Университет ИТМО

Факультет программной инженерии и компьютерной техники

Информационная безопасность

Отчет по лабораторной работе № 5

«Атака на алгоритм шифрования RSA, основанная на Китайской теореме об остатках»

Вариант 24

Выполнил студент группы P34302

Ким Даниил Кванхенович

Проверил преподаватель

Рыбаков Степан Дмитриевич

Санкт-Петербург 2024

Содержание

Цель работы:

- изучить атаку на алгоритм шифрования RSA посредством Китайской теоремы об остатках.

Порядок выполнения работы:

1. Ознакомьтесь с теорией в подразделе «Атака на основе Китайской теоремы об остатках».
2. Получите вариант задания у преподавателя. Экспонента для всех вариантов e = 3.
3. Используя Китайскую теорему об остатках, получите исходный текст.
4. Результаты и промежуточные вычисления значений для любых трех блоков шифрованного текста оформите в виде отчета.

Данные варианта:

Вариант					
24					
Модуль сравнения			**Блок шифротекста**		
N_1	**N_2**	**N_3**	**C_1**	**C_2**	**C_3**
590059443367	586035939793	582032534407	534935192069 586334468916 575821575470 158445010924 168022188272 419451618702 403150327598 462915818163 156960926738 423280293357 308065052008	70956316615 196061328294 472946437612 167175113770 213280914294 97582680057 87487791156 319786583031 526032348303 561873181810 93452497746	547351293988 558349441596 209735294323 257527905634 328543700761 241383661927 318686253990 391540759391 124252499803 400043751247 36326931192

Выполнение:

Описание алгоритма:

RSA – асимметричный алгоритм блочного шифрования основанный на вычислительной сложности разложения на множители больших чисел (задача факторизации). При шифровании данных применяется открытый ключ, а при дешифровании достаточно обладать закрытым ключом. Обратимость шифрования за счет теоремы Эйлера и ограничений, накладываемых на генерируемые ключи.

В алгоритме используются следующие основные переменные:

- ***p** и **q*** – большие простые числа используемые для генерации ключей.
- ***N*** – модуль сравнения равный сумме p и q. Является частью обоих ключей – закрытого и открытого.
- ***E*** – большое простое число, взаимно простое со значением *функции Эйлера φ()* от числа *N*. В контексте алгоритма *RSA* называется *открытой экспонентой*. Входит в состав открытого ключа и используется при шифровании данных.
- ***d*** – число обратное к числу *E* по модулю *φ(N)*. В контексте алгоритма *RSA* называется *секретной экспонентой*. Входит в состав закрытого ключа и используется при дешифрации данных.
- ***m*** – передаваемый блок сообщения. Значение не должно превышать модуль *N*.
- ***C*** – блок передаваемого шифротекста.

Используемые ключи:

- Открытый ключ: $\{N; E\}$
- Закрытый ключ: $\{N; d\}$

Числа p, q и *φ(N)* используются для генерации ключей и не используются в процессе шифровании и дешифровании данных.

Используемые функции:

- Функция шифрования: $m^E \bmod N = C$
- Функция дешифрования: $\mathrm{C}^d \bmod N = m$

Достоинства метода:

- Нет необходимости в надежном канале для передачи секретного ключа.
- В больших сетях число ключей в ассиметричной системе значительно меньше, чем в симметричной.

Недостатки метода:

- Необходимы большие ключи по сравнению с симметричными алгоритмами шифрования для обеспечения того же уровня криптостойкости.
- Работа алгоритма подразумевает потребление существенно большего количества вычислительных ресурсов.

Листинг разработанного модуля RSA:

Вспомогательные функции для генерации ключей и вычисления значений используемых алгоритмом переменных:

```
def getN(_p1, _p2):
    _N = _p1 * _p2
    return _N


def getEuler(_p1, _p2):
    _Euler = (_p1 - 1) * (_p2 - 1)
    return _Euler


def getD(_p1, _p2, _e):
    _EulerN = (_p1 - 1) * (_p2 - 1)
    _D = pow(_e, -1, _EulerN)
    return _D


def encodeM(_m, _e, _n):
    _m_encoded = (_m ** _e) % _n
    return _m_encoded


def decodeM(_m, _d, _n):
    _m_decoded = (_m ** _d) % _n
    return _m_decoded
```

Основная функция разработанного модуля RSA:

```
def RSA():
    p1 = int(input("[ВВОД] Введите секрет - простое число p"))
    p2 = int(input("[ВВОД] Введите секрет - простое число q"))

    N = getN(p1, p2)
    Euler = getEuler(p1, p2)

    E = int(input(
        "[ВВОД] Введите число E взаимно простое с ϕ(N) = "
        + str(Euler)))

    d = getD(p1, p2, E)

    print("[ВЫВОД] Открытый ключ = [N = {0}, E = {1}]"
          .format(N, E))
    print("[ВЫВОД] Закрытый ключ = [N = {0}, d = {1}]"
          .format(N, d))

    m = int(input("[ВВОД] Введите код m меньше, чем N = "
                  + str(N)))

    m_encoded = encodeM(m, E, N)

    m_decoded = decodeM(m_encoded, d, N)

    print("[РЕЗУЛЬТАТ] Открытый текст:       " + str(m))
    print("[РЕЗУЛЬТАТ] Расшифрованный текст: " + str(m_decoded))
```

Вывод программы:

```
[ВВОД] Введите секрет - простое число p>? 7
[ВВОД] Введите секрет - простое число q>? 17
        N = 119
        ϕ(N) = 96
[ВВОД] Введите число E взаимно простое с ϕ(N) = 96>? 5
       d = 77
[ВЫВОД] Открытый ключ = [N = 119, E = 5]
[ВЫВОД] Закрытый ключ = [N = 119, d = 77]
[ВВОД] Введите код m меньше, чем N = 119>? 19
       C = 66
       m` = 19
[РЕЗУЛЬТАТ] Открытый текст:       19
[РЕЗУЛЬТАТ] Расшифрованный текст: 19
```

Описание уязвимости:

При неправильном или неоптимальном выборе значений переменных для генерации ключей становятся возможны специальные криптографические атаки, такие как атака *малых экспонент*.

Положим, что в качестве открытого ключа используется единственное малое значение модуля – $E = 3$ и случайное N. В таком случае, если одна сторона отсылает одинаковое сообщение некоторым пользователям, путь и зашифрованное открытым ключом каждого из них, злоумышленник, подслушав шифротексты и открытые ключи, имеет возможность восстановить исходный текст.

Пусть было прослушано 3 шифротекста C_1, C_2 и C_3 соответственно. Можно составить систему, где значение C определено функцией шифрования:

- $C_1 = m^3 \pmod{N_1}$
- $C_2 = m^3 \pmod{N_2}$
- $C_3 = m^3 \pmod{N_3}$

Согласно китайской теореме об остатках, для каждого из N и C найдется единственное число X, которое при делении на N даёт остаток C:

- $X = C_1 \pmod{N_1}$
- $X = C_2 \pmod{N_2}$
- $X = C_3 \pmod{N_3}$

и удовлетворяет сравнению:

- $X = \sum C * M * M\_INV$,

  где $M = N^{-1} * \prod N$, а M_INV – обратное ему по модулю.

Т.к. по значение исходных данных не может превышать модуль N, решение X совпадет с m^3. Зная m^3 и вычислив кубических корень, можно восстановить исходный текст.

Промежуточные вычисления значений:

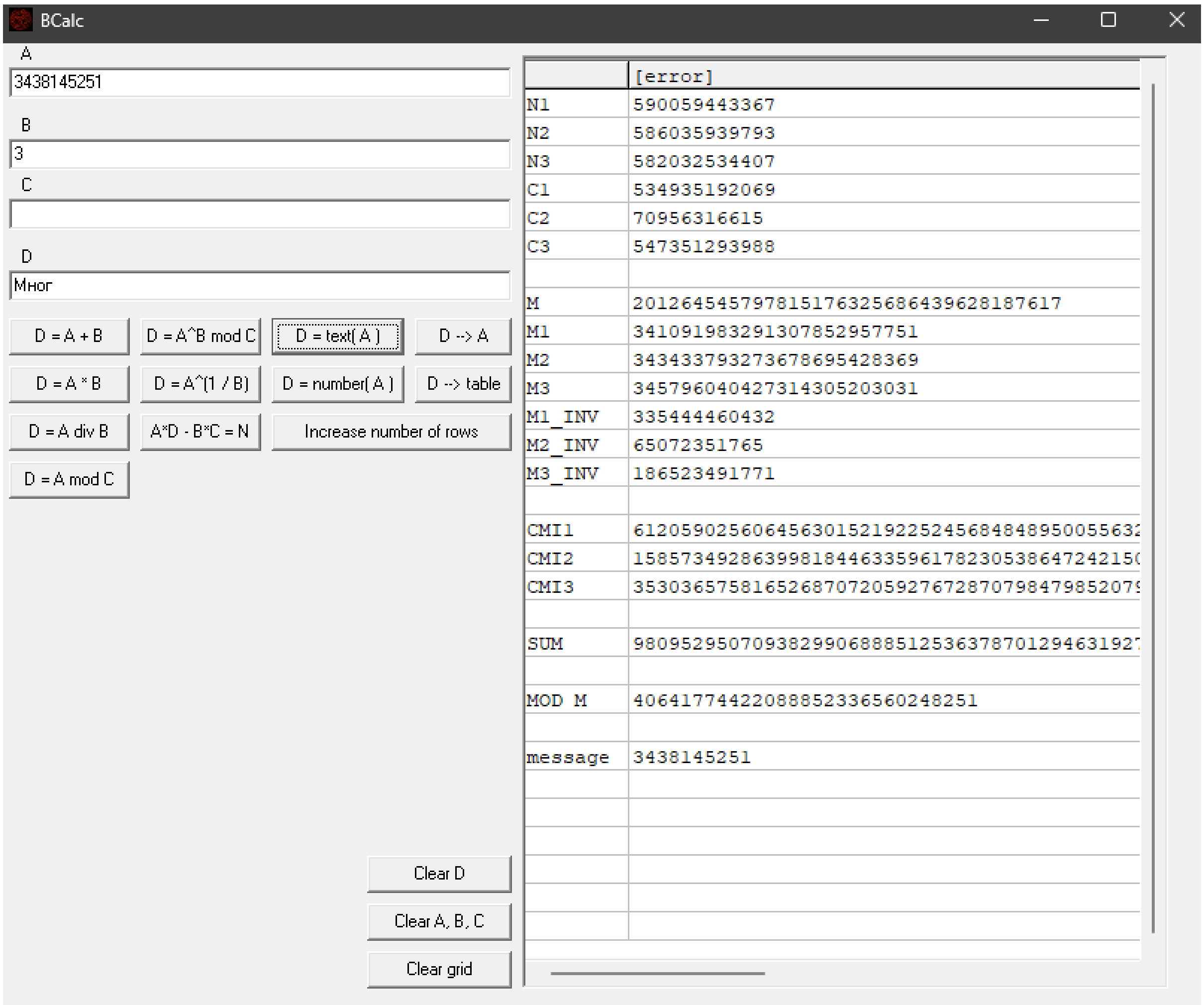

Рисунок 1. Выполнение вычислений в программе BCalc

Листинг разработанного модуля:

```
def win1251_decoder(digit):
    return digit.to_bytes(4, byteorder='big').decode('cp1251')


e = 3
N1 = 590059443367
N2 = 586035939793
N3 = 582032534407
C1 = […]
C2 = […]
C3 = […]


# Шаг 1
M = N1 * N2 * N3
# Шаг 2
M1 = N2 * N3
M2 = N1 * N3
M3 = N1 * N2
# Шаг 3
M1_INVERSE = pow(M1, -1, N1)
M2_INVERSE = pow(M2, -1, N2)
M3_INVERSE = pow(M3, -1, N3)

for i in range(11):
    # Шаг 4
    CMI1 = C1[i]*M1*M1_INVERSE
    CMI2 = C2[i]*M2*M2_INVERSE
    CMI3 = C3[i]*M3*M3_INVERSE

    S = CMI1 + CMI2 + CMI3
    S_MOD_M = S % M

    message = S_MOD_M ** (1/e)

    print(win1251_decoder( round(message)), end="")
```

Вывод программы:

```
Многие анализаторы имеют генераторы трафика
```

Вывод:

В ходе данной лабораторной работы был закреплен материал об алгоритмах асимметричного блочного шифрования, в частности об RSA. Был получен опыт его программной реализации. Так же была проиллюстрирована уязвимость алгоритма при неоптимальном выборе значений переменных, используемых алгоритмом. Была проиллюстрирована атака посредством малых экспонент или атака, основанная на Китайской теореме об остатках.