УНИВЕРСИТЕТ ИТМО

ФГАОУ ВО «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

Лабораторная работа №2

Блочное симметричное шифрование

Лабушев Тимофей

Группа P3402

Санкт-Петербург

2021

Цель работы

Изучение структуры и основных принципов работы современных алгоритмов блочного симметричного шифрования, приобретение навыков программной реализации блочных симметричных шифров.

Задание

Вариант 7. Реализовать систему симметричного блочного шифрования, позволяющую шифровать и дешифровать файл на диске с использованием блочного шифра **TEA** в режиме шифрования **CBC**.

Описание шифра

TEA является блочным шифром на основе сети Фейстеля с 64-битными блоками и 128-битным ключом.

При шифровании блок и ключ разделяются на две половины. На одной итерации шифра считается два раунда. Сначала правая половина блока изменяется функцией от первой половины ключа, результат складывается по модулю 2^32 с левой половиной блока. Затем левая половина блока изменяется функцией от второй половины ключа, результат складывается с правой половиной блока. Как можно заметить, отличием **TEA** от других шифров на основе сети Фейстеля является использование сложения по модулю 2^32, а не 2, для аккумуляции результата.

В функции преобразования используются битовые сдвиги (блок перестановок) и сложение с дополнительной константой, выведенной из золотого сечения (0x9e3779b9). Константа домножается на индекс итерации, что способствует предотвращению определенного класса атак.

Описание режима шифрования

Режим шифрования описывает то, как блочный шифр применяется к исходным данным размером более одного блока.

В режиме **CBC** каждый последующий блок *перед шифрованием* складывается по модулю 2 с предыдущим *зашифрованным* блоком. При обработке первого блока должна использоваться уникальная последовательность (**IV** — *initialization vector*, вектор инициализации).

При дешифровании, *дешифрованный блок* складывается по модулю 2 с предыдущим *зашифрованным блоком* (IV при обработке первого блока). Таким образом, при неправильном указании IV будет утерян только первый блок данных.

Интерфейс взаимодействия с программой шифрования

Режим: Зашифровать

Число итераций: 32

Ключ (0x...): 0x313

Вектор инициализации (0x...): 0x42

Файл: Choose File aaiw-excerpt

Результат работы программы

Размер данных: 304 байт

Результат: Download... aaiw-excerpt

```
tea_decrypt (generic function with 1 method)
```

```
begin
    using Test, PlutoUI

    struct CipherConfig
        key::UInt128
        num_iterations::UInt32
    end

    function tea_encrypt(config::CipherConfig, block::UInt64)::UInt64
        k1::UInt32, k2::UInt32, k3::UInt32, k4::UInt32 =
            (config.key >> 96 % UInt32,
             config.key >> 64 % UInt32,
             config.key >> 32 % UInt32,
             config.key % UInt32)
        # Split the block into the left and right halves
        l::UInt32, r::UInt32 = (block >> 32 % UInt32, block % UInt32)

        # Delta, derived from the golden ratio
        delta::UInt32 = 0x9e3779b9
        # Sum accumulates delta * current iteration in a variable
        # to avoid the multiplication on each iteration
        sum::UInt32 = 0

        for i in 1:1:config.num_iterations
            sum += delta
            l += (r << 4 + k1) ⊻ (r + sum) ⊻ (r >> 5 + k2)
            r += (l << 4 + k3) ⊻ (l + sum) ⊻ (l >> 5 + k4)
        end

        # Put the left and right halves back into a single block, now encrypted
        UInt64(l) << 32 | r
    end

    function tea_decrypt(config::CipherConfig, block::UInt64)::UInt64
        k1::UInt32, k2::UInt32, k3::UInt32, k4::UInt32 =
            (config.key >> 96 % UInt32,
             config.key >> 64 % UInt32,
             config.key >> 32 % UInt32,
             config.key % UInt32)
        l::UInt32, r::UInt32 = (block >> 32 % UInt32, block % UInt32)

        delta::UInt32 = 0x9e3779b9
        # To reverse the addition of delta, the value on each iteration is
        # computed as `delta * (num iterations - current iteration)`
        log2_num_rounds = 31 - leading_zeros(config.num_iterations)
        sum::UInt32 = delta << log2_num_rounds

        for i in 1:1:config.num_iterations
            r -= (l << 4 + k3) ⊻ (l + sum) ⊻ (l >> 5 + k4)
            l -= (r << 4 + k1) ⊻ (r + sum) ⊻ (r >> 5 + k2)
            sum -= delta
        end

        UInt64(l) << 32 | r
    end
end
```

```
cbc_decrypt (generic function with 1 method)
```

```
begin
    function cbc_encrypt(config::CipherConfig, iv::UInt64, i::IO, o::IO)
        xor_with = iv

        while !eof(i)
            block = read(i, UInt64)
            block ⊻= xor_with
            enc_block = tea_encrypt(config, block)
            write(o, enc_block)
            xor_with = enc_block
        end
    end

    function cbc_decrypt(config::CipherConfig, iv::UInt64, i::IO, o::IO)
        xor_with = iv

        while !eof(i)
            enc_block = read(i, UInt64)
            block = tea_decrypt(config, enc_block)
            block ⊻= xor_with
            write(o, block)
            xor_with = enc_block
        end
    end
end
```

Тестирование алгоритма

```
Test Summary:             | Pass  Total
encryption and decryption |    2      2
```

```
with_terminal() do
    @testset "encryption and decryption" begin
        conf = CipherConfig(UInt128(0x666), UInt32(32))
        iv = UInt64(0x313)

        orgbuf = IOBuffer(reinterpret(UInt8, [1,2,3,4,5,6]))
        encbuf = IOBuffer()
        decbuf = IOBuffer()

        cbc_encrypt(conf, iv, orgbuf, encbuf)
        seekstart(encbuf)
        cbc_decrypt(conf, iv, encbuf, decbuf)

        @test reinterpret(UInt64, encbuf.data) == [
            0x61137e4fdb64c60b, 0xda3ca992451598ef, 0x9c3928cfffa5794d,
            0x6110330d7c396aae, 0xae22265919967777, 0xfb0e9c9dc7055110
        ]
        @test orgbuf.data == decbuf.data
    end
end
```

Вывод

В ходе выполнения работы была рассмотрена теоретическая основа блочного шифра TEA и режима шифрования CBC. Реализованы алгоритмы шифрования и дешифрования.