

Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 1
по дисциплине «Криптографические методы защиты информации»
Построение криптографических операций в полях Галуа

Студент гр. БИБ232
Николаев Михаил
«5» Января 2025 г.

Руководитель
Заведующий кафедрой
информационной безопасности
киберфизических систем канд. техн.
наук, доцент
_____ О.О. Евсютин
« » 2025 г.

Москва 2025

Оглавление

1 ЦЕЛЬ РАБОТЫ.....	3
2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.....	3
2.1 Поля Галуа.....	3
2.2 Построение аффинного шифра над полем Галуа.....	4
3. Особенности программной реализации	5
4 Демонстрация работы программной реализации	10
5 Выводы о проделанной работе.....	14

1 ЦЕЛЬ РАБОТЫ

Целью данной работы является приобретение навыков программной реализации операций над многочленами в полях Галуа для построения криптографических преобразований.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Поля Галуа

Поле Галуа называется поле F_{p^n} , полученное расширением простого конечного поля F_p посредством неприводимого многочлена $f \in F_p[X]$ степени n . Мощность поля Галуа составляет p^n . Элементами поля Галуа являются многочлены, принадлежащие кольцу многочленов над полем F_p , степень которых строго меньше n . Принадлежность многочлена кольцу многочленов $F_p[X]$ означает, что коэффициенты при степенях данного многочлена являются элементами поля F_p . Таким образом, поле Галуа F_{p^n} состоит из всевозможных остатков от деления многочленов, заданных над полем F_p , на неприводимый многочлен $f \in F_p[X]$ степени n , и в общем случае может быть записано так: $F_{p^n} = \{0, 1, \dots, p-1, x, x+1, \dots, x+(p-1), \dots, (p-1)x^{n-1} + (p-1)x^{n-2} + \dots + (p-1)\}$.

В поле Галуа определены две операции: сложение и умножение.

Чтобы сложить два многочлена–элемента поля Галуа F_{p^n} , необходимо сложить их коэффициенты при соответствующих степенях и привести полученные значения по модулю p .

Чтобы перемножить два многочлена–элемента поля Галуа F_{p^n} , необходимо каждый член одного многочлена умножить на каждый член второго многочлена, привести подобные, привести коэффициенты при степенях полученного многочлена по модулю p и выполнить деление полученного многочлена, степень которого может быть выше $n-1$, на неприводимый многочлен $f \in F_p[X]$ степени n . Остаток от такого деления и будет представлять собой результат перемножения двух исходных элементов поля Галуа.

Известно, что мультипликативная группа $F_{p^n}^*$ поля Галуа F_{p^n} , включающая все ненулевые элементы поля, представляет собой циклическую группу порядка $q = p^n - 1$. Это означает, что каждый элемент данной группы может быть представлен в виде некоторой степени образующего элемента $\alpha \in F_{p^n}^*$: $F_{p^n}^* = \langle \alpha \rangle = \{1, \alpha, \alpha^2, \dots, \alpha^{q-1}\}$. Количество образующих элементов группы $F_{p^n}^*$ может быть определено через функцию Эйлера как $\varphi(q)$. Разложение элементов мультипликативной группы поля Галуа по степеням образующего позволяет реализовать операцию умножения многочленов в поле Галуа более простым образом. Пусть даны два многочлена $u, v \in F_{p^n}^* = \langle \alpha \rangle$, причем

известно, что $u = \alpha^k$, $v = \alpha^l$. Тогда умножение данных многочленов может быть выполнено по следующей формуле:

$$u \cdot v = \alpha^k \alpha^l = \alpha^{(k+l) \bmod q}. \quad (1)$$

2.2 Построение аффинного шифра над полем Галуа

Математический аппарат полей Галуа широко используется для конструирования криптографических операций в современных симметричных шифрах. В качестве основных примеров можно привести шифры AES и Кузнечик. В шифре AES один из этапов основного криптографического преобразования состоит в замене байтов мультипликативно обратными значениями в группе $F_{2^8}^*$. В шифре Кузнечик в качестве одного из базовых преобразований используется свертка 16-байтового слова в один байт посредством линейного преобразования в поле Галуа F_{2^8} .

Наиболее простым примером шифра, который может быть построен над полем Галуа, является аффинный шифр, основанный на так называемом аффинном преобразовании.

Пусть A – это алфавит, используемый для представления сообщений, подлежащих шифрованию. Символы данного алфавита представляются в виде элементов поля Галуа F_{p^n} одним из двух возможных способов:

- Параметры поля Галуа F_{p^n} выбираются таким образом, чтобы выполнялось равенство $|A| = p^n$. Это не всегда возможно для алфавитов естественных языков, поэтому можно работать с усеченным или расширенным алфавитом.

- Независимо от используемого алфавита естественного языка сообщение представляется в виде двоичной последовательности, которая разбивается на n -разрядные блоки. Отдельно взятый блок рассматривается как символ сообщения, подлежащий замене с помощью аффинного шифра. Алфавит, составленный из таких символов, будет иметь мощность 2^n , поэтому он может быть представлен в виде поля Галуа F_{2^n} .

Тогда открытый текст после перехода от исходного алфавита A к полю Галуа F_{p^n} может быть обозначен $x = (x_1, \dots, x_l)$, соответствующий шифртекст – $y = (y_1, \dots, y_l)$, где $x_i, y_i \in F_{p^n}$, $i = \overline{1, l}$. В качестве ключа аффинного шифра, построенного над полем Галуа F_{p^n} , выступает пара значений $k = (\alpha, \beta)$, $\alpha \in F_{p^n}^*$, $\beta \in F_{p^n}$, и ключевое пространство имеет вид $K = F_{p^n}^* \times F_{p^n}$.

Зашифрование отдельного символа открытого текста осуществляется по следующей формуле:

$$y_i = \alpha x_i + \beta, i = \overline{1, l}. \quad (2)$$

Расшифрование символа шифртекста осуществляется по формуле

$$x_i = (y_i - \beta)\alpha^{-1}, i = \overline{1, l}, \quad (3)$$

где $\alpha^{-1} \in F_{p^n}^*$ – элемент поля Галуа, мультипликативно обратный к α .

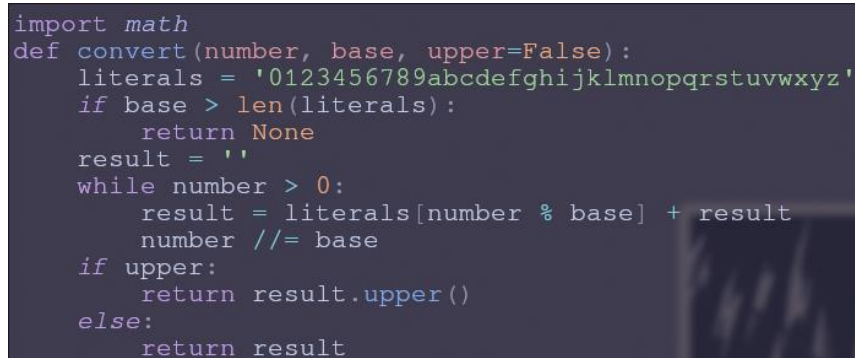
3. ОСОБЕННОСТИ ПОСТРОЕНИЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

Реализация полей Галуа и операций над ними, а также аффинного шифра, выполненная на языке программирования Python, без использования компонентов библиотек.

Одной из особенностей данной реализации является необходимость учета модификаций алфавитов для составления языков, которые могут быть либо сокращенными, либо расширенными. Это важно про

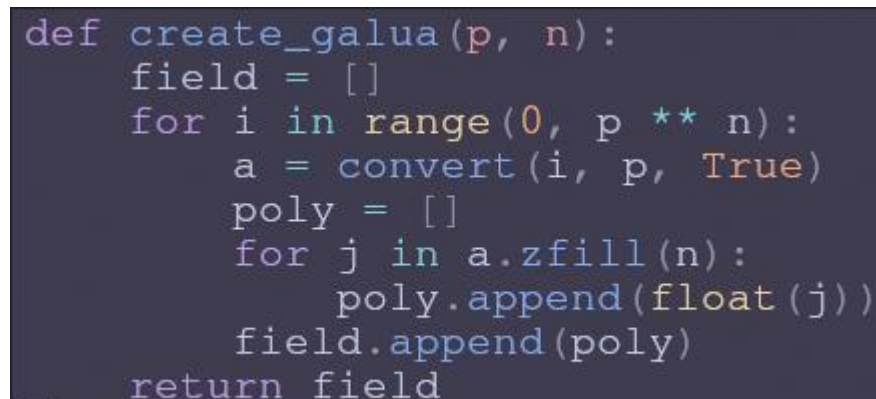
Код можно условно разделить на две основные

1. Поля Галуа и операции с ними
2. Аффинный шифр: процесс шифрования и дешифрования



```
import math
def convert(number, base, upper=False):
    literals = '0123456789abcdefghijklmnopqrstuvwxyz'
    if base > len(literals):
        return None
    result = ''
    while number > 0:
        result = literals[number % base] + result
        number //= base
    if upper:
        return result.upper()
    else:
        return result
```

Рисунок 1 – Программная функция производства, которая преобразует число в многочлен.



```
def create_galua(p, n):
    field = []
    for i in range(0, p ** n):
        a = convert(i, p, True)
        poly = []
        for j in a.zfill(n):
            poly.append(float(j))
        field.append(poly)
    return field
```

Рисунок 2 – Код функции, реализующей создание поля Галуа.

```
def nepr(p, n):
    b = []
    for i in range(p**n, p**(n+1)):
        a = convert(i, p, True)
        poly = []
        flag = 0
        for j in a:
            poly.append(int(j))
        for k in range(p):
            sum = 0
            for l in range(len(poly)):
                sum = sum + poly[l] * (k**(n+1-l-1))
            if (sum % p == 0):
                flag = 1
        if (flag == 0):
            b.append(poly)
    if (len(b) != 0):
        return b
    else:
        return 0
```

Рисунок 3 – Код функции, реализующей поиск неприводимого многочлена, подстановку многочленов, инверсию многочлена и массивов многочленов

```
def poly_sum(a, b, n):
    zip_sum = zip(a, b)
    l = list(map(sum, zip_sum))
    result = []
    for i in range(len(l)):
        x = l[i]
        result.append(x*n)
    return result
```

Рисунок 4 – Код функции, реализующей сложение многочленов.

```
def poly_mult(a, b, p, polynom):
    result = [0] * (len(a) + len(b) - 1)
    for i in range(len(a)):
        for j in range(len(b)):
            result[i+j] += a[i] * b[j]
    for i in range(len(result)):
        result[i] %= p
    result = div_poly(result, polynom)
    result = remove_trailing_zeros(result)
    while len(result) < len(a):
        result.append(0)
    for i in range(len(result)):
        result[i] = int(result[i]) % p
    return result
```

Рисунок 5 – Код функции, реализующей умножение многочленов.

```

def find_multip(galua, p, n, neprevod):
    b = []
    for i in range(1, len(galua)):
        if galua[i][1:] == [0]*(len(galua[i])-1) and galua[i][0]==1:
            continue
        flag = 0
        proizved = galua[i]
        st = 1
        stepeni = []
        stepeni.append((proizved, st))
        while flag==0:
            proizved = poly_mult(proizved, galua[i], p, neprevod)
            st += 1
            stepeni.append((proizved, st))
            if proizved[1:] == [0]*(len(proizved)-1) and proizved[0]==1:
                flag = 1
        if st == p**n - 1:
            b.append((galua[i], stepeni))
    return b

```

Рисунок 6 – Код функции, реализующей нахождение образующих элементов мультипликативной группы.

```

def div_poly(a, b, p):
    c1 = []
    c2 = []
    for i in a:
        c1.append(i)
    for i in b:
        c2.append(i)
    c1 = remove_trailing_zeros(c1)
    c2 = remove_trailing_zeros(c2)
    lc1 = len(c1)
    lc2 = len(c2)

    if b == [0] * len(b):
        raise ValueError("Division by zero")
    elif lc1 < lc2:
        return a
    elif lc1 == lc2 and c1[-1] < c2[-1]:
        c1, c2 = c2, c1
    while len(c1) >= len(c2):
        k = c1[-1]/c2[-1]
        i = len(c1)-1
        j = len(c2) - 1
        while i >= 0 and j >= 0:
            c1[i] -= c2[j] * k
            i -= 1
            j -= 1
        c1 = remove_trailing_zeros(c1)
    while len(c1) != len(c2):
        c1.append(0)
    for i in range(len(c1)):
        while c1[i] < 0:
            c1[i] += p
        if c1[i] % 1 != 0:
            c1[i] = frac_module(p, c1[i])
    return c1

```

Рисунок 7 – Функция «Деление полиномов», реализующая нахождение остатка от деления двух многочленов.

```

def alpha_in(stepen):
    print('Ввод альфа: Введите', steppen, 'коэффициентов, начиная с свободного члена')
    key_alpha = []
    for i in range(stepen):
        s = -1
        while s < 0 or s >= steppen:
            s = float(input())
            if s < 0 or s >= steppen:
                print('Попробуйте ввести число еще раз')
        key_alpha.insert(0, s)
    if (key_alpha == [0]*len(key_alpha)):
        return ValueError("error error ошибка ОШИБКА")
    return key_alpha

```

Рисунок 8 – Код функции, реализующей проверку значения альфа для ключа шифрования/расшифрования.


```
def opposite(field, k, nepriv, p):
    one = [0] * len(field[0])
    one[0] = 1
    for i in range(1, len(field)):
        if poly_mult(field[k], field[i], p, nepriv) == one:
            return field[i]
```

Рисунок 9 – Код функции, реализующей нахождение обратного многочлена.

```
def beta_in(stepen):
    print('Ввод бета: Введите ', stepen, 'коэффициентов начиная со свободного члена')
    key_beta = []
    for i in range(stepen):
        s = -1
        while s < 0 or s >= stepen:
            s = float(input())
            if s < 0 or s >= stepen:
                print('Попробуйте ввести число еще раз!')
        key_beta.insert(0, s)
    return key_beta
```

Рисунок 10 – Код функции, реализующей проверку значения бета для ключа шифрования/расшифрования.

```
def affine_shifr(message, alphabet, stepen, key_alpha, key_beta, neprevodim):
    shift_text = ''
    for i in range(0, len(message)):
        if message[i] not in (list(alphabet)):
            shift_text += str(message[i])
        else:
            for j in range(0, len(alphabet)):
                if message[i] == alphabet[j]:
                    буква = convert(j, 2)
                    буква_galua = []
                    for j in буква.zfill(stepen):
                        буква_galua.insert(0, float(j))
                    proizved = poly_mult(буква_galua, key_alpha, 2, neprevodim)
                    sum = poly_sum(proizved, key_beta, 2)
                    index = 0
                    for i in range(len(sum)):
                        index = int(index + 2**i * sum[int(i)])
                    shift_text += alphabet[index%len(alphabet)]
                    break
    return shift_text
```

Рисунок 11 – Код функции, реализующей шифрование текста аффинным шифром над полем Галуа.



```
def affine_shifr_decode(message, alphabet, stepen, key_alpha, key_beta, neprevodim):
    shifr_text = ''

    for i in range(0, len(message)):
        if message[i] not in (list(alphabet)):
            shifr_text += message[i]
        else:
            for j in range(0, len(alphabet)):
                if message[i] == alphabet[j]:
                    bukva = convert(j, 2)
                    bukva_galua = []
                    for k in bukva.zfill(stepen):
                        bukva_galua.insert(0, float(k))

                    minus_beta = key_beta
                    sum = poly_sum(bukva_galua, minus_beta, 2)
                    galua = create_galua(2, stepen)

                    obrat_alpha = opposite(galua, galua.index(key_alpha), neprevodim, 2)
                    prozved = poly_mult(sum, obrat_alpha, 2, neprevodim)

                    index = 0
                    for i in range(len(prozved)):
                        index = int(index + (2 ** i) * prozved[int(i)])
                    shifr_text += alphabet[index % len(alphabet)]
                    break

    return shifr_text
```

Рисунок 12 – Код функции, реализующей дешифрование текста аффинным шифром над полем Галуа.

4 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

При запуске программы необходимо выбрать действие, которое необходимо выполнить с помощью кода: **зашифровать** или **дешифровать сообщение** .

Алфавит для шифрования и дешифрования следует выбирать таким образом, чтобы его размер был равен двойному размеру. В противном случае процесс шифрования может оказаться некорректным.

В качестве входного текста для использования использовалась строка: «**alaska young**» .

Процесс включает в себя следующие этапы:

1. Выбор действия (шифрование или дешифрование).
2. Задание алфавита (в данном случае — алфавит размером 32 символа).
3. Выполнение шифрования или дешифрования текста с использованием двух алфавитов и заданных параметров (например, ключей шифрования).

Результаты работы программы отображают преобразованный текст (зашифрованный или расшифрованный), что позволяет оценить корректность реализации.

```

Хотите поработать с полем – press [y]
Хотите поработать с шифрованием – press [n]
n
Работа с шифром
Введите сообщение
alaska young
Ввод альфа: Введите 5 коэффициентов, начиная с свободного члена
1
1
0
0
0
Ввод бета: Введите 5 коэффициентов начиная со свободного члена
1
1
1
1
1
1
Все возможные неприводимые элементы:
1 [1, 0, 0, 0, 1, 1]
2 [1, 0, 0, 1, 0, 1]
3 [1, 0, 1, 0, 0, 1]
4 [1, 0, 1, 1, 1, 1]
5 [1, 1, 0, 0, 0, 1]
6 [1, 1, 0, 1, 1, 1]
7 [1, 1, 1, 0, 1, 1]
8 [1, 1, 1, 1, 0, 1]
Введите номер элемента с которым будем работать
2
Что будем делать?
Шифровать – press [y]
Дешифровать – press [n]
y
Зашифрованное сообщение:
xkxrcx pnidv

```

Рисунок 13 – Результат выполнения программы шифрования с использованием аффинного шифра над полем Галуа.

```

Хотите поработать с полем – press [y]
Хотите поработать с шифрованием – press [n]
n
Работа с шифром
Введите сообщение
xkxrcx pnidv
Ввод альфа: Введите 5 коэффициентов, начиная с свободного члена
1
1
0
0
0
Ввод бета: Введите 5 коэффициентов начиная со свободного члена
1
1
1
1
1
1
Все возможные неприводимые элементы:
1 [1, 0, 0, 0, 1, 1]
2 [1, 0, 0, 1, 0, 1]
3 [1, 0, 1, 0, 0, 1]
4 [1, 0, 1, 1, 1, 1]
5 [1, 1, 0, 0, 0, 1]
6 [1, 1, 0, 1, 1, 1]
7 [1, 1, 1, 0, 1, 1]
8 [1, 1, 1, 1, 0, 1]
Введите номер элемента с которым будем работать
2
Что будем делать?
Шифровать – press [y]
Дешифровать – press [n]
n
Дешифрованное сообщение:
alaska young

```

Рисунок 14 – Результат выполнения программы дешифрования с использованием аффинного шифра над полем Галуа.
Полученный результат подтверждает корректность работы программы.

Следующим заданием надо было принимать на вход значения p и n , определяющие поле Галуа, и строить и отображать соответствующее поле Галуа F_{p^n} .

Были выбраны такие параметры, как $p = 3$, $n = 2$, значит $|A| = 3^2$. Параметры можно менять в коде.

The image shows a terminal window with a Python script named 'crypt1.py' being executed. The script prompts the user to choose between working with a field or encryption. The user selects 'y' for field work. It then asks for the field size parameters p and n, which are entered as 3 and 2 respectively. The script calculates the Galois field F_{3^2} and lists all 9 non-zero elements. The user selects element 2, which is [1, 1, 2]. The script then lists the powers of this element, showing they form a cyclic group of order 8.

```
> python crypt1.py
Хотите поработать с полем - press [y]
Хотите поработать с шифрованием - press [n]
y
Все многочлены представлены в виде [число, x, x^2, x^3...]
Задайте размеры Поля Галуа
Последовательно введите параметры p и n
3
2
Поле Галуа::
[[0.0, 0.0], [0.0, 1.0], [0.0, 2.0], [1.0, 0.0], [1.0, 1.0], [1.0, 2.0], [2.0, 0.0], [2.0, 1.0], [2.0, 2.0]]
Все возможные неприводимые элементы:
1 [1, 0, 1]
2 [1, 1, 2]
3 [1, 2, 2]
4 [2, 0, 2]
5 [2, 1, 1]
6 [2, 2, 1]
Введите номер элемента с которым будем работать
2
Используемый неприводимый элемент поля: [1, 1, 2]
Выберите образующий группы:
1 [0.0, 1.0]
2 [0.0, 2.0]
3 [1.0, 2.0]
4 [2.0, 1.0]
3
Степень 1 [1.0, 2.0]
Степень 2 [2, 2]
Степень 3 [0, 1]
Степень 4 [2, 0]
Степень 5 [2, 1]
Степень 6 [1, 1]
Степень 7 [0, 2]
Степень 8 [1, 0]
```

Рис. 15 - Результат работы с полем Галуа

Можно увидеть, что, действительно, поле Галуа состоит из 9 вышепоказанных на рисунке 5 многочленов.


```

3 [1, 2, 2]
4 [2, 0, 2]
5 [2, 1, 1]
6 [2, 2, 1]
Введите номер элемента с которым будем работать
2
Используемый неприводимый элемент поля: [1, 1, 2]
Выберите образующий группы:
1 [0.0, 1.0]
2 [0.0, 2.0]
3 [1.0, 2.0]
4 [2.0, 1.0]
3
Степень 1 [1.0, 2.0]
Степень 2 [2, 2]
Степень 3 [0, 1]
Степень 4 [2, 0]
Степень 5 [2, 1]
Степень 6 [1, 1]
Степень 7 [0, 2]
Степень 8 [1, 0]
Если вы хотите выполнить действия с многочленами - press [y]
Если больше задач нет - press [n]
y
Введите два многочлена 1+ x +2x^2 + 3 x^3 ... в виде 123...
Обратите внимание длина многочлена - 2
1
4
Если хотите выполнить сложение - press [y]
Умножение - press [n]
y
Результат сложения: [2]
Если вы хотите выполнить действия с многочленами - press [y]
Если больше задач нет - press [n]

```

Рисунок 16– Результат работы кода для сложения двух полиномов

```

Если вы хотите выполнить действия с многочленами - press [y]
Если больше задач нет - press [n]
y
Введите два многочлена 1+ x +2x^2 + 3 x^3 ... в виде 123...
Обратите внимание длина многочлена - 2
2
3
Если хотите выполнить сложение - press [y]
Умножение - press [n]
n
Результат умножения: [0]
Если вы хотите выполнить действия с многочленами - press [y]
Если больше задач нет - press [n]

```

Рисунок 17– Результат работы кода для умножения двух полиномов

5 ВЫВОДЫ О ПРОДЕЛАННОЙ РАБОТЕ

В рамках практической работы были введены такие понятия, как “поле Галуа”, “аффинный шифр над полем Галуа”, рассмотрены их особенности. Были программно реализованы операции над многочленами в полях Галуа для построения криптографических преобразований. Также были реализованы шифрование и дешифрование текста с помощью аффинного шифра над полем Галуа. Если сравнивать его с обычным аффинным шифром, который мы выполняли в прошлом году, такой вариант кажется более надежным, однако также одинаковые символы он шифрует одинаково, что наделяет его теми же уязвимостями, что и обычный аффинный шифр.