

Отчёт по лабораторной работе №2:

Шифры перестановки

**Дисциплина: Математические основы защиты информации и
информационной безопасности**

Манаева Варвара Евгеньевна

Содержание

1	Общая информация о задании лабораторной работы	4
1.1	Цель работы	4
1.2	Задание [1]	4
2	Теоретическое введение [2]	5
2.1	Шифры и симметричные шифры	5
3	Выполнение лабораторной работы [1]	6
3.1	Шифрование гаммированием с конечной гаммой	6
3.1.1	1. Предобработка данных исходного текста	8
3.1.2	2. Предобработка гаммы	9
3.1.3	3. Шифровка/расшифровка исходного текста	10
3.1.4	4. Вывод функции	11
3.1.5	Проверка работы функции	12
4	Выводы	13
	Список литературы	14

Список иллюстраций

3.1 Результат работы реализованной функции шифрования гаммирования с конечной гаммой 12

1 Общая информация о задании лабораторной работы

1.1 Цель работы

Ознакомиться с шифрованием гаммированием и его математическими основами.

1.2 Задание [1]

1. Реализовать шифрование гаммированием с конечной гаммой.

2 Теоретическое введение [2]

2.1 Шифры и симметричные шифры

Первоначальное сообщение от одного пользователя к другому названо исходным текстом; сообщение, передаваемое через канал, названо зашифрованным текстом. Чтобы создать зашифрованный текст из исходного текста, отправитель использует алгоритм шифрования и совместный ключ засекречивания. Для того чтобы создать обычный текст из зашифрованного текста, получатель использует алгоритм дешифрования и тот же секретный ключ. Мы будем называть совместное действие алгоритмов шифрования и дешифрования шифровкой. Ключ — набор значений (чисел), которыми оперируют алгоритмы шифрования и дешифрования.

Обратите внимание, что шифрование симметричными ключами использует единственный ключ (ключ, содержащий непосредственно набор кодируемых значений) и для кодирования и для дешифрования. Кроме того, алгоритмы шифрования и дешифрования — инверсии друг друга. Если P — обычный текст, C — зашифрованный текст, а K — ключ, алгоритм кодирования $E_k(x)$ создает зашифрованный текст из исходного текста.

Алгоритм же дешифрования $D_k(x)$ создает исходный текст из зашифрованного текста. Мы предполагаем, что $E_k(x)$ и $D_k(x)$ обратны друг другу. Они применяются, последовательно преобразуя информацию из одного вида в другой и обратно.

3 Выполнение лабораторной работы

[1]

3.1 Шифрование гаммированием с конечной гаммой

В классической реализации шифрования гаммированием используется псевдослучайная последовательность (ПСП), которая имеет некоторый цикл повторения в связи с особенностями построения. В задании лабораторной предлагается рассмотреть альтернативный случай шифрования гаммированием – шифрование гаммированием с конечной гаммой. Таким образом, вместо параметров a , γ_0 , b , которые бы задавали ПСП, предлагается задать некоторое кодовое слово, построенное на том же алфавите, что и зашифрованное сообщение. Такое слово можно расшифровать в некоторые значения гаммы. Если гамма короче слова, необходимо просто повторять символы гаммы циклически с начала до тех пор, пока получившаяся последовательность не покроет полностью шифруемое сообщение.

Исходный код написан на языке Julia [3]. Код функции, осуществляющей шифрование гаммированием с конечной гаммой, представлен ниже.

```
function finiteGammaEncoding(text, gamma_code, isToBeEncoded::Bool)
    alphabet = vcat(1040:1045, 1025, 1046:1071, 32:33, 44, 46, 63, 1072:1077, 1105, 10
    filt_text = filter(x -> findfirst(isequal(Int(only(x))), alphabet) != nothing, text)
    separated_text = Int.(only.(split(filt_text, "")))
    n = length(separated_text)
```

```

t_nums = [findfirst(isequal(separated_text[i]), alphabet) for i in 1:n]
for i in 1:n
    if t_nums[i] > 38
        t_nums[i] -= 38
    end
end
println("The text to be encoded:\n", join(Char.([alphabet[t_nums[i]] for i in 1:n]
g_nums = [findfirst(isequal(i), alphabet) for i in Int.(only.(split(gamma_code, "
m = length(g_nums)
if isToBeEncoded
    encoded_nums = [alphabet[mod(t_nums[i] + g_nums[mod(i-1, m)+1]-1, 38)+1] for i
else
    encoded_nums = [alphabet[mod(t_nums[i] - g_nums[mod(i-1, m)+1]-1, 38)+1] for i
end
encoded_text = "" * join(Char.(encoded_nums))
return encoded_text
end

```

Разберём подробно работу функции.

На вход функция принимает 3 параметра:

- text – исходный текст;
- gamma_code – конечная гамма в виде кодового слова или фразы;
- isToBeEncoded – переменная логического типа, изменяющая поведение работы функции в зависимости от того, был ли наш текст зашифрован до этого или нет.

Функцию саму можно поделить на несколько смысловых частей:

1. Предобработка данных исходного текста;
2. Предобработка гаммы;

3. Шифровка/расшифровка исходного текста;
4. Вывод функции.

3.1.1 1. Предобработка данных исходного текста

Предобработка исходного текста включает в себя фильтрацию от символов, не принадлежащих алфавиту, а также изменение регистра символов. Реализовано это с помощью части функции, которая представлена ниже.

```
alphabet = vcat(1040:1045, 1025, 1046:1071, 32:33, 44, 46, 63, 1072:1077, 1105, 1078:1109)
filt_text = filter(x -> findfirst(isequal(Int(only(x))), alphabet) != nothing, text)
separated_text = Int.(only.(split(filt_text, ")))
n = length(separated_text)
t_nums = [findfirst(isequal(separated_text[i]), alphabet) for i in 1:n]
for i in 1:n
    if t_nums[i] > 38
        t_nums[i] -= 38
    end
end
println("The text to be encoded:\n", join(Char.([alphabet[t_nums[i]] for i in 1:n])))
# <...>
```

Переменная `alphabet` ограничивает алфавит текста именно теми символами, численные коды которых записаны в переменной, а именно:

1. Кириллицей заглавного регистра: А,Б,В,Г,Д,Е,Ё,Ж,З,И,Й,К,Л,М,Н,О,П,Р,С,Т,У,Ф,Х,Ц,Ч,Ш,Щ,Ъ,Ы,Ь,Э,Ю;
2. Знаками препинания: ' ','', '!', '?';
3. Кириллицей строчного регистра: а,б,в,г,д,е,ё,ж,з,и,й,к,л,м,н,о,п,р,с,т,у,ф,х,ц,ч,ш,щ,ъ,ы,ь,э,ю,

Следующим после задания алфавита этапом используется функция `filter(x -> findfirst(isequal(Int(only(x))), alphabet) != nothing, text)`, которая фильтрует исходный текст, убирая символы, которых нет в алфавите `alphabet`.

Далее получившийся текст разделяется по символам, а каждый символ обращается в своё численное значение.

Задаётся переменная `n`, хранящая длину отфильтрованного текста.

Задаётся переменная `t_num`, которая обозначает порядковый номер каждого символа в алфавите. Именно над этими числами и проводится операции шифрования гаммирования.

Далее по всей длине `t_num` пробегается простой цикл, который заменяет строчные символы кириллицы на заглавные (именно поэтому строчные символы кириллицы добавлены в конец алфавита).

После всех преобразований текста выводится промежуточное сообщение "The text to be encoded:", в котором демонстрируется сообщение, которое в действительности будет закодировано.

3.1.2 2. Предобработка гаммы

Предобработка исходного текста включает в себя преобразование гаммы в последовательность символов, которая затем переводится в числа. Реализовано это с помощью части функции, которая представлена ниже.

```
# <...>
```

```
g_nums = [findfirst(isequal(i), alphabet) for i in Int.(only.(split(gamma_code, "")))]
```

```
m = length(g_nums)
```

```
# <...>
```

Вместительная строчка предобработки, которая задаёт `g_nums`, делает несколько вещей:

1. `split(gamma_code, "")` – разделяет гамму на подстроки каждого символа;
2. `only.<...>` – преобразует каждый символ (который хранится как подстрока) в символьный формат (`Char`);
3. `Int.<...>` – преобразует каждый символ в его числовой код;

4. `for i in <...>` – циклом пробегается по каждому элементу строки;
5. `findfirst(isequal(i), alphabet)` – ищет местоположение символа в алфавите или возвращает `nothing`, если не находит его.

Далее просто вводится переменная длины конечной гаммы `m` для ограничения условия о том, что конечная гамма может быть меньше текста, в связи с чем её придётся повторять несколько раз.

3.1.3 3. Шифровка/расшифровка исходного текста

Собственно шифровка/расшифровка исходного текста включает в себя сложение по модулю мощности алфавита символов гаммы и символов исходного текста. Реализовано это с помощью части функции, которая представлена ниже.

```
#< ...>
if isToBeEncoded
    encoded_nums = [alphabet[mod(t_nums[i] + g_nums[mod(i-1, m)+1]-1, 38)+1] for i in 1:n]
else
    encoded_nums = [alphabet[mod(t_nums[i] - g_nums[mod(i-1, m)+1]-1, 38)+1] for i in 1:n]
end
#< ...>
```

Реализация лишь в одном знаке зависит от того, шифруется ли сообщение или расшифровывается (гамма прибавляется, если текст шифруется, и вычитается, если текст расшифровывается).

Для каждого символа исходного текста осуществляются следующие операции:

1. `for i in 1:n` – цикл проходит по каждому символу исходного текста;
2. `g_nums[mod(i-1, m)+1]` – гамма должна быть зациклена по длине исходного текста, для чего происходит проверка того, какой остаток от деления даёт порядковый номер элемента исходного текста минус 1, после чего прибавляется единица, и этот индекс используется для задания символа гаммы,

который будет использоваться для сложения с данным элементом исходного текста;

3. `mod(t_nums[i] +/- <...>-1, 38)+1` – мы задаём новое значение порядкового номера рассматриваемого элемента в алфавите. При использовании сложения полученные символы считаются зашифрованными, при использовании вычитания – расшифрованными;
4. `alphabet[<...>]` – и финальным элементом мы задаём собственно численное значение рассматриваемого символа, которое можно использовать для преобразования в символьный формат данных.

Функция остатка от деления в программе используется в виде `mod(number-1, base)+1` в связи с особенностями операции остаток от деления. Так, при классическом использовании `mod(number, base)` значения остатка от деления лежат в диапазоне от 0 до `base-1`, при этом ещё и остаток от деления 0 будет обозначать, что число `number` делится на `base` нацело. Операция, когда мы сначала отнимаем от числа единицу, затем пропускаем через операцию остатка от деления и затем обратно прибавляем единицу, напрямую отражает все получающиеся остатки в диапазон от 1 до `base`, где остаток вида `base` обозначает, что число делится `number` делится на `base` нацело.

3.1.4 4. Вывод функции

Для создания вывода функции вектор численных значений символов зашифрованного текста преобразуется в формат `Char`, после чего символы объединяются в единую строку и выводятся из функции. Реализовано это с помощью части функции, которая представлена ниже.

```
# <...>
encoded_text = "" * join(Char.(encoded_nums))
return encoded_text
```

3.1.5 Проверка работы функции

При проверке корректности реализации важно учитывать, что шифрование гаммированием относится к симметричным шифрам. Для проверки изначальное сообщение мы пропускаем через функции шифровки и расшифровки с одними и теми же параметрами (кодовым словом, которое играет роль гаммы при шифровании). Так мы должны получить шифрокод после запуска функции шифрования первый раз, и изначальное сообщение после запуска функции второй раз с теми же параметрами на входе (исключая собственно параметр функции, задающий направление шифровки/расшифровки).

```
coded_text = finiteGammaEncoding("приКАЗ", "ГАММА", true)
println("The result of encoding:\n", coded_text, "\n\n")
decoded_text = finiteGammaEncoding(coded_text, "ГАММА", false)
println("The result of decoding:\n", decoded_text)
```

Результат работы кода представлен ниже (рис. 3.1).

```
[83]: finiteGammaEncoding (generic function with 2 methods)

[84]: coded_text = finiteGammaEncoding("приКАЗ", "ГАММА", true)
      println("The result of encoding:\n", coded_text, "\n\n")
      decoded_text = finiteGammaEncoding(coded_text, "ГАММА", false)
      println("The result of decoding:\n", decoded_text)

      The text to be encoded:
      ПРИКАЗ
      The result of encoding:
      УСЦШБЛ

      The text to be encoded:
      УСЦШБЛ
      The result of decoding:
      ПРИКАЗ
```

Рис. 3.1: Результат работы реализованной функции шифрования гаммирования с конечной гаммой

4 Выводы

В результате работы мы ознакомились со способом шифрования гаммированием и его математическими основами, а также реализовали шифрование гаммированием с конечной гаммой.

Также были записаны скринкасты:

На RuTube:

- Весь плейлист
- Выполнения лабораторной работы
- Запись создания отчёта
- Запись создания презентации
- Защита лабораторной работы

На Платформе:

- Весь плейлист
- Выполнения лабораторной работы
- Запись создания отчёта
- Запись создания презентации
- Защита лабораторной работы

Список литературы

1. Лабораторная работа №3. Шифрование гаммированием [Электронный ресурс]. RUDN, 2024. URL: https://esystem.rudn.ru/pluginfile.php/2368510/mod_folder/content/0/lab03.pdf.
2. Математика криптографии и теория шифрования [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/552/408/info>.
3. Julia 1.10 Documentation [Электронный ресурс]. 2024. URL: <https://docs.julialang.org/en/v1/>.