# NSR/AS Lab 5 – Public Key Cryptography – Dylan Rodwell – 105341089

Dylan Rodwell
(105341089)
105341089@student.swin.edu.au

## Abstract

This lab session used MATLAB to break a cypher text (c) encrypted with a simple RSA public key system. By exploiting the key's small modulus size (n), it demonstrates how the algorithm can be compromised by factoring n into it's secret prime components (p, q). These components can then be used to calculate Euler's totient (x = ϕ(n)), and use it to brute force the private exponent (d), such that (d*e) mod x == 1. MATLAB was used to create a script that automates this process and decrypts the message. The routine *decryptString.m* was used to decrypt the message. It's "decryptString" function takes n, d, and c in as parameters and outputs the decrypted cypher (c) in plain text.

*-Key words: RSA, cryptography, public key*

## Introduction to Public Key Cryptography

Public key cryptography uses two virtual keys to encrypt and decrypt messages. The public key is used to encrypt a message, and only the related private key can be used to decrypt the message.

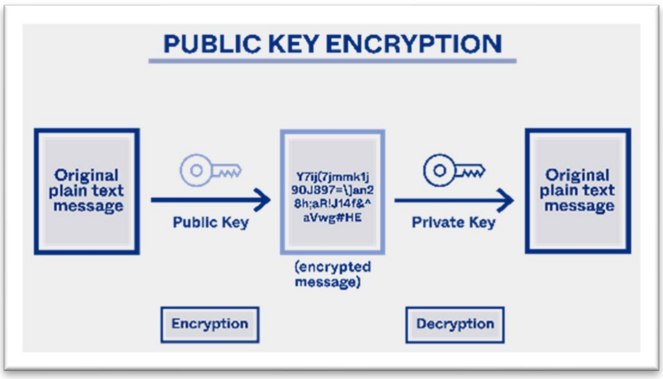

PUBLIC KEY ENCRYPTION
Original plain text message
Public Key
Y7ij(7jmmk1j 90J897=\]an2 8h;aR!J14f&^ aVwg#HE
(encrypted message)
Private Key
Original plain text message
Encryption
Decryption

*Figure 1. [1] Public Key Encryption Diagram.*

To generate the keys, the RSA algorithm is used. In RSA, two large positive prime numbers are chosen and multiplied to create the modulus. The public key consists of this modulus and an encrypted exponent. The private key also uses this modulus and a decrypted exponent. The security of the keys depends on the difficulty of factoring the modulus. If the prime numbers selected are small or poorly chosen, as demonstrated in this lab, it is easy to factor out the primes from the modulus and calculate the private key.

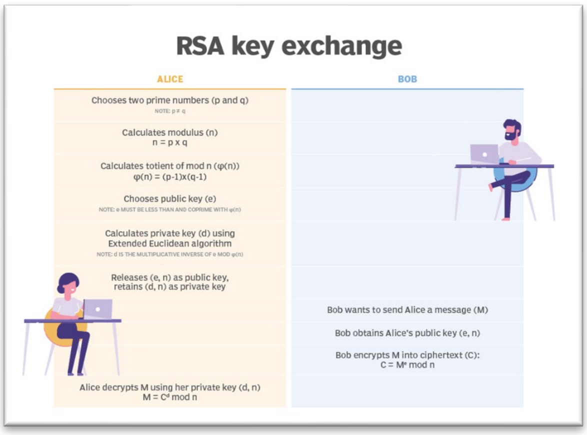
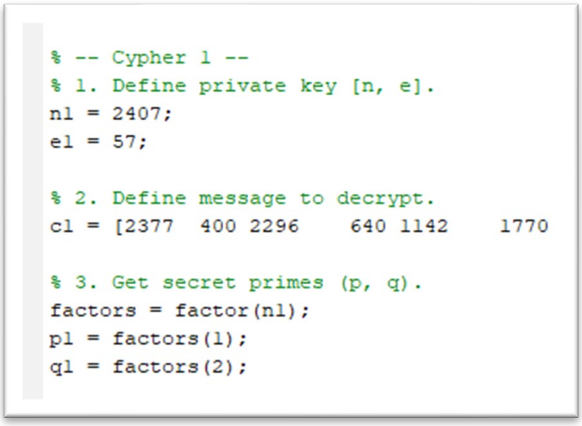

RSA key exchange
ALICE
BOB
Chooses two prime numbers (p and q)
NOTE: p ≠ q
Calculates modulus (n)
n = p x q
Calculates totient of mod n (φ(n))
φ(n) = (p-1)x(q-1)
Chooses public key (e)
NOTE: e MUST BE LESS THAN AND COPRIME WITH φ(n)
Calculates private key (d) using
Extended Euclidean algorithm
NOTE: d IS THE MULTIPLICATIVE INVERSE OF e MOD φ(n)
Releases (e, n) as public key,
retains (d, n) as private key
Bob wants to send Alice a message (M)
Bob obtains Alice's public key (e, n)
Bob encrypts M into ciphertext (C):
C = Mᵉ mod n
Alice decrypts M using her private key (d, n)
M = Cᵈ mod n

*Figure 2. [2] RSA Key Exchange diagram.*

## Breaking the RSA Algorithm

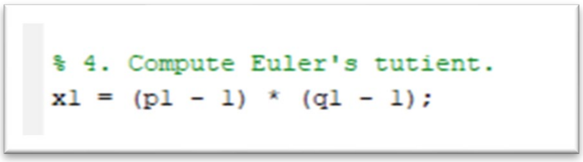
The first step to break the weak RSA key is to factor the prime numbers from the modulus "n", giving us "p" and "q".



```
% -- Cypher 1 --
% 1. Define private key [n, e].
n1 = 2407;
e1 = 57;

% 2. Define message to decrypt.
c1 = [2377   400 2296    640 1142    1770

% 3. Get secret primes (p, q).
factors = factor(n1);
p1 = factors(1);
q1 = factors(2);
```

*Figure 3. Break 1.*

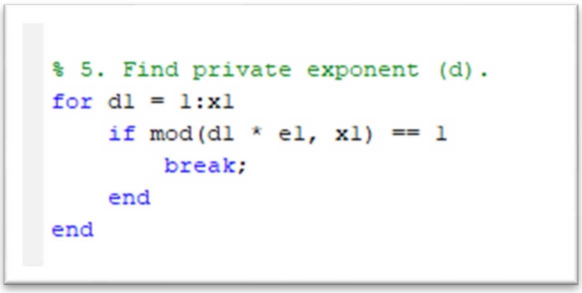
The next step is to compute Euler's totient "x = (p-1) * (q-1)".



```
% 4. Compute Euler's tutient.
x1 = (p1 - 1) * (q1 - 1);
```

*Figure 4. Break 2.*

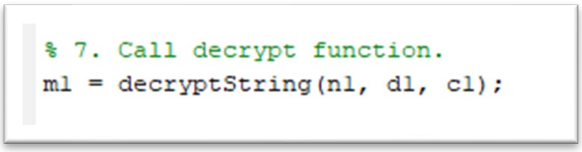
After finding "x", we can use a loop to brute force finding the decrypted exponent "d". We know that "d" is the correct value by computing the modular inverse "mod(d*e, x) == 1".



```
% 5. Find private exponent (d).
for d1 = 1:x1
    if mod(d1 * e1, x1) == 1
        break;
    end
end
```

*Figure 5. Break 3.*

Now we use the "decryptString" routine to decrypt the message "c" into plain text. The routine takes in the modulus "n", the decrypted exponent "d", and the message vector "c".

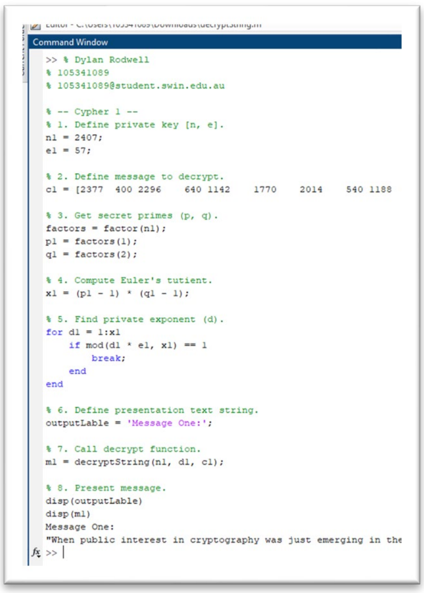


```
% 7. Call decrypt function.
m1 = decryptString(n1, d1, c1);
```

*Figure 6. Break 4.*

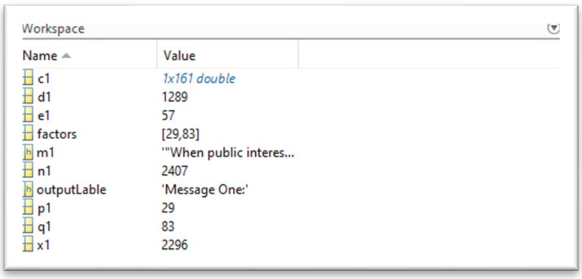
## Results

### Message 1:



```
>> % Dylan Rodwell
% 105341089
% 105341089@student.swin.edu.au

% -- Cypher 1 --
% 1. Define private key [n, e].
n1 = 2407;
e1 = 57;

% 2. Define message to decrypt.
c1 = [2377  400 2296    640 1142    1770    2014    540 1188

% 3. Get secret primes (p, q).
factors = factor(n1);
p1 = factors(1);
q1 = factors(2);

% 4. Compute Euler's tutient.
x1 = (p1 - 1) * (q1 - 1);

% 5. Find private exponent (d).
for d1 = 1:x1
    if mod(d1 * e1, x1) == 1
        break;
    end
end

% 6. Define presentation text string.
outputLable = 'Message One:';

% 7. Call decrypt function.
m1 = decryptString(n1, d1, c1);

% 8. Present message.
disp(outputLable)
disp(m1)
Message One:
"When public interest in cryptography was just emerging in the
>> |
```

*Figure 7. Cypher 1 code.*



| Name ▲ | Value |
|---|---|
| c1 | 1x161 double |
| d1 | 1289 |
| e1 | 57 |
| factors | [29,83] |
| m1 | '"When public interes... |
| n1 | 2407 |
| outputLable | 'Message One:' |
| p1 | 29 |
| q1 | 83 |
| x1 | 2296 |

*Figure 8. Cypher 1 variables.*

**Public Key 1=** [2407, 57]

**p1=** 29

**q1=** 83

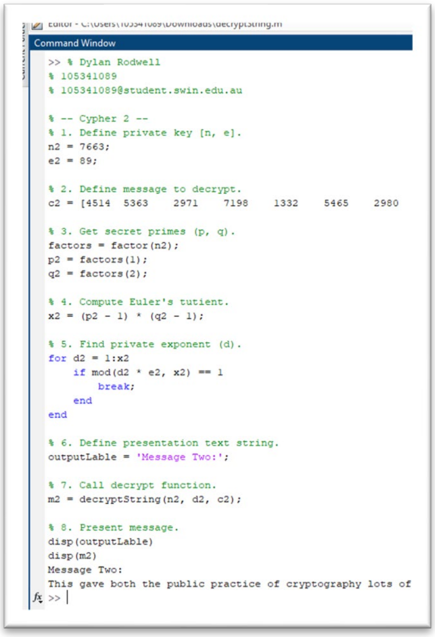
**x1= (p1 - 1)*(q1 - 1)=** 2296

**d1=** 1289

**Message1=**

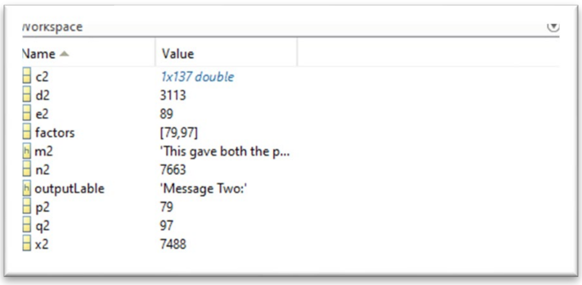
“When public interest in cryptography was just emerging in the late seventies and early eighties, the National Security Agency made several attempts to quash it.

### Message 2:



```
>> % Dylan Rodwell
% 105341089
% 105341089@student.swin.edu.au

% -- Cypher 2 --
% 1. Define private key [n, e].
n2 = 7663;
e2 = 89;

% 2. Define message to decrypt.
c2 = [4514  5363    2971    7198    1332    5465    2980

% 3. Get secret primes (p, q).
factors = factor(n2);
p2 = factors(1);
q2 = factors(2);

% 4. Compute Euler's tutient.
x2 = (p2 - 1) * (q2 - 1);

% 5. Find private exponent (d).
for d2 = 1:x2
    if mod(d2 * e2, x2) == 1
        break;
    end
end

% 6. Define presentation text string.
outputLable = 'Message Two:';

% 7. Call decrypt function.
m2 = decryptString(n2, d2, c2);

% 8. Present message.
disp(outputLable)
disp(m2)
Message Two:
This gave both the public practice of cryptography lots of
>> |
```

*Figure 9. Cypher 2 code.*



| Name ▲ | Value |
|---|---|
| c2 | 1x137 double |
| d2 | 3113 |
| e2 | 89 |
| factors | [79,97] |
| m2 | 'This gave both the p... |
| n2 | 7663 |
| outputLable | 'Message Two:' |
| p2 | 79 |
| q2 | 97 |
| x2 | 7488 |

*Figure 10. Cypher 2 variables.*

**Public Key 2=** [7663, 89]

**p2=** 79

**q2=** 97

**x2= (p2 - 1)*(q2 - 1)=** 7488

**d2=** 3113

**Message2=**

This gave both the public practice of cryptography lots of unexpected publicity." Whitfield Diffie in "Applied Cryptography,". (abridged)

## Conclusion

This lab demonstrates how easy it is to break RSA encryption when the key size is small. This is done by factoring the modulus and computing the modular inverse to find the decrypted exponent.

The decrypted message confirms that our attempts of cracking the cypher was successful and shows that for more security, longer keys should be used. This can be accomplished by simply picking much larger prime numbers to calculate the modulus.

## References

[1] “Public Key Encryption: What Is Public Cryptography? - Okta AU & NZ,” *www.okta.com*. https://www.okta.com/au/identity-101/public-key-encryption/.

[2] M. Cobb, “RSA algorithm (Rivest-Shamir-Adleman),” *TechTarget*, Nov. 2021. https://www.techtarget.com/searchsecurity/definition/RSA.