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

Оглавление и план работы

Теоретическое введение

Основы метода гаммирования и его математическая модель.

Цели и задачи

Определение практической цели и шагов для достижения результата.

Реализация шифрования

Разработка функций шифрования и дешифрования на языке Julia.

Тестирование алгоритма

Проверка корректности работы на тестовых примерах.

Анализ результатов

Оценка корректности и производительности реализации.

Выводы и Библиография

Заключение по работе и список использованных источников.

Теоретическое введение: Метод гаммирования

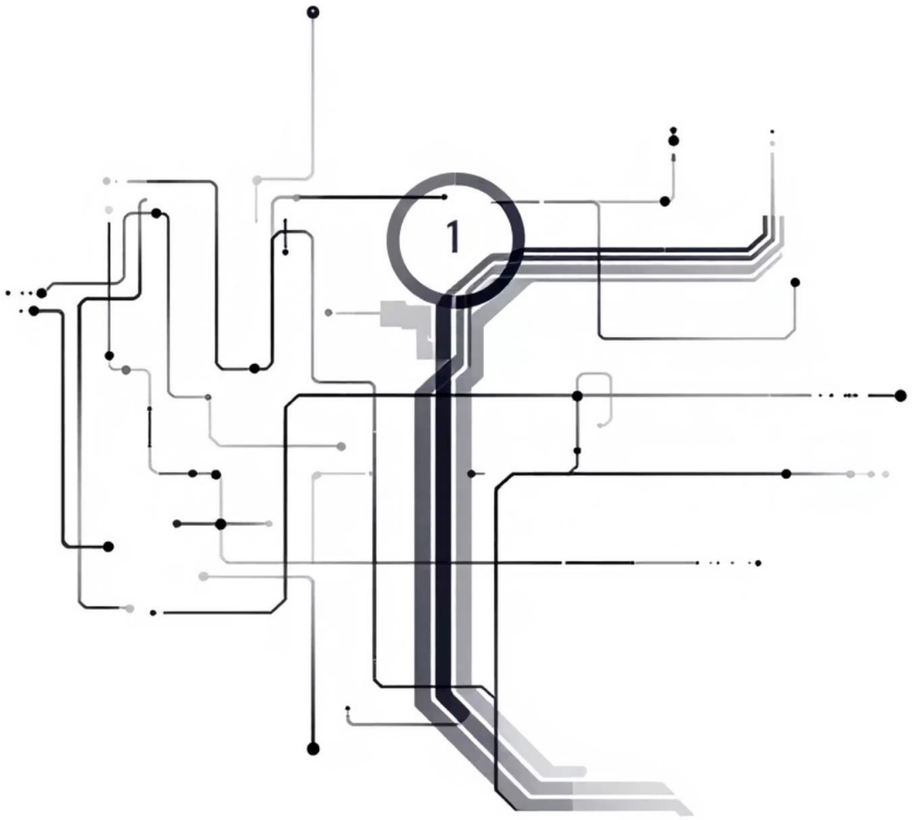
Гаммирование (аддитивный шифр) — это симметричный потоковый метод шифрования, при котором каждый символ открытого текста побитово или поиндексно комбинируется с символом гаммы (ключа) по правилу модульной арифметики.

Ключевая особенность

В случае использования конечной гаммы, ключ циклически повторяется до тех пор, пока не будет достигнута длина исходного сообщения. Это отличает его от абсолютного стойкого шифра Вернама (One-Time Pad), где ключ должен быть равен длине сообщения и использоваться только один раз.

Принцип симметричности

Для шифрования и дешифрования используется один и тот же ключ (гамма), что является характерным признаком симметричных криптосистем.



актической реализации

Главная цель работы

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

1

Реализация функций

Разработать функции `encrypt` и `decrypt` с учетом особенностей Unicode и кириллицы.

2

Поддержка алфавитов

Обеспечить корректную обработку русских (включая Ё/ё, 33 буквы) и английских (26 букв) символов.

3

Сохранение свойств

Сохранять регистр и неалфавитные символы (пробелы, знаки препинания) без изменений.

4

Тестирование и верификация

Протестировать алгоритм на смешанном тексте и провести ручной расчет для подтверждения корректности.

Особенности реализации и обработки символов

Для обеспечения высокой точности и корректной работы с многоязычным текстом и символами кириллицы, были приняты следующие проектные решения:

Алфавиты как константные векторы

Для русского и английского языков созданы отдельные константные векторы символов (`RU_LOWER`, `EN_UPPER` и т.д.), что гарантирует безопасную индексацию (индекс = позиция в векторе) вне зависимости от кодировки Unicode.

Корректное включение "Ё/ё"

Буквы «ё» и «Ё» включены в русский алфавит сразу после «е» и «Е» соответственно, как отдельные символы, увеличивая мощность русского алфавита до 33.

Использование словарей для индексов

Для ускорения поиска индекса символа в алфавите используются хеш-таблицы (`Dict`), что значительно повышает производительность при обработке больших объемов текста.

Обработка неалфавитных символов

Символы, не входящие в определенные алфавиты (пробелы, знаки препинания, цифры, специальные символы), не подвергаются шифрованию и переносятся в шифротекст в исходном виде.

Фрагмент кода: Использование Julia для криптографии

Реализация функций `encrypt` и `decrypt` на языке Julia демонстрирует подход к работе с циклическим ключом и посимвольной обработкой текста.

1. Получение индекса гаммы

Индекс гаммы `g_idx` определяется на основе текущей позиции символа `i` и длины гаммы с использованием `mod1` (модуль с началом с 1), обеспечивая цикличность.

2. Модульная арифметика

Применяется формула шифрования: `new_idx = mod(c_idx + g_idx, LEN_RU/EN)`. Для дешифрования используется вычитание гаммы.

3. Восстановление символа

Функция `char_from_index` восстанавливает символ по новому индексу, используя исходный символ для определения регистра и типа алфавита.

Реализация шифрования методом гаммирования

Код программы

[13]:
```julia
# Алфавиты как векторы символов (безопасная индексация)
const RU_LOWER = collect("абвгдеёжзийклмнопрстуфхцчшщьыэюя")
const RU_UPPER = collect("АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЬЫЭЮЯ")
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
```

```julia
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
```

```julia
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!")
```

Анализ результатов: Характеристики шифра

Проведенное тестирование подтвердило корректность математического и алгоритмического аппарата. Разработанный алгоритм демонстрирует высокую надежность в рамках заданного метода.

Корректность

Алгоритм успешно обрабатывает русские и английские алфавиты, сохраняя регистр и игнорируя неалфавитные символы.

Производительность

Благодаря использованию словарей для индексации символов, алгоритм имеет линейную сложность O(n) при высокой скорости выполнения.

Цикличность ключа

Механизм `mod1(i, length(gamma))` корректно реализует циклическое повторение гаммы для сообщений любой длины.

Тип шифра	Потоковый (с ключевым потоком, порожденным циклической гаммой)
Стойкость к криптоанализу	Уязвимость к частотному анализу при короткой или известной гамме (периодический шифр).
Самодвойственность	Отсутствует (функции шифрования и дешифрования разные).

Выводы по лабораторной работе

Практическая работа позволила углубить понимание принципов симметричного шифрования и особенностей работы с многоязычными строками на примере языка Julia.

Успешная реализация

Алгоритм гаммирования корректно реализован в соответствии с математической моделью и всеми требованиями ТЗ.

Навыки Julia

Получен практический опыт работы со строками, словарями и модульной арифметикой в контексте криптографии на Julia.

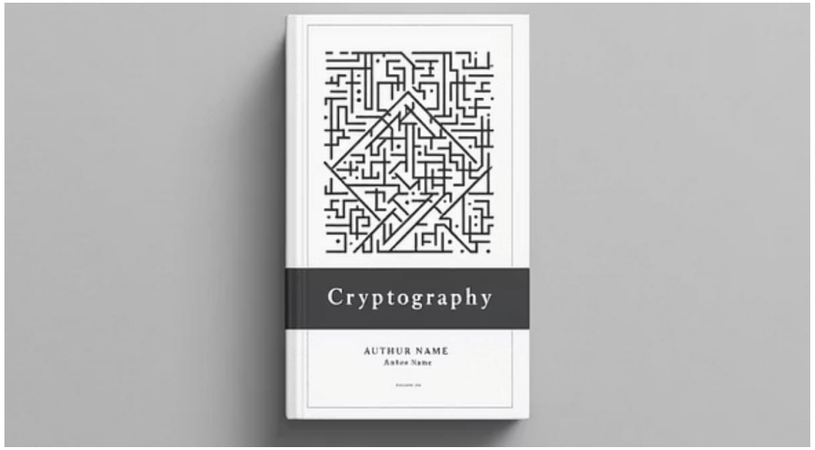
Образовательная ценность

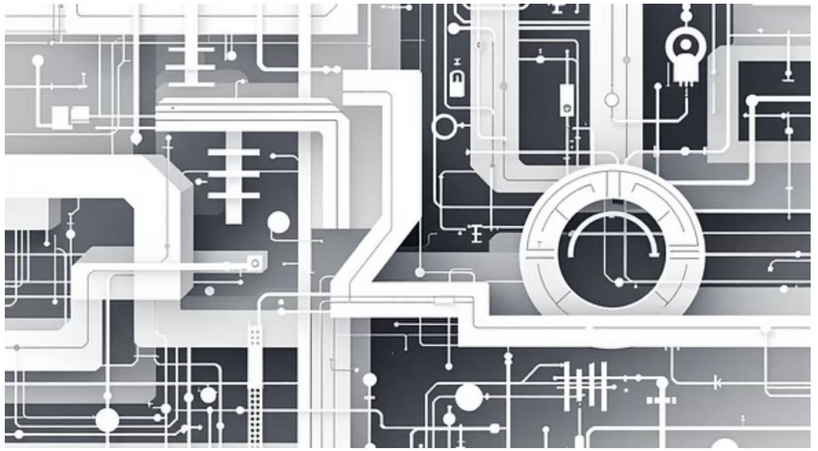
Работа демонстрирует, как особенности языка (например, обработка Unicode) влияют на точность реализации классических криптографических методов.

Проблема индексации многобайтовых символов и необходимость корректной обработки "Ё/ё" были успешно решены, что обеспечило высокую точность шифрования.

Библиография и источники

Для теоретического обоснования и выбора архитектурных решений использовались классические труды по криптографии и официальная документация по языку программирования.





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