# NSR/AS Lab 5 – Cryptography

Kris

Part 1: RSA Algorithm

Rivest, R., A. Shamir and L. Adleman designed the RSA algorithm. It is an asymmetric cryptography algorithm and works with the use of two different keys (Public Key ***[n,e]*** and Private Key ***[n,d]***).

To describe the above more clearly, an example of asymmetric cryptography can be:

1. Bob and Alice want to communicate. Bob sends his public key to Alice and requests for some data.
2. Alice encrypts the data using Bob’s public key and her private key and sends the encrypted data back to Bob.
3. Bob receives this data and decrypts it using his private key and Alice’s public key.

Despite everyone being able to see the public keys and the encrypted message send, no one apart from Bob can decrypt and view the real data.

To create the public key:

Select two numbers ***p*** and ***q*** that are prime and relatively large. The first component of the public key is *n* that can be computed **n = p\*q**. Next, calculate **x = (p-1) \* (q-1)***.* Select an exponent **e** that needs to fulfil three criterions: (i)an integer *(1 <* ***e*** *<* **x**), (ii) not a factor of **x** and (iii) is relatively prime to **x**. **Public key** is then **[n,e]**

To create the private key:

Calculate the value of **d** such that **d\*e mod x = 1***.* Thus, the**Private key** is then **[n,d]**

The message ***m*** can be encrypted to cipher text ***c*** as **c = me mod n** and the cipher text ***c*** can be decrypted to data ***m*** as **m= cd mod n**

Manually calculating the value of ‘d’ using Extended Euclidean Algorithm:

In our lab, there was a challenge where we were only provided with ‘n’ and ‘d’ was unknown. The following steps can be taken to achieve the value of ‘d’:

Step 1:

Public key: [n,e] => [2407,57]

Private key: [n,d] => [2407,d]

Step 2:

n = 2407 => factor(n) = 29 83

thus **p = 29** and **q = 83**

Step 3:

x = (p-1)(q-1)

x = 28 \* 82

**x = 2296**

Step 4:

factor(x) = 2 2 2 7 41

factor(57) = 3 19

Since there are no common factors between x and 57, they are therefore relatively prime.

Thus **e = 57**

Step 5:

1. Select d such that d \* e mod x = 1

Using Extended Euclidean Algorithm and Inverse Modulo, we can derive:

d \* e = 1 (mod 2296)

d \* 57 = 1 (mod 2296)

d = 1/57 (mod 2296)

d = 57^-1 (mod 2296)

1. Try and find if e and n have greatest common divisor (gcd), that is if gcd(e,n) = 1

2296 = 40 x 57 + 16

57 = 3 x 16 + 9

16 = 1 x 9 + 7

9 = 1 x 7 + 2

7 = 3 x 2 + 1

2 = 2 x 1 + 0

Therefore gcd(e,n) = 1

1. 1 = 7 + 2(-3)

1 = 7 + [9 + 7(-1)](-3) **Subs: 2 = 9 + 7(-1)**

1 = 7 + 9(-3) + 7(3)

1 = 7(4) + 9(-3)

1 = [16 + 9(-1)](4) + 9(-3) **Subs: 7 = 16 + 9(-1)**

1 = 16(4) + 9(-4) + 9(-3)

1 = 16(4) + 9(-7)

1 = 16(4) + [57 + 16 (-3)](-7) **Subs: 9 = 57 + 16 (-3)**

1 = 16(4) + 57(-7) + 16 (21)

1 = 16(25) + 57(-7)

1 = [2296 + 57(-40)](25) + 57(-7) **Subs: 16 = 2296 + 57(-40)**

1 = 2296(25) + 57(-1000) + 57(-7)

1 = 2296(25) + 57(-1007)

**2296(25) mod 2296 = 0** and (-1007) could be represented by 1289 **(which is the same as 2296 – 1007)**

We can rearrange the above as: d \* e mod x = 1

(1289)(57) mod 2296 = 1

Therefore **d = 1289**.

The above can laso be used for getting the value of ‘d’ for the next encrypted message as follows:

Step 1:

Public key: [n,e] => [7663,89]

Private key: [n,d] => [7663,d]

Step 2:

n = 7663 => factor(n) = 79 97

thus **p = 79** and **q = 97**

Step 3:

x = (p-1)(q-1)

x = 78 \* 96

**x = 7448**

Step 4:

**e = 57**

Step 5:

1. Select d such that d \* e mod x = 1

Using Extended Euclidean Algorithm and Inverse Modulo, we can derive:

d \* e = 1 (mod 7663)

d \* 89 = 1 (mod 7663)

d = 1/89 (mod 7663)

d = 89-1 (mod 7663)

1. Try and find if e and n have greatest common divisor (gcd), that is if gcd(e,n) = 1

7663 = 89(84) + 12

89 = 12(7) + 5

12 = 5(2) + 2

5 = 2(2) + 1

Therefore gcd(e,n) = 1

1. 1 = 5 + 2(-2)

1 = 5 + [12 + 5(-2)](-2) **Subs: 2 = 12 + 5(-2)**

1 = 5 + 12(-2) + 5(4)

1 = 5(5) + 12(-2)

1 = [89 + 12(-7)](5) + 12(-2) **Subs: 5 = 89 + 12(-7)**

1 = 89(5) + 12(-35) + 12(-2)

1 = 89(5) + 12(-37)

1 = 89(5) + [7488 + 89 (-84)](-37) **Subs: 12 = 7488 + 89 (-84)**

1 = 89(5) + 7488 (-37) + 89 (3108)

1 = 89(3113) + 7488 (-37)

**7448(-37) mod 7448 = 0**

We can rearrange the above as: d \* e mod x = 1

(3113)(89) mod 7448 = 1

Therefore **d = 3113**.

Part 2: decipher(n,e,c) Function

function [m] = decipher(n,e,c)

% Given 'n', 'e' and 'c', calculate value of 'd' then decipher to get the text message

% Author: Komal Krishneil Sharma

% Code calculates d from d\*e mod n = 1 then,

% uses value of 'd' to get: m = c\*d mod n

%declare an array and assign the alphabets to it

alphabets = ['ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz:(). '];

%get the factorials of 'n' and compute value of 'x'

nFactorial = factor(n);

x = (nFactorial(1)-1)\*(nFactorial(2)-1);

%using 'e' and 'x', perform Brute Force to get value of 'd'

for counter = 1:n

if (mod(counter\*e,x) == 1)

d = counter;

end

end

%an array to store our numeric cipher text

positionOfCharacter = [1:length(c)];

%using the index value from the above for loop, the next part

%calculates m = c^d mod n and fills the 'm' values in the

%positionOfCharacter array

for counter = 1:length(c)

s = 1; t = c(counter); u = d;

while (u > 0)

if (mod(floor(u),2) == 1)

s = mod(floor(s)\*floor(t), n);

end

u = floor(u/2);

t = floor(mod(t\*t, n));

end

positionOfCharacter(counter) = s;

end

% Convert positionOfCharacter to alphabets that can be read as a

% message 'm'

for i = 1:length(positionOfCharacter)

if positionOfCharacter(i) > length(alphabets)

decipherMessage(i) = '.';

else

decipherMessage(i) = alphabets(positionOfCharacter(i));

end

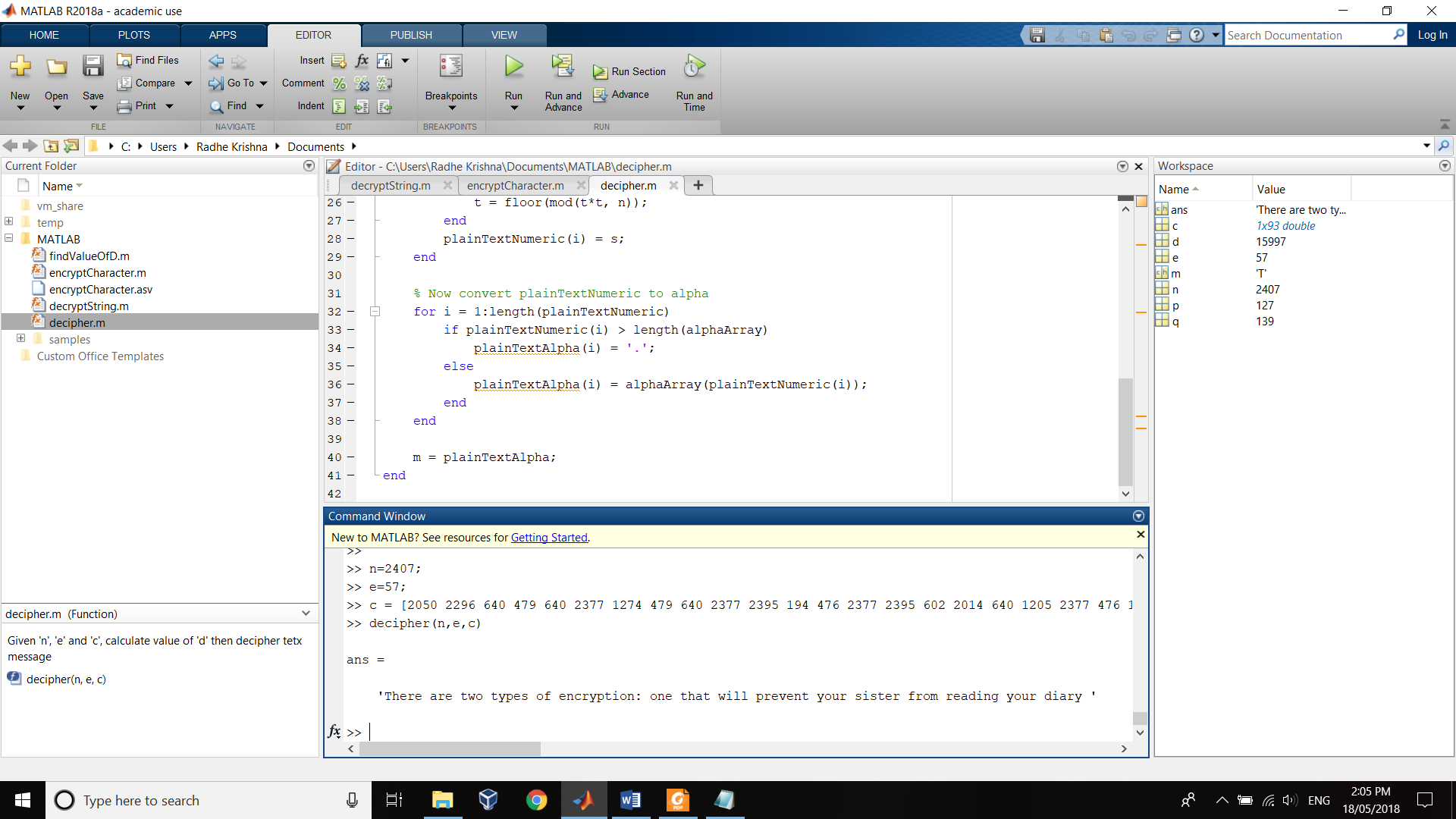
end

%assigning the decipherMessage to the function return value 'm'

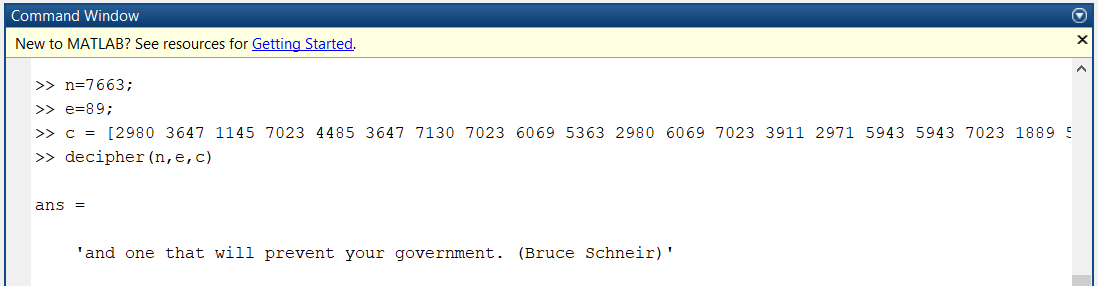
m = decipherMessage;

end

Part 3: First Decrypted Message



Part 4: Second Decrypted Message



Part 5: encryptCharacter(n,e,m) Function

function [c] = encryptCharacter(n, e, m)

% Given public key (n, e), encrypt message 'm' into cipher text 'c'

% Author: Komal Krishneil Sharma

% Code calculates c = m^e mod n and uses some clever maths

% to calculate it quickly and to avoid rounding errors

%declare an array and assign the alphabets to it

alphabets = ['ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz:(). '];

%nested for loop that compares each character from 'm' with the

%alphabets in the above array and assigns its index to a variable

for counter=1:length(m)

for position=1:length(alphabets)

if (m(counter)==alphabets(position))

positionOfCharacter(counter)= position;

end

end

end

%buffer where we store our numeric cipher text

cipherTextNumeric = [1:length(m)];

%using the index value from the above for loop, the next part

%calculates c = m^e mod n and fills the 'c' values in the

%plainTextNumeric array

for counter = 1:length(m)

s = 1; t = positionOfCharacter(counter); u = e;

while (u > 0)

if (mod(floor(u),2) == 1)

s = mod(floor(s)\*floor(t), n);

end

u = floor(u/2);

t = floor(mod(t\*t, n));

end

cipherTextNumeric(counter) = s;

end

%assigning the array to the function return value 'c'

c = cipherTextNumeric;

end

Screen shot of the output from encryptCharacter(n,e,m) Function

