

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 ПОСТРОЕНИЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ.....	5
4 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММНОЙ РЕАЛИЗАЦИИ.....	15
5 ВЫВОДЫ О ПРОДЕЛАННОЙ РАБОТЕ.....	18
6 ИТОГОВЫЙ КОД.....	19

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 $F_{p^n}^\times$ 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 $a \in F_{p^n}^\times$: $F_{p^n}^\times = \langle a \rangle = \{1, a, a^2, \dots, a^{q-1}\}$. Количество образующих элементов группы $F_{p^n}^\times$ 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 $u, v \in F_{p^n}^i = \langle \alpha \rangle$, 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 $\alpha^{-1} \in F_p^*$ – элемент поля Галуа, мультипликативно обратный к α .

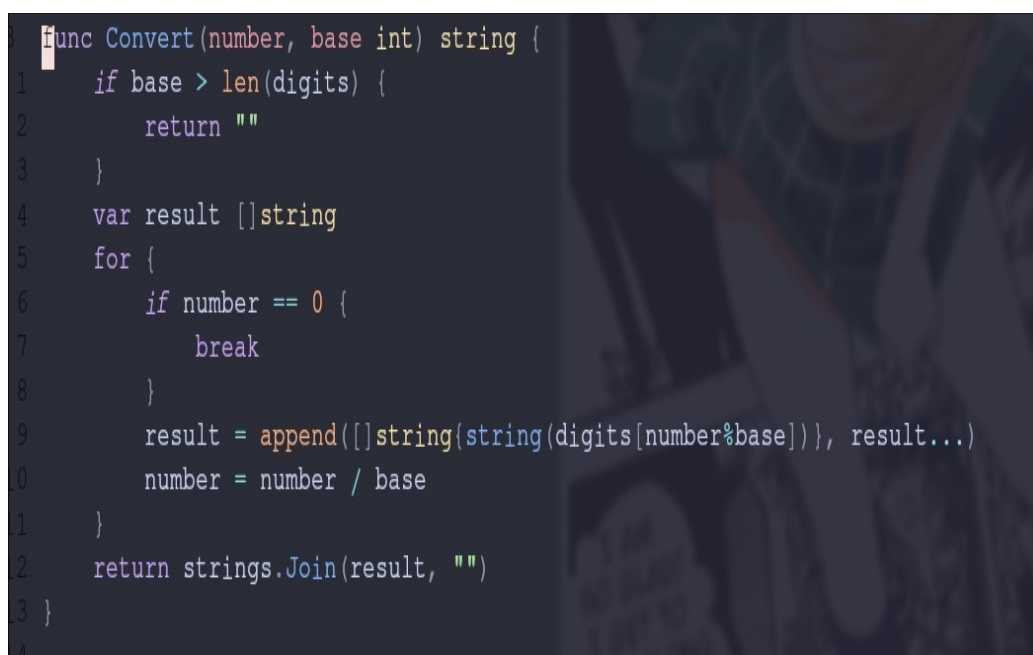
3 ОСОБЕННОСТИ ПОСТРОЕНИЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

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.

Код можно условно разделить на две основные

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



```
func Convert(number, base int) string {
1   if base > len(digits) {
2       return ""
3   }
4   var result []string
5   for {
6       if number == 0 {
7           break
8       }
9       result = append([]string{string(digits[number%base])}, result...)
0       number = number / base
1   }
2   return strings.Join(result, "")
3 }
```

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

```
func CreateGalua(p, n int) [][]int {  
    field := [][]int{}  
    for i := 0; i < Power(p, n); i++ {  
        a := Convert(i, p)  
        poly := []int{}  
        zeroa := ZeroFill(a, n)  
        zeroa = ZeroFill(zeroa, n)  
        zeroa = ZeroFill(zeroa, n)  
        for _, z := range zeroa {  
            poly = append(poly, int(z-'0'))  
        }  
        field = append(field, poly)  
    }  
    return field  
}
```

Рисунок 2 – Код функции, реализующей создание поля Галуа.

```

func Nprevodim(p, n int) ([][]int, error) {
    b := [][]int{}
    for i := Power(p, n); i < Power(p, n+1); i++ {
        a := Convert(i, p)
        flag := 0
        poly := []int{}
        for _, j := range a {
            poly = append(poly, int(j-'0'))
        }
        for j := 0; j < p; j++ {
            sum := 0
            for l := 0; l < len(poly); l++ {
                sum = sum + poly[l]*(Power(j, n+1-l-1))
            }
            if sum%p == 0 {
                flag = 1
            }
        }

        if n%2 == 0 {
            galua := CreateGalua(p, n)
            galual := galua[1:]
            for _, j := range galual {
                if j[1:] == nil {
                    continue
                }
                res, err := PolyDiv(poly, j, p)
                if res == nil {
                    flag = 1
                }
                if err != nil {
                    return b, errors.New("Нет неприводимых значений")
                }
            }
        }
        if flag == 0 {
            b = append(b, poly)
        }
    }
    if len(b) != 0 {
        return b, nil
    }
    return b, errors.New("Нет неприводимых значений")
}

```

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

```
func PolySum(a, b []int, n int) []int {
    m := min(len(a), len(b))
    res := []int{}
    for i := 0; i < m; i++ {
        x := (a[i] + b[i]) % n
        res = append(res, x)
    }
    return res
}
```

Рисунок 4 – Код функции, реализующей сложение многочленов.

```
func PolyMult(a, b []int, p int, nepr []int) []int {
    result := make([]int, len(a)+len(b)-1)
    for i := range a {
        for j := range b {
            result[i+j] += a[i] * b[j]
        }
    }
    for i := range result {
        result[i] %= p
    }
    result, err := PolyDiv(result, nepr, p)
    if err != nil {
        panic(err)
    }
    result = RemoveZeroes(result)
    for {
        if len(result) >= len(a) {
            break
        }
        result = append(result, 0)
    }
    for i := range result {
        result[i] %= p
    }
    return result
}
```

Рисунок 5 – Код функции, реализующей умножение многочленов.


```

func FindMultip(galua [][]int, p, n int, neprevod []int) [][]interface{} {
    b := [][]interface{}{}

    for i := 1; i < len(galua); i++ {
        if len(galua[i]) > 1 && AllZeroes(galua[i][1:]) && galua[i][0] == 1 {
            continue
        }

        flag := false
        proizved := append([]int(nil), galua[i]...)
        st := 1
        steps := [][]interface{}{{proizved, st}}

        for !flag {
            proizved = PolyMult(proizved, galua[i], p, neprevod)
            st++
            steps = append(steps, []interface{}{proizved, st})

            if len(proizved) > 1 && AllZeroes(proizved[1:]) && proizved[0] == 1 {
                flag = true
            }
        }

        if st == (Power(p, n) - 1) {
            b = append(b, []interface{}{galua[i], steps})
        }
    }

    return b
}

```

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

```

func PolyDiv(a, b []int, p int) ([]int, error) {

    c1 := []int{}
    c2 := []int{}
    for _, i := range a {
        c1 = append(c1, i)
    }
    for _, i := range b {
        c2 = append(c2, i)
    }
    c1 = RemoveZeroes(c1)
    c2 = RemoveZeroes(c2)
    if b == nil {
        return nil, errors.New("Division by zero")
    }
    if len(c1) < len(c2) {
        return a, nil
    }

    if len(c1) == len(c2) && c1[len(c1)-1] < c2[len(c2)-1] {
        c1, c2 = c2, c1
    }

    for len(c1) >= len(c2) {
        i := len(c1) - 1
        j := len(c2) - 1
        k := c1[i] / c2[j]
        for i >= 0 && j >= 0 {
            c1[i] -= c2[j] * k
            i--
            j--
        }
        c1 = RemoveZeroes(c1)
    }
    for len(c1) != len(c2) {
        c1 = append(c1, 0)
    }

    for i := range c1 {
        for c1[i] < 0 {
            c1[i] += p
        }
    }
    return c1, nil
}

```

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

```

func Alpha(stepen int) []int {
    fmt.Println("Ввод альфа: введите", stepen, "коэффициентов, начиная с свободного члена")
    key_alpha := []int{}
    for i := 0; i < stepen; i++ {
        s := -1
        for s < 0 || s >= stepen {
            r := bufio.NewReader(os.Stdin)
            char1, _, err := r.ReadRune()
            if err != nil {
                fmt.Println("i have a problem with your input")
            }
            s = int(char1 - '0')
            if s < 0 || s >= stepen {
                fmt.Println("try to enter again", s)
            }
        }
        if len(key_alpha) > 0 {
            key_alpha = slices.Insert(key_alpha, 0, s)
        } else {
            key_alpha = append(key_alpha, s)
        }
    }
    if key_alpha == nil {
        panic("Error ERROR ОШИБКА ОШИБКА")
    }
    return key_alpha
}

```

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

```

func Opposite(galua [][]int, k int, nepr []int, p int) []int {
    for i := 1; i < len(galua); i++ {
        temp := PolyMult(galua[k], galua[i], p, nepr)

        one := make([]int, len(temp))
        one[0] = 1
        if SliceEq(temp, one) {
            return galua[i]
        }
    }
    return nil
}

```

Рисунок 9 – Код функции, реализующей нахождение обратного многочлена.

```

func Beta(stepen int) []int {
    fmt.Println("Ввод бета: введите", stepen, "коэффициентов, начиная с свободного члена")
    key_beta := []int{}
    for i := 0; i < stepen; i++ {
        s := -1
        for s < 0 || s >= stepen {
            r := bufio.NewReader(os.Stdin)
            char1, err := r.ReadByte()
            if err != nil {
                fmt.Println("i have a problem with your input")
            }
            s = int(char1 - '0')
            if s < 0 || s >= stepen {
                fmt.Println("try to enter again", s)
            }
        }
        if len(key_beta) > 0 {
            key_beta = slices.Insert(key_beta, 0, s)
        } else {
            key_beta = append(key_beta, s)
        }
    }
    if key_beta == nil {
        panic("Error ERROR ОШИБКА ОШИБКА")
    }
    return key_beta
}

```

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

```

func AffineEncode(message,
    alphabet string,
    stepen int,
    key_alpha, key_beta []int,
    nepr []int) string {
    shifr := []string{}
    alph := strings.Split(alphabet, "")

    for i := 0; i < len(message); i++ {
        if !strings.Contains(alphabet, string(message[i])) {
            shifr = append(shifr, string(message[i]))
        } else {
            for k, j := range alphabet {
                if string(message[i]) == string(j) {
                    буква := Convert(k, 2)
                    буква_galua := []int{}
                    temp := ZeroFill(буква, stepen)
                    for _, z := range temp {
                        if len(буква_galua) > 0 {
                            буква_galua = slices.Insert(буква_galua, 0, int(z-'0'))
                        } else {
                            буква_galua = append(буква_galua, int(z-'0'))
                        }
                    }
                    multiplication := PolyMult(буква_galua, key_alpha, 2, nepr)
                    sum := PolySum(multiplication, key_beta, 2)
                    index := 0
                    for il, _ := range sum {
                        ■ simplify range expression
                        index = index + Power(2, il)*sum[il]
                    }
                    shifr = append(shifr, alph[index%len(alphabet)])
                    break
                }
            }
        }
    }

    return strings.Join(shifr, "")
}

```

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

```

func AffineDecode(message,
    alphabet string,
    stepen int,
    key_alpha, key_beta []int,
    nepr []int) string {

    shifr := []string{}
    alph := strings.Split(alphabet, "")

    for i := 0; i < len(message); i++ {
        if !strings.Contains(alphabet, string(message[i])) {
            shifr = append(shifr, string(message[i]))
        } else {
            for k, j := range alphabet {
                if string(message[i]) == string(j) {
                    bukva := Convert(k, 2)
                    bukva_galua := []int{}
                    temp := ZeroFill(bukva, stepen)
                    temp = ZeroFill(temp, stepen)
                    for _, z := range temp {
                        if len(bukva_galua) > 0 {
                            bukva_galua = slices.Insert(bukva_galua, 0, int(z-'0'))
                        } else {
                            bukva_galua = append(bukva_galua, int(z-'0'))
                        }
                    }
                    sum := PolySum(bukva_galua, key_beta, 2)
                    galua := CreateGalua(2, stepen)

                    galuaindex := FindSlice(key_alpha, galua)
                    if galuaindex == -1 {
                        panic("WE HAVE A PROBLEM")
                    }
                    oppositeAlpha := Opposite(galua, galuaindex, nepr, 2)

                    multiplication := PolyMult(sum, oppositeAlpha, 2, nepr)
                    index := 0
                    for i2 := range multiplication {
                        index = int(index + Power(2, i2)*multiplication[i2])
                    }
                    shifr = append(shifr, alph[index%len(alphabet)])
                    break
                }
            }
        }
    }

    return strings.Join(shifr, "")
}

```

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.


```
→ go run .
Введите "y", если надо поработать с полем и "n", если с шифром
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 с свободного члена
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
Сначала шифруем
Зашифрованное сообщение 6y6sa6 3dhfi
Теперь расшифруем
Расшифрованное сообщение alaska young
```

Рисунок 13 – Результат выполнения программы шифрования 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.


```

hse/crypt/prakt1 on | main [?] via 🐛 v1.23.5 via 🐛 v3.13.1 took 4s
) go run .
Введите "y", если надо поработать с полем и "n", если с шифром
y
Все многочлены представлены в виде [число, x, x^2, x^3...]
Введите через пробел числа p, n
3 2
Размеры поля: p = 3 n = 2
Поле Галуа
[[0 0] [0 1] [0 2] [1 0] [1 1] [1 2] [2 0] [2 1] [2 2]]
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
1
Вами выбран неприводимый элемент [1 0 1]
Образующие группы
1 [1 1]
2 [1 2]
3 [2 1]
4 [2 2]
Введите номер элемента с которым будем работать
2
Выбран [2 1]
Степень 1 [2 1]
Степень 2 [0 1]
Степень 3 [2 2]
Степень 4 [2 0]
Степень 5 [1 2]
Степень 6 [0 2]
Степень 7 [1 1]
Степень 8 [1 0]
Если вы хотите выполнить действия с многочленами - press [y]
Если больше задач нет - press [n]
Программа выполнена успешно

```

Рисунок 14 - Результат работы с полем Галуа

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.

6 ИТОГОВЫЙ КОД

```
package main
```

```
import (
```

```
    "bufio"
```

```
    "errors"
```

```
    "fmt"
```

```
    "math"
```

```
    "os"
```

```
    "slices"
```

```
    "strings"
```

```
)
```

```
var digits string = "0123456789abcdefghijklmnopqrstuvwxyz"
```

```
func SliceEq(a, b []int) bool {
```

```
    if len(a) != len(b) {
```

```
        return false
```

```
    }
```

```
    for i := range a {
```

```
        if a[i] != b[i] {
```

```
            return false
```

```
        }
```

```
    }  
  
    return true  
}  
  
func FindSlice(a []int, b [][]int) int {  
    for i, j := range b {  
        if SliceEq(a, j) {  
            return i  
        }  
    }  
  
    return -1  
}
```

```
func modInverse(a, p int) int {  
    t, newT := 0, 1  
    r, newR := p, a  
    for newR != 0 {  
        quotient := r / newR  
        t, newT = newT, t-quotient*newT  
        r, newR = newR, r-quotient*newR  
    }  
  
    if r > 1 {
```

```
        panic("Элемент не обратим")

    }

    if t < 0 {

        t += p

    }

    return t

}
```

```
func AsIntegerRatio(num float64) (int, int) {

    a, b := num, 1.0

    for a != math.Floor(a) {

        a *= 10

        b *= 10

    }

    return int(a), int(b)

}
```

```
func AllZeroes(arr []int) bool {

    for _, v := range arr {

        if v != 0 {

            return false

        }

    }

}
```

```
    }  
  
    return true  
}
```

```
func Power(a, b int) int {  
  
    res := 1  
  
    for i := 0; i < b; i++ {  
  
        res = res * a  
  
    }  
  
    return res  
}
```

```
func TrimLeftChar(s string) string {  
  
    for i := range s {  
  
        if i > 0 {  
  
            return s[i:]  
  
        }  
  
    }  
  
    return s[:0]  
}
```

```
func RemoveZeroes(a []int) []int {
```

```
    if AllZeroes(a) {  
        return nil  
    }  
    for {  
        if a[len(a)-1] != 0 {  
            return a  
        }  
        a = a[:len(a)-1]  
    }  
}
```

```
func Convert(number, base int) string {  
    if base > len(digits) {  
        return ""  
    }  
    var result []string  
    for {  
        if number == 0 {  
            break  
        }  
        result = append([]string{string(digits[number%base])}, result...)  
        number = number / base  
    }  
}
```

```
    }

    return strings.Join(result, "")
}

func ZeroFill(s string, n int) string {

    r := strings.Split(s, "")

    for i := 0; i < n-len(s); i++ {

        r = append([]string{"0"}, r...)

    }

    return strings.Join(r, "")
}
```

```
func CreateGalua(p, n int) [][]int {

    field := [][]int{}

    for i := 0; i < Power(p, n); i++ {

        a := Convert(i, p)

        poly := []int{}

        zeroa := ZeroFill(a, n)

        zeroa = ZeroFill(zeroa, n)

        zeroa = ZeroFill(zeroa, n)

        for _, z := range zeroa {
```

```

        poly = append(poly, int(z-'0'))

    }

    field = append(field, poly)

}

return field

}

```

```

func PolySum(a, b []int, n int) []int {

    m := min(len(a), len(b))

    res := []int{}

    for i := 0; i < m; i++ {

        x := (a[i] + b[i]) % n

        res = append(res, x)

    }

    return res

}

```

```

func Nprevodim(p, n int) ([][]int, error) {

    b := [][]int{}

    for i := Power(p, n); i < Power(p, n+1); i++ {

        a := Convert(i, p)

        flag := 0

```

```

poly := []int{}

for _, j := range a {

    poly = append(poly, int(j-'0'))

}

for j := 0; j < p; j++ {

    sum := 0

    for l := 0; l < len(poly); l++ {

        sum = sum + poly[l]*(Power(j, n+1-l-1))

    }

    if sum%p == 0 {

        flag = 1

    }

}

if n%2 == 0 {

    galua := CreateGalua(p, n)

    galua1 := galua[1:]

    for _, j := range galua1 {

```

```
        if j[1:] == nil {  
            continue  
        }  
  
        res, err := PolyDiv(poly, j, p)  
  
        if res == nil {  
            flag = 1  
        }  
  
        if err != nil {  
            return b, errors.New("Нет неприводимых значений")  
        }  
    }  
}  
  
if flag == 0 {  
    b = append(b, poly)  
}  
}  
  
if len(b) != 0 {  
    return b, nil  
}  
  
return b, errors.New("Нет неприводимых значений")  
}
```

```
func PolyMult(a, b []int, p int, nepr []int) []int {  
    result := make([]int, len(a)+len(b)-1)  
    for i := range a {  
        for j := range b {  
            result[i+j] += a[i] * b[j]  
        }  
    }  
    for i := range result {  
        result[i] %= p  
    }  
    result, err := PolyDiv(result, nepr, p)  
    if err != nil {  
        panic(err)  
    }  
    result = RemoveZeroes(result)  
    for {  
        if len(result) >= len(a) {  
            break  
        }  
        result = append(result, 0)  
    }  
}
```

```

    for i := range result {
        result[i] %= p
    }

    return result
}

func FindMultip(galua [][]int, p, n int, neprevod []int) [][]interface{} {
    b := [][]interface{} {}

    for i := 1; i < len(galua); i++ {
        if len(galua[i]) > 1 && AllZeroes(galua[i][1:]) && galua[i][0] == 1 {
            continue
        }

        flag := false

        proizved := append([]int(nil), galua[i]...)

        st := 1

        steps := [][]interface{} {{proizved, st}}

        for !flag {
            proizved = PolyMult(proizved, galua[i], p, neprevod)

            st++
        }
    }
}

```

```

        steps = append(steps, []interface{} {proizved, st})

        if len(proizved) > 1 && AllZeroes(proizved[1:]) && proizved[0] == 1 {

            flag = true

        }

    }

    if st == (Power(p, n) - 1) {

        b = append(b, []interface{} {galua[i], steps})

    }

}

return b

}

func FracModule(p int, num float64) float64 {

    a, b := AsIntegerRatio(num)

    a %= p

    if a < 0 {

        a += p

    }

    for i := 1; i < p; i++ {

```

```
        if (b*i)%p == 1 {  
            b = i  
            break  
        }  
    }  
    result := float64((a * b) % p)  
    if result < 0 {  
        result += float64(p)  
    }  
    return result  
}
```

```
func PolyDiv(a, b []int, p int) ([]int, error) {
```

```
    c1 := []int{}  
    c2 := []int{}  
    for _, i := range a {  
        c1 = append(c1, i)  
    }  
    for _, i := range b {
```

```

        c2 = append(c2, i)

    }

    c1 = RemoveZeroes(c1)

    c2 = RemoveZeroes(c2)

    if b == nil {

        return nil, errors.New("Division by zero")

    }

    if len(c1) < len(c2) {

        return a, nil

    }


    if len(c1) == len(c2) && c1[len(c1)-1] < c2[len(c2)-1] {

        c1, c2 = c2, c1

    }


    for len(c1) >= len(c2) {

        degDiff := len(c1) - len(c2)

        inv := modInverse(c2[len(c2)-1], p)

        k := (c1[len(c1)-1] * inv) % p

        for i := 0; i < len(c2); i++ {

            index := i + degDiff

            c1[index] = (c1[index] - k*c2[i]) % p

```

```

        if c1[index] < 0 {
            c1[index] += p
        }
    }

    c1 = RemoveZeroes(c1)
}

for len(c1) != len(c2) {
    c1 = append(c1, 0)
}

for i := range c1 {
    for c1[i] < 0 {
        c1[i] += p
    }
}

return c1, nil
}

```

```

func Opposite(galua [][]int, k int, nepr []int, p int) []int {
    for i := 1; i < len(galua); i++ {
        temp := PolyMult(galua[k], galua[i], p, nepr)
    }
}

```

```
        one := make([]int, len(temp))

        one[0] = 1

        if SliceEq(temp, one) {

            return galua[i]

        }

    }

    return nil

}
```

```
func Alpha(stepen int) []int {

    fmt.Println("Ввод альфа: введите", stepen, "коэффициентов, начиная с свободного члена")

    key_alpha := []int{}

    for i := 0; i < stepen; i++ {

        s := -1

        for s < 0 || s >= stepen {

            r := bufio.NewReader(os.Stdin)

            char1, _, err := r.ReadRune()

            if err != nil {

                fmt.Println("i have a problem with your input")

            }

            s = int(char1 - '0')

            if s < 0 || s >= stepen {
```

```
        fmt.Println("try to enter again", s)

    }

}

if len(key_alpha) > 0 {

    key_alpha = slices.Insert(key_alpha, 0, s)

} else {

    key_alpha = append(key_alpha, s)

}

}

if key_alpha == nil {

    panic("Error ERROR ОШИБКА ОШИБКА")

}

return key_alpha

}
```

```
func Beta(stepen int) []int {

    fmt.Println("Ввод бета: введите", stepen, "коэффициентов, начиная с свободного члена")

    key_beta := []int{}

    for i := 0; i < stepen; i++ {

        s := -1

        for s < 0 || s >= stepen {

            r := bufio.NewReader(os.Stdin)
```

```
    char1, _, err := r.ReadRune()

    if err != nil {

        fmt.Println("i have a problem with your input")

    }

    s = int(char1 - '0')

    if s < 0 || s >= stepen {

        fmt.Println("try to enter again", s)

    }

}

if len(key_beta) > 0 {

    key_beta = slices.Insert(key_beta, 0, s)

} else {

    key_beta = append(key_beta, s)

}

}

if key_beta == nil {

    panic("Error ERROR ОШИБКА ОШИБКА")

}

return key_beta

}
```

```
func AffineEncode(message,
```

```

alphabet string,

stepen int,

key_alpha, key_beta []int,

nepr []int) string {

shifr := []string{}

alph := strings.Split(alphabet, "")

for i := 0; i < len(message); i++ {

    if !strings.Contains(alphabet, string(message[i])) {

        shifr = append(shifr, string(message[i]))

    } else {

        for k, j := range alphabet {

            if string(message[i]) == string(j) {

                bukva := Convert(k, 2)

                bukva_galua := []int{}

                temp := ZeroFill(bukva, stepen)

                for _, z := range temp {

                    if len(bukva_galua) > 0 {

                        bukva_galua = slices.Insert(bukva_galua, 0,

int(z-'0'))

                    } else {

                        bukva_galua = append(bukva_galua, int(z-'0'))

                    }

                }

            }

        }

    }

}

}

```

```

    }

    multiplication := PolyMult(bukva_galua, key_alpha, 2, nepr)

    sum := PolySum(multiplication, key_beta, 2)

    index := 0

    for i1 := range sum {

        index = index + Power(2, i1)*sum[i1]

    }

    shifr = append(shifr, alph[index%len(alphabet)])

    break

    }

    }

    }

    }

    return strings.Join(shifr, "")

}

func AffineDecode(message,

    alphabet string,

    stepen int,

    key_alpha, key_beta []int,

```

```

nepr []int) string {

shifr := []string{}

alph := strings.Split(alphabet, "")

for i := 0; i < len(message); i++ {

    if !strings.Contains(alphabet, string(message[i])) {

        shifr = append(shifr, string(message[i]))

    } else {

        for k, j := range alphabet {

            if string(message[i]) == string(j) {

                bukva := Convert(k, 2)

                bukva_galua := []int{}

                temp := ZeroFill(bukva, stepen)

                temp = ZeroFill(temp, stepen)

                for _, z := range temp {

                    if len(bukva_galua) > 0 {

                        bukva_galua = slices.Insert(bukva_galua, 0,

int(z-'0'))

                    } else {

                        bukva_galua = append(bukva_galua, int(z-'0'))

                    }

                }

            }

        }

    }

}

```

```

sum := PolySum(bukva_galua, key_beta, 2)

galua := CreateGalua(2, stepen)


galuaindex := FindSlice(key_alpha, galua)

if galuaindex == -1 {

    panic("WE HAVE A PROBLEM")

}

oppositeAlpha := Opposite(galua, galuaindex, nepr, 2)


multiplication := PolyMult(sum, oppositeAlpha, 2, nepr)

index := 0

for i2 := range multiplication {

    index = int(index + Power(2, i2)*multiplication[i2])

}

shifr = append(shifr, alph[index%len(alphabet)])

break

}

}

}

}

```

```
    return strings.Join(shifr, "")
}

func WorkPole() {

    fmt.Println("Все многочлены представлены в виде [число, x, x^2, x^3...]")

    fmt.Println("Введите через пробел числа p, n")

    in := bufio.NewReader(os.Stdin)

    var p, n int

    fmt.Fscan(in, &p, &n)

    fmt.Println("Размеры поля: p =", p, "n =", n)

    pole := CreateGalua(p, n)

    fmt.Println("Поле Галуа")

    fmt.Println(pole)

    neprev, err := Nprevodim(p, n)

    if err != nil {

        panic(err)

    }

    fmt.Println("Неприводимые элементы:")

    for c, i := range neprev {
```

```

        fmt.Println(c+1, i)

    }

    fmt.Println("Введите номер элемента с которым будем работать")

    neprChosenNumber := 1

    fmt.Fscan(in, &neprChosenNumber)

    nepr := neprev[neprChosenNumber-1]


    fmt.Println("Вами выбран неприводимый элемент", nepr)


    obraz := FindMultip(pole, p, n, nepr)

    fmt.Println("Образующие группы")

    for c, i := range obraz {

        fmt.Println(c+1, i[0])

    }


    fmt.Println("Введите номер элемента с которым будем работать")

    obrazChosen := 0

    fmt.Fscan(in, &obrazChosen)

    fmt.Println("Выбран", obraz[obrazChosen][0])

    steps, ok := obraz[obrazChosen][1].([][]interface{})

    if !ok {

```

```
        fmt.Println("Ошибка приведения типа")

        return

    }

    length := len(steps)

    for i := 0; i < length; i++ {

        fmt.Println("Степень", steps[i][1], steps[i][0])

    }

    flag := false

    for !flag {

        fmt.Println("Если вы хотите выполнить действия с многочленами - press [y]")

        fmt.Println("Если больше задач нет - press [n]")

        inp := 'n'

        switch inp {

        case 'y':

            fmt.Println("Введите два многочлена  $1 + x + 2x^2 + 3x^3 \dots$  в виде 123...")

            fmt.Println("\nОбратите внимание длина многочлена -", n)

        case 'n':

            flag = true

        }

    }
```

```
    }  
  
    fmt.Println("Программа выполнена успешно")  
}
```

```
func WorkShifr() {  
  
    fmt.Println("Работа с шифром")  
  
  
    fmt.Println("Введите сообщение")  
  
    in := bufio.NewReader(os.Stdin)  
  
    var message string  
  
    alphabet := "123456abcdefghijklmnopqrstuvwxyz"  
  
    input, err := in.ReadString('\n')  
  
    if err != nil {  
  
        fmt.Println("Ошибка чтения:", err)  
  
        return  
    }  
  
    input = strings.TrimSpace(input)  
  
    message = input  
  
  
    stepen := 0  
  
  
    for Power(2, stepen) < len(alphabet) {
```

```
        stepen += 1

    }

    key_a := Alpha(stepen)

    key_b := Beta(stepen)


    fmt.Println("Все возможные неприводимые элементы")

    nepr_all, err := Nprevodim(2, stepen)

    if err != nil {

        fmt.Println("ОШибка Ошиткбка", err)

        return

    }

    for i, j := range nepr_all {

        fmt.Println(i+1, j)

    }


    fmt.Println("Введите номер элемента с которым будем работать")

    nepr_c := 0

    fmt.Fscan(in, &nepr_c)


    nepr_chosen := nepr_all[nepr_c]


    fmt.Println("Сначала шифруем")
```

```
    shifr := AffineEncode(message, alphabet, stepen, key_a, key_b, nepr_chosen)

    fmt.Println("Зашифрованное сообщение", shifr)


    fmt.Println("Теперь расшифруем")


    deshifr := AffineDecode(shifr, alphabet, stepen, key_a, key_b, nepr_chosen)

    fmt.Println("Расшифрованное сообщение", deshifr)

}


func main() {

    in := bufio.NewReader(os.Stdin)

    fmt.Println(`Введите "y", если надо поработать с полем и "n", если с шифром`)

    var what string

    fmt.Fscan(in, &what)


    switch what {

    case "y":

        WorkPole()

    case "n":

        WorkShifr()

    }
```

}