Лабораторная работа №3

Математические основы защиты информации и информационной безопасности

Николаев Дмитрий Иванович, НПМмд-02-24

Содержание

1 Цель работы **5**

2 Теоретическое введение **6**
2.1 Шифрование гаммированием 6

3 Выполнение лабораторной работы **9**

4 Выводы **11**

Список литературы **12**

Список иллюстраций

3.1 Код реализации алгоритма шифрования конечной гаммой на Julia 9
3.2 Код реализации алгоритма дешифрования сообщения, зашифрованного конечной гаммой на Julia 10
3.3 Результат кода реализации алгоритма шифрования конечной гаммой на Julia 10

Список таблиц

1 Цель работы

Изучить работу алгоритма шифрования гаммированием конечной гаммой, а также реализовать его программно.

2 Теоретическое введение

2.1 Шифрование гаммированием

Из всех схем шифрования простейшей и наиболее надежной является схема однократного использования. Формируется m- разрядная случайная двоичная последовательность — ключ шифра. Отправитель производит побитовое сложение по модулю два (mod 2) ключа

$$k = k_1 k_2 \dots k_i \dots k_m$$

и m-разрядной двоичной последовательности:

$$p = p_1 p_2 \dots p_i \dots p_m,$$

соответствующей посылаемому сообщению:

$$c_i = p_i \oplus k_i, i = \overline{1, m},$$

где p_i — i-й бит исходного текста, k_i — i-й бит ключа, $\oplus$ – операция побитового сложения (XOR), c_i — i-й бит получившейся криптограммы

$$c = c_1 c_2 \dots c_i \dots c_m.$$

Операция побитного сложения является обратимой, т.е. $(x \oplus y) \oplus y = x$, поэтому дешифрование осуществляется повторным применением операции $\oplus$

к криптограмме

$$p_i = c_i \oplus k_i, i = \overline{1, m}.$$

Основным недостатком такой схемы является равенство объема ключевой информации и суммарного объема передаваемых сообщений. Данный недостаток можно убрать, использовав ключ в качестве “зародыша”, порождающего значительно более длинную ключевую последовательность.

Гаммирование — процедура наложения при помощи некоторой функции F на исходный текст гаммы шифра, т.е. псевдослучайной последовательности (ПСП) с выходом генератора G. Псевдослучайная последовательность по своим статистическим свойствам неотличима от случайной последовательности, но является детерминированной, т.е. известен алгоритм ее формирования. Чаще всего в качестве функции F берется операция поразрядного сложения по модулю два или по модулю N (N — число букв алфавита открытого текста).

Простейший генератор псевдослучайной последовательности можно представить рекуррентным соотношением:

$$\gamma_i = a \cdot \gamma_{i-1} + b \,(\text{mod}\, m), i = 1, \dots, m,$$

где γ_i — i-й член последовательности псевдослучайных чисел, a, γ_0, b — ключевые параметры. Такая последовательность состоит из целых чисел от 0 до $m - 1$. Если элементы γ_i и γ_j совпадут, то совпадут и последующие участки: $\gamma_{i+1} = \gamma_{j+1}$, $\gamma_{i+2} = \gamma_{j+2}$. Таким образом, ПСП является периодической. Знание периода гаммы существенно облегчает криптоанализ. Максимальная длина периода равна m. Для ее достижения необходимо удовлетворить следующим условиям:

1. b и m — взаимно простые числа;
2. $a - 1$ делится на любой простой делитель числа m;
3. $a - 1$ кратно 4, если m кратно 4.

Стойкость шифров, основанных на процедуре гаммирования, зависит от характеристик гаммы — длины и равномерности распределения вероятностей появления знаков гаммы.

При использовании генератора ПСП получаем бесконечную гамму. Однако, возможен режим шифрования конечной гаммы. В роли конечной гаммы может выступать фраза. Как и ранее, используется алфавитный порядок букв, т.е. буква “а” имеет порядковый номер 1, “б” — 2 и т.д.

Например, зашифруем слово “ПРИКАЗ” (“16 17 09 11 01 08”) гаммой “ГАММА” (“04 01 13 13 01”). Будем использовать операцию побитового сложения по модулю 33 (mod 33). Получаем:

$$
\begin{aligned}
c_1 &= 16 + 4 \mod 33 = 20 \\
c_2 &= 17 + 1 \mod 33 = 18 \\
c_3 &= 9 + 13 \mod 33 = 22 \\
c_4 &= 11 + 13 \mod 33 = 24 \\
c_5 &= 1 + 1 \mod 33 = 2 \\
c_6 &= 8 + 4 \mod 33 = 12
\end{aligned}
$$

Криптограмма: “УСХЧБЛ” (“20 18 22 24 02 12”).

3 Выполнение лабораторной работы

Следуя указаниям [1], реализуем алгоритм шифрования гаммированием конечной гаммой и его расшифрование на Julia ([3.1,3.2]), в результате получим следующий вывод ([3.3]).

```julia
# Функция для преобразования строки в числовое представление на основе алфавитного порядка
alphabet = 'а':'я'
function Text_to_Numbers(Text::String, Alphabet::StepRange{Char, Int64} = alphabet)::Vector{Any}
    numbers = []
    for char in lowercase(Text)
        push!(numbers, findfirst(c -> c == char, Alphabet))
    end
    return numbers
end

# Функция для преобразования числового представления обратно в строку
function Numbers_to_Text(Numbers::Vector{Any}, Alphabet::StepRange{Char, Int64} = alphabet)::String
    text = ""
    for number in Numbers
        text *= alphabet[number]
    end
    return lowercase(text)
end

"""Шифрование конечной гаммой"""
function Cipher_Gamma(Message::String, Gamma::String, Alphabet::StepRange{Char, Int64} = alphabet)::String
    Message_Numbers = Text_to_Numbers(Message, Alphabet)
    Gamma_Numbers = Text_to_Numbers(Gamma, Alphabet)
    modulo = length(Alphabet)
    Encrypted_Numbers = []
    for i ∈ 1:length(Message_Numbers)
        encrypted_number = (Message_Numbers[i] + Gamma_Numbers[(i-1) % length(Gamma_Numbers) + 1]) % modulo
        push!(Encrypted_Numbers, encrypted_number == 0 ? modulo : encrypted_number)
    end
    return Numbers_to_Text(Encrypted_Numbers, Alphabet)
end

```

Рис. 3.1: Код реализации алгоритма шифрования конечной гаммой на Julia

```julia
"""Дешифрование сообщения, зашифрованного конечной гаммой"""
function Decipher_Gamma(Encrypted_Message::String, Gamma::String, Alphabet::StepRange{Char, Int64} = alphabet)::String
    Encrypted_Numbers = Text_to_Numbers(Encrypted_Message, Alphabet)
    Gamma_Numbers = Text_to_Numbers(Gamma, Alphabet)
    modulo = length(Alphabet)
    Message_Numbers = []
    for i ∈ 1:length(Encrypted_Numbers)
        message_number = (Encrypted_Numbers[i] - Gamma_Numbers[(i-1) % length(Gamma_Numbers) + 1]) % modulo
        push!(Message_Numbers, message_number == 0 ? modulo : message_number)
    end
    return Numbers_to_Text(Message_Numbers, Alphabet)
end

# Пример. Сообщение для шифрования
message = "ПРИКАЗ"  # ("16 17 09 11 01 08")
gamma = "ГАММА"     # ("04 01 13 13 01")
println("Исходное сообщение: ", message, "; Конечная гамма шифрования: ", gamma)
# Шифрование
ciphertext = Cipher_Gamma(message, gamma)
println("Зашифрованное сообщение: ", ciphertext)
# Дешифрование
decrypted_message = Decipher_Gamma(ciphertext, gamma)
println("Расшифрованное сообщение: ", decrypted_message)
```

gamma::String = "ГАММА"

Рис. 3.2: Код реализации алгоритма дешифрования сообщения, зашифрованного конечной гаммой на Julia

```
PS C:\Users\User\Documents\work\study\2024-2025\Математические
thbase-infosec\labs\lab03\report\report> julia .\gamma.jl
Исходное сообщение: ПРИКАЗ; Конечная гамма шифрования: ГАММА
Зашифрованное сообщение: усхчбл
Расшифрованное сообщение: приказ
```

Рис. 3.3: Результат кода реализации алгоритма шифрования конечной гаммой на Julia

4 Выводы

В ходе выполнения лабораторной работы я изучил работу алгоритма шифрования гаммированием конечной гаммой, а также реализовал его программно.

Список литературы

1. Лабораторная работа № 3. Шифрование гаммированием [Электронный ресурс]. Саратовский государственный университет имени Н.Г. Чернышевского, 2024.