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 № 4

по дисциплине «Криптографические методы защиты информации»

Криптосистемы с открытым ключом

Студент гр. БИБ224
Николаев М.А
«16» марта 2025 г.

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

Москва 2025

СОДЕРЖАНИЕ

1 Задание на практическую работу

Целью данной работы является приобретение навыков программной реализации криптосистем с открытым ключом.

В рамках практической работы необходимо выполнить следующее:

1) написать программную реализацию одной из следующих асимметричных криптосистем (по выбору студента) с использованием больших чисел:
   - RSA;
   - Рабина;
   - Эль-Гамаля;
2) изучить методы криптоанализа выбранной криптосистемы;
3) реализовать (вручную или программно) не менее одной атаки на выбранную криптосистему, исключая наивную переборную атаку, для случая, когда параметры криптосистемы не являются большими числами;
4) подготовить отчет о выполнении работы.

Программа должна обладать следующей функциональностью:

1) принимать на вход файл, содержащий открытый текст, подлежащий зашифрованию, или шифртекст, подлежащий расшифрованию;
2) принимать на вход ключевую пару (открытый ключ, закрытый ключ);
3) давать пользователю возможность сгенерировать ключевую пару;
4) осуществлять зашифрование или расшифрование введенного текста по выбору пользователя.

Отчет должен содержать следующие составные части:

1) раздел с заданием;
2) раздел с краткой теоретической частью;
3) раздел с двумя-тремя примерами «ручного» шифрования для произвольных последовательностей символов;
4) раздел с результатами работы программы для тех же последовательностей символов, что и в предыдущем разделе;
5) раздел с подробным описанием реализованной атаки на криптосистему с приведением численных результатов;
6) раздел с выводами о проделанной работе.

2 Теория

Асимметричные криптосистемы основываются на использовании двух ключей: приватного ключа для дешифрования, который выбирается пользователем и остается в секрете, и публичного ключа для шифрования, который генерируется на основе приватного ключа и может быть передан другому пользователю. Алгоритмы шифрования и генерации ключей разработаны таким образом, чтобы было сложно вычислить приватный ключ, зная только публичный.

В данной работе представлена реализация криптосистемы Эль-Гамаля. Данная криптосистема в настоящее время является одной из основных криптосистем с открытым ключом. В отличие от криптосистем RSA и Рабина, она основана на задаче дискретного логарифмирования в конечной абелевой группе: мультипликативной группе простого конечного поля ❑ или группе точек эллиптической кривой . Кроме того, в криптосистеме Эль-Гамаля существуют некоторые открытые параметры, называемые параметрами домена, которые могут использоваться определенными группами пользователей. Параметры домена для случая построения криптосистемы Эль-Гамаля над ❑: — большое простое число, такое, что делится на другое простое число меньшей битовой длины; — элемент мультипликативной группы поля ❑, порождающий большое число элементов ❑.

Алгоритм генерации ключей.

1. Пользователь *A* задает параметры домена и .
2. Пользователь *A* выбирает случайное число *x* в интервале .
3. Пользователь *A* вычисляет
4. Открытый ключ пользователя *A* есть .
5. Закрытый ключ пользователя *A* есть .

Алгоритм зашифрования.

1. Пользователь *B* получает аутентичную копию открытого ключа пользователя A — число .
2. Пользователь *B* представляет сообщение в виде числа в интервале . В общем случае сообщение может быть разбито на блоки, каждый из которых представляется своим числом.
3. Пользователь *B* вычисляет сеансовый ключ в интервале .
4. Пользователь *B* вычисляет и .
5. Пользователь *B* отправляет пару пользователю *A*.

Алгоритм расшифрования.

1. Пользователь *A* получает шифртекст — пару .

2. Пользователь *A*, используя свой секретный ключ, вычисляет открытый текст по следующей формуле:

3 Примеры ручного шифрования

Входные данные: текст m = “alaska young”

Зашифрование:

1) Вычисляем открытый ключ:

Т.к. находится между 2 в 8 и 9 степени, то его длина равна 9 бит, поэтому размер одного блока будет равен одному байту, то есть 8 бит. Так же для полного восстановления текста, в конец будем добавлять 1 и после него количество нулей, недостающих до полного блока. Сеансовыми ключами будет 3.

2) Шифрование:

Для каждого символа шифртекст вычисляется так:

Символ „a“, ASCII = 97,

Символ „l“, ASCII = 108,

Символ „a“, ASCII = 97, c = 222

Символ „s“, ASCII = 115, c = 88

Символ „k“, ASCII = 107, c = 115

Символ „a“, ASCII = 97, c = 222

Символ „ “, ASCII = 32, c = 185

Символ „y“, ASCII = 121, c = 141

Символ „o“, ASCII = 111, c = 248

Символ „u“, ASCII = 117, c = 8

Символ „n“, ASCII = 110, c = 288

Символ „g“, ASCII = 103, c = 275

Зашифрованное сообщение = [64, 222, 64, 75, 64, 222, 64, 88, 64, 115, 64, 222, 64, 185, 64, 141, 64, 248, 64, 8, 64, 288, 64, 275] =

Расшифрование:

4 Пример работы программы

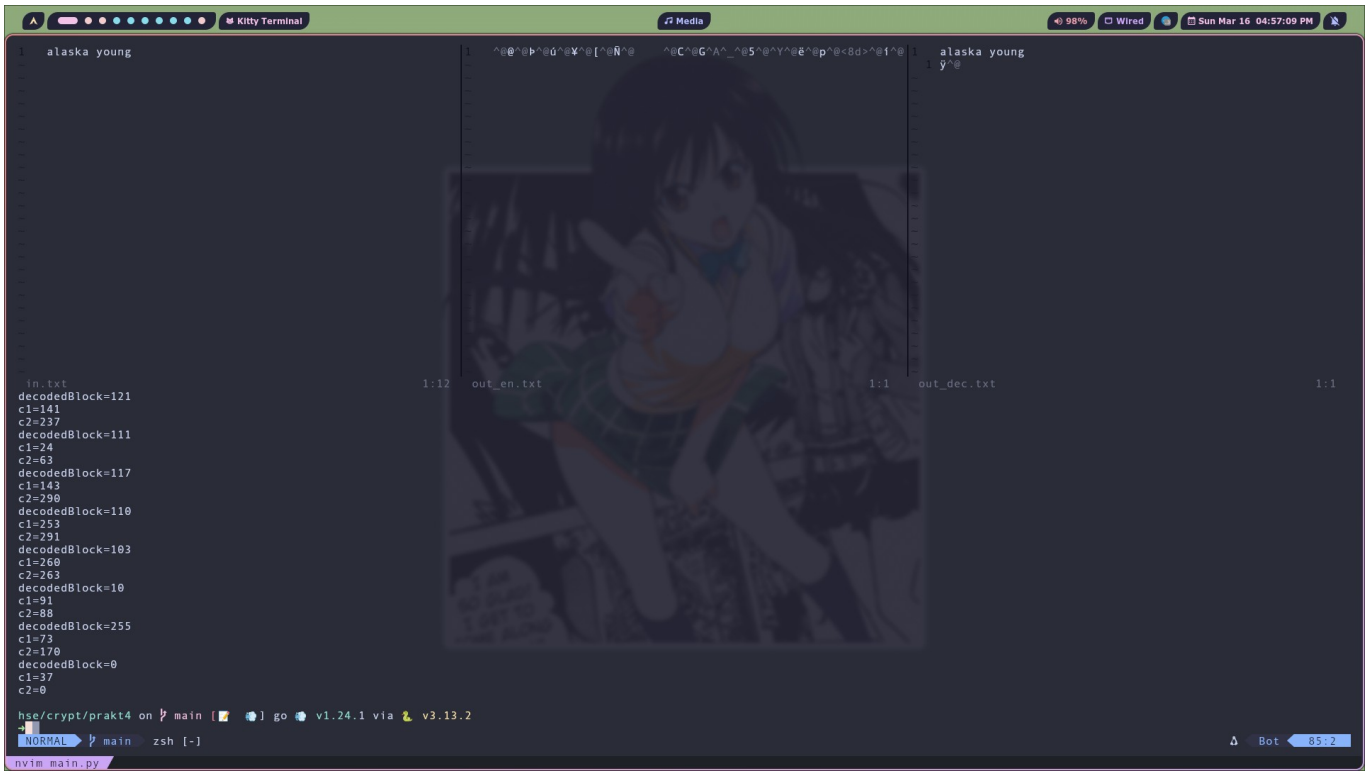

Рисунок 1

5 Криптоанализ

Примером простой атаки может служить “атака одинаковых сессионных ключей”. Предположим, мы имеем 1 и тот же сессионный ключ на 2 разных сообщения, назовём его k. Остальные параметры соответственно будут одинаковые тоже. Тогда рассмотрим шифртекст, который получается в таком случае:

При помощи этой формулы мы спокойно сможем узнать изначальный вид сообщения, которое будет передано с тем же сессионным ключом.

Другим примером может служить атака малого модуля. Её суть состоит в том, что будем искать решение уравнения , зная . Но не полным перебором, а алгоритмом baby step giant step.

Пусть есть уравнение в циклической группе. Положим, что , где заранее выбранная константа (обычно используют округленное вверх значение квадратного корня из p), очевидно любое из промежутка [0, p) можно представить в такой форме. Тогда уравнение принимает вид

Алгоритм предварительно вычисляет для некоторых i. Потом корректируется значение m и пробуются различные значения i.

Проведём атаку:

В данном случае . Наша цель – x. Сначала определим m

Затем мы вычислим от 1 до m и занесём информацию в виде структуры данных hash table (аналог данной структуры данных в питоне будет dict, а в c/c++ - map, для удобства далее будем называть выбранную структуру словарём). Словарь будет иметь следующий вид, ключ - , значение .

Например: если , мы получаем

Теперь у нас есть список пар от 0 до квадратного корня из . Вычислим . Согласно одному из следствий малой теоремы Ферма . Затем мы итерируемся по значениям пока мы не найдем совпадения в таблице. Тогда мы умножаем число на и добавляем число, которое сопоставлено в таблице.

Число 42 есть в словаре, значит

Таким образом данный алгоритм эффективнее для перебора чем просто метод грубой силы. Его сложность можно оценить как Но при достаточно больших p он всё равно будет занимать больше миллиардов и миллиардов и … лет.

6 Выводы о проделанной работе

В этой работе мы программно сделали криптосистему Эль-Гамаля и провели ее криптоанализ. Если длина ключа невелика, его можно определить атакой малого модуля за значительно меньшее время, чем при использовании грубой силы. Однако при большой длине ключа, время вычисления остается неприемлемо долгим, ведь для полного перебора максимально допустимого по длине ключа в этой системе (4096 бит) потребуется операций, а при использовании самого оптимального алгоритма, существующего на данный момент, скорость едва ли позволит взломать ключ до смерти вселенной. Практически все традиционные методы криптоанализа неэффективны против этой криптосистемы, что делает ее достаточно надежной, и поэтому она до сих пор находит применение, как например в цифровых подписях или ГОСТ от 1994 года. Тем не менее, у нее есть некоторые недостатки, в частности, отсутствие стандартов реализации. Это означает, что не все продукты, использующие ее, обладают высокой стойкостью, что позволяет злоумышленникам успешно взламывать этот шифр в уязвимых реализациях.

ПРИЛОЖЕНИЕ А.

Программная реализация

```
package main


import (
	"fmt"
	"io/ioutil"
	"math/rand"
	"os"
	"time"
)


func modPow(a, x, n int) int {
	result := 1
	for x > 0 {
		if x%2 == 1 {
			result = (result * a) % n
		}
		a = (a * a) % n
		x /= 2
	}
	return result
}


func checkPrimeFerma(n int) bool {
	return modPow(2, n-1, n) == 1
}


func checkPrime(n int) bool {
	if n == 2 {
```

```
		return true
	}
	if n == 1 || n%2 == 0 {
		return false
	}
	if !checkPrimeFerma(n) {
		return false
	}
	for d := 3; d*d <= n; d += 2 {
		if n%d == 0 {
			return false
		}
	}
	return true
}

func extendedEuclid(a, b int) (int, int, int) {
	if b == 0 {
		return a, 1, 0
	}
	gcd, x1, y1 := extendedEuclid(b, a%b)
	x := y1
	y := x1 - (a/b)*y1
	return gcd, x, y
}

func generateOpenKey(p, g, k int) int {
	return modPow(g, k, p)
}
```

```
func encodeBlock(h, m, p, g, k int) (int, int) {
	c1 := modPow(g, k, p)
	c2 := (m * modPow(h, k, p)) % p
	return c1, c2
}

func decodeBlock(x, c1, c2, p int) int {
	c1Exp := modPow(c1, x, p)
	_, _, inv := extendedEuclid(p, c1Exp)
	inv = ((inv % p) + p) % p
	result := (c2 * inv) % p
	return result
}

func readFileBytes(filename string) ([]byte, error) {
	return ioutil.ReadFile(filename)
}

func writeFileBytes(filename string, data []byte) error {
	return ioutil.WriteFile(filename, data, 0644)
}

func encrypt(byteString []byte, key, p, g int) ([]int, int) {
	bitLen := 0
	temp := p - 2
	for temp > 0 {
		bitLen++
		temp /= 2
	}
	lenMsg := (bitLen - 1) / 8
```

```
	if lenMsg < 1 {
		lenMsg = 1
	}

	byteString = append(byteString, 255)
	padding := lenMsg - (len(byteString) % lenMsg)
	for i := 0; i < padding; i++ {
		byteString = append(byteString, 0)
	}

	result := []int{}
	for i := 0; i < len(byteString); i += lenMsg {
		blockInt := 0
		for j := 0; j < lenMsg; j++ {
			blockInt = (blockInt << 8) | int(byteString[i+j])
		}
		k := rand.Intn(p-3) + 2
		c1, c2 := encodeBlock(key, blockInt, p, g, k)
		fmt.Printf("block=%d\nk=%d\n", blockInt, k)
		result = append(result, c1, c2)
	}
	return result, lenMsg
}

func decrypt(byteString []byte, key, p, g int) []int {
	bitLen := 0
	temp := p - 2
	for temp > 0 {
		bitLen++
		temp /= 2
```

```
	}
	lenMsg := (bitLen - 1) / 8
	if lenMsg < 1 {
		lenMsg = 1
	}
	result := []int{}
	blockSize := 2 * (lenMsg + 1)
	for i := 0; i < len(byteString); i += blockSize {
		c1 := 0
		for j := 0; j < lenMsg+1; j++ {
			c1 = (c1 << 8) | int(byteString[i+j])
		}
		c2 := 0
		for j := 0; j < lenMsg+1; j++ {
			c2 = (c2 << 8) | int(byteString[i+lenMsg+1+j])
		}
		decodedBlock := decodeBlock(key, c1, c2, p)
		fmt.Printf("decodedBlock=%d\nc1=%d\nc2=%d\n", decodedBlock, c1, c2)
		result = append(result, decodedBlock)
	}
	return result
}

func writeIntsToFile(result []int, filename string, lenMsg int) error {
	data := []byte{}
	for _, num := range result {
		b := make([]byte, lenMsg)
		for i := lenMsg - 1; i >= 0; i-- {
			b[i] = byte(num & 0xFF)
			num >>= 8
```

```
            }
            data = append(data, b...)
        }
        return writeFileBytes(filename, data)
}

func main() {
        rand.Seed(time.Now().UnixNano())

        p := 293
        g := 4
        closedKey := 5
        openKey := 0

        fmt.Println("Выберите режим работы:")
        fmt.Println("1. Шифрование/дешифрование")
        fmt.Println("2. Генерация открытого ключа")

        var mode int
        _, err := fmt.Scan(&mode)
        if err != nil {
            fmt.Println("Ошибка ввода:", err)
            os.Exit(1)
        }

        switch mode {
        case 1:
            inBytes, err := readFileBytes("in.txt")
            if err != nil {
                fmt.Println("Ошибка чтения in.txt:", err)
```

```
                return
            }
            openKey = generateOpenKey(p, g, closedKey)
            encrypted, lenMsg := encrypt(inBytes, openKey, p, g)
            err = writeIntsToFile(encrypted, "out_en.txt", lenMsg+1)
            if err != nil {
                fmt.Println("Ошибка записи out_en.txt:", err)
                return
            }
            encBytes, err := readFileBytes("out_en.txt")
            if err != nil {
                fmt.Println("Ошибка чтения out_en.txt:", err)
                return
            }
            decrypted := decrypt(encBytes, closedKey, p, g)
            err = writeIntsToFile(decrypted, "out_dec.txt", 1)
            if err != nil {
                fmt.Println("Ошибка записи out_dec.txt:", err)
                return
            }
        case 2:
            fmt.Println("Открытый ключ:", generateOpenKey(p, g, closedKey))
        default:
            fmt.Println("Неверный режим")
        }
}
```