволяє програмі ефективно використовувати ресурси.

Результати тестування підтвердили, що розроблена програма коректно шифрує вхідний текст та безпомилково відновлює його після дешифрування, що доводить правильність обраних алгоритмів та математичних методів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Математичні методи криптології: Навчальний посібник [Електронний ресурс] (Для студентів техн. спец. вищ. навч. закл.) / [А.Д. Кожухівський, І.Д. Горбенко, Г.І. Гайдур, О.А. Кожухівська, В.В. Марченко]; М-во освіти і науки України, Державний університет телекомунікацій. Київ: ДУТ, 2021 – 244 с.
2. Ігнатович А.О. Критерій ефективності для визначення стійкості блокових шифрівна основі внесених змін статистичних характеристик шифрованого тексту / Ігнатович А.О., Глухова О.В., Лозинський А.Я., Яремчук Р.І. // АСІТ'5 "Сучасні комп'ютерні інформаційні технології". ТНЕУ. – Тернопіль. 22-23 травня 2015. – С. 167-168.

Поліщук Максим Олександрович – студентка групи 2БС – 25б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: 04-25-128.stud@vntu.edu.ua

Науковий керівник: ***Клєопа Ірина Анатоліївна*** – pHd, доцент, кафедра вищої математики, Вінницький національний технічний університет, м. Вінниця, Хмельницьке шосе, 95, e-mail: paceka08@vntu.edu.ua

Polishchuk Macksym Oleksandrovych – student of group 2SS – 25b, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: 04-25-128.stud@vntu.edu.ua

Scientific supervisor: ***Klieopa Iryna Anatoliivna*** – PhD, Associate Professor, Department of Higher Mathematics, Vinnytsia National Technical University, Vinnytsia, Khmelnytske Shosse, 95, e-mail: paceka08@vntu.edu.ua