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

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. **«АТАКА ПО ВРЕМЕНИ НА КРИПТОСИСТЕМУ RSA»**
2. по дисциплине «Криптографические методы защиты информации»
3. Выполнил
4. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

* + - 1. Преподаватель

1. ассистент Ярмак А.В.

<*подпись*>

1. Санкт-Петербург
2. 2021

**Цель работы**

изучение атак на RSA, основанных на неправильной реализации криптосистемы, выполнение атаки по времени на криптосистему RSA.

**Задание**

* Получить у преподавателя вариант задания и разработать программу **П-1**, которая выполняет атаку по времени на некорректную реализацию RSA.
* Предоставленный бинарный файл выполняет расшифрование поданного на вход шифртекста алгоритмом RSA, используя фиксированный неизвестный закрытый ключ, и возвращает соответствующий открытый текст и время, затраченное на расшифрование.
* В качестве входных данных программа **П-1** должна принимать параметры и (число запросов для вычисления одного бита), модуль , на выходе – выдавать найденный множитель , значение закрытого показателя .

**Ход работы**

Параметры l (Neighborhood size) =6

s=1 (Sample size)

delta=500000

Чтобы достичь стабильного времени дешифрования (малое отклонение времени), параметр s должен быть не большим. Тогда получится найти delta.

Параметр l должен быть чуть больше, ведь путём его увеличения, мы можем уменьшить zero-one-gap (связан с дельтой) и лучше различать 0 или 1.

Вариант 4 предоставляет n=0xbdca61f2bdc1b76bda3b7811df28052e0501ad5aad2d206e6b8eb5e578939692d2a7f679fb194104ef79bb90c71f9b1f66c02f830b0b9afad9a527a0ae8fedfb78a178e3fad6a98c9cf95cceafda851eb24fd881c45ec3bd4429a8364f993c4ba1f69020442d00c018b72f8b56edd00dc586f5a485859b752e56cc853361457f

e=0x010001

В результате выполнения временной атаки было получено q=11736881569571073867692224734711427178345958674570067830728999705090754578720902306809364298085544632649347560416171575972262793021360506748752259755533591

в 10 системе счисления.

Далее для того чтобы убедиться, что был найден правильный q сверимся с подсказкой, которую выслал преподаватель,где был указан, что старший байт делителя n – это e0. Но для этого переведем полученный q в 16-ричную систему счисления:

|  |
| --- |
|  |

Рисунок 1 – Сверяем q с подсказкой преподавателя.

Это означает, что q был найден верно.

Подсчитаем кол-во запросов для вычисления **одного бита** q алгоритмом Брамли-Бонея. Запросы на cryptor посылаются посылаются с помощью метода interact. Метод interact посылает шифртекст внутри цикла s дважды. Параметр s=1, тогда вложенный цикл посылает 2 (1\*2=2) раза запрос. Вложенный цикл обрамлён внешним циклом l, который вызывается 6 раз, так как l=6. Тогда потребовалось 12 запросов (2\*6=12) для вычисления одного бита q.

Пример вычисления трёх битов числа q

Рассмотрим восьмой (8) бит. При этом управляющий шаг j равен 7.

g=11731831938699772462127271873430115361544445093018344205508116263256543526314353604701640010896040499228777875913175294497034647460453248703129442880323584

Чтобы получить q\_ нужно поставить 1 на нужную позицию. Для этого совершаем побитовое смещение на 504 позиции (511-7=504) и делаем побитовую дизъюнкцию, для того чтобы единица “встала” на место справа к старшим разрядам (которые уже подобраны на предыдущих шагах). Тогда g\_=11784206188426110732047482908579356947979911365755033242139848925146081667056828397579772332373254965743192062859216255633182123565187414991982699321753600

R=13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649006084096

Нулевая итерация во внешнем цикле(l)

Вложенный цикл (s)

Среднеарифметическое времени от шифртекста u\_g равно 100000.0

Среднеарифметическое времени от шифртекста u\_g\_ равно 8000

К 6 итерации получаем значения

u\_g=17062311631683047089769752015118405108242679930587175863361544351485109285353800399162012707593734088519303440683338733739551160108677324105944388412195873331360454910493927168002238981660902504336493359283155528056186097235798769982255977457026874537748070988803533451828819505788602976679506456053000085002

u\_g\_=84220649666613620625976563291981246906553601949620295809975884103345028360338278284929469640513336057259280973279686964619321274574770722709707606766883971479661680835129287109332101620702676407772908299770890630323163337126459055142821695199230763488068467711241893477257836796530636923093789644983905198607

Время запроса от u\_g [100000.0, 106000.0, 111000.0, 106000.0, 111000.0, 95000.0]

Время запроса от u\_g\_ [8000.0, 5000.0, 6000.0, 4000.0, 9000.0, 5000.0]

Тогда суммарное время запросов равно от u\_g равно 629000.0, а от u\_g\_= 37000.0

Таким образом, 629000.0 - 37000.0 =592000.0 дельта оказалась “большой” и выставляется 0.

**9-ый бит:**

j=8

Теперь единица ставится на 503 позицию (511-j). При этом g=11731831938699772462127271873430115361544445093018344205508116263256543526314353604701640010896040499228777875913175294497034647460453248703129442880323584

Тогда g\_=11758019063562941597087377391004736154762178229386688723823982594201312596685591001140706171634647732485984969386195775065108385512820331847556071101038592

Нулевая итерация во внешнем цикле(l)

Вложенный цикл (s)

Среднеарифметическое времени от шифртекста u\_g равно 100000.0

Среднеарифметическое времени от шифртекста u\_g\_ равно 4000.0

К 6 итерации получаем значения

u\_g=17062311631683047089769752015118405108242679930587175863361544351485109285353800399162012707593734088519303440683338733739551160108677324105944388412195873331360454910493927168002238981660902504336493359283155528056186097235798769982255977457026874537748070988803533451828819505788602976679506456053000085002

u\_g\_=11727921141249029829690007116857652174525704030782652152013100483391173271120273057753593099896600834419717689048796690772680501309102600031698640029806764769994949359648080149905649136937827 8290472228832491597444229776164514497257528061408971555933630280445942829768373496151509415281540127177106201831359180

Время запроса от u\_g [100000.0, 106000.0, 111000.0, 106000.0, 111000.0, 95000.0]

Время запроса от u\_g\_ [4000.0, 4000.0, 2000.0, 3000.0, 2000.0, 5000.0]

Тогда суммарное время запросов равно от u\_g равно 629000.0, а от u\_g\_= 20000.0

Таким образом, 629000.0 - 20000.0= 609000.0 дельта оказалась “большой” и выставляется 0.

**10-ый бит**

j=9

Теперь единица ставится на 502 позицию (511-j). При этом

g=11731831938699772462127271873430115361544445093018344205508116263256543526314353604701640010896040499228777875913175294497034647460453248703129442880323584

g\_=11744925501131357029607324632217425758153311661202516464666049428728928061499972302921173091265344115857381422649685534781071516486636790275342756990681088

Нулевая итерация во внешнем цикле(l)

Вложенный цикл (s)

Среднеарифметическое времени от шифртекста u\_g равно 100000.0

Среднеарифметическое времени от шифртекста u\_g\_ равно 5000.0

К 6 итерации получаем значения

u\_g=17062311631683047089769752015118405108242679930587175863361544351485109285353800399162012707593734088519303440683338733739551160108677324105944388412195873331360454910493927168002238981660902504336493359283155528056186097235798769982255977457026874537748070988803533451828819505788602976679506456053000085002

u\_g\_=67170761522086672693334911591847463426749860119206848691746274592698420998278265488348971853279871216358240165585652820733178086599851662211465394355131760515654974253487364333529365175519590397404361095887376486142981130875148013755158693214291404084014258465816650912662485507601942258403341781127415722091

Время запроса от u\_g [100000.0, 106000.0, 111000.0, 106000.0, 111000.0, 95000.0]

Время запроса от u\_g\_ [5000.0, 2000.0, 5000.0, 5000.0, 4000.0]

Тогда суммарное время запросов равно от u\_g равно 629000.0, а от u\_g\_= 28000.0

Таким образом, 629000.0 - 28000.0= 601000.0 дельта оказалась “большой” и выставляется 0.

Время, потраченное на реализацию атаки = 54 секунды

Суммарное количество запросов = 6 112

(4 за первый цикл, 509\*12(кол-во запросов на 1 бит))

Вычисленное значение закрытого показателя d=

### Контрольные вопросы

1. Сформулируйте методы противодействия атакам по времени для криптосистемы RSA.

Из-за того что атака основана на предположении, что различные операции выполняются за разное время, можно измеряя время вычислений получить информацию о секретном ключе. Тогда можно защитится следующим образом:

* RSA blinding вычисляет до самой операции дешифрования , где r – случайное число, e – секретная экспонента, g – шифрованный текст. Затем x расшифровывается как обычно с последующим делением на r, то есть . А так как r случайное и x случайное время расшифрования не должно показывать информацию о ключе. И r всегда должен быть новым случайным числом при каждой расшифровке. Но это снижает производительность на 2%-10%.
* Квантование. Требовать от всех вычислений RSA быть квантоваными, то есть всегда выполняться за заранее определенный квант времени. Например, брать максимум от всех дешифрований.

1. Чем обусловлен выбор значения в алгоритме Монтгомери?

Произведение Монтгомери необходимо для ускорения выполнения операций умножения и возведения в степень, нужных при возведении числа в степень по модулю. Операции умножения и деления на R выполняются очень быстро, так как при  представляют собой просто сдвиги бит. Таким образом алгоритм Монтгомери быстрее обычного вычисления {\displaystyle a\cdot b\mod {n}}, которое содержит деление на n.

**Вывод**

В данной работе мы познакомились со способом атаки на RSA по side-channel, где используются факт того, что различные операции в устройстве выполняются за различное время. Мы провели свою собственную атаку, где подтвердили, что таким образом можно действительно найти фактор N, (то есть q) и затем найти и сам секретный ключ d. А также указали, как можно защититься от такого вида атаки.

**Листинг**

import subprocess  
  
class Cryptor:  
 def \_\_init\_\_(self, exe\_path):  
 self.exe\_path = exe\_path  
 self.process = None  
 self.stdin = None  
 self.stdout = None  
 self.interactions = 0  
  
 def run(self):  
 self.process = subprocess.Popen(args=self.exe\_path, stdout=subprocess.PIPE, stdin=subprocess.PIPE)  
 self.stdout = self.process.stdout  
 self.stdin = self.process.stdin  
  
 def interact(self, c):  
 self.interactions += 1  
 print(**"This is interactions "**,self.interactions)  
 line = **'{0:x}**\r\n**'**.format(c).encode()  
 self.stdin.write(line)  
 self.stdin.flush()  
 time = int(self.stdout.readline())  
 message = int(self.stdout.readline().strip(), 16)  
 return message, time  
  
 def close(self):  
 if self.process:  
 self.process.kill()

from math import log  
from time import time  
from random import randint  
from crpt import Cryptor  
  
*# атака Брамли−Боне*def Bramli\_Bone\_attack(g, l, s, R, j, delta, cryptor, n):  
 print(**'Подбираю {}-ый бит'**.format(j + 1))  
 t\_1\_list = []  
 t\_2\_list = []  
 g\_ = g | (1 << (511 - j)) *#(1) ставим бит 1 на позицию(управляется циклом) и пропускаем побитовой дизъюнк,чтобы получить новый g'* for i in range(0, l): *#(2)* u\_g = ((g + i) \* pow(R, -1, n)) % n *# преобразование в форму монтгомери с ближайшими соседями* u\_g\_ = ((g\_ + i) \* pow(R, -1, n)) % n  
 t\_1 = t\_2 = 0  
 for i in range(0, s):  
 message, time = cryptor.interact(u\_g)  
 t\_1 += time  
 message, time = cryptor.interact(u\_g\_)  
 t\_2 += time  
 t\_1\_list.append(t\_1 / s) *# mean from 5.2, среднеарифметическое* t\_2\_list.append(t\_2 / s)  
 T\_g = sum(t\_1\_list)  
 T\_g\_ = sum(t\_2\_list)  
 d = abs(T\_g - T\_g\_)  
 print(**'delta ='**, d, end=**', '**)  
 if d < delta: *# если разница маленькая , то бит i равен 1; t(g)<t(g\_) время дешифрования g оказалось меньше g\_* print(**'на {}-ый бит ставлю 1'**.format(j + 1))  
 return g\_, d  
 else: *# если дельта "большая", то бит от q это нуль; g>g\_ время дешифрования g оказалось больше g\_* print(**'на {}-ый бит ставлю 0'**.format(j + 1))  
 return g, d  
  
  
*# выполняет атаку по времени на некорректную реализацию RSA*def program1(n, e, l, s, delta):  
 cryptor = Cryptor(**'C:**\\**Users**\\**Elvin**\\**Downloads**\\**cryptor\_v4.exe'**)  
 cryptor.run()  
  
*# вычисляем начальное приближение g для множителя q = (2 \*\* (log(n, 2) // 2))  
# при помощи этого приближения вычисляем R - степень двойки > приближение* R = int(2 \*\* ((log(n, 2) // 2) + 1)) *#правая граница для g  
  
 # первые 3 бита (всего 512) необходимо подобрать так,  
 # чтобы время расширования было было минимальным.* minimum\_time = 10 \*\* 10  
 for number in range(4, 8):  
 g = number << 509 *# создаём на конце нули, а начало такое же - 3 бита -- подбор g относительно q* message, time = cryptor.interact(g)  
 if time < minimum\_time:  
 minimum\_time, minimum\_time\_g = time, g  
 g = minimum\_time\_g  
 print(**'g ='**, g)  
  
 *# далее алгоритм Брамли−Боне,  
 # подбирает остальные биты числа.  
  
 # первые три бита подобраны. далее  
 # с 3 по 511 позицию подбираем остальные* for i in range(3, 512):  
 g, delta\_ = Bramli\_Bone\_attack(g, l, s, R, i, delta, cryptor, n)  
  
 cryptor.close()  
  
 *# найти закрытый ключ RSA* q = g  
 p = n // q  
 print(**'p = {}**\n**q = {}**\n**n = {}**\n**p \* q = {}'**.format(p, q, n, p \* q))  
 return pow(e, -1, (p - 1) \* (q - 1)) *# выражаем d*n = 0xbdca61f2bdc1b76bda3b7811df28052e0501ad5aad2d206e6b8eb5e578939692d2a7f679fb194104ef79bb90c71f9b1f66c02f830b0b9afad9a527a0ae8fedfb78a178e3fad6a98c9cf95cceafda851eb24fd881c45ec3bd4429a8364f993c4ba1f69020442d00c018b72f8b56edd00dc586f5a485859b752e56cc853361457f  
e = 0x010001l = 6 *# neighborhood size*s = 1 *# sample size*delta = 500000 *# для шага 4 из дока*timer = time()  
d = program1(n, e, l, s, delta)  
print(**'d ='**, d)  
timer = time() - timer  
message = randint(10 \*\* 10, 10 \*\* 11)  
cipher = pow(message, e, n)  
decipher = pow(cipher, d, n)  
if message == decipher:  
 print(**'Атака прошла успешно, затрачено {} секунд'**.format(timer))  
else:  
 print(**'Атака прошла неуспешно'**)