Лабораторная работа №7

Элементы криптографии. Однократное гаммирование

Дмитрий Сергеевич Шестаков

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Освоить на практике применение режима однократного гаммирования.

2 Теоретическое введение

Предложенная Г. С. Вернамом так называемая «схема однократного использования (гаммирования)» (рис. 7.1) является простой, но надёжной схемой шифрования данных. Гаммирование представляет собой наложение (снятие) на открытые (зашифрованные) данные последовательности элементов других данных, полученной с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных. Иными словами, наложение гаммы — это сложение её элементов с элементами открытого (закрытого) текста по некоторому фиксированному модулю, значение которого пред- ставляет собой известную часть алгоритма шифрования. В соответствии с теорией криптоанализа, если в методе шифрования используется однократная вероятностная гамма (однократное гаммирование) той же длины, что и подлежащий сокрытию текст, то текст нельзя раскрыть. Даже при раскрытии части последовательности гаммы нельзя получить информацию о всем скрываемом тексте.

Наложение гаммы по сути представляет собой выполнение операции сложения по модулю 2 (XOR) (обозначаемая знаком $\oplus$) между элементами гаммы и элементами подлежащего сокрытию текста. Напомним, как рабо- тает операция XOR над битами: $0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0$.

Такой метод шифрования является симметричным, так как двойное прибавление одной и той же величины по модулю 2 восстанавливает исходное. Открытый текст имеет символьный вид, а ключ — шестнадцатеричное представление. Ключ также можно представить в символьном виде, воспользовавшись таблицей ASCII-кодов. [1]

3 Выполнение лабораторной работы

Написали программу на языке C++

1. Написали функцию шифрующую входную строку по ключу

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <cassert>

std::string Codec(const std::vector<std::string>& key,
                  const std::string& input
                    ) {
    assert(key.size() == input.size());

    std::string res;

    for(size_t i = 0; i < input.size(); ++i)
    {
        int64_t ki, ci;

        std::istringstream iss(key[i]);

        iss >> std::hex >> ki;

        ci = int(input[i]);

        res += char(ci ^ ki);
    }

    return res;
}
```

Рис. 3.1: Шифрование

2. Написали функцию вычисляющую ключ по входной строке и зашифрованному сообщению

```cpp
std::vector<std::string> GetKey(const std::string& input,
                                const std::string& output
                                ){
    assert(input.size() == output.size());
    std::vector<std::string> key;
    key.reserve(input.size());

    for(size_t i = 0; i < input.size(); ++i) {
        int64_t ii, oi, ki;

        ii = int(input[i]);
        oi = int(output[i]);

        ki = ii ^ oi;

        std::ostringstream ss;
        ss << std::hex << ki;

        key.push_back(ss.str());
    }

    return key;
}
```

Рис. 3.2: Поиск ключа

3. Получили следующий ключ

```
❯ g++ -std=c++20 crypto.cpp
❯ ./a.out
Input msg: Dima
Key: 05 17 7f 0e
Coded msg: A~o
Coded size: 4
Recovered key: 5 17 7f e %
```

Рис. 3.3: Результат

4 Выводы

Освоили на практике применение режима однократного гаммирования.

Список литературы

1. Кулябов Д.С., Королькова А. В., Геворкян М. Н. Лабораторная работа No 6. Мандатное разграничение прав в Linux [Электронный ресурс]. URL: https://esystem.rudn.ru/pluginfile.php/2090210/mod_resource/content/2/006-lab_selinux.pdf.