Отчет по Лабораторной работе №2 по предмету Математические основы зищаты информации и информационной безопасности

Лобов Михаил Сергеевич

Содержание

Цель работы

Изучить шифры перестановки

Задание

Программно реализовать на языке Julia шифры: 1. Маршрутное шифрование 2. Шифрование с помощью решеток 3. Таблица Вижнера

Теоретическое введение

Маршрутное шифрование

Открытый текст разбивается на блоки равной длины, состоящие из числа символов, равного произведению mn. Если последний блок получится меньше остальных, то в него следует дописать требуемое количество произвольных символов. Составляется таблица размерности mn. Блоки вписываются построчно в таблицу. Криптограмма получается выписыванием букв из таблицы в соответствии с некоторым маршрутом. Ключом такой криптограммы является маршрут и числа m и n. Обычно буквы выписывают по столбцам, которые упорядочивают согласно паролю: внизу таблицы приписывается слово из n неповторяющихся букв и столбцы нумеруются по алфавитному порядку букв пароля.

Шифрование с помощью решеток

Суть этого способа заключается в следующем. Выбирается натуральное число $k>1$, строится квадрат размерности $k \times k$ и построчно заполняется числами $1, 2, \ldots, k^2$. В качестве примера рассмотрим квадрат размерности $k=2$.

1 2
3 4

Повернем его по часовой стрелке на 90° и присоединим к исходному квадрату справа.

1	2	3	1
3	4	4	2

Проделаем еще дважды такую процедуру и пришьем получившиеся квадраты снизу. Получился большой квадрат размерности $2k$.

1	2	3	1
3	4	4	2
2	4	3	4
1	3	2	1

Далее из большого квадрата вырезаются клетки, содержащие числа от 1 до k^2. В каждой клетке должно быть только одно число. Получается своего рода решето. Шифрование осуществляется следующим образом. Решето накладывается на чистый квадрат $2k \times 2k$ и в прорези вписываются буквы исходного текста по порядку их следования. Когда заполнятся все прорези, решето поворачивается на 90° и вписывание букв продолжается. После третьего поворота все клетки большого квадрата окажутся заполненными. Подобрав подходящий пароль (число букв пароля должно равняться k^2 и они не должны повторяться), выпишем буквы по столбцам. Очередность столбцов определяется алфавитным порядком букв пароля.

Пример. Исходный текст — *договор подписали*; пароль — *шифр*. С применением вышеуказанной решетки за пять шагов получаем следующую криптограмму.

д	*о*	*д*	*о*
г	*о*	*а*	*в*
о	*р*	*п*	*и*
п	*о*	*д*	*п*
с	*а*	*л*	*и*
и	*ш*	*ф*	*р*

Получившаяся криптограмма: `ОВОРДЛГПАПИОСДОИ`. Важно отметить, что число клеток подбирается в соответствии с количеством букв в исходном тексте. В идеальном случае $k^2 = n$. Если такого равенства достичь невозможно, можно либо дописать произвольную букву к последнему слову открытого текста, либо убрать её.””

Таблица Виженера

В 1585 году французский криптограф Блез Виженер опубликовал свой метод шифрования в «Трактате о шифрах». Шифр считался нераскрываемым до 1863 года, когда австриец Фридрих Казиски взломал его.

Открытый текст разбивается на блоки длины n. Ключ представляет собой последовательность из n натуральных чисел: $a_1, a_2, \ldots, a_n$. Далее в каждом блоке первая буква циклически сдвигается вправо по алфавиту на a_1 позиций, вторая буква — на a_2 позиций, последняя — на a_n позиций. Для лучшего запоминания в качестве ключа можно

взять осмысленное слово, а алфавитные номера входящих в него букв использовать для осуществления сдвигов. Рассмотрим еще одну схему построения шифра Виженера.

В нижеприведенной таблице в строчках записаны буквы русского алфавита. При переходе от одной строки к другой происходит циклический сдвиг на одну позицию. Исходный текст: *криптография серьезная наука*; пароль — *математика*. Пароль записывается с повторениями над буквами сообщения.

м	а	т	е	м	а	т	и	к	а	м	а	т	е	м	а	т	и	к	а	м	а	т	е	м	а
к	р	и	п	т	о	г	р	а	ф	и	я	с	е	р	ь	е	з	н	а	я	н	а	у	к	а

АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ

БВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБ

ВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВ

ГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГ

ДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГД

ЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕ

ЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖ

ЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗ

ИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИ

ЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙ

КЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙК

ЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛ

МНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМ

НОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМН

ОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНО

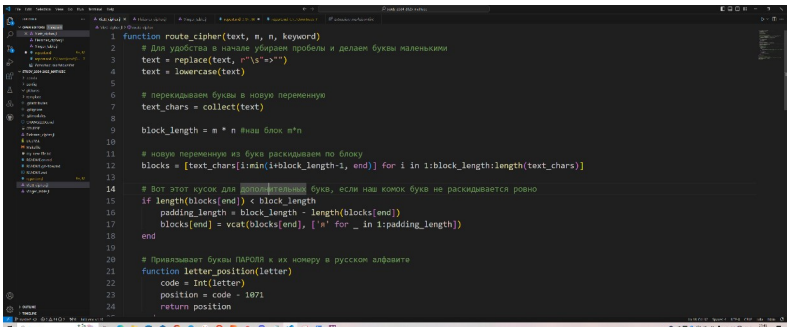
ПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОП

В горизонтальном алфавите находим букву «к», а в вертикальном — букву «м». На пересечении столбца и строки в таблице расположена буква «ц». Далее переходим к буквам «р» и «а» соответственно. В итоге получается следующая криптограмма:

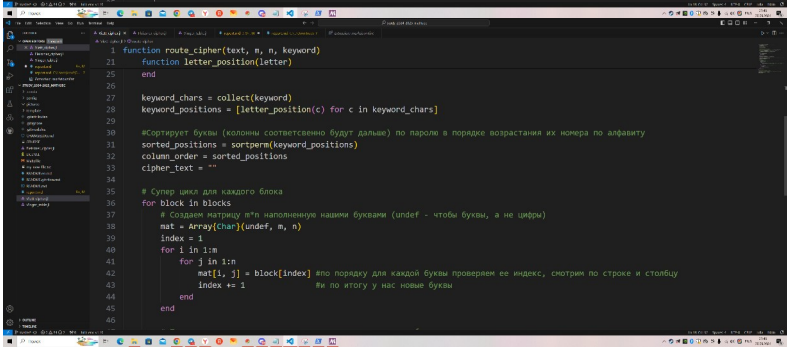
ЦРЬФЯЮХШКФФЯДКЭЬЧПЧАЛНТЩЦА.

Выполнение лабораторной работы

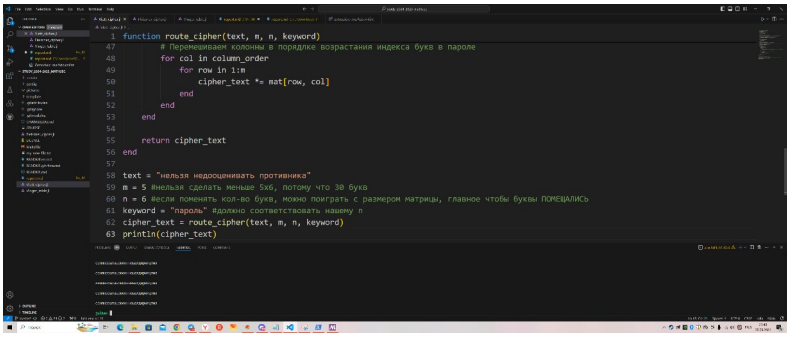
Написаны программы на языке Julia.



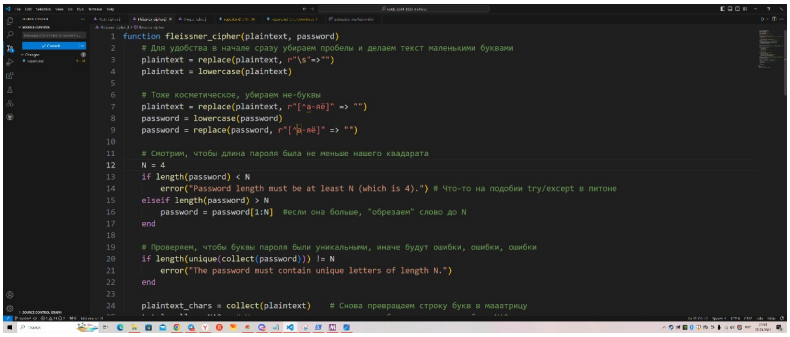
```julia
function route_cipher(text, m, n, keyword)
    # Для удобства в начале убираем пробелы и делаем буквы маленькими
    text = replace(text, r"\s"=>"")
    text = lowercase(text)

    # перекидываем буквы в новую переменную
    text_chars = collect(text)

    block_length = m * n #наш блок m*n

    # новую переменную из букв раскидываем по блоку
    blocks = [text_chars[i:min(i+block_length-1, end)] for i in 1:block_length:length(text_chars)]

    # Вот этот кусок для дополнительных букв, если наш комок букв не раскидывается ровно
    if length(blocks[end]) < block_length
        padding_length = block_length - length(blocks[end])
        blocks[end] = vcat(blocks[end], ['я' for _ in 1:padding_length])
    end

    # Привязывает буквы ПАРОЛЯ к их номеру в русском алфавите
    function letter_position(letter)
        code = Int(letter)
        position = code - 1071
        return position
```



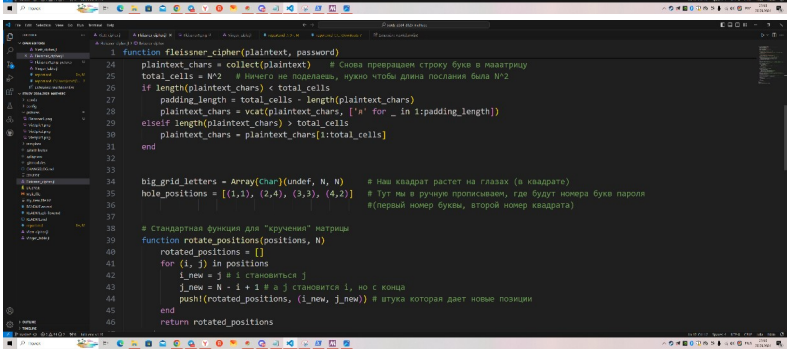
```julia
function route_cipher(text, m, n, keyword)
    function letter_position(letter)
    end

    keyword_chars = collect(keyword)
    keyword_positions = [letter_position(c) for c in keyword_chars]

    #Сортирует буквы (колонны соответственно будут дальше) по паролю в порядке возрастания их номера по алфавиту
    sorted_positions = sortperm(keyword_positions)
    column_order = sorted_positions
    cipher_text = ""

    # Супер цикл для каждого блока
    for block in blocks
        # Создаем матрицу m*n наполненную нашими буквами (undef - чтобы буквы, а не цифры)
        mat = Array{Char}(undef, m, n)
        index = 1
        for i in 1:m
            for j in 1:n
                mat[i, j] = block[index] #по порядку для каждой буквы проверяем ее индекс, смотрим по строке и столбцу
                index += 1               #и по итогу у нас новые буквы
            end
        end
```



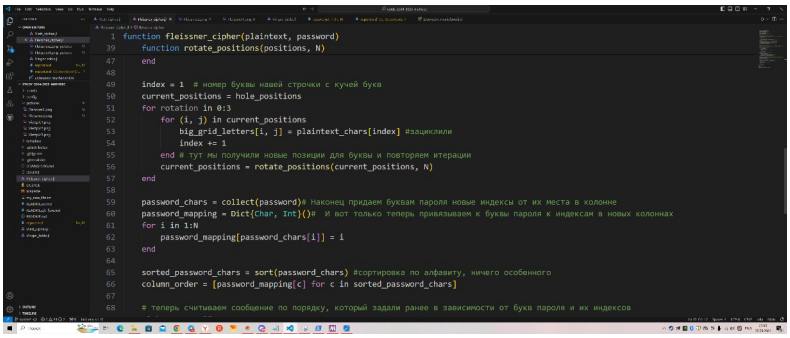
```julia
function route_cipher(text, m, n, keyword)
        # Перемешиваем колонны в порядке возрастания индекса букв в пароле
        for col in column_order
            for row in 1:m
                cipher_text *= mat[row, col]
            end
        end
    end

    return cipher_text
end

text = "нельзя недооценивать противника"
m = 5 #нельзя сделать меньше 5х6, потому что 30 букв
n = 6 #если поменять кол-во букв, можно поиграть с размером матрицы, главное чтобы буквы ПОМЕЩАЛИСЬ
keyword = "пароль" #должно соответствовать нашему n
cipher_text = route_cipher(text, m, n, keyword)
println(cipher_text)
```



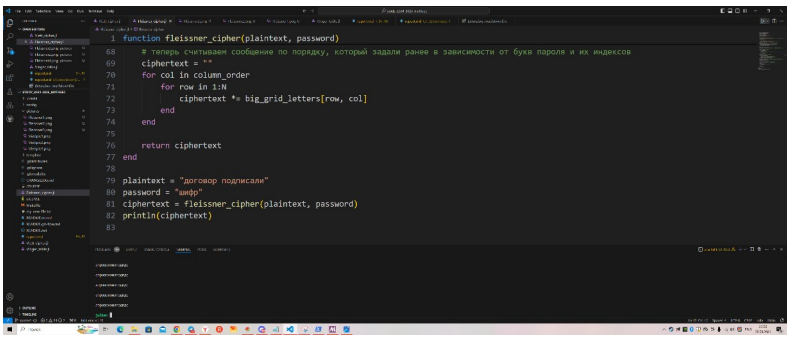
```julia
function fleissner_cipher(plaintext, password)
    # Для удобства в начале сразу убираем пробелы и делаем текст маленькими буквами
    plaintext = replace(plaintext, r"\s"=>"")
    plaintext = lowercase(plaintext)

    # Тоже косметическое, убираем не-буквы
    plaintext = replace(plaintext, r"[^а-яё]" => "")
    password = lowercase(password)
    password = replace(password, r"[^а-яё]" => "")

    # Смотрим, чтобы длина пароля была не меньше нашего квадрата
    N = 4
    if length(password) < N
        error("Password length must be at least N (which is 4).") # Что-то на подобии try/except в питоне
    elseif length(password) > N
        password = password[1:N]  #если она больше, "обрезаем" слово до N
    end

    # Проверяем, чтобы буквы пароля были уникальными, иначе будут ошибки, ошибки, ошибки
    if length(unique(collect(password))) != N
        error("The password must contain unique letters of length N.")
    end

    plaintext_chars = collect(plaintext)    # Снова превращаем строку букв в маатрицу
```



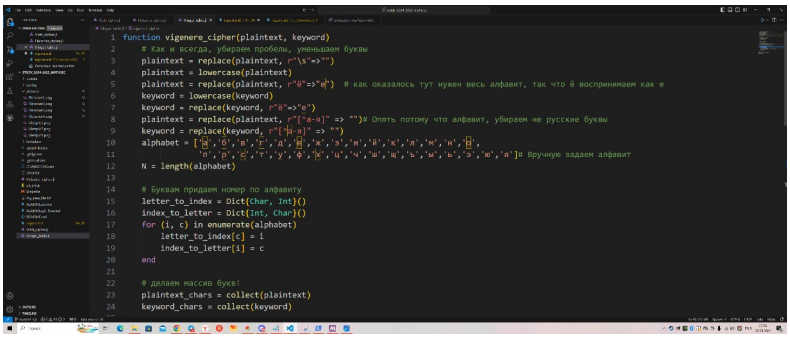
```julia
function fleissner_cipher(plaintext, password)
    plaintext_chars = collect(plaintext)    # Снова превращаем строку букв в маатрицу
    total_cells = N^2  # Ничего не подсказка, нужно чтобы длина послания была N^2
    if length(plaintext_chars) < total_cells
        padding_length = total_cells - length(plaintext_chars)
        plaintext_chars = vcat(plaintext_chars, ['я' for _ in 1:padding_length])
    elseif length(plaintext_chars) > total_cells
        plaintext_chars = plaintext_chars[1:total_cells]
    end


    big_grid_letters = Array{Char}(undef, N, N)     # Наш квадрат растет на глазах (в квадрате)
    hole_positions = [(1,1), (2,4), (3,3), (4,2)]   # Тут мы в ручную прописываем, где будут номера букв пароля
                                                    #(первый номер буквы, второй номер квадрата)

    # Стандартная функция для "кручения" матрицы
    function rotate_positions(positions, N)
        rotated_positions = []
        for (i, j) in positions
            i_new = j # i становиться j
            j_new = N - i + 1 # а j становится i, но с конца
            push!(rotated_positions, (i_new, j_new)) # штука которая дает новые позиции
        end
        return rotated_positions
```



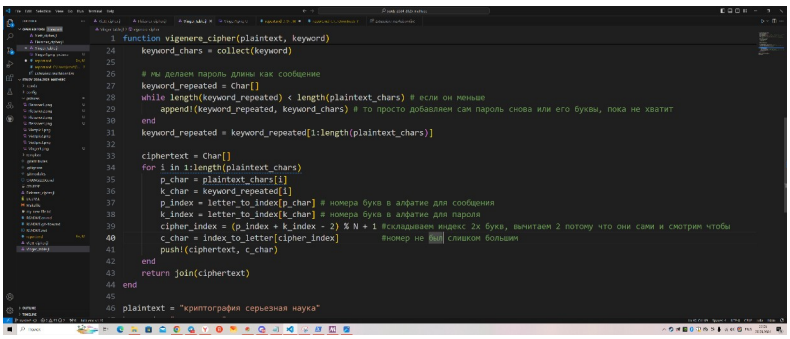
```julia
function fleissner_cipher(plaintext, password)
    function rotate_positions(positions, N)
    end

    index = 1  # номер буквы нашей строчки с кучей букв
    current_positions = hole_positions
    for rotation in 0:3
        for (i, j) in current_positions
            big_grid_letters[i, j] = plaintext_chars[index] #зациклили
            index += 1
        end # тут мы получили новые позиции для буквы и повторяем итерации
        current_positions = rotate_positions(current_positions, N)
    end

    password_chars = collect(password)# Наконец придаем буквам пароля новые индексы от их места в колонне
    password_mapping = Dict{Char, Int}()# И вот только теперь привязываем к буквы пароля к индексам в новых колоннах
    for i in 1:N
        password_mapping[password_chars[i]] = i
    end


    sorted_password_chars = sort(password_chars) #сортировка по алфавиту, ничего особенного
    column_order = [password_mapping[c] for c in sorted_password_chars]

    # теперь считываем сообщение по порядку, который задали ранее в зависимости от букв пароля и их индексов
```



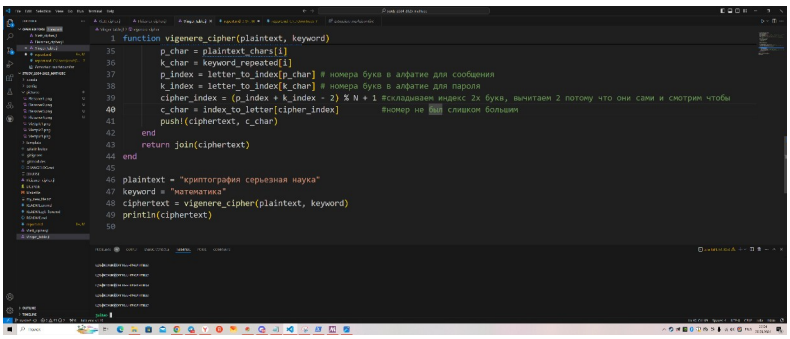
```julia
function fleissner_cipher(plaintext, password)
    # теперь считываем сообщение по порядку, который задали ранее в зависимости от букв пароля и их индексов
    ciphertext = ""
    for col in column_order
        for row in 1:N
            ciphertext *= big_grid_letters[row, col]
        end
    end

    return ciphertext
end

plaintext = "договор подписали"
password = "шифр"
ciphertext = fleissner_cipher(plaintext, password)
println(ciphertext)
```



```julia
function vigenere_cipher(plaintext, keyword)
    # Как и всегда, убираем пробелы, уменьшаем буквы
    plaintext = replace(plaintext, r"\s"=>"")
    plaintext = lowercase(plaintext)
    plaintext = replace(plaintext, r"ё"=>"е")  # как оказалось тут нужен весь алфавит, так что ё воспринимаем как е
    keyword = lowercase(keyword)
    keyword = replace(keyword, r"ё"=>"е")
    plaintext = replace(plaintext, r"[^а-я]" => "")# Опять потому что алфавит, убираем не русские буквы
    keyword = replace(keyword, r"[^а-я]" => "")
    alphabet = ['а','б','в','г','д','е','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','я']# Вручную задаем алфавит
    N = length(alphabet)

    # Буквам придаем номер по алфавиту
    letter_to_index = Dict{Char, Int}()
    index_to_letter = Dict{Int, Char}()
    for (i, c) in enumerate(alphabet)
        letter_to_index[c] = i
        index_to_letter[i] = c
    end

    # делаем массив букв!
    plaintext_chars = collect(plaintext)
    keyword_chars = collect(keyword)
```



```julia
function vigenere_cipher(plaintext, keyword)
    keyword_chars = collect(keyword)

    # мы делаем пароль длины как сообщение
    keyword_repeated = Char[]
    while length(keyword_repeated) < length(plaintext_chars) # если он меньше
        append!(keyword_repeated, keyword_chars) # то просто добавляем сам пароль снова или его буквы, пока не хватит
    end
    keyword_repeated = keyword_repeated[1:length(plaintext_chars)]

    ciphertext = Char[]
    for i in 1:length(plaintext_chars)
        p_char = plaintext_chars[i]
        k_char = keyword_repeated[i]
        p_index = letter_to_index[p_char] # номера букв в алфатие для сообщения
        k_index = letter_to_index[k_char] # номера букв в алфатие для пароля
        cipher_index = (p_index + k_index - 2) % N + 1 #складываем индекс 2х букв, вычитаем 2 потому что они сами и смотрим чтобы
        c_char = index_to_letter[cipher_index]          #номер не был слишком большим
        push!(ciphertext, c_char)
    end
    return join(ciphertext)
end

plaintext = "криптография серьезная наука"
```



```julia
function vigenere_cipher(plaintext, keyword)
        p_char = plaintext_chars[i]
        k_char = keyword_repeated[i]
        p_index = letter_to_index[p_char] # номера букв в алфатие для сообщения
        k_index = letter_to_index[k_char] # номера букв в алфатие для пароля
        cipher_index = (p_index + k_index - 2) % N + 1 #складываем индекс 2х букв, вычитаем 2 потому что они сами и смотрим чтобы
        c_char = index_to_letter[cipher_index]          #номер не был слишком большим
        push!(ciphertext, c_char)
    end
    return join(ciphertext)
end

plaintext = "криптография серьезная наука"
keyword = "математика"
ciphertext = vigenere_cipher(plaintext, keyword)
println(ciphertext)
```

Выводы

По итогу проделанной работы были написаны 3 программы для каждого алгоритма шифрования. Программы успешно работают, без ошибок и могут принимать на вход различные данные, т.е. не привязаны к конкретным паролям или сообщениям.

Список литературы