МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА №2.4
“Атака на алгоритм шифрования RSA, основанная на Китайской теореме об остатках”

по дисциплине

“Информационная безопасность”

Студент:

Алексеев Даниил Иннокентьевич

Группа P34302

Преподаватель:

Рыбаков Степан Дмитриевич

Санкт-Петербург

2023

Цели работы:

- Изучить базовые принципы RSA шифрования
- Потренировать важные инструменты криптографии: теорема Эйлера, алгоритм Евклида
- Изучить атаку на алгоритм шифрования RSA посредством Китайской теоремы об остатках
- Найти открытый текст шифротекста

Задание (вариант 16)

Для имеющихся трех зашифрованных сообщений, общей экспонеты и известных взаимопростых N_1,N_2,N_3 найти исходный текст.

e = 3 – экспонента, используемая в шифровании.

Отправитель	N	Шифротекст
1	519445678909	302279248041 398777422648 382393465830 109346520792 393648988334 83456507369 503695835656 409770589873 483819180150 358939341533 402486907104 347176414967 1633679742
2	522088422619	48522238217 116578598684 98210011370 452947538650 113090002659 130683028799 170075383039 19947030841 458406287083 178964953872 500143943025 189689940709 218613469572
3	523328119219	129856570412 82270781294 140695444887 510689827054 42634086860 516267119547 5616396143 8388941434 73724586316 290433741122 102266925300 75736288391 406132000561

Китайская теорема об остатках

Для взаимопростых $\mathbf{n_1,n_2,\ldots,n_k}$ существует однозначное соответствие некого числа **a** с числами $\mathbf{a_1,a_2,\ldots,a_k}$, такое что $\mathbf{a_i \equiv a \bmod n_1}$

Число а можно вывести:

По условию теоремы **a** должно быть таким что при делении на $\mathbf{n_1}$ будет остаток $\mathbf{a_1}$, это условие работает для:

$$a=a_1 \cdot Y_1\text{, где } Y_1 \text{ такое что } Y_1 \bmod n_1 \equiv 1$$

В то же время, должны выполнятся равенства и для других $\mathbf{a_i}$. Модульная арифметика позволяет добиться этого простым сложением:

$$a=a_1 \cdot Y_1+a_2 \cdot Y_2+a_3 \cdot Y_3$$

Однако теперь при взятии остатка от деления **a** на $\mathbf{n_i}$ помимо остатка из i-го слагаемого, остатки других слагаемых будут не нулевыми и нарушат равенство. Поэтому необходимо модифицировать формулу. Для каждого i-го слагаемого необходимо добавить конструкцию, которая будет нейтрализовывать (обнулять) остаток при делении этого числа на $\mathbf{n_j}$, где $\mathbf{j \neq i}$. Таким числом является $X_i=\frac{n_1 \cdot n_2 \cdot \ldots \cdot n_3}{n_i}$

Чтобы не нарушать предыдущие свойства, которое привносились числами $\mathbf{Y_i}$ для отдельных слагаемых, надо представаить $\mathbf{Y_i}$ в виде произведения $\mathbf{X_i}$ некоторое $\mathbf{x_i}$. Т.е. $Y_i=X_i \cdot x_i$. Если подставить это в условие $\mathbf{Y_i \bmod n_i \equiv 1}$, то получим что число $\mathbf{x_i}$ должно быть обратным по модулю с числом $\mathbf{X_i}$. Будем записывать это как $Y_i=X_i \cdot inv_{n_i}(X_i)$

В результате получаем:

$$a=a_1 \cdot X_1 inv_{n_1}(X_1)+a_2 \cdot X_2 inv_{n_2}(X_2)+a_3 \cdot X_3 inv_{n_3}(X_3)$$

Для такого числа, например, взятие остатка от деления на $\mathbf{n_2}$ приведет к обнулению 1-го и 3-го слагаемого, и к обращению второго слагаемого в $\mathbf{a_2}$. Таким образом, мы нашли формулу позволяющую найти число **a** удовлетворяющее условию Китайской теоремы об остатках.

Решение задачи

Атака выполняется на основе того, что нам даны все условия для использования свойства КТО: взаимопростые числа N_1,N_2,N_3; набор чисел C_1,C_2,C_3; и число y – сообщение, которое необходимо вычислить при помощи выведенной ранее формулы.

Расшифруем блоки, подставляя числа в формулу. Выполним это для данных при помощи программы на Kotlin с использованием библиотек для работы с большими числами:

```kotlin
import java.io.File
import java.math.BigDecimal
import java.math.BigInteger
import java.math.MathContext


fun main(args: Array<String>) {

    val text = File(args[0]).readText().split("\n")
    val blocks_n = text[0].toInt()

    val n1: BigInteger = BigInteger.valueOf(text[1].toLong())
    val n2: BigInteger = BigInteger.valueOf(text[2].toLong())
    val n3: BigInteger = BigInteger.valueOf(text[3].toLong())


    var offset: Int = 4
    val msg1: ArrayList<BigInteger> = ArrayList()
    val msg2: ArrayList<BigInteger> = ArrayList()
    val msg3: ArrayList<BigInteger> = ArrayList()

    for (i in 0..<blocks_n) {
        val e = text[offset + i].toLong()
        msg1.add(BigInteger.valueOf(e))
    }

    offset += blocks_n
```

```
    for (i in 0..<blocks_n) {
        val e = text[offset + i].toLong()
        msg2.add(BigInteger.valueOf(e))
    }

    offset += blocks_n
    for (i in 0..<blocks_n) {
        val e = text[offset + i].toLong()
        msg3.add(BigInteger.valueOf(e))
    }

    val r = getUncrypted(msg1[0], msg2[0], msg3[0], n1, n2, n3)

    for (i in 0..<blocks_n) {
        val r = getUncrypted(msg1[i], msg2[i], msg3[i], n1, n2, n3)
        println(r)
    }
}

fun getUncrypted(
    C1: BigInteger,
    C2: BigInteger,
    C3: BigInteger,
    N1: BigInteger,
    N2: BigInteger,
    N3: BigInteger
): BigInteger {

    val M0: BigInteger = N1.multiply(N2).multiply(N3)
    val m1: BigInteger = N2.multiply(N3)
    val m2: BigInteger = N1.multiply(N3)
    val m3: BigInteger = N1.multiply(N2)
    val n1: BigInteger = m1.modInverse(N1)
    val n2: BigInteger = m2.modInverse(N2)
    val n3: BigInteger = m3.modInverse(N3)
```

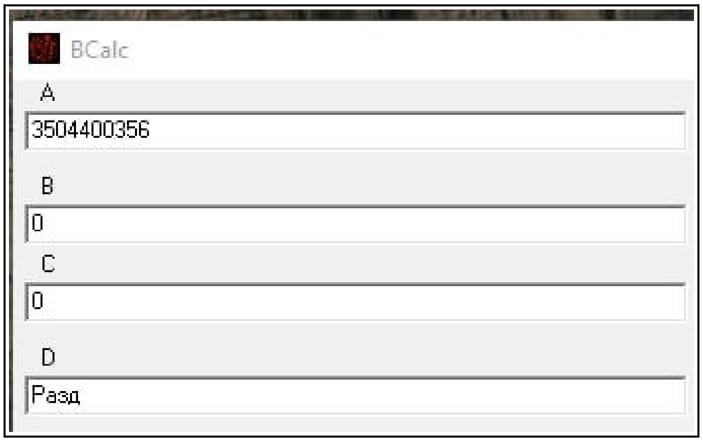
```
    val s1: BigInteger = C1.multiply(n1).multiply(m1)
    val s2: BigInteger = C2.multiply(n2).multiply(m2)
    val s3: BigInteger = C3.multiply(n3).multiply(m3)

    val s: BigInteger = (s1.add(s2).add(s3)).mod(M0)
    val res: BigDecimal = cuberoot(s.toBigDecimal())
    return res.toBigInteger()
}

fun cuberoot(b: BigDecimal?): BigDecimal {
    val mc = MathContext(40)
    var x = BigDecimal("1", mc)

    for (i in 0 until 1000) {
        x = x.subtract(
            x.pow(3, mc)
                .subtract(b, mc)
                .divide(
                    BigDecimal("3", mc).multiply(
                        x.pow(2, mc), mc
                    ), mc
                ), mc
        )
    }
    return x
}
```

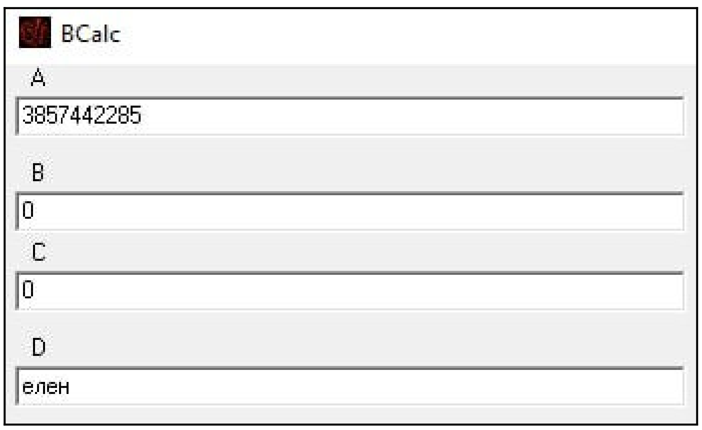
После исполнения програмым получены расшифрованные блоки:

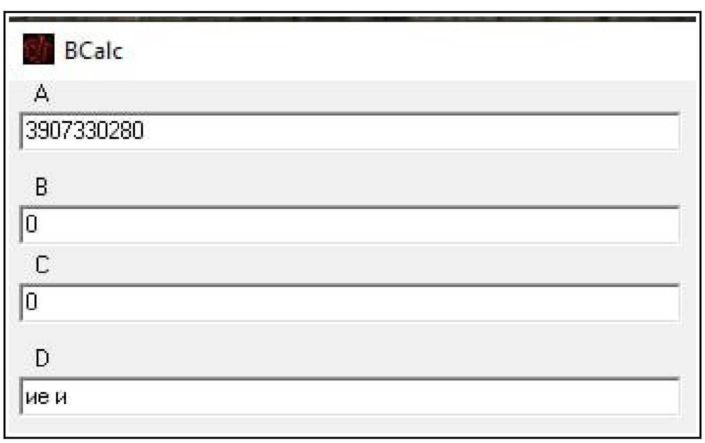
```
3504400356
3857442285
3907330280
4075742190
3808163104
4058179044
3857574121
552594976
4092653282
3992972320
```

```
4059167723
4009225710
3911196704
```

Воспользуемся программой BCalc для преобразования чисел в текст:







и т.д.

Полученный текст

Разделение итоговых сведений по уровням ссылочной

Вывод

В ходе выполнения лабораторной работы я ознакомился с основами шифрования RSA и способом атаки на алгоритм на основе китайской теоремы об остатках.