Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Криптографические методы обеспечения информационной безопасности»

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4

«Асимметричные криптосистемы»

Выполнил:

Полевцов Артем Сергеевич, студент группы N34511

(подпись)

Проверил:

Волков Александр Григорьевич, инженер ФБИТ

(отметка о выполнении)

(подпись)

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: изучить основные принципы работы асимметричных криптосистем на примере алгоритма RSA.

Задачи практической работы

1. Проанализировать эмуляцию алгоритма RSA и примитивных атак на шифр, используя Cryptool 2. Выделить основные необходимые настройки шифра и требуемые ограничения на параметры.

2. Программно реализовать и модифицировать любую асимметричную криптосистему. В случае отсутствия опыта программирования подойдет реализация алгоритма на псевдокоде или в виде блок схем, включающих основные этапы алгоритма с отображением формул и основных математических действий. Атаки и модификации, приведенные ниже, указаны для RSA. Если атака или модификация не применима для реализуемого алгоритма разрешается найти любую альтернативу (атаки, применимой к алгоритму; модификации для ускорения алгоритма и дополнительной защиты).

3. Для созданной реализации криптосистемы предлагается провести примитивный криптоанализ на устойчивость к следующим атакам, а также сделать минимальные модификации по оптимизации (ускорению процессов шифрования, дешифрования, процесса генерации ключей).

1 АСИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ

1.1 Ход работы

1.1.1 Анализ алгоритма rsa при помощи утилиты cryptool

Открытый текст:

Metasploit is a powerful tool used by network security professionals to do penetration tests, by system administrators to test patch installations, by product vendors to implement regression testing, and by security engineers across industries.

Шифротекст:

87 76 B8 2F AE 5D 07 4E DF 46 BC 3C 53 84 D2 A0 56 60 2A D0 B7 02 F6 43 18 0B C1 72 7D 9D 54 78 0B 52 A6 1A 28 1E 1F 21 D1 2E 91 21 EF E3 0C 1D 7B 18 26 D5 DC 3A BE 04 76 49 3D 9D 8A 15 4D 11 34 13 A0 F7 50 BF 8F A2 6B F2 60 D7 13 83 EB F6 D2 A3 42 B1 E5 6E E1 DE 76 51 44 FD 68 3E 76 ED 9B 30 38 D7 EC E9 08 C4 9F 49 8D 35 18 E3 08 82 21 D0 CA 5D BD C7 41 C4 DD 1B D8 EC 1A 5D 61 B4 AC D0 1F FE BB 59 6B 42 8B 30 FA 44 E6 11 26 1E 51 61 30 F5 CF 89 5D E2 60 C9 9E 05 24 9C 95 B6 0D DB 76 32 13 09 3A A9 AD AF B8 4C 54 B4 7D E2 5D 39 8A 2D C7 EE D0 32 63 C6 43 94 55 18 A1 0E 6B 96 22 AD 50 4B 75 11 BD C3 A9 E9 67 32 41 B5 F4 FD 9B CB 83 76 DA 92 63 32 CC 8D 2D 1B B4 74 FF C8 8D 9A 02 C6 47 CD 65 D7 7B C6 61 73 B5 3E AC 7F 03 2E 3E C0 5A F9 D0 7C 45 31 86 AB E0 03

Процедура создания публичного и приватного ключей:

1. Выбираем два случайных простых числа p и q
2. Вычисляем их произведение: N = p * q
3. Вычисляем функцию Эйлера: φ(N) = (p-1) * (q-1)
4. Выбираем число e (обычно простое, но необязательно), которое меньше φ(N) и является взаимно простым с φ(N) (не имеющих общих делителей друг с другом, кроме 1).
5. Ищем число d, обратное числу e по модулю φ(N) .Т.е. остаток от деления (d*e) и φ(N) должен быть равен 1. Найти его можно через расширенный алгоритм Евклида.

Возвращаемся в Cryptool:

Устанавливаем следующие значения

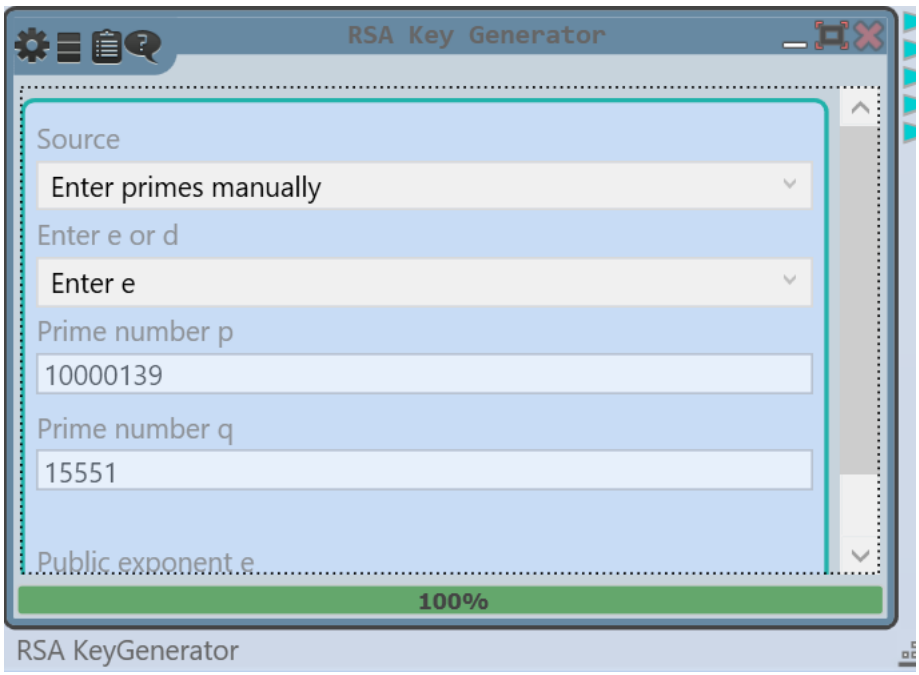

Рисунок 1 – Настройки RSA

Получаем:

p = 10000139

q = 15551

N = 10000139 * 15551 = 155 512 161 589

φ(N) = (10000139 – 1) * (15551 – 1) = 155502161450

e = 23

d = 87892526037

Закрытый ключ: {23, 155502161450}

Открытый ключ: {87892526037, 155502161450}

Шифрование:

При шифровании используется открытый ключ и открытый текст

Можно вычислить по формуле:

$c = E(m) = m^e (\mathrm{mod}\ N)$

Дешифрование:

При дешифровании используется закрытый ключ и шифротекст

Можно вычислить по формуле:

$m = D(c) = c^d (\mathrm{mod}\ N)$

1.1.2 Атака на алгоритм rsa на основе общего делителя для модуля

Возьмем два модуля шифрования: N1, N2, где N1 = p * q1, N2 = p * q2.

Отсюда мы можем найти p с помощью расширенного алгоритма Евклида:

p = gcd(N1, N2).

Пример:

p = 6369793

q1 = 4519289

q2 = 5374951

N1 = 28 786 935 437 177

N2 = 34 237 325 255 143

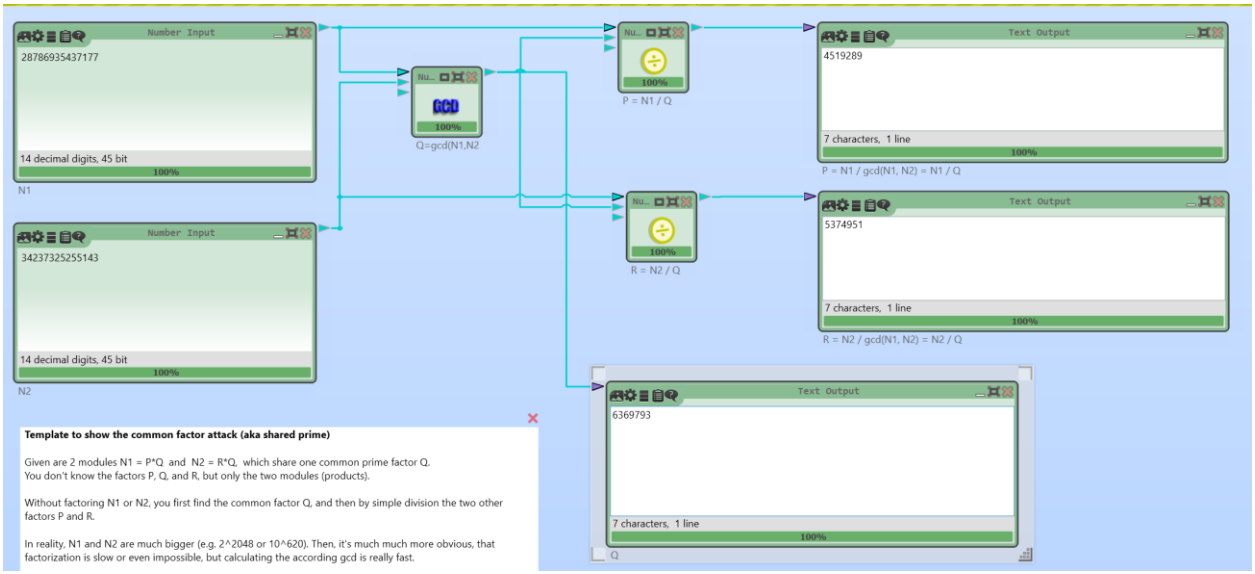

Рисунок 2 – Атака на RSA

Как мы видим из рисунка выше, нам удалось найти общий делитель двух модулей.

1.1.3 Программная реализация rsa

```
Plaintext: 123
Encrypted message: 1968
Decrypted message: 123
```

Рисунок 3 – Проверка реализации алгоритма

Выше мы видим запуск моей собственной реализации алгоритма RSA на языке C++. Как можем заметить, результат корректный. Листинг программы указан в конце работы в приложении.

1.1.4 Модификация программной реализации

1. Функция быстрого возведения в степень

В алгоритме уже содержится функция быстрого возведения в степень. В коде она выглядит следующим образом:

```
int powerMod(int base, int exponent, int modulus) {
    int result = 1;
    base = base % modulus;

    while (exponent > 0) {
        if (exponent % 2 == 1) {
            result = (result * base) % modulus;
        }
        exponent = exponent >> 1;
        base = (base * base) % modulus;
    }

    return result;
}
```

Рисунок 4 – Быстрое возведение в степень

Алгоритм построен на формуле:

$A^n = (A^{n/2})^2 = A^{n/2} * A^{n/2}$

То есть для четного n можно получить результат, выполнив всего $\log_2 n$ перемножений, что уже дает логарифмическую сложность. А в случае, когда n нечетно, приведем его к четному виду с помощью формулы:

$A^n = A^{n-1} * A^n$

2. Китайская теорема об остатках

```
int decrypt(int cipher, int d, int n) {
    int p, q;
    // Разложение n на простые множители p и q
    for (p = 2; p * p <= n; p++) {
        if (n % p == 0) {
            q = n / p;
            break;
        }
    }
    // Вычисление частных ключей
    int dp = d % (p - 1);
    int dq = d % (q - 1);
    int qInv = modInverse(q, p);

    // Вычисление остатков
    int mp = powerMod(cipher, dp, p);
    int mq = powerMod(cipher, dq, q);

    // Использование Китайской теоремы об остатках для нахождения оригинального сообщения
    int m = (qInv * (mp - mq) % p + p) % p;

    // Восстановление оригинального сообщения
    return mq + m * q;
}
```

Рисунок 4 - Китайская теорема об остатках

3. Атаки методом Ферма

Ключ RSA уязвим, если два простых числа p и q находятся близко. Если простые числа генерируются независимо друг от друга и случайным образом, то вероятность того, что они будут близки, ничтожно мала.

Однако функции генерации ключей RSA могут реализовывать ошибочный алгоритм, например такой:

Сгенерировать случайное простое число X;

Найти следующее простое число и присвоить его q;

Найти следующее простое число и присвоить его p;

Для стандартных размеров ключей RSA разница p и q — тысячи или меньше. Такой алгоритм является уязвимым для метода факторизации Ферма.

Реализация алгоритма тестирования на уязвимость на языке Python:

```
def fermat_attack(n):
    # Проверка входных данных
    if n < 2 or not isinstance(n, int):
        return "Неверный ввод"
    # Инициализация переменных
    x = math.ceil(math.sqrt(n)) # Округление вверх
    y = x**2 - n
    # Поиск квадратов
    while not math.isqrt(y)**2 == y: # Проверка, является ли y точным квадратом
```

```
            x += 1
            y = x**2 - n
        # Проверка уязвимости
        if x - math.isqrt(y) < n**(1/4):
            p = x + math.isqrt(y)
            q = x - math.isqrt(y)
            return p, q
        else:
return "n не уязвим"
```

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы был изучен и реализован алгоритм RSA. Была проведена атака на данный алгоритм. Также были реализованы модификации данного алгоритма, направленные на улучшение защищенности данного алгоритма в целях противодействия злоумышленникам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бабенко, Л. К. Современные алгоритмы блочного шифрования и методы их анализа / Л.К. Бабенко, Е.А. Ищукова. - М.: Гелиос АРВ, 2015. - 376 с.

2. Бабенко, Л.К. Современные интеллектуальные пластиковые карты / Л.К. Бабенко. - М.: Гелиос АРВ, 2015. - 921 с.

3. Болотов, А. А. Элементарное введение в эллиптическую криптографию. Протоколы криптографии на эллиптических кривых / А.А. Болотов, С.Б. Гашков, А.Б. Фролов. - М.: КомКнига, 2012. - 306 с.

4. Бузов, Геннадий Алексеевич Защита информации ограниченного доступа от утечки по техническим каналам / Бузов Геннадий Алексеевич. - М.: Горячая линия - Телеком, 2016. - 186 с.

ПРИЛОЖЕНИЕ

```
#include <iostream>
#include <cstdlib>
#include <ctime>

// Функция для проверки на простоту
bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) return false;
    }
    return true;
}

// Функция для поиска простого числа
int generatePrime() {
    int prime;
    do {
        prime = rand() % 100 + 50; // Генерация случайного числа (для примера)
    } while (!isPrime(prime));
    return prime;
}

// Функция для вычисления НОД (алгоритм Евклида)
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

```
// Функция для нахождения обратного числа по модулю (расширенный алгоритм Евклида)
int modInverse(int a, int m) {
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;

    if (m == 1) return 0;

    while (a > 1) {
        q = a / m;
        t = m;
        m = a % m, a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }

    if (x1 < 0) x1 += m0;

    return x1;
}

// Быстрое возведение в степень по модулю
int powerMod(int base, int exponent, int modulus) {
    int result = 1;
    base = base % modulus;

    while (exponent > 0) {
        if (exponent % 2 == 1) {
            result = (result * base) % modulus;
        }
        exponent = exponent >> 1;
        base = (base * base) % modulus;
    }
```

```
    return result;
}

// Генерация ключей RSA
void generateRSAKeys(int& e, int& d, int& n) {
    srand(static_cast<unsigned int>(time(0)));

    // Генерация простых чисел p и q
    int p = generatePrime();
    int q = generatePrime();

    // Вычисление модуля n
    n = p * q;

    // Вычисление функции Эйлера (φ(n))
    int phi = (p - 1) * (q - 1);

    // Выбор открытой экспоненты e
    do {
        e = rand() % phi;
    } while (gcd(e, phi) != 1);

    // Вычисление закрытой экспоненты d
    d = modInverse(e, phi);
}

// Шифрование сообщения
int encrypt(int message, int e, int n) {
    return powerMod(message, e, n);
}

// Дешифрование сообщения
int decrypt(int cipher, int d, int n) {
    return powerMod(cipher, d, n);
}
```

```
int main() {
    int e, d, n;

    // Генерация ключей
    generateRSAKeys(e, d, n);

    // Пример использования
    int message = 21;
    std::cout << "Plaintext: " << message << std::endl;

    // Шифрование
    int cipher = encrypt(message, e, n);
    std::cout << "Encrypted message: " << cipher << std::endl;

    // Дешифрование
    int decryptedMessage = decrypt(cipher, d, n);
    std::cout << "Decrypted message: " << decryptedMessage << std::endl;

    return 0;
}
```