Федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский университет ИТМО"

Лабораторная работа №2

по дисципение "Информационная безопасность"

Блочное симметричное шифрование

Вариант 10

Выполнил: студент Саржевский И.А.

Группа: P3402

Преподаватель: к.т.н., доцент
Маркина Т.А.

г. Санкт-Петербург

2021 г.

Лабораторная работа №2

Блочное симметричное шифрование

Цель работы

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

Задание

Вариант: 4(д). Реализовать систему блочного шифрования, позволяющую шифровать и дешифровать файл на диске с использованием блочного шифра ГОСТ 28147-89 в режиме шифрования OFB.

Данный блочный шифр реализован на основе итерационной схемы Фейстеля. Для шифрования необходимо задать один 256-битный ключ, который в последствии разбивается на 8 32-х битных подключей, которые используются в 32-х раундах шифрования в определенной последовательности - в первых 23-х раундах подключи циклически повторяются по очереди, и в последнем проведении подключи выбираются с начиная с последнего. На каждом раунде шифрования, блок данных складывается с ключом по модулю 2^{32}, и результат подается на узлы таблицы замен. Нет четкого требования к алгоритму формирования этих узлов, было принято решение взять узлы замен определенные Техническим комитетом по стандартизации 'Криптографическая защита информации' Росстандарта. Полученное на выходе число циклически сдвигается на 11 разрядов вправо.

Режим шифрования OFB предполагает выработку гаммы. Зашифрованный текст получается применением операции сложения по модулю 2 с блоком открытого текста. Каждая последующая гамма, использующаяся для шифрования следующего блока данных, вырабатывается на основании предыдущей используя алгоритм шифрования (в данном случае ГОСТ 28147-89). При этом, первая гамма вырабатывается на основании блока данных, задаваемого извне (IV). Расшифровка производится аналогично шифрованию, при использовании одинакового IV повторное шифрование зашифрованного текста произведет исходные данные.

Для инициализации работы алгоритма требовались две константы извне - IV для генерации первой гаммы и 256-битный ключ. Было принято решение, что система будет принимать файл с секретной фразой, первый 8 байт которой будут считаться за IV, а следующие 32 байта - за ключ. Таким образом, стороны передачи должны договориться только об этой секретной фразе. Исходя из назначения, данная фраза должна иметь длинну не менее 40 байт, это 40 латинских символов в кодировке UTF8.

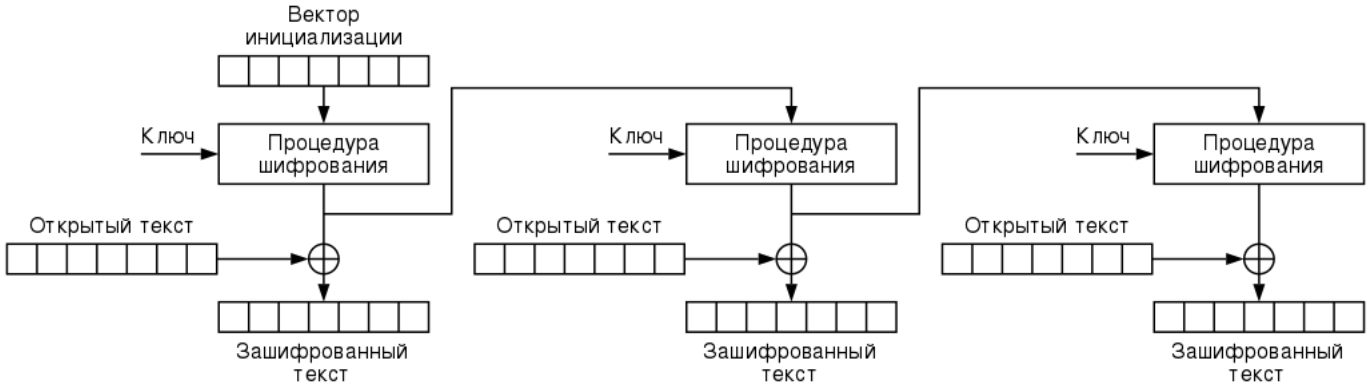

Рис. 1: Схема шифрования OFB

Листинг разработанной программы

main.rs

```
1  extern crate getopts;
2  use getopts::Options;
3  use std::env;
4
5  mod encrypt;
6
7  // Just parsing cmd args and calling encrypt function
8  fn main() {
9      let args: Vec<String> = env::args().collect();
10
11     let mut opts = Options::new();
12     opts.optopt("f", "file", "input file path", "input.txt");
13     opts.optopt("s", "sec_file", "passphase file path", "secret.txt");
14     opts.optopt("o", "out_file", "output file path", "output");
15
16     let matches = match opts.parse(&args[1..]) {
17         Ok(m) => { m }
18         Err(f) => { panic!(f.to_string()) }
19     };
20
21     if !matches.opt_present("f") {
22         println!("You must specify the input file path!");
23         return;
24     }
25
26     if !matches.opt_present("s") {
27         println!("You must specify the passphrase file path!");
28         return;
29     }
30
31     if !matches.opt_present("o") {
32         println!("You must specify the output file path!");
33         return;
34     }
35
36     let input_file = matches.opt_str("f").unwrap();
37     let output_file = matches.opt_str("o").unwrap();
38     let secret_file = matches.opt_str("s").unwrap();
39
40     let (iv, k) = encrypt::get_iv_and_ks(secret_file)
41         .expect("Secret phrase must contain at least 40 characters!");
42
43     encrypt::encrypt(input_file, output_file, iv, k);
44 }
```

encrypt.rs

```
use std::fs;
use std::io::prelude::*;
use std::fs::File;

// Replacement units according to ГОСТ 34.12-2018
const S: [[u32; 16]; 8] = [
    [0xC, 0x4, 0x5, 0x2, 0xA, 0x5, 0xB, 0x9, 0xE, 0x8, 0xD, 0x7, 0x0, 0x3, 0xF, 0x1],
    [0x6, 0x8, 0x2, 0x3, 0x9, 0xA, 0x5, 0xC, 0x1, 0xE, 0x4, 0x7, 0xB, 0xD, 0x0, 0xF],
    [0xB, 0x3, 0x5, 0x8, 0x2, 0xF, 0xA, 0xD, 0xE, 0x1, 0x7, 0x4, 0xC, 0x9, 0x6, 0x0],
    [0xC, 0x8, 0x2, 0x1, 0xD, 0x4, 0xF, 0x6, 0x7, 0x0, 0xA, 0x5, 0x3, 0xE, 0x9, 0xB],
    [0x7, 0xF, 0x5, 0xA, 0x8, 0x1, 0x6, 0xD, 0x0, 0x9, 0x3, 0xE, 0xB, 0x4, 0x2, 0xC],
    [0x5, 0xD, 0xF, 0x6, 0x9, 0x2, 0xC, 0xA, 0xB, 0x7, 0x8, 0x1, 0x4, 0x3, 0xE, 0x0],
    [0x8, 0xE, 0x2, 0x5, 0x6, 0x9, 0x1, 0xC, 0xF, 0x4, 0xB, 0x0, 0xD, 0xA, 0x3, 0x7],
    [0x1, 0x7, 0xE, 0xD, 0x0, 0x5, 0x8, 0x3, 0x4, 0xF, 0xA, 0x6, 0x9, 0xC, 0xB, 0x2]
];

// check array boundaries, return 0 if out of range
fn get_array_value(array: &Vec<u8>, i: usize, offset: usize) -> u64 {
    let len = array.len();
    if i + offset >= len {
        return 0;
    }
    return array[i + offset] as u64;
}

// convert 8 bytes by given offset to u64 number
fn as_u64_le(array: &Vec<u8>, offset: usize) -> u64 {
    return
        (get_array_value(array, 0, offset) <<  0) +
        (get_array_value(array, 1, offset) <<  8) +
        (get_array_value(array, 2, offset) << 16) +
        (get_array_value(array, 3, offset) << 24) +
        (get_array_value(array, 4, offset) << 32) +
        (get_array_value(array, 5, offset) << 40) +
        (get_array_value(array, 6, offset) << 48) +
        (get_array_value(array, 7, offset) << 56);
}

// convert 4 bytes by given offset to u32 number
fn as_u32_le(array: &Vec<u8>, offset: usize) -> u32 {
    return
        ((array[0 + offset] as u32) <<  0) +
        ((array[1 + offset] as u32) <<  8) +
        ((array[2 + offset] as u32) << 16) +
        ((array[3 + offset] as u32) << 24);
}

// the encryption function according to ГОСТ 28147-89
fn gost_func(block: u32, key: u32) -> u32 {
    let tmp = (block as u64 + key as u64) as u32;
    let mut mask: u32 = 0x0000000F;
    let mut result: u32 = 0;
    for i in 0..8 {
        let cur_num = (tmp & mask) >> (i * 4);
        result |= S[i][cur_num as usize] << (i * 4);
        mask = mask << 4;
    }
    return result.rotate_left(11);
}

// basic implementation of Feistel cipher
fn gost(block: u64, k: [u32; 8]) -> u64 {
    let mut left = (block >> 32) as u32;
    let mut right = block as u32;
    for i in 0..32 {
        let cur_key = if i < 24 {k[i % 8]} else {k[7 - (i % 8)]};
        let tmp = right ^ gost_func(left, cur_key);
        right = left;
        left = tmp;
    }
    return (right as u64) | ((left as u64) << 32);
```

```
}

// the OFB implementation
pub fn encrypt(filename: String, out_filename: String, iv: u64, k: [u32; 8]) {
    let file_bytes = fs::read(filename)
        .expect("Cannot read input file, check the spelling");

    let byte_count = file_bytes.len();
    let mut encoded_bytes = 0;

    let mut res: Vec<u8> = Vec::new();

    let mut gamma = gost(iv, k); // calc first gamma from iv and k
    let mut block = as_u64_le(&file_bytes, 0); // get the first block of text
    let mut encoded = block ^ gamma; // encode the text
    res.extend_from_slice(&encoded.to_le_bytes()); // save encoded bytes
    encoded_bytes += 8;
    // main encoding loop
    while encoded_bytes < byte_count {
        gamma = gost(gamma, k); // calc new gamma from old one and k
        block = as_u64_le(&file_bytes, encoded_bytes); // get the next block of text
        encoded = block ^ gamma; // encode the text
        res.extend_from_slice(&encoded.to_le_bytes()); // save encoded bytes
        encoded_bytes += 8;
    }

    let mut pos = 0;
    let mut buffer = File::create(out_filename).unwrap();
    // dump the encoded bytes to the output file
    while pos < byte_count {
        let bytes_written = buffer.write(&res[pos..byte_count]).unwrap();
        pos += bytes_written;
    }
}

// to initiate 64 bit IV and 256 bit key I use the file
// with memorable passphrase. It needs to be at least
// 40 bytes long to fit 8 bytes of IV and 32 bytes of key
pub fn get_iv_and_ks(filename: String) -> Option<(u64, [u32; 8])> {
    let bytes = fs::read(filename)
        .expect("Cannot read passphrase file, check the spelling");
    if bytes.len() <= 40 {
        return None;
    }
    let iv = as_u64_le(&bytes, 0); // use first 8 bytes of file as IV
    let mut k: [u32; 8] = [0; 8]; // use next 32 bytes of file as 8 4-byte subkeys
    for i in 0..8 {
        k[i] = as_u32_le(&bytes, 8 + i * 4);
    }

    return Some((iv, k));
}
```

Результаты работы программы

```
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳ cat secret.txt
All work and no play makes Jack a dull boy
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳ head -c 400 input.txt
A merry little surge of electricity piped by automatic alarm from the mood organ beside his bedawakened
 Rick  Deckard.  Surprised  -  it  always  surprised  him  to  find  himself  awake  without  priornotic
e - he rose from the bed, stood up in his multicolored pajamas, and stretched. Now, in her bed,his wife
Iran opened her gray, unmerry eyes, blinked, then groaned and shut her eyes again.    "Y%
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳ ./lab2 -f input.txt -s secret.txt -o output
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳ head -c 20 output
[illegible]
{%
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳ ./lab2 -f output -s secret.txt -o decrypted.txt
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳ head -c 400 decrypted.txt
A merry little surge of electricity piped by automatic alarm from the mood organ beside his bedawakened
 Rick  Deckard.  Surprised  -  it  always  surprised  him  to  find  himself  awake  without  priornotic
e - he rose from the bed, stood up in his multicolored pajamas, and stretched. Now, in her bed,his wife
Iran opened her gray, unmerry eyes, blinked, then groaned and shut her eyes again.    "Y%
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳ diff input.txt decrypted.txt
keker (~/code/itmo-4th-year/infosec/lab2/target/debug) master
↳
```

Рис. 2: Результат работы программы

Вывод

В ходе выполнения данной лабораторной работы были изучены принципы работы блочно-симметричных алгоритмов шифрования, а также разработана программа, имплементирующая шифр ГОСТ 28147-89 в режиме шифрования OFB.