Лабораторная работа №2

Шифрование методом гаммирования с конечной гаммой

Дисциплина: Математические основы защиты информации и информационной безопасности (МОЗИиИБ)
Автор: Фатеева Елизавета Артёмовна
Группа: НПИмд-01-24
Преподаватель: Кулябов Дмитрий Сергеевич
Дата выполнения: 11.10.2025

Оглавление

Теоретическое введение

Метод гаммирования

Гаммирование — это симметричный метод шифрования, при котором каждый символ открытого текста комбинируется с символом **гаммы** (ключа) по правилу модульной арифметики. При использовании **конечной гаммы** ключ циклически повторяется до длины исходного сообщения.

Математически:

- **Шифрование**:
  [
  C_i = (P_i + G_i) \mod N
  ]
- **Дешифрование**:
  [
  P_i = (C_i - G_i) \mod N
  ]

где:

- (P_i) — индекс i-го символа открытого текста,
- (C_i) — индекс i-го символа шифротекста,
- (G_i) — индекс i-го символа гаммы,
- (N) — мощность алфавита языка символа.

Особенности реализации

- Поддерживаются **русский** (включая **ё/Ё**) и **английский** алфавиты.
- Русский алфавит содержит **33 буквы**, английский — **26**.
- Буква **ё** идёт сразу после **е**, **Ё** — после **Е**.
- Неалфавитные символы (пробелы, знаки препинания, цифры) **не шифруются**.
- Регистр и язык исходного символа **сохраняются** в шифротексте.

Цели и задачи

Цель работы:

Изучение и практическая реализация **метода гаммирования с конечной гаммой** на языке программирования **Julia** с поддержкой русского и английского алфавитов.

Задачи:

1. Реализовать функции шифрования и дешифрования с учётом особенностей Unicode и кириллицы.
2. Обеспечить корректную обработку букв **ё/Ё** как отдельных символов.
3. Сохранять регистр и неалфавитные символы без изменений.
4. Протестировать работу алгоритма на различных примерах, включая смешанный текст.
5. Провести ручной расчёт для верификации корректности.

Реализация шифрования методом гаммирования

Код программы

```
# Алфавиты как векторы символов (безопасная индексация)
const RU_LOWER = collect("абвгдеёжзийклмнопрстуфхцчшщъыьэюя")
const RU_UPPER = collect("АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ")
const EN_LOWER = collect("abcdefghijklmnopqrstuvwxyz")
const EN_UPPER = collect("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

const LEN_RU = length(RU_LOWER)  # 33
const LEN_EN = length(EN_LOWER)  # 26

# Словари для быстрого поиска индексов
const RU_LOWER_IDX = Dict(c => i-1 for (i, c) in enumerate(RU_LOWER))
const RU_UPPER_IDX = Dict(c => i-1 for (i, c) in enumerate(RU_UPPER))
const EN_LOWER_IDX = Dict(c => i-1 for (i, c) in enumerate(EN_LOWER))
const EN_UPPER_IDX = Dict(c => i-1 for (i, c) in enumerate(EN_UPPER))

function get_char_index(c::Char)
    return get(RU_LOWER_IDX, c,
           get(RU_UPPER_IDX, c,
           get(EN_LOWER_IDX, c,
           get(EN_UPPER_IDX, c, nothing))))
end
```

```
function get_alphabet_type(c::Char)
    if haskey(RU_LOWER_IDX, c) || haskey(RU_UPPER_IDX, c)
        return :ru
    elseif haskey(EN_LOWER_IDX, c) || haskey(EN_UPPER_IDX, c)
        return :en
    else
        return :other
    end
end

function char_from_index(idx::Int, original_char::Char)
    if haskey(RU_LOWER_IDX, original_char)
        return RU_LOWER[idx + 1]
    elseif haskey(RU_UPPER_IDX, original_char)
        return RU_UPPER[idx + 1]
    elseif haskey(EN_LOWER_IDX, original_char)
        return EN_LOWER[idx + 1]
    else
        return EN_UPPER[idx + 1]
    end
end

function encrypt(plaintext::String, gamma::String)::String
    isempty(gamma) && throw(ArgumentError("Гамма не может быть пустой"))
    gamma_chars = collect(gamma)
    gamma_indices = Int[]
    for c in gamma_chars
        idx = get_char_index(c)
        idx === nothing && throw(ArgumentError("Символ гаммы '$c' не
поддерживается."))
        push!(gamma_indices, idx)
    end

    plaintext_chars = collect(plaintext)
    ciphertext = Char[]

    for (i, c) in enumerate(plaintext_chars)
        c_type = get_alphabet_type(c)
        if c_type == :other
            push!(ciphertext, c)
        else
            c_idx = get_char_index(c)::Int
            g_idx = gamma_indices[mod1(i, length(gamma_indices))]
            new_idx = if c_type == :ru
                mod(c_idx + g_idx, LEN_RU)
            else
                mod(c_idx + g_idx, LEN_EN)
            end
            push!(ciphertext, char_from_index(new_idx, c))
        end
    end
    return String(ciphertext)
```

```
end

function decrypt(ciphertext::String, gamma::String)::String
    isempty(gamma) && throw(ArgumentError("Гамма не может быть пустой"))
    gamma_chars = collect(gamma)
    gamma_indices = Int[]
    for c in gamma_chars
        idx = get_char_index(c)
        idx === nothing && throw(ArgumentError("Символ гаммы '$c' не
поддерживается."))
        push!(gamma_indices, idx)
    end

    ciphertext_chars = collect(ciphertext)
    plaintext = Char[]

    for (i, c) in enumerate(ciphertext_chars)
        c_type = get_alphabet_type(c)
        if c_type == :other
            push!(plaintext, c)
        else
            c_idx = get_char_index(c)::Int
            g_idx = gamma_indices[mod1(i, length(gamma_indices))]
            new_idx = if c_type == :ru
                mod(c_idx - g_idx, LEN_RU)
            else
                mod(c_idx - g_idx, LEN_EN)
            end
            push!(plaintext, char_from_index(new_idx, c))
        end
    end
    return String(plaintext)
end

println("✅ Алгоритм гаммирования реализован!")
```

```
WARNING: redefinition of constant Main.RU_LOWER. This may fail, cause incorrect
answers, or produce other errors.
WARNING: redefinition of constant Main.RU_UPPER. This may fail, cause incorrect
answers, or produce other errors.
WARNING: redefinition of constant Main.EN_LOWER. This may fail, cause incorrect
answers, or produce other errors.
WARNING: redefinition of constant Main.EN_UPPER. This may fail, cause incorrect
answers, or produce other errors.
WARNING: redefinition of constant Main.RU_LOWER_IDX. This may fail, cause
incorrect answers, or produce other errors.
WARNING: redefinition of constant Main.RU_UPPER_IDX. This may fail, cause
incorrect answers, or produce other errors.
WARNING: redefinition of constant Main.EN_LOWER_IDX. This may fail, cause
incorrect answers, or produce other errors.
WARNING: redefinition of constant Main.EN_UPPER_IDX. This may fail, cause
incorrect answers, or produce other errors.


✅ Алгоритм гаммирования реализован!
```

Тестирование алгоритма

Интерактивный тестер

```
function gamma_tester()
    println("=== ГАММИРОВАНИЕ ===")
    while true
        println("\n1 - Зашифровать")
        println("2 - Расшифровать")
        println("3 - Выйти")
        print("Выберите действие: ")
        choice = readline()

        if choice == "3"
            break
        elseif choice in ("1", "2")
            print("Введите текст: ")
            text = readline()
            print("Введите гамму (ключ): ")
            gamma = readline()

            if choice == "1"
                result = encrypt(text, gamma)
                println("Зашифровано: ", result)
            else
                result = decrypt(text, gamma)
                println("Расшифровано: ", result)
            end
        else
            println("Неверный выбор!")
        end
    end
end

println("✅ Тестер готов! Для запуска введите: gamma_tester()")
```

```
✅ Тестер готов! Для запуска введите: gamma_tester()
```

```
 gamma_tester()
```

```
=== ГАММИРОВАНИЕ ===

1 - Зашифровать
2 - Расшифровать
3 - Выйти
Выберите действие:

stdin>  1


Введите текст:

stdin>  I love шоколадки!


Введите гамму (ключ):
```

```
stdin>  Ключик


Зашифровано: T qmep дмвчцкпиа!

1 - Зашифровать
2 - Расшифровать
3 - Выйти
Выберите действие:

stdin>


Неверный выбор!

1 - Зашифровать
2 - Расшифровать
3 - Выйти
Выберите действие:

stdin>  2


Введите текст:

stdin>  T qmep дмвчцкпиа!


Введите гамму (ключ):

stdin>  Ключик


Расшифровано: I love шоколадки!

1 - Зашифровать
2 - Расшифровать
3 - Выйти
Выберите действие:

stdin>  3
```

Анализ результатов

Корректность реализации

Алгоритм корректно обрабатывает:

- Русские и английские буквы в любом регистре.
- Буквы **ё/Ё** как отдельные символы.

- Пробелы, знаки препинания, цифры — без изменений.
- Гамму любого допустимого состава (только буквы).

Характиристика шифра

Характеристика	Гаммирование
Тип	Потоковый (с ключом)
Ключ	Строка (гамма)
Устойчивость к анализу	Выше (при длинной гамме)
Самодвойственность	Нет

Производительность

Алгоритм имеет линейную сложность **O(n)**, где *n* — длина текста. Использование словарей и предварительного преобразования строк в `Vector{Char}` обеспечивает эффективную работу даже с длинными сообщениями.

Выводы

1. **Успешная реализация**: Алгоритм гаммирования корректно реализован на языке Julia с учётом всех требований.
2. **Поддержка Unicode**: Решена проблема индексации многобайтовых символов через использование `collect()`.
3. **Корректная обработка ё/Ё**: Буквы включены в алфавит на правильных позициях.
4. **Практическая применимость**: Разработанный тестер позволяет легко шифровать и дешифровать тексты.
5. **Образовательная ценность**: Работа углубляет понимание классических методов симметричного шифрования и особенностей работы со строками в Julia.

Библиография

1. Менезес А., ван Ооршот П., Ванстон С. *Прикладная криптография*. — М., 2002.
2. Смарт Н. *Криптография*. — М., 2005.
3. Бабаш А.В., Шанкин Г.П. *Криптография*. — М., 2007.
4. Сингх С. *Книга шифров. Тайная история шифров и их расшифровки*. — М., 2007.
5. Bezanson J. et al. *Julia: A Fresh Approach to Numerical Computing* // SIAM Review. — 2017.
6. Официальная документация Julia: https://docs.julialang.org