Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 1

“Компьютерная реализация блочных шифров

на примере DES и ГОСТ”

Выполнил студент гр. 753504

Горбачёнок К. Н.

Проверил

Протько М.И.

Минск, 2020

# Постановка задачи

* Изучить теоретические сведения о блочных шифрах DES и ГОСТ 28147-89
* Создать программу, читающую данные из файла и шифрующие (дешифрующие) их с помощью DES(двойной и тройной) и ГОСТ 28147-89

# Описание используемых алгоритмов

DES (англ. Data Encryption Standard) — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт (FIPS 46-3). Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований. Для DES рекомендовано несколько режимов:

ECB (англ. electronic code book) — режим «электронной кодовой книги» (простая замена);

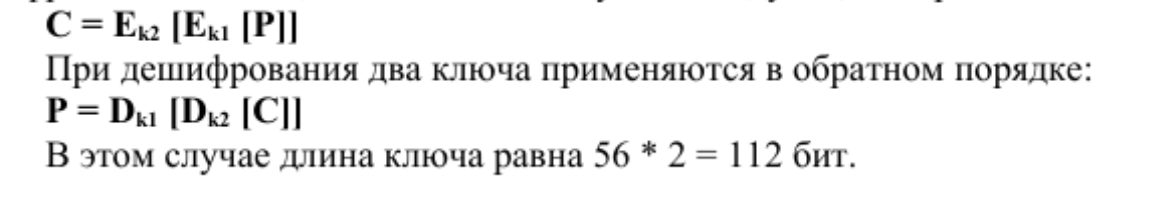
CBC (англ. cipher block chaining) — режим сцепления блоков;

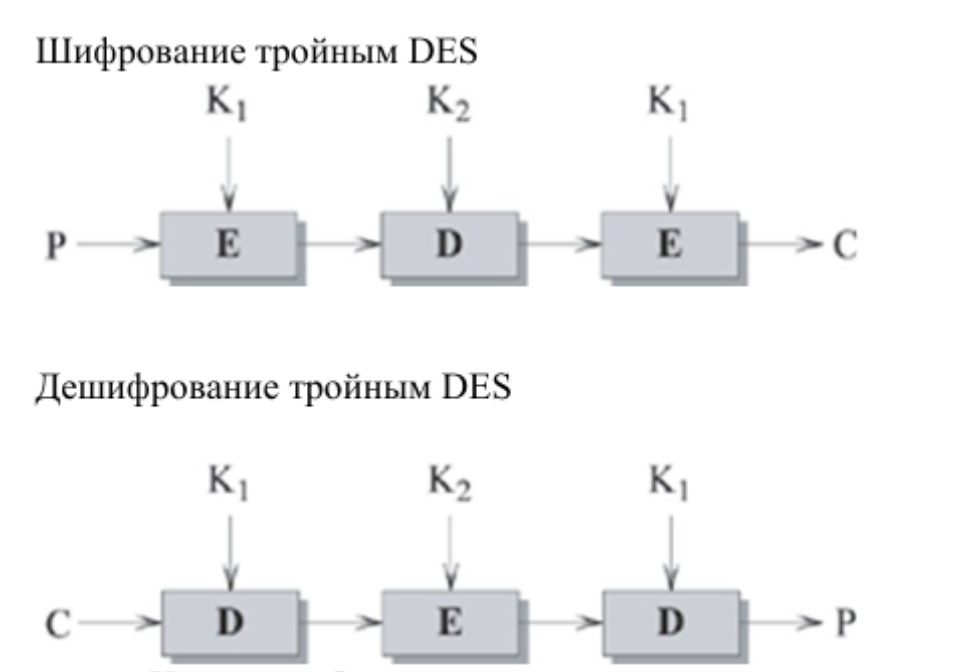
CFB (англ. cipher feed back) — режим обратной связи по шифротексту;

OFB (англ. output feed back) — режим обратной связи по выходу;

Counter Mode (CTR) — режим счётчика.

Прямым развитием DES в настоящее время является алгоритм Triple DES (3DES). В 3DES шифрование/расшифровка выполняются путём троекратного выполнения алгоритма DES.

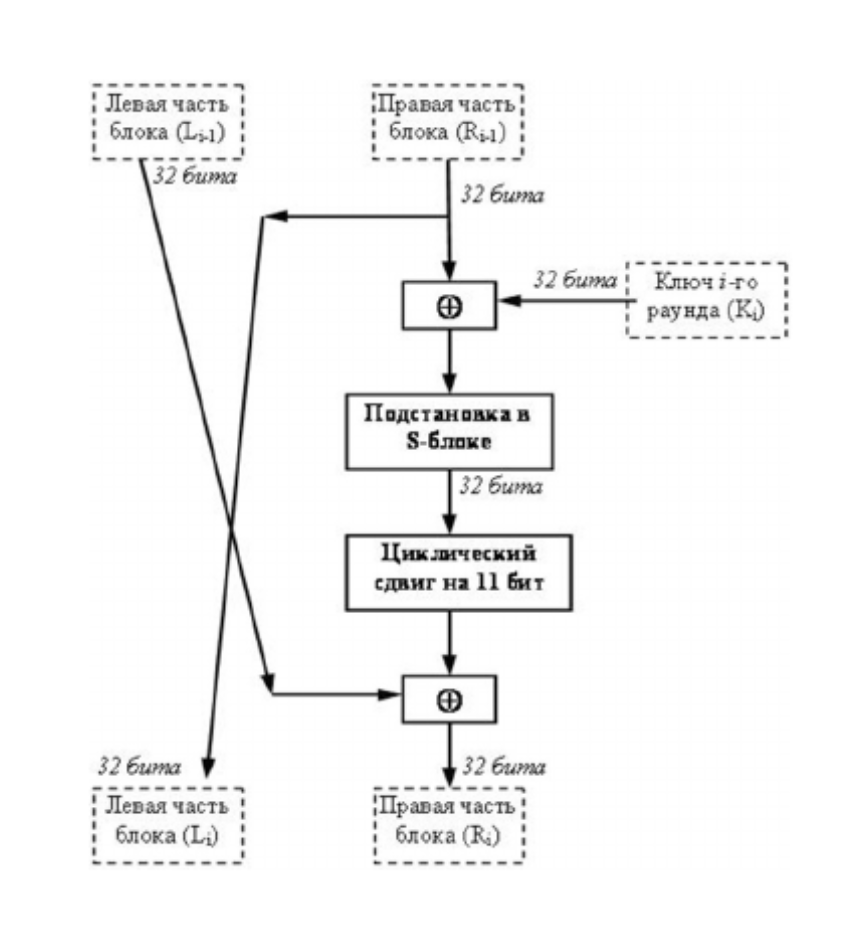
Для увеличения длины ключа DES используют повторно с разным ключами, это называется **двойным DES.**

**Triple DES (3DES)** — симметричный блочный шифр, созданный Уитфилдом Диффи, Мартином Хеллманом и Уолтом Тачманном в 1978 году на основе алгоритма DES с целью устранения главного недостатка последнего — малой длины ключа (56 бит), который может быть взломан методом полного перебора ключа. Скорость работы 3DES в 3 раза ниже, чем у DES, но криптостойкость намного выше — время, требуемое для криптоанализа 3DES, может быть в миллиард раз больше, чем время, нужное для вскрытия DES. 3DES используется чаще, чем DES, который легко взламывается при помощи сегодняшних технологий (в 1998 году организация Electronic Frontier Foundation, используя специальный компьютер DES Cracker, вскрыла DES за 3 дня). 3DES является простым способом устранения недостатков DES. Алгоритм 3DES построен на основе DES, поэтому для его реализации возможно использовать программы, созданные для DES. Официальное название алгоритма, используемое в стандартах - TDEA или Triple DEA (англ. Triple Data Encryption Algorithm). Однако, термин "3DES" используется более широко поставщиками, пользователями и разработчиками криптосистем.

**ГОСТ 28147-89** «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» — устаревший (см. хронологию использования) государственный стандарт союза ССР (а позже межгосударственный стандарт СНГ), описывающий алгоритм симметричного блочного шифрования и режимы его работы.

Является примером DES-подобных криптосистем, созданных по классической итерационной схеме Фейстеля.

# Блок-схемы алгоритмов

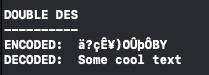
***Блок-схема алгоритма DES***

***Блок-схема алгоритма ГОСТ 28147-89***

# Результаты работы программы

# Исходный файл с текстом:

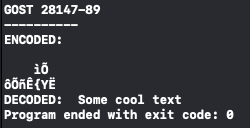


****

**Двойной DES:**

**Тройной DES:**

**ГОСТ 28147-89:**



# Программный код

//

// DES.swift

// mzi

//

// Created by Kirill Gorbachyonok on 9/13/20.

//

**import** Foundation

**class** DES {

**private** **static** **let** pc1 = [

57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

]

**private** **static** **let** shiftTable = [

1: 1, 2: 1, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2,

9: 1, 10: 2, 11: 2, 12: 2, 13: 2, 14: 2, 15: 2, 16: 1

]

**private** **static** **let** pc2 = [

14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

]

**private** **static** **let** ip = [

58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7,

]

**private** **static** **let** e = [

32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12, 13,

12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21, 22,

23, 24, 25, 24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1

]

**private** **static** **let** s1 = [

[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],

[0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],

[4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],

[15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]

]

**private** **static** **let** s2 = [

[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],

[3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],

[0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],

[13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]

]

**private** **static** **let** s3 = [

[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],

[13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],

[13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],

[1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]

]

**private** **static** **let** s4 = [

[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],

[13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],

[10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],

[3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]

]

**private** **static** **let** s5 = [

[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],

[14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],

[4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],

[11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]

]

**private** **static** **let** s6 = [

[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],

[10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],

[9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],

[4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]

]

**private** **static** **let** s7 = [

[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],

[13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],

[1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],

[6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]

]

**private** **static** **let** s8 = [

[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],

[1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],

[7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],

[2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]

]

**private** **static** **let** p = [

16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,

2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25

]

**private** **static** **let** fp = [

40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25

]

**private** **static** **func** keys(for key: String) -> [String] {

**let** kkey = key.binary

**let** newKey = pc1.map({ kkey[$0 - 1] }).reduce("", +)

**var** splitedKeys = [separateOnBlocks(of: 28, this: newKey)]

stride(from: 1, to: 17, by: 1).forEach { i **in**

**let** (ci, di) = (splitedKeys[i - 1][0], splitedKeys[i - 1][1])

splitedKeys.append(

[ci << shiftTable[i]!, di << shiftTable[i]!]

)

}

**return** splitedKeys

.dropFirst()

.map { $0[0] + $0[1] }

.map { key **in**

pc2.map { key[$0 - 1] }.reduce("", +)

}

}

**private** **static** **func** s(**\_** index: Int, bValue: String) -> String {

**let** i = Int(bValue[0] + bValue[-1], radix: 2)!

**let** j = Int(bValue[1..<bValue.count - 1], radix: 2)!

**switch** index {

**case** 1:

**return** binaryConvert(element: s1[i][j])

**case** 2:

**return** binaryConvert(element: s2[i][j])

**case** 3:

**return** binaryConvert(element: s3[i][j])

**case** 4:

**return** binaryConvert(element: s4[i][j])

**case** 5:

**return** binaryConvert(element: s5[i][j])

**case** 6:

**return** binaryConvert(element: s6[i][j])

**case** 7:

**return** binaryConvert(element: s7[i][j])

**case** 8:

**return** binaryConvert(element: s8[i][j])

**default**:

fatalError("WRONG INDEX")

}

}

**private** **static** **func** f(**\_** block: String, **\_** key: String) -> String {

assert(block.count == 32 && key.count == 48)

**let** extended = e.map { block[$0 - 1] }.reduce("", +)

**let** xor = key ^ extended

**let** sBloxUsed = separateOnBlocks(of: 6, this: xor)

.enumerated()

.map { s($0 + 1, bValue: $1) }

.reduce("", +)

**return** p.map({ sBloxUsed[$0 - 1] }).reduce("", +)

}

**static** **func** binaryConvert(element: UInt32) -> String {

**let** result = String(element, radix: 2)

**return** String(repeatElement("0", count: 8 - result.count)) + result

}

**static** **func** binaryConvert(element: Int) -> String {

**let** result = String(element, radix: 2)

**return** String(repeating: "0", count: 4 - result.count) + result

}

**static** **func** separateOnBlocks(of size: Int, this string: String) -> [String] {

**var** newString = string

**if** newString.count % size != 0 {

**let** zeroAddAmount = size - (newString.count % size)

newString = String(repeatElement("0", count: zeroAddAmount)) + string

}

**return** newString.returnBlocks(of: size)

}

**public** **static** **func** encode(this string: String, using key: String) -> String {

assert(key.count >= 8, "KEY LENGTH ERROR")

**let** allKeys = keys(for: key)

**let** result = separateOnBlocks(of: 64, this: string.binary).map { s -> String **in**

**let** withPermutations = ip.map({ s[$0 - 1] }).reduce("", +)

**var** rounds = [separateOnBlocks(of: 32, this: withPermutations)]

allKeys.enumerated().forEach { i, key **in**

**let** (prevLi, prevRi) = (rounds[i][0], rounds[i][1])

**let** li = prevRi

**let** ri = prevLi ^ f(prevRi, key)

rounds.append([li, ri])

}

**let** encoded = rounds.last!.reduce("", +)

**return** fp.map({ encoded[$0 - 1] }).reduce("", +)

}

**return** result.reduce("", +).humanReadable

}

**public** **static** **func** decode(this string: String, using key: String) -> String {

assert(key.count >= 8, "KEY LENGTH ERROR")

**let** allKeys = keys(for: key).reversed()

**let** result = separateOnBlocks(of: 64, this: string.binary).map { s -> String **in**

**let** withPermutations = ip.map({ s[$0 - 1] }).reduce("", +)

**var** rounds = [separateOnBlocks(of: 32, this: withPermutations)]

allKeys.enumerated().forEach { i, key **in**

**let** (prevLi, prevRi) = (rounds[i][0], rounds[i][1])

**let** ri = prevLi

**let** li = prevRi ^ f(prevLi, key)

rounds.append([li, ri])

}

**let** encoded = rounds.last!.reduce("", +)

**return** fp.map({ encoded[$0 - 1] }).reduce("", +)

}

**return** result.reduce("", +).humanReadable.filter { $0 != "\0" }

}

}

//

// GOST.swift

// mzi

//

// Created by Kirill Gorbachyonok on 9/28/20.

//

**import** Foundation

**class** GOST {

**private** **static** **let** s = [

4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3,

14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9,

5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11,

7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3,

6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2,

4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14,

13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12,

1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12

]

**public** **static** **func** encode(this string: String, using key: String) -> String {

assert(key.count >= 256, "KEY LENGTH ERROR")

**let** allKeys = keys(for: key)

**let** result = separateOnBlocks(of: 64, this: string.binary).map { s -> String **in**

**let** withPermutations = ip.map({ s[$0 - 1] }).reduce("", +)

**var** rounds = [separateOnBlocks(of: 32, this: withPermutations)]

allKeys.enumerated().forEach { i, key **in**

**let** (prevLi, prevRi) = (rounds[i][0], rounds[i][1])

**let** li = prevRi

**let** ri = prevLi ^ f(prevRi, key)

rounds.append([li, ri])

}

**let** encoded = rounds.last!.reduce("", +)

**return** fp.map({ encoded[$0 - 1] }).reduce("", +)

}

**return** result.reduce("", +).humanReadable

}

**public** **static** **func** decode(this string: String, using key: String) -> String {

assert(key.count >= 256, "KEY LENGTH ERROR")

**let** allKeys = keys(for: key).reversed()

**let** result = separateOnBlocks(of: 64, this: string.binary).map { s -> String **in**

**let** withPermutations = ip.map({ s[$0 - 1] }).reduce("", +)

**var** rounds = [separateOnBlocks(of: 32, this: withPermutations)]

allKeys.enumerated().forEach { i, key **in**

**let** (prevLi, prevRi) = (rounds[i][0], rounds[i][1])

**let** ri = prevLi

**let** li = prevRi ^ f(prevLi, key)

rounds.append([li, ri])

}

**let** encoded = rounds.last!.reduce("", +)

**return** fp.map({ encoded[$0 - 1] }).reduce("", +)

}

**return** result.reduce("", +).humanReadable.filter { $0 != "\0" }

}

}

//

// main.swift

// mzi

//

// Created by Kirill Gorbachyonok on 9/13/20.

//

**import** Foundation

**func** getFileData() -> (String, String, String)? {

**guard** **let** strings = readFromFile(named: "text", withExtension: "txt") **else** { **return** **nil** }

**return** (strings[0], strings[1], strings[2])

}

**func** doubleDES() {

print("\nDOUBLE DES")

print(repeatElement("-", count: 10).reduce("", +))

**guard** **let** (string, key1, key2) = getFileData() **else** { fatalError("Info read error") }

**let** encodedString = DES.encode(this: DES.encode(this: string, using: key1), using: key2)

print("ENCODED: ", encodedString)

**let** decoded = DES.decode(this: DES.decode(this: encodedString, using: key2), using: key1)

print("DECODED: ", decoded)

}

**func** tripleDES() {

print("\nTRIPLE DES")

print(repeatElement("-", count: 10).reduce("", +))

**guard** **let** (string, key1, key2) = getFileData() **else** { fatalError("Info read error") }

**let** encodedString = DES.encode(

this: DES.decode(

this: DES.encode(

this: string,

using: key1

),

using: key2

),

using: key1

)

print("ENCODED: ", encodedString)

**let** decoded = DES.decode(

this: DES.encode(

this: DES.decode(

this: encodedString,

using: key1

),

using: key2

),

using: key1

)

print("DECODED: ", decoded)

}

**func** gost() {

print("\nGOST 28147-89")

print(repeatElement("-", count: 10).reduce("", +))

**guard** **let** (string, key, **\_**) = getFileData() **else** { fatalError("Info read error") }

**let** encodedString = GOST.encode(this: string, using: key)

print("ENCODED: ", encodedString)

**let** decoded = GOST.decode(this: encodedString, using: key)

print("DECODED: ", decoded)

}

doubleDES()

tripleDES()

gost()

# Вывод

В ходе данной лабораторной работы я изучил принципы работы алгоритмы блочного шифрования двойной DES, тройной DES и ГОСТ 28147-89 и реализовал их программно.

Двойной DES позволяет увеличить длину ключа, однако подвержен атаке “встречи посередине”, это проблему решает тройной DES.