SAN JOSÉ STATE
UNIVERSITY

Cryptanalysis Project Report

SmartCard RSA

Team Members: Suchita Deshmukh, Sunita Rajain

Contents

Email Id: - esuchitad@gmail.com, work.sunita@gmail.com

1. Introduction

RSA is one of the first practicable public-key cryptosystems and is widely used for secure data transmission. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret.

There are three stages of RSA

- Key Generation
- Encryption
- Decryption

KEY GENERATION

Choose large prime no. p & q

N=p*q

Choose e relatively prime to (p-1)*(q-1)

$d=e^{-1} \bmod (p-1)(q-1)$

ENCRYPTION

Public key :(N, e)

Private Key :(N, d)

Encrypt $C= M^e \bmod N$

Send the cipher text

DECRYPTION

(Simple Decryption)

Decrypt: $M=C^d \bmod N$

(Decryption – CRT)

$dp=d \bmod (p-1)$

$dq=d \bmod (q-1)$

$M1=C^{dp} \bmod p$

$M2=C^{dq} \bmod q$

$q_{inv} = q^{-1} \bmod p$

$h = (M1-M2) * q_{inv} \bmod p$

$M = M2 + h*q$

2. Problem Statement

The main goal of the project is to find the private key corresponding to the given public parameters in a RSA cryptosystem. Then decrypt the given cipher text to obtain the plaintext message.

To make the decryption process faster and easier, the large value‘d’ is broken into two smaller values as given below:

$dp = d \bmod (p-1)$

Email Id: - esuchitad@gmail.com, work.sunita@gmail.com

dq = d mod (q-1)
So instead of computing C^d, we need to compute C^{dp} and C^{dq}. Since dp and dq are much smaller than d, the time taken for these two computations will be lesser than the time taken to compute C^d.

3. RSA Smart Card Challenge

3.1 Challenge Input

n:
754897633930553141282899857600681866214922805622879200942600741532095172369703415725185597971533449465576145454201026891283601494488262771597328435528850907122893849517975920947769168699513594157953875730587323943476340412264776041510220894422630321715512241190592912467594811186260668310257307289597639739 39

e:
469338399365132038068145347854304403991210600419615635342389212892232158868331557489233299427681847931270303573352344887529519989519523007529616547910449217149861920712860366622338225861165737738348959555817678858854614025782811877120692451547034409961678170022156066582605740293538370972729222479829581201 99

cipher:
173031155883857832318555429145944360337062106115921429941481958471613803696941488982943213166198891614656467413426229517477920203919193963771666003900426919164603320386632028416160054267446603995379157422243007576103500689520821974573699856595568244001341478833431818842913777635823879570674041452802213337 02

3.2 Challenge Output

p:
701821858668278061740899294204909450739854598857375537671181400567219714881432010521555081577661748223362975536471433064812791144077520510104268529038222 7
q:
107562570844257757170939019104030341887090690249505236819400825633065209822620177698655669491618574115710315614196027838827367341947190105805551418434810 57

dp: 593411
dq: 597111

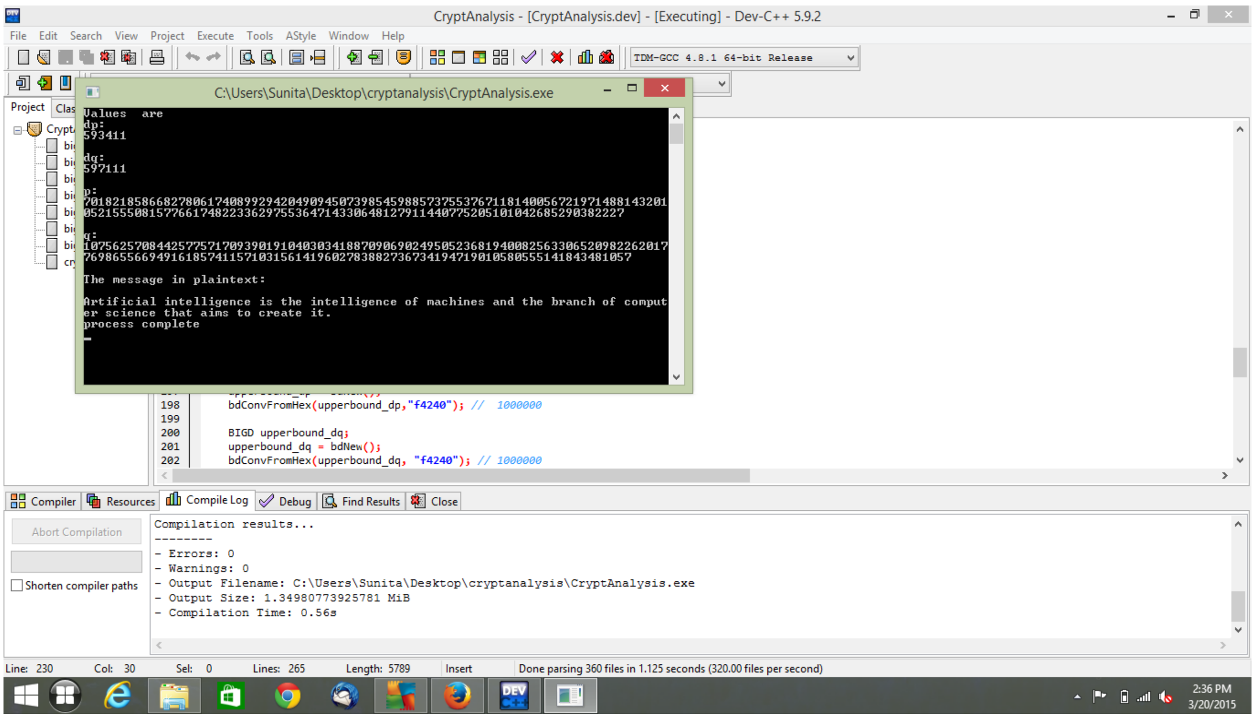
Message:
885128680775033329921124738660371610209821294886834044479337855278387763234754532897721164631891727154835265472264502152947818487828260807612033761689775362313220126531707691992170081760506423102743350462795307416443535572503479062543849178114783483681468395916972879049220 6

Email Id: - esuchitad@gmail.com, work.sunita@gmail.com

Message in plaintext:

Artificial intelligence is the intelligence of machines and the branch of computer science that aims to create it.

3.3 Output Snapshot



3.4 Link to Hall of Fame (Smartcard RSA)

Our number is 47

[Mystery Twist Hall of Fame]

4. Solution and approaches to solution

For a given pair of values for (dp, dq), we need to check if it will fit in the RSA cryptosystem with our (N, e) values.
We used a sample message Test_Msg = 48454c4c4f

$M_p = Test\_Msg^{dp} \bmod N$
$M_q = Test\_Msg^{dq} \bmod N$

M_Mp = Test_Msg - M_p
M_Mq = Test_Msg - M_q
p = GCD(M_Mp, N)
q = GCD(M_Mq, N)
if p*q = N, we have obtained the correct values of dp and dq.

Now we can get the original message M as follows:
$M1 = Cipher^{dp} \bmod p$
$M2 = Cipher^{dq} \bmod q$
$q_{inv} = q^{-1} \bmod p$
$h = (M1-M2) * q_{inv} \bmod p$
Message, M = M2 + hq

4.1 Factors of N

Generate random primes p and q and see if p*q = N. Since N is 1024 bits long, this approach did not did not give us any result.

4.2 Random generation of dp and dq

In this approach, we randomly generated integers of length 512 bits (Since N is 1024 bits long, we assumed p and q would be atmost 512 bits long) for the values dp and dq. This pair was given as input to the Check () method to see if it gives the correct result. This approach also did not give us any result.

4.3 Iterative approach

In this approach, we fixed an upper limit for both dp and dq and iteratively tried every combination, starting from dp = 1. Our method was successful for dp = 593411 and dq=597111.
To confirm these values, we tried encrypting several sample messages. The decryption of each of the cipher text generated the corresponding sample message and hence it was verified that we got the correct values of dp and dq.

5. Some Other Attacks that did not fit in our case

5.1 Low Public Exponent

Public exponent e is huge, therefore we can't apply this attack.

5.2 Franklin-Reiter Related Message Attack

Franklin and Reiter found a clever attack when Bob sends Alice related messages using the same modulus. M1, M2 ϵ Z^*_N are two distinct messages satisfying M1 =f (M2) mod N for some publicly known polynomial f ϵ $Z_N[x]$. Since C1=M1^e mod N and M2 is root of polynomial

g1(x) =f (x) ^e – C1 ϵ Z_N[x]. Similarly, M2 is root of polynomial g2(x) =f (x) ^e – C2 ϵ Z_N[x]. The linear factor x – M2 divides both polynomial. We can use the Euclidean algorithm to compute gcd of g1 and g2. If the gcd is linear, M2 is found.
For e > 3 the attack takes time quadratic in e. As our value of e is huge, we can't apply this attack.

5.3 Partial Key Attack

If e < √N and a quarter of bits of d is known then it is possible to reconstruct whole d. This approach is also not applicable in our problem as e is very large and no bit of d is known to us.

5.4 Timing Attack

By measuring the time taken by RSA smart card to decrypt the cipher text, the decryption exponent d can be discovered. This attack is based on repeated squaring algorithm which works as shown below:-
Let d_0, d_1, d_2, d_3……d_{n-1}be binary representation of d
z=M (message)
for j= 1 to n
if(d_j== 1) then
z = mod (zM,N)
end if
x = mod (z^2, N)
next j;
return z
If d_j=1 then two operations are performed namely z = mod(z^2,N) and z=mod(zM,N) whereas if d_j =0 then only one operation is performed namely z = mod(z^2,N)
By measuring the time difference, bits of d can be exposed. This method was not feasible in our case as we need the physical device for it.

5.5 Random Faults

A random glitch is introduced in the system by power spike or some other method to generate c' instead of c where c' consists of a glitch. Then by finding gcd (M, c'-M) the factors of N can be obtained. But for this attack full knowledge of M is required.

5.6 Chosen Cipher Text Attack

In this method to decrypt a cipher text c1, we can choose a random integer $c2$ mod n, and compute $c3=c1/c2 \pmod n$; Then we need to obtain m2 and m3, corresponding to $c2$ and $c3$.
C1=m1^e (mod n) → m1^e = C1 (mod n)
C2=m2^e (mod n) → m2^e = C2 (mod n)
C3=c1/c2 (mod n) → c1 = c2c3 (mod n)
m1 =m2 m3 (mod n)
Theoretically we understand how it is working but we could not implement it.

Email Id: - esuchitad@gmail.com, work.sunita@gmail.com

All the attacks explained above either requires hardware or work on the weakness of public and private exponent length. However none of the attack was possible in our case as n and e value are very large.

6. Implementation Details

As our values of n and e were very large, it was not possible to perform mathematical operations by the general C Math library functions so we used a free library for manipulating big integers in C. This library has implementations for big integer operations like add, subtract, multiply, mod, exponent etc. We have included the following files from this library in our project:

1. bigd.c
2. bigd.h
3. bigdRand.c
4. bigdRand.h
5. bigdigits.c
6. bigdigits.h
7. bigdigitsRand.c
8. bigdigitsRand.h

The methods implemented by us are in the file CryptAnalysis.c.
These are some of the important methods in our implementation:
Check()
Test the current dp and dq values and see if they generate the correct p and q values. If they do, print the values of dp, dq, p and q. Call Decrypt() and PrintMessage() methods.
BruteForce():
Iteratively call Check()
Decrypt():
Apply the dp and dq values to the input cipher and get the original message.
PrintMessage()
Extract the message byte by byte and then convert each byte into a character. Store it in a character array and print all the characters when done.

7. References

1. Key recovery for CRT implementation of RSA
2. Twenty years of attack on RSA cryptosystem
3. http://www.cs.utsa.edu/~wagner/laws/ARSAFast.html
4. BigDigits in C
5. Chapter 6 Information Security Principles and Practices Second Edition, Mark Stamp, November 12,2013
6. http://crypto.stackexchange.com/questions/1408/cracking-plain-rsa-without-private-key

Email Id: - esuchitad@gmail.com, work.sunita@gmail.com