Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Защита информации»

Профиль: «Автоматизированные системы обработки информации и управления»

Семестр 7

ОТЧЕТ

по лабораторной работе №2

Тема: «Алгоритм RSA»

Выполнил: студент группы АСУ-17-1б

Хохряков Денис

Проверил: доцент кафедры ИТАС

Шереметьев В.Г.\_\_\_\_\_\_\_\_\_\_

Дата\_\_\_\_\_\_

# ЦЕЛЬ РАБОТЫ

Получить практические навыки по использованию ассиметричных алгоритмов шифрования на примере использования алгоритма RSA.

# ЗАДАНИЕ

Выполнить шифрование строки исходного текста методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двадцати, выполнив условие случайности p и q для каждого нового шифрования.

# КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Алгоритм RSA был разработан еще в 1977 году, и остается самым распространенным ассиметричным криптоалгоритмом в мире. Самый яркий пример – использование RSA в соединении HTTPS при передаче ключа по протоколу TLS.

Алгоритм основывается на вычислительной сложности задачи факторизации (разложении в простые множители) больших целых чисел (размер числа > 512 бит).

Сначала необходимо получить открытый и закрытый ключи:

1. Выбираются два простых числа p и q
2. Вычисляется их произведение n(=p\*q)
3. Выбирается произвольное число e (e<n), такое, что НОД(e,(p-1)(q-1))=1, то есть e должно быть взаимно простым с числом (p-1)(q-1).
4. Методом Евклида решается в целых числах уравнение e\*d+(p-1)(q-1)\*y=1. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d,y), каждая из которых является решением уравнения в целых числах.
5. Два числа (e,n) – публикуются как открытый ключ.
6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e,n).

Ключи RSA в наше время передаются в бинарном формате (.DER), а хранятся в читабельном формате base64 (.PEM)

Оба ключа RSA формирует принимающая сторона, и высылает публичный ключ (e, n) отправляющей стороне.

Дальше шифрование происходит следующим образом

1. Отправитель разбивает свое сообщение на блоки, равные k=log2(n).
2. Дальше он каждый блок представляет в виде числа m, и вычисляет шифротекст:

c = m­e mod n

А дешифрование на принимающей стороне происходит так:

1. Вычисляются исходные блоки информации:

m = c d mod n

1. Полученные блоки объединяются в исходное сообщение

Сложность данного алгоритма заключается в генерации больших простых чисел, а также в методе остатка от возведения числа в степень (решается с помощью алгоритма быстрого возведения в степень по модулю)

# ХОД РАБОТЫ

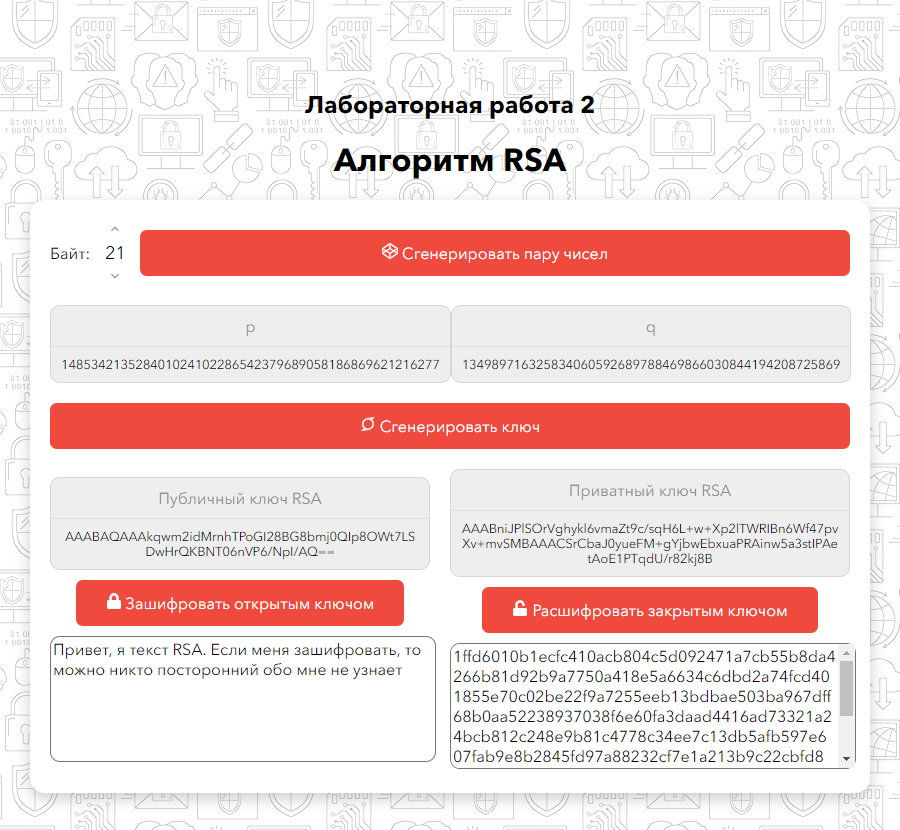
Первым шагом нужно определиться с методом формирования простых чисел. Можно воспользоваться древним методом решета Эратосфена (просто вычеркивать простые числа и их множители до корня из n). Но для больших чисел это слишком затратная операция. Поэтому поступим следующим образом:

1. Сгенерируем случайное число
2. Проверим, делится ли оно нацело на простые числа от 2 до 101.
3. Если не делится, то проверяем дальше тестом Миллера-Рабина, где количество раундов зависит от длины числа.
4. Если число не проходит хотя бы один тест, то увеличиваем его на единицу и повторяем операцию 2-4.

Затем генерируем данным методом p и q, и, взяв e = 17, вычисляем d.

Таким образом, публичный ключ составляет {e, n}, а приватный – {d, n}. Сконвертируем их в формат base64, использовав 0 как разделитель.

Затем, используем метод шифрования или дешифрования, в зависимости от того, что нам необходимо.



При этом мы шифруем текст так:

1. Делим текст на блоки длиной log2(n) бит. (То есть смотрим кол-во бит в числе n).
2. Дальше, с помощью операции сдвига и сложения записываем каждый блок текста в большое число m (m < n).
3. Шифруем данное число по методу шифрования RSA с помощью публичного ключа, получаем зашифрованные блоки c (c < n)
4. Записываем данные блоки в 16-чном формате
5. При расшифровке читаем данные блоки и переводим их в большие числа
6. По методу дешифрования RSA с помощью закрытого ключа получаем блоки текста.
7. С помощью операции остатка от деления получаем необходимые коды символов, которые объединяем в строку

Также можно проверить, что при изменении какого-либо символа шифр-текста портится один блок исходного текста, и его уже не восстановить



# ЛИСТИНГ ПРОГРАММЫ (Язык – JavaScript)

const primeLow = [3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101];

const primeLowMap = primeLow.map((i) => new bn(i));

//Генерация простого числа

module.exports.generatePrime = function(size) {

let b = new bn(rand(size));

while(b.isEven())

b = b.shrn(1);

let increment = new bn(2);

for(let i = 0; i < 300; i++){

for(let c of primeLowMap)

if(b.mod(c).isZero())

continue;

if(mr.test(b))

break;

b = b.add(increment);

}

return BigInt('0x'+b.toString('hex'));

}

//Быстрое возведение в степень (a ^ b mod n)

module.exports.fastModularExponentiation = function(a, b, n) {

a = a % n;

var result = 1n;

var x = a;

while(b > 0){

var leastSignificantBit = b % 2n;

b = b / 2n;

if (leastSignificantBit == 1n) {

result = result \* x;

result = result % n;

}

x = x \* x;

x = x % n;

}

return result;

};

export function generateKeys (p, q) {

const n = p\*q;

const euler = (p-1n)\*(q-1n);

const e = 17n;

const d = calcSecretExponent(e, euler);

return { public: {e, n}, private: {d, n} };

}

//Разбиваем строку на блоки

export const **encrypt** = (str, publicKey) => {

const size = publicKey.n.toString(16).length-4;

let k = 1n;

let m = 0n;

let array = [];

let s = 0;

for(let i = 0; i < str.length; i++){

m = m + BigInt(str.charCodeAt(i)) \* k;

k \*= 65536n;

s += 4;

if(s >= size){

array.push(m);

k = 1n;

s = 0;

m = 0n;

}

}

if(s >= 0)

array.push(m);

const {e, n} = publicKey;

const arr = [];

for(let i = 0; i < array.length; i++){

const a = fastModularExponentiation(array[i], e, n);

arr.push(a);

}

let outString = "";

for(let i = 0; i < array.length; i++){

let q = array[i];

while(q > 0){

outString += String.fromCharCode(Number(q%65536n));

q /= 65536n;

}

}

console.log(outString);

return arr;

}

export const **decrypt** = (arr, privateKey) => {

const {d, n} = privateKey;

console.log(d, n);

let outString = "";

for(let i = 0; i < arr.length; i++){

let q = fastModularExponentiation(arr[i], d, n);

console.log(q);

while(q > 0){

outString += String.fromCharCode(Number(q%65536n));

q /= 65536n;

}

}

return outString;

}