1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. **Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «АТАКИ НА КРИПТОСИСТЕМУ RSA БЕЗ РАЗЛОЖЕНИЯ ЧИСЛА НА МНОЖИТЕЛИ»
2. по дисциплине «Криптографические методы защиты информации»
3. Выполнил
4. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. ассистент Ярмак А.В.

<*подпись*>

1. Санкт-Петербург
2. 2021

**Цель работы**

Изучение уязвимостей криптосистемы RSA по отношению к атакам без разложения на множители, генерация параметров криптосистемы RSA.

**Задачи**

1. Разработать программу (**П-1**), которая реализует разложение составного числа на множители по известным показателям RSA и вычисляет закрытый ключ другого пользователя в случае общего модуля *n*.
2. Разработать программу (**П-2**), которая реализует атаку Винера на криптосистему RSA.
3. Разработать программу (**П-3**), которая, в соответствии с вариантом задания, реализует одну из следующих атак:
4. бесключевое дешифрование сообщения в случае малого порядка *e* в .
5. Разработать программу (**П-4**), которая осуществляет генерацию параметров криптосистемы RSA и проверить CryptoTool.

**Ход работы**

**Атака в случае общего модуля n**

Если пользователи А и В имеют общий модуль n, то пользователь B может найти секретное разложение на множители, то есть p и q. А так же и значение параметра (секретный ключ (n,d)). Тогда атаку можно описать следующим образом: число представить в виде , где s- нечетное. Затем выбирается случайное число а в диапазоне .

Вне цикла получим квадрат по модулю n от и сохраняем в b. Далее в цикле l (эль) раз выполняется , пока . А если окажется, что равен b равен -1, то заново находим случайное число а и повторяем алгоритм. Если все верно, то затем находим функцией НОД значения p и q. Зная p и q находим . А затем находим параметр d секретного ключа : . Таким образом, находится секретное разложение p и q, а так же .

|  |
| --- |
|  |

Рисунок 1 – Использованные параметры.

Это означает, что значения n должны быть разными.

**Атака Винера на криптосистему RSA**

В целях ускорения алгоритма секретный показатель d делают малым(Тогда расшифровывающая экспонента e будет большой). Однако такой способ шифрования приводит к снижению стойкости. Поэтому возможна атака Винера:

по открытому ключу можно вычислить секретный показатель d. А p,q имеют примерно одинаковое значение(занимают примерно одинаковое количество байт), удовлетворяющее диапазону .

И если , то значение закрытого показателя может быть вычислено за полиномиальное от n время.

Тогда алгоритм можно описать следующим образом:

Сгенерируем небольшую d и найдем e по формуле . Взломщику подается на вход шифрующая экспонента e и n. В процедуре сгенерируем открытый текст m и зашифруем его открытым ключом. Затем представим в виде непрерывной дроби, то есть запишем в список целые части от деления. После этого по формуле в цикле. Предполагаемым d (или Q) расшифровываем ранее зашифрованный текст m и проверяем совпали ли они. Если это так, то мы нашли d.

|  |
| --- |
|  |

Рисунок 2 – Предполагаемый d совпал с оригинальным d.

Таким образом, d не должен быть маленьким.

**Бесключевое дешифрование сообщения в случае малого порядка *e.***

Для ускорения процесса шифрования иногда используются малые открытые показатели e. Тогда возможна атака из-за малого значения показателя e. На вход атакующей процедуре подаётся открытый ключ и шифртекст. Тогда до цикла сохраняем поданный на вход шифртекст. В цикле перед зашифрованием шифртекста() сохраняем его в отдельной переменной cprev и при совпадении ново-полученного шифртекста с шифртекстом, сохраненным до цикла, возвращаем cprev. (это и есть ). Таким образом, получаем открытый текст.

Сгенерируем малые значения n и e, и зашифруем сообщение m=12345. Тогда алгоритм возвращает:

|  |
| --- |
|  |

Рисунок 3 – Успешное дешифрование и совпадение открытых текстов.

Таким образом, е не должен быть малым.

**Генерация собственных параметров RSA**

Для генерации простых чисел воспользуемся функцией GetStrongPrime, импортированной из Cryptodome. На вход эта функция принимает количество бит (не меньше 512). Таким образом были получены p и q.

Далее сгенерируем открытый ключ (и секретный) для проверки с помощью CrypTool. Для этого в процедуру передадим только что сгенерированные p,q. Найдём границы для e: нижняя граница найдена сравнением, для того чтобы сузить границы поиска е и начать с наибольшего минимального. А верхняя граница – это функция . Перебор в цикле продолжается пока e взаимно простой с .

512 бит

p = 0xe168c3c35857fed74ace925365660ed3bab5d7238ae31a35aabde39f0879da7cd7a1dc7d9cc6b356d39cb48b919758a73d239b2b0f037f7fc0a7eaf44b062d31

q = 0xd979132f4350d3693caff42664631afc39eb9578d24274ff1de910c370873b20dfe59ca88ec21081ac98f81c5f13e70023c01e33fa6e18586485bf644f0bfdcd

n = 0xbf7c695981405f270229a55aedeb5a478ed56592050d36356807aa2b9209ec07c094182bb7cf5d04949cc3ee6e77f2119676dcb86181d7ba0d0351c2b08a3b820fe0f2ee07dfa84e6fdc5b162b970c51edf6a8854db0614017ebc3ada454e4a84a33dfc399bcf4927425522946d81f3f79ba05080d4703873de6eb1f3eb69d3d

e = 0x63a35b3aa4dfb4cdeba4a888efd4c265aa533158f1b6109e9483a7e683475ebca12277e589c58eb0214929923bff8260a4fb896358ec361e8894c0f2b242d9168c22f4bed3255a103a4fcd177c5f269b8b315d06f2f3d16afcbf0fcb15bed04d92075814a106b7c3410b3cc8a49111077b5c8e1fcb19ad31e2de1fa07d14b051

|  |
| --- |
|  |

Рисунок 4 – Быстрый взлом при 512 бит.

2048 бит

p = 

q = 

n = 

e = 

|  |
| --- |
|  |

Рисунок 5 – За 7 часов факторизация не окончена.

### Контрольные вопросы

1. При каком условии возможна атака Винера?

* Значение закрытого показателя d мало и
* Простые p,q имеют примерно одинаковую разрядность(длину байт)
* Взломщику известна шифрующая экспонента e.

1. Перечислите методы противодействия атаке Винера.

* Значение закрытого показателя d должно быть
* Увеличить разрядность p и q, тогда количество α из непрерывных дробей будет больше

1. Почему числа  и  необходимо выбирать таким образом, чтобы  имели большие простые делители?

* Для того, чтобы обеспечить стойкость против атаки повторным шифрованием.
* Для того, чтобы усложнить поиск разложения числа при помощи известных алгоритмов факторизации. Например, если число сильно составное, то эффективно работает p-1 метод Полларда.

1. Предложите способы защиты от атаки бесключевого дешифрования широковещательных сообщений без отказа от использования общих малых закрытых показателей.

Чтобы расшифровать сообщение, необходимо чтобы количество адресатов ( а значит и шифртекстов ) было не меньше открытого показателя . То есть для взлома нужно, чтобы k>e.

Тогда для обеспечения криптостойкости системы необходимо взять , при этом может оставаться малым, если также мало.

1. Обеспечивают ли часто используемые открытые показатели вида  стойкость к методу криптоанализа на основе частично известных открытых текстов?

Сложность подобной атаки полиномиальная – , а минимальный размер открытого показателя для криптосистемы RSA (для RSA-512) – , что много больше . Таким образом, стойкость системы при использовании подобного открытого показателя не обеспечивается. В настоящее время, требуется использовать размеры открытого показателя порядка 2\*1011

**Вывод**

В результате данной лабораторной работы были изучены возможные атаки на криптосистему RSA из-за неверного выбора параметров для криптосистемы. Это означает, что выбор параметров для криптосистемы является ответственной задачей.

**Листинг**

import logging  
import random  
import math  
import sys  
from Cryptodome.Util.number import inverse  
from datetime import datetime  
from logging import info  
from gen\_params import \*  
from main1 import RSA  
  
logging.basicConfig(stream=sys.stdout, level=logging.INFO, format=**"%(message)s"**)  
  
  
def Alg2\_1(n, ea, eb, db):  
 N = eb \* db - 1  
 f = 0  
 s = N  
 while s % 2 == 0: *# поиск показателя степени, проверкой на четность, на последней итерации s - нечетное число (1)* f += 1  
 s //= 2  
 while True:  
 a = random.randint(2, n) *# случ а* b = pow(a, s, n) *# a^s(mod n) (2)* t = b  
 b = pow(b, 2, n) *# b^2(mod n)* while b != 1: *#инвариат цикла, b меняется в тееле цикла, а 1(mod n) нет, поэтому по нему и сверяем (3)* t = b  
 b = pow(b, 2, n) *# многократное действие b^2(mod n) и есть b^(2^l)(mod n)* if t != n - 1: *# если не выполнено, то это означает, что неудовлетворяет, что t=-1 (mod n). Если вып. ,значит нашли верное значение и возврата на п.2 не будет* break  
 p = math.gcd(t + 1, n) *# (4)* q = math.gcd(t - 1, n) *# (4)* phi = (p - 1) \* (q - 1)  
 d = inverse(ea, phi) *# d=e^(-1)mod n* return p, q, d *# делители p и q числа n и закртый ключ другого пользователя d\_a*def contFrac(n, d):  
 cf = []  
 while True:  
 cf.append(n // d) *# сохраняем цулю часть от деления* rem = n % d *# находим остаток от деления* n = d *# а n стал делителем* d = rem *# становится остатком* if d == 0: *# пока остаток не будет равен нулю* break  
 return cf  
  
  
def wiener(n, e):  
 m = random.randint(2, n) *# генерим случайную переменную-открытй текст* me = pow(m, e, n) *# зашифровываем её RSA* cf = contFrac(e, n) *# представляем e и n в виде непрерывной дроби е/n* cf = cf[1:]  
 Q\_1 = 0 *# q1=a1, при a1=0* Q\_0 = 1 *# по определению* for a in cf: *# берем альфу из списка альф* Q = a \* Q\_0 + Q\_1 *# умножаем по формуле* m1 = pow(me, Q, n) *# (m^e)^Q (mod n) предполагаем, что получили верный открытй текст* if m1 == m: *# если предположенный открытй текст совпадает с изначальным открытм текстом, то мы нашли d, то есть Q* return Q  
 Q\_1 = Q\_0  
 Q\_0 = Q  
 return -1  
  
  
def Alg2\_4(n, c, e):  
 c0 = c  
 while True:  
 cprev = c  
 c = pow(c, e, n)  
 if c == c0:  
 return int(cprev)  
  
  
def main():  
 *#gen\_prime\_test(256)  
 #gen\_prime\_test(512)  
 #gen\_prime\_test(1024)  
 #gen\_prime\_test(2048)* db = 0x77E693EAA7C0DFD69AEB21130E0DF891178FA230CC906D095D06A1830164E4EF6375295EAA6A19FAD30E7BB4972FFBDB71A937AA2CEE3BC1ADA1C57B30A217A7  
 eb = 0x35A362569BE3465B0FA287859CBA9DB13764C3DCE77853FC63734DCA752A6232838A4ED699C52F86649E695CDD8F09E76956985F67FA57D6FABD19C1CE3F9617  
 n = 0x78D8EFEF0A397FD92863E4C4DC70928B1EBACCE29FBBB3F8E661863461F56C0B31388ED75B31BDDA9712E2D0B595109483BC0096DF3F24ACC61CD527E7F0C52D  
  
 ea = 5685263797886913133055847610347187504480134889004652024264626436304933782114662123911569869356533305197344321092228167848937365536641903309711355548938559  
 da = 5226817542677159244553252195007976004700966434512429664764434720961625989142294250652544960709510363258991954113748694974976144452897286512889855916603071  
  
 ec = 3797575492591847160166531613910594781660890857981463889314985346172721230652388858388733802861236016930543825262888128179555525095674526709832617718246093  
 dc = 5  
 random.seed(datetime.now())  
 info(**"Start"**)  
 info(**f"n =** {hex(n)}\n**ea =** {hex(ea)}\n**eb =** {hex(eb)}\n**db =** {hex(db)}**"**)  
 p, q, da1 = Alg2\_1(n, ea, eb, db)  
 info(**f"p =** {hex(p)}\n**q =** {hex(q)}\n**p\*q =** {hex(p \* q)} {p \* q == n}\n**da =** {hex(da1)}\n**da==da1 ->** {da == da1}**"**)  
*#------------------------------------2* print(**"wiener"**)  
 dc = 2  
 while math.gcd(dc, (p - 1) \* (q - 1)) != 1: *# генерируем новый d, чтобы он удовлетворял: d < 1/3(n)^1/4 и был взаимнопросытм с phi* dc = random.randint(0.1 \* n \*\* (1 / 4), 0.3 \* n \*\* (1 / 4))  
 ec = inverse(dc, (p - 1) \* (q - 1)) *# тогда находим и его е* info(**f"check** {pow(pow(5, dc, n), ec, n)}**"**) *# быстро зашифровали и расшифровали и получли сразу открытый текст* info(**f"dc =** {dc}\n**ec =** {ec}**"**) *# как в снарте: небольшая d и большая е* dc1 = wiener(n, ec) *# взломщику дана шифрующая экспонента е и n* info(**f"dc1 =** {dc1} {dc == dc1}**"**) *# проверяем,что этот d и есть dc(который d < 1/3(n)^1/4)  
#-----------------------------------2  
  
#-----------------------------3  
 #m = 346#2#234913748356* m = 123456*#2#234913748356  
 #n1=61\*53* n1=983\*563  
 e=49  
 c = pow(m, e, n1)  
 m2 = Alg2\_4(n1, c, e)  
 info(**f"m2 =** {m2} {m == m2}**"**)  
*#----------------------------3* q1 = gen\_prime(2048)  
 p1 = gen\_prime(2048)  
 *#p1 = 11805639674883327389697576828531382092379989958487602683441100520350303251334412940606356298195506804331004286195037100488503367678093787202514691966774577  
 #q1 = 11389982538815619941440368107386417795256506064244111841928907833067376651190762898414040405242336151354943488105465521902170088901777994439235249111825869* info(**f"good params**\n**p =** {hex(p1)}\n**q =** {hex(q1)}\n**n =** {hex(p1 \* q1)}**"**)  
 ea, da, n = gen\_keys(p1, q1)  
 eb, db, n = gen\_keys(p1, q1)  
 info(**f"ea =** {hex(ea)}\n**da =** {hex(da)}\n**eb =** {hex(eb)}\n**db =** {hex(db)}\n**"**)  
 info(**f"eb!=ea** {eb != ea}**"**)  
 info(**f"db!=da** {db != da}**"**)  
  
 *#p, q, da1 = Alg2\_1(n, ea, eb, db)  
 #info(f"p = {p}\nq = {q}\np\*q = {p \* q} {p \* q == n}\nda = {da1}\nda==da1 -> {da == da1}")  
  
 #dc1 = wiener(n, ec)  
 #info(f"dc1 = {dc1} {dc == dc1}")*main()