**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**Уфимский государственный авиационный технический университет**

**Кафедра вычислительной математики и кибернетики**

**Расчетно-графическая работа по курсу**

**«Защита информации»**

Вариант 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ф.И.О. | Дата | Оценка | Подпись |
| Выполнил студент группы ПРО-301в | Доронин С.Г. |  |  |  |
| Принял: | Иванова Л.Ш. |  |  |  |

Уфа – 2014 г.

# Задание на расчетно-графическую работу

Изучить и программно реализовать криптографический алгоритм IDEA.

# Содержание

[Введение 4](#_Toc389568226)

[Алгоритм шифрования / расшифрования 4](#_Toc389568227)

[Текст программы 7](#_Toc389568228)

[Класс шифрования IDEA 7](#_Toc389568229)

[Класс с функциями-помощниками для шифрования 8](#_Toc389568230)

[Текст используемых источников 11](#_Toc389568231)

# Введение

Первый вариант шифра IDEA, предложенный Ксуеджа Лай и Джеймсом Масси, появился в 1990 году. Он назывался РЕS (Prorosed Encryption Standard, предложенный стандарт шифрования). В следующем году, после демонстрации Бихамом и Шамиром возможностей дифференциального криптоанализа, авторы усилили свой шифр против такого вскрытия и назвали новый алгоритм IPES (Improved Prorosed Encryption Standard, улучшенный предложенный стандарт шифрования). В 1992 году название IPES было изменено на IDEA (International Data Encryption Algorithm, международный алгоритм шифрования данных).

IDEA основывается на некоторых впечатляющих теоретических положениях и, хотя криптоанализ добился некоторых успехов в отношении вариантов с уменьшенным количеством этапов, алгоритм все еще кажется сильным.

IDEA является блочным шифром, он работает с 64-битовыми блоками открытого текста. Длина ключа - 128 битов. Для шифрования и дешифрирования используется один и тот же алгоритм.

Как и другие, уже рассмотренные блочные шифры IDEA использует и запутывание, и рассеяние. Флософия, лежащая в основе проекта, представляет собой "объединение операций из различных алгебраических групп". Смешиваются три алгебраические группы:

- XOR

- Сложение по модулю 216

- Умножение по модулю 216 + 1

# Алгоритм шифрования / расшифрования

Шифрование начинается с того, что 64-битовый блок данных делится на четыре 16-битовых подблока: Х1, Х2,, Х3 и Х4. Эти четыре подблока становятся входными данными для первого этапа алгоритма. Всего в алгоритме восемь этапов. На каждом этапе четыре подблока подвергаются операциям XOR, сложениям и умножениям друг с другом и с шестью 16-битовыми подключами. Между этапами обмениваются местами второй и третий подблоки. Наконец четыре подблока объединяются с четырьмя подключами в окончательном преобразовании. На каждом этапе события происходят в следующей последовательности:

(1) Перемножаются Х1 и первый подключ.

(2) Складываются Х2 и второй подключ.

(3) Складываются Х3 и третий подключ.

(4) Перемножаются Х4 и четвертый подключ.

(5) Выполняется XOR над результатами этапов (1) и (3).

(6) Выполняется XOR над результатами этапов (2) и (4).

(7) Перемножаются результаты этапа (5) и пятый подключ.

(8) Складываются результаты этапов (6) и (7).

(9) Перемножаются результаты этапа (8) и шестой подключ.

(10) Складываются результаты этапов (7) и (9).

(11) Выполняется XOR над результатами этапов (1) и (9).

(12) Выполняется XOR над результатами этапов (3) и (9).

(13) Выполняется XOR над результатами этапов (2) и (10).

(14) Выполняется XOR над результатами этапов (4) и (10).

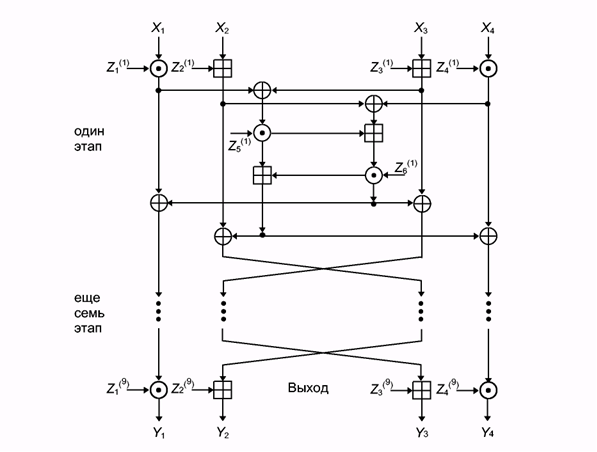


Рисунок 1. Схема IDEA

Xi: 16-битовый подблок открытого текста

Yi: 16-битовый подблок шифротекста

Zj(i):16-битовый подблок ключа

⊕: побитовое "исключающее или" (XOR) 16-битовых подблоков

□: сложение по модулю 216 16-битовых целых

Ο: умножение по модулю 216+1 16-битовых целых при условии, что нулевой подблок соответствует 216

Выходом этапа являются четыре подблока - результаты действий (11), (12), (13) и (14).

После восьмого этапа выполняется заключительное преобразование:

1. Перемножаются *Х1* и первый подключ.
2. Складываются *Х2* и второй подключ.
3. Складываются *Х3* и третий подключ.
4. Перемножаются *Х4* и четвертый подключ.

Наконец четыре подблока снова соединяются, образуя шифротекст.

Также несложно создавать подключи. Алгоритм использует 52 из них (шесть для каждого из восьми этапов и еще четыре для заключительного преобразования). Сначала 128-битовый ключ делится на восемь 16-битовых подключей. Это первые восемь подключей алгоритма (шесть для первого этапа и два - для второго). Затем ключ циклически сдвигается налево на 25 битов и снова делится на восемь подключей. Первые четыре используются на этапе 2, а оставшиеся четыре - на этапе 3. Ключ циклически сдвигается налево на 25 битов для получения следующих восьми подключей, и так до конца алгоритма.

Дешифрирование выполняется точно также за исключением того, что подключи инвертируются и слегка изменяются. Подключи при дешифрировании представляют собой обратные значения ключей шифрования по отношению к операциям либо сложения, либо умножения. Эти вычисления могут занять некоторое время, но их нужно выполнить один раз для каждого ключа дешифрирования. В Таблице представлены подключи шифрования и соответствующие им подключи дешифрования:

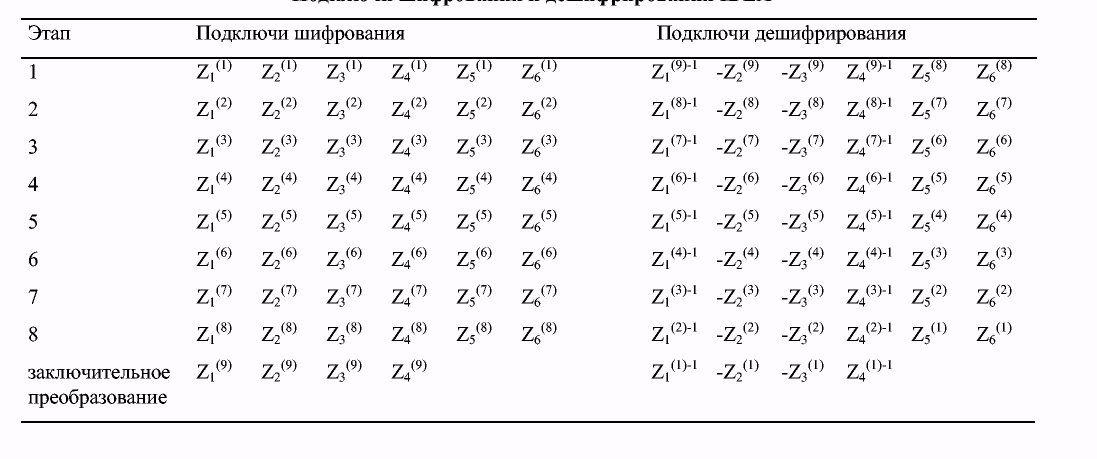


Рисунок 2. Подключи шифрования и дешифрования IDEA

# Текст программы

## Класс шифрования IDEA

**object** IDEA {

/\*\* Отладка \*/

**val** DebugEnable **=** **false**

/\*\* Размер ключа \*/

**val** KeySize **=** 128

/\*\* Размер частей ключа \*/

**val** KeyPartSize **=** 16

/\*\* Кол-во подключей \*/

**val** SubKeysCount **=** 52

/\*\* Кол-во блоков подключей \*/

**val** SubKeysBlocksCount **=** 6

/\*\* Размер блока \*/

**val** BlockSize **=** 64

/\*\* Кол-во подблоков \*/

**val** SubBlocksCount **=** 4

/\*\* Размер подблоков \*/

**val** SubBlocksSize **=** 16

/\*\* Генерация ключа \*/

**def** generateKey() **=** {

**val** key **=** BitNumber(0)

**for** (i **<-** 1 to 4) {

**val** part **=** BitNumber(Array(Random.nextInt()))

key.join(part)

}

key

}

}

/\*\*

\* Алгоритм шифрования данных IDEA

\* http://ru.wikipedia.org/wiki/IDEA

\*/

**class** IDEA(key: BitNumber, encrypt: Boolean) **extends** IDEATools {

**import** dataprotection.rgr.idea.IDEA.\_

assert(key.size **==** KeySize)

/\*\* Подключи \*/

**private** **val** subKeys **=** **if** (encrypt) generateSubKeys **else** invertKey(generateSubKeys)

assert(subKeys.map(\_.size).sum **==** SubKeysCount)

/\*\* Подключи в виде строки (для отлдаки) \*/

**val** subKeysStr **=** subKeys.map{ e **=>**

e.map(\_.toHexStr).mkString(", ")

}.mkString("\n")

/\*\* Шифрование данных \*/

**def** processBlocks(data: BitNumber) **=** {

**val** out **=** BitNumber(0)

data.split(BlockSize).map(processBlock).foreach(out.join)

out

}

/\*\* Генерация подключей \*/

**private** **def** generateSubKeys **=** {

**val** tmpSubKeys **=** ListBuffer[BitNumber]()

**var** shiftKey **=** key

**for** (i **<-** 1 to SubKeysBlocksCount) {

tmpSubKeys **++=** shiftKey.split(KeyPartSize)

shiftKey **=** shiftKey **<<** 25

}

**val** sk **=** shiftKey.split(KeyPartSize)

tmpSubKeys **++=** sk.take(4)

// Разбитие на блоки по 6 подключей

tmpSubKeys.sliding(SubKeysBlocksCount, SubKeysBlocksCount).toList

}

/\*\* Шифрование/расшифрование блока \*/

**private** **def** processBlock(data: BitNumber) **=** {

assert(data.size **==** BlockSize)

**var** d **=** data.split(SubBlocksSize)

**if** (DebugEnable) println(s"0. " **+** d.map(\_.toHexStr).mkString(", "))

assert(d.size **==** SubBlocksCount)

/\*\* Выделение последних 16 бит \*/

**val** m **=** (bitNumber: BitNumber) **=>** bitNumber.last(SubBlocksSize)

**for** (i **<-** 0 until 8) {

**val** k **=** subKeys(i)

**val** s1 **=** mul(d(0), k(0))

**val** s2 **=** m(d(1) **+** k(1))

**val** s3 **=** m(d(2) **+** k(2))

**val** s4 **=** mul(d(3), k(3))

**val** s5 **=** s1 xor s3

**val** s6 **=** s2 xor s4

**val** s7 **=** mul(s5, k(4))

**val** s8 **=** m(s6 **+** s7)

**val** s9 **=** mul(s8, k(5))

**val** s10 **=** m(s7 **+** s9)

**val** d0 **=** s1 xor s9

**val** d1 **=** s3 xor s9

**val** d2 **=** s2 xor s10

**val** d3 **=** s4 xor s10

d **=** List(d0, d1, d2, d3)

**if** (DebugEnable) println(s"${i+1}. " **+** d.map(\_.toHexStr).mkString(", "))

}

**val** k **=** subKeys(8)

**val** d0 **=** mul(d(0), k(0))

**val** d1 **=** d(2) **+** k(1)

**val** d2 **=** d(1) **+** k(2)

**val** d3 **=** mul(d(3), k(3))

// Склеивание подблоков

**val** dList **=** List(d0, d1, d2, d3).map(m)

**val** outBits **=** BitNumber(0)

dList.foreach(outBits.join)

outBits

}

}

## Класс с функциями-помощниками для шифрования

/\*\*

\* Утилита для шифра IDEA

\*/

**trait** IDEATools {

/\*\* 2 в 16 + 1 \*/

**private** **val** Base **=** 0x10001

/\*\* Маска для выделения последних 16 бит из Int \*/

**private** **val** Mask **=** 0xffff

/\*\* Умножение по модулю 2 в 16 + 1 \*/

**def** mul(a: BitNumber, b: BitNumber) **=** {

**var** x **=** a.toInt

**var** y **=** b.toInt

**if** (x **==** 0) x **=** Base **-** y

**else** **if** (y **==** 0) x **=** Base **-** x

**else** {

**val** p **=** x \* y

y **=** p **&** Mask

x **=** p **>>>** SubBlocksSize

x **=** y **-** x **+** (if (y **<** x) 1 **else** 0)

}

**val** res **=** BitNumber(Array(x **&** Mask))

res.last(SubBlocksSize)

}

/\*\*

\* Функция инвертирует подключи шифрования в подключи расшифрования.

\* Она таже включает в себя функции для нахождения мультипликативного обратного и аддитивно обратного.

\*/

**def** invertKey(inKeySeq: Seq[Seq[BitNumber]]) **=** {

/\*\* Промежуточные значения \*/

**var** t1, t2, t3, t4 **=** 0

/\*\* Входной ключ в виде Int \*/

**val** inKey **=** inKeySeq.flatMap(\_.toSeq).map(\_.toInt)

/\*\* Инвертированный ключ \*/

**val** key **=** **new** Array[Int](SubKeysCount)

/\*\* Индекс входного ключа \*/

**var** inKeyIndex **=** 0

/\*\* Индекс выходного ключа \*/

**var** outKeyIndex **=** SubKeysCount

/\*\* Возвращает подключ входного ключа и увеличивает индекс (inKeyIndex) \*/

**val** getSubKey **=** () **=>** {val value **=** inKey(inKeyIndex); inKeyIndex **+=** 1; value}

/\*\* Уменьшает индекс (outKeyIndex) и устанавливает подключ выходного ключа \*/

**val** setSubKey **=** (value: Int) **=>** {outKeyIndex **-=** 1; key(outKeyIndex) **=** value}

/\*\* Набор операций 1 \*/

**val** set1 **=** () **=>** {

t1 **=** getSubKey(); t2 **=** getSubKey()

setSubKey(t2); setSubKey(t1)

}

/\*\* Набор операций 2 \*/

**val** set2 **=** () **=>** {

t1 **=** mulInv(getSubKey()); t2 **=** addInv(getSubKey())

t3 **=** addInv(getSubKey()); t4 **=** mulInv(getSubKey())

setSubKey(t4); setSubKey(t3)

setSubKey(t2); setSubKey(t1)

}

set2()

**for** (i **<-** 1 to 7) {

set1()

t1 **=** mulInv(getSubKey()); t2 **=** addInv(getSubKey())

t3 **=** addInv(getSubKey()); t4 **=** mulInv(getSubKey())

/\* NB: Order \*/

setSubKey(t4); setSubKey(t2)

setSubKey(t3); setSubKey(t1)

}

set1()

set2()

key.map(e **=>** BitNumber(Array(e)).last(KeyPartSize)).sliding(SubKeysBlocksCount, SubKeysBlocksCount).toList.map(\_.toList)

}

/\*\*

\* Вычисляет мультипликативное обратное используя алгоритм Евклида для нахождения НОД.

\* Ноли и один self inverse.

\* <p>

\* x \* mulInv(x) == 1 (modulo BASE)

\*/

**private** **def** mulInv(xIn: Int): Int **=** {

**var** t0 **=** 0

**var** t1 **=** 0

**var** q **=** 0

**var** y **=** 0

**var** x **=** xIn

**if** (x **<** 2) **return** x

t0 **=** 1

t1 **=** Base / x

y **=** Base **%** x

**while** (y **!=** 1) {

q **=** x / y

x **=** x **%** y

t0 **=** (t0 **+** (t1 \* q)) **&** Mask

**if** (x **==** 1) {

**return** t0

}

q **=** y / x

y **=** y **%** x

t1 **=** (t1 **+** (t0 \* q)) **&** Mask

}

(1 **-** t1) **&** Mask

}

/\*\*

\* Возвращает аддитивное обратное для x.

\* <p>

\* x + addInv(x) == 0

\*/

**private** **def** addInv(x: Int) **=** (0 **-** x) **&** Mask

}

# Текст используемых источников

1. ru.wikipedia.org [Электронный ресурс] // Википедия – Свободная энциклопедия - <http://ru.wikipedia.org/wiki/IDEA> Режим доступа – свободный.
2. protect.htmlweb.ru [Электронный ресурс] // Защита информации в компьютерных системах - <http://protect.htmlweb.ru/idea.htm> Режим доступа – свободный.
3. kriptografea.narod.ru [Электронный ресурс] // Криптография - <http://kriptografea.narod.ru/IDEA.html> Режим доступа – свободный.