Network Security and Resilience

NSR/AS Lab 4 – Public Key Cryptography

Xuan Tuan Minh Nguyen
103819212

Abstract

This report is written in order to delves into the heart of RSA algorithm, which is one of the foundation algorithm of public-key cryptography. It will discuss the fundamental procedures that created RSA, which includes the generation process of the resistant key pairs by empowering large prime numbers. While the public key, which is always available to be used publicly, serves as the gateway for encryption. Private key on the other hand, will not be used publicly, but instead it is a critical component that helps breaking the encrypted messages. To demonstrate RSA algorithm, MATLAB is used as the main programming language to showcase the process of decrypting the encrypted messages with different public keys. The code that will be used in this report unveils the interesting mathematical behind the RSA algorithm of achieving the private key from the public key. By exploring this mathematic mechanisms, we will achieve a deeper understanding of the powerful and genious procedure that made up the RSA algorithm and its essential role in safely encapsulating data communication. Furthermore, the abstract also focuses on vulnerabilities and considerations for using RSA securely in real-word scenarios. It shows the importance of using appropriate key size and employing best practices for key generation, storage and management. Finally, the abstract underscores the connection of RSA in the digital age.

Introduction to Public Key Cryptography

1. *An Introduction*

Public Key Cryptography is the concept that is dated back into the middle of 1970s to offer preventing public data being access prohibitively, with the massive contribution from Whitfield Diffie and Martin Hellman, who invented the asymmetric key algorithms in 1976 [1]. In 1977, a group of researchers at Massachusetts Institute of Technology (Ronald **Rivest**, Adi **Shamir** and Leonard **Adleman**) has refined this idea, brought a new standardized method for public key cryptography, which is well-known nowaday as RSA (Rivest, Shamir and Adleman) algorithm [2].

2. Procedure of generating the keys

The Rivest, Shamir and Adleman (RSA) algorithm is an advanced asymmetric cryptogrpahic algorithm that contains two main components, which are the public key and private key. The public key, as by its name, used for encrypting publicly while private key is used for decryption [3]. The procedure belows illustrate the execution procedure of RSA Algorithm.

1. Pick two different prime numbers, which in this case $\boldsymbol{p}$ and $\boldsymbol{q}$.
   - Both of the prime numbers, which must be integer, must be chosen randomly and must have the same number of bits for security reasons.
2. Compute $\boldsymbol{n}$, which will be used as the modulus of both private and public keys. Moreover, $\boldsymbol{n}$ will be the determination of the length of the key.
   - $\boldsymbol{n}$ could be calculated by multiplying the randomly picked prime number $\boldsymbol{(n = p \times q)}$.
3. Obtain the $\boldsymbol{\phi(n)}$ by using **Euler's totient function** $\boldsymbol{(\phi(n) = ((p-1) \times (q-1))}$.
4. Determine the **pubic exponent** $\boldsymbol{e}$ such that $\boldsymbol{(gcd(e, \phi(n)) = 1)}$ and $\boldsymbol{(1 < e < \phi(n))}$, which e and $\phi(n)$ must be **co-primitive** (have no common factors except 1).
   - In most case, $\boldsymbol{e}$ is commonly used with small bit length ; i.e. $(\boldsymbol{2^{16} + 1})$. However, with the use of much smaller value than mentioned, such as $(\boldsymbol{2^2 + 1})$, could expose the vulnerability.
5. Choose the **private exponent** $\boldsymbol{d}$ that satisfies $\boldsymbol{d \times e (mod(\phi(n)) = 1}$; i.e. d is the multiplicative inverse of $\boldsymbol{e(mod(\phi(n))}$.
   - The **private exponent** $\boldsymbol{d}$ is usally determined using the extended Euclidean algorithm.

6. While the pair $(\boldsymbol{n}, \boldsymbol{e})$ that is calculated in step 2 and step 4 will be used as the **public key** encryption. The pair $(\boldsymbol{n}, \boldsymbol{d})$**,** computed in step 2 and step 5, will be the **private key** used for decrypting.

3. Encryption procedure

The cipher $\boldsymbol{C}$, that a message $\boldsymbol{M}$ will be encrypted to using the public key $(\boldsymbol{n}, \boldsymbol{e})$ described on the step 6 could be described in the given formula.

$$\boldsymbol{C = M^e mod(n)}$$

4. Decryption procedure

A message $\boldsymbol{M}$, which is decrypted from cipher $\boldsymbol{C}$ using the private key $(\boldsymbol{n}, \boldsymbol{d})$ illustrated on the step 6 could be described in the given formula.

$$\boldsymbol{M = C^d mod(n)}$$

Breaking the RSA Algorithm

In this section, we will be going to breakdown the RSA Code using MATLAB as the main programming language. We are handed two ciphers, $\boldsymbol{c}$ and $\boldsymbol{c_2}$, along with two pairs of private keys $(\boldsymbol{n}, \boldsymbol{e})$ and $(\boldsymbol{n_2}, \boldsymbol{e_2})$ described in the *Figure 1* and *Figure 2*. Our mission is to decrypt the message behind the two given ciphers. In Figure 1, the first two variables, $\boldsymbol{n}$ and $\boldsymbol{e}$, are the private key of the cipher $\boldsymbol{c}$, while the matrix $\boldsymbol{c}$ is the cipher $\boldsymbol{c}$. All the given variables will be passed into the *decryptCipher()* function in line 18 to process the decrypting process and retrieve the decrypted message. *Figure 2* illustrates the decryption process for cipher c_2, which has the same initialization procedure as given in *Figure 1*. The major difference here is the private key $(\boldsymbol{n_2}, \boldsymbol{e_2})$ that is distinct for cipher $\boldsymbol{c_2}$. It then also passes into the *decryptCipher()* function for further processing.


```
% 1. Definition of 'n' and 'e' in relation to first text.

n = 2407; % In this case, 'n' will be act as the product of two large positive prime numbers p and q.

e = 57; % In this case, 'e' will be act as the part of public key [n, e] that is used in RSA encryption algorithm.

% 2. First message to decrypt.

% Array 'c' contains encrypted integers representing blocks of encrypted text.
c = [2050 2296 640 479 640 2377 1274 479 640 2377 2395 194 476 2377 2395 ...
    602 2014 640 1205 2377 476 1888 2377 640 1142 1421 479 602 2014 2395 ...
    586 476 1142 749 2377 476 1142 640 2377 2395 2296 1274 2395 2377 194 ...
    586 1285 1285 2377 2014 479 640 1904 640 1142 2395 2377 602 476 540 479 ...
    2377 1205 586 1205 2395 640 479 2377 1888 479 476 2011 2377 479 640 1274 ...
    1741 586 1142 1019 2377 602 476 540 479 2377 1741 586 1274 479 602 2377];

% 3. Call to decryptCipher function
z = decryptCipher(n, e, c);
textstring = "Message one:"
% 4. Displays function outputs and presentation messages.

disp (textstring);  % Displays the 'Message one:'.

disp (z);  % Displays the decrypted message, expect output:
           % Message one:
           % There are two types of encryption: one that will prevent your sister from reading your diary
```

Figure 1: Cipher 1 code in MATLAB

```
% 1. Definition of 'n' and 'e' in relation to second text

n2 = 7663; % In this case, 'n' will be act as the product of two large positive prime numbers p and q.

e2 = 89;  % In this case, 'e' will be act as the part of public key [n, e] that is used in RSA encryption algorithm.


% 2. Second message to decrypt

% Array 'c2' contains encrypted integers representing blocks of encrypted text.
c2 = [2980 3647 1145 7023 4485 3647 7130 7023 6069 5363 2980 6069 7023 3911 2971 ...
      5943 5943 7023 1889 5561 7130 454 7130 3647 6069 7023 3243 4485 2957 5561 ...
      7023 5465 4485 454 7130 5561 3647 1883 7130 3647 6069 656 7023 6689 2206 ...
      5561 2957 4580 7130 7023 6238 4580 5363 3647 7130 2971 5561 1603];


% 3. Calls for decryptCipher function
z2 = decryptCipher(n2, e2, c2);

% 4. Displays function outputs and presentation messages
textstring2 = "Message two:";
disp (textstring2);  % Displays the 'Message two:'.

disp (z2); % Displays the decrypted message, expect output:
           % Message two:
           % and one that will prevent your government. (Bruce Schneir)
```

Figure 2: Cipher 2 code in MATLAB

In *Figure 3*, the *decryptCipher()* function is made in order to avoid creating the same decrypting procedure for multiple ciphers and increasing the re-usability. It accepts three parameters $(\boldsymbol{n}, \boldsymbol{e}, \boldsymbol{c})$**,** where the parameters $(\boldsymbol{n}, \boldsymbol{e})$ represents the private key factor and $\boldsymbol{c}$ is the cipher that need to be decrypted. In the line number 2, $\boldsymbol{n}$ is being factorized, and stored into two variables $\boldsymbol{p}$ and $\boldsymbol{q}$ in line 3 and 4 respectively, which proofs the fact that $\boldsymbol{n}$ is a result from the multiplication of $\boldsymbol{p}$ and $\boldsymbol{q}$. Line 6 obtains the $\boldsymbol{\phi(n)}$ by using Euler's totient function, which is a major step to generate the key as described in the section *Procedure of generating the keys*. The next line initiates a loop that iterates **[1,** $\boldsymbol{\phi(n)}$**],** searching for the multiplicative inverse of $\boldsymbol{e}$, such that $\boldsymbol{d} = (\boldsymbol{gcd}(\boldsymbol{e}, \boldsymbol{\phi(n)}) = \boldsymbol{1})$**.** Once the value is satisfied with the condition, the value of $\boldsymbol{d}$ will be assigned to the current value of $\boldsymbol{i}$, and the loop will be terminated. Finally, the function returns the *decryptMsg* value that contains the decrypted string that employed from the written fucntion *decryptString()*.

```
function decryptMsg = decryptCipher(n, e, c)
    g = factor(n); % Factorize the given 'n' and store the prime number in array 'g'.
    p = g(1); % Extract the first largest positive prime number.
    q = g(2); % Extract the second largest positive prime number.

    phiOfN = ((p - 1) * (q - 1)); % Calculate the Euler's totient function phi(n).
    for i = 1:n % Iterate to 'n' to find 'd'.
        if mod((i * e), phiOfN) == 1 % Condition to find the multiplicative inverse:
                                     % (d*e) mod (phi(n)) equals 1 or not.
            d = i; % Assigns 'i' to 'd' once modular inverse is found.
        end
    end
    decryptMsg = decryptString(n, d, c) % Calls for decryptstring function and return the value.
end
```

Figure 3: decryptCipher() function

Results

This section focuses on inspecting the results of code execution. Upon executing the code for the first cipher, which is illustrated in *Figure 1*, a message containing 93 characters is shown, as illustrated in *Figure 4*. *Figure 5* depicts the workspace of the corresponding executed cipher, where the given cipher $\boldsymbol{c}$ is shown as a 93 elements vector. With the public key $\boldsymbol{n}$ of 2407, the two obtained factors that stored respectively in $\boldsymbol{p}$ and $\boldsymbol{q}$ are 29 and 83, which are two different prime factors that satisfies $(\boldsymbol{n = p \times q})$**.** These variables are then used for calculates $\boldsymbol{\phi(n)}$, which is an essential component for deriving the value of $\boldsymbol{d}$. The value of $\boldsymbol{i}$ indicates that the condition is met after looping 1289 times, thus assigning the value of $\boldsymbol{d}$ that will be used as the private key for decrypting process.

```
Message one:
There are two types of encryption: one that will prevent your sister from reading your diary
```

Figure 4: Output of cipher $\boldsymbol{c}$

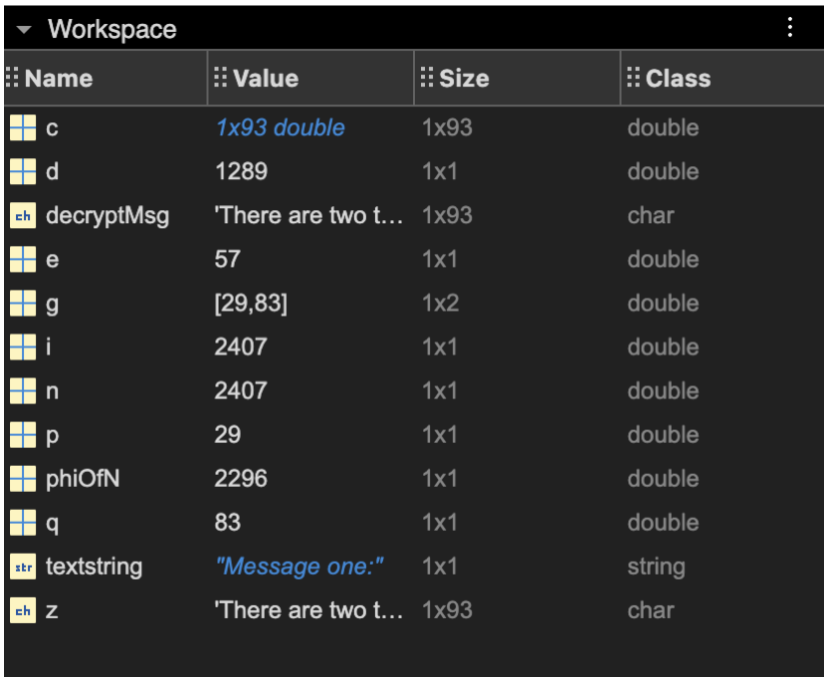
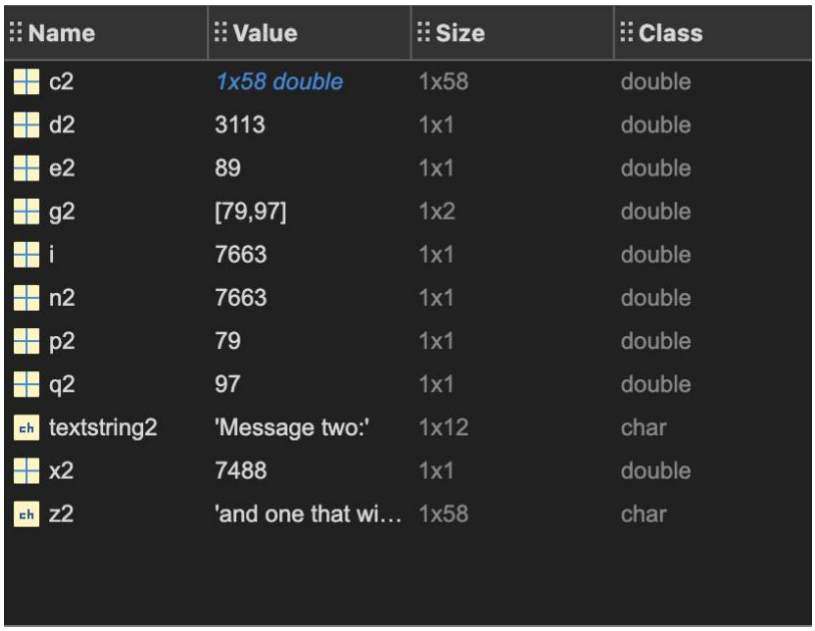


Figure 5: Workspace of cipher $\boldsymbol{c}$

The outcome for cipher $\boldsymbol{c_2}$ is illustrated in *Figure 6*, resulting in a different and shorter value compared to cipher $\boldsymbol{c}$. In *Figure 7*, it is clear that the length of cipher $\boldsymbol{c_2}$ only contains 58 elements. Although using the same function *decryptCipher()*, cipher $\boldsymbol{c_2}$ prompts different values to all variables within the function, as described in *Figure 7*. With the new public key in cipher $\boldsymbol{c_2}$, the value of private key is also changed, with the variable $\boldsymbol{d}$ set at 3113.

```
Message two:
and one that will prevent your government. (Bruce Schneir)
```

Figure 6: Output of cipher $\boldsymbol{c_2}$



Name	Value	Size	Class
c2	*1x58 double*	1x58	double
d2	3113	1x1	double
e2	89	1x1	double
g2	[79,97]	1x2	double
i	7663	1x1	double
n2	7663	1x1	double
p2	79	1x1	double
q2	97	1x1	double
textstring2	'Message two:'	1x12	char
x2	7488	1x1	double
z2	'and one that wi…	1x58	char

Figure 7: Workspace of cipher $\boldsymbol{c_2}$

Conclusion

To conclude, this report is written to provide an extensive overview of the procedure that made up the RSA encryption algorithm. With the use of MATLAB, this report has shown the successful of executing the encryption and decryption processes for two different ciphers. The received message content from both ciphers $\boldsymbol{c}$ and $\boldsymbol{c_2}$ has illustrated the efficacy and reliability of the RSA algorithm in the context of transmitting data securely. By utilizing the magic of prime factorization and modular arithmetic, the encryption and decryption functions could accurately processed the ciphers, safeguarding the confidentiality and integrity of the information. Through this implementation, it underscores the applicability of RSA encryption in real-world scenarios, marking its status as a strong foundation of nowadays cryptography.

References

[1] J. Schneider, “A brief history of cryptography: Sending secret messages throughout time,” *IBM Blog*, Jan. 05, 2024. https://www.ibm.com/blog/cryptography-history/

[2] “Understanding the RSA algorithm,” *ar5iv* (accessed May 20, 2024). https://ar5iv.labs.arxiv.org/html/2308.02785

[3] S. Wickramasinghe, “RSA Algorithm in Cryptography: Rivest Shamir Adleman Explained,” *Splunk-Blogs*, May 15, 2023. https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html

Diagrams

```
function m = decryptString(n, d, c)
% given private key (n, d), decrypt cypher text c into message m
% Author: Philip Branch
% Code calculates m = mod(c^d, n) . Uses some clever maths
% to calculate it quickly and to avoid rounding errors

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

plainTextNumeric = [1:length(c)];  %buffer where we store our numeric plain text
for i = 1:length(c)
    s = 1; t = c(i); 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;
    plainTextNumeric(i) = s;
end

% Now convert plainTextNumeric to alpha
for i = 1:length(plainTextNumeric)
    if plainTextNumeric(i) > length(alphaArray)
        plainTextAlpha(i) = '.';
    else
       plainTextAlpha(i) = alphaArray(plainTextNumeric(i));
    end
end
m = plainTextAlpha;
end
```

Figure 8: decryptString() function (Written by Philip Branch)

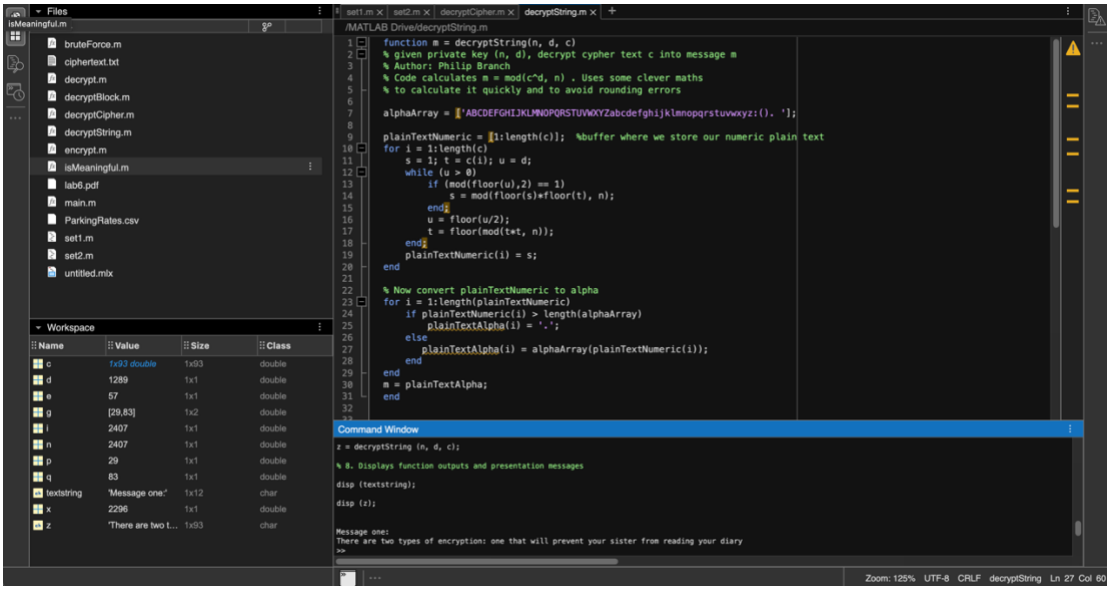


***Figure 9: Execution of cipher* c**

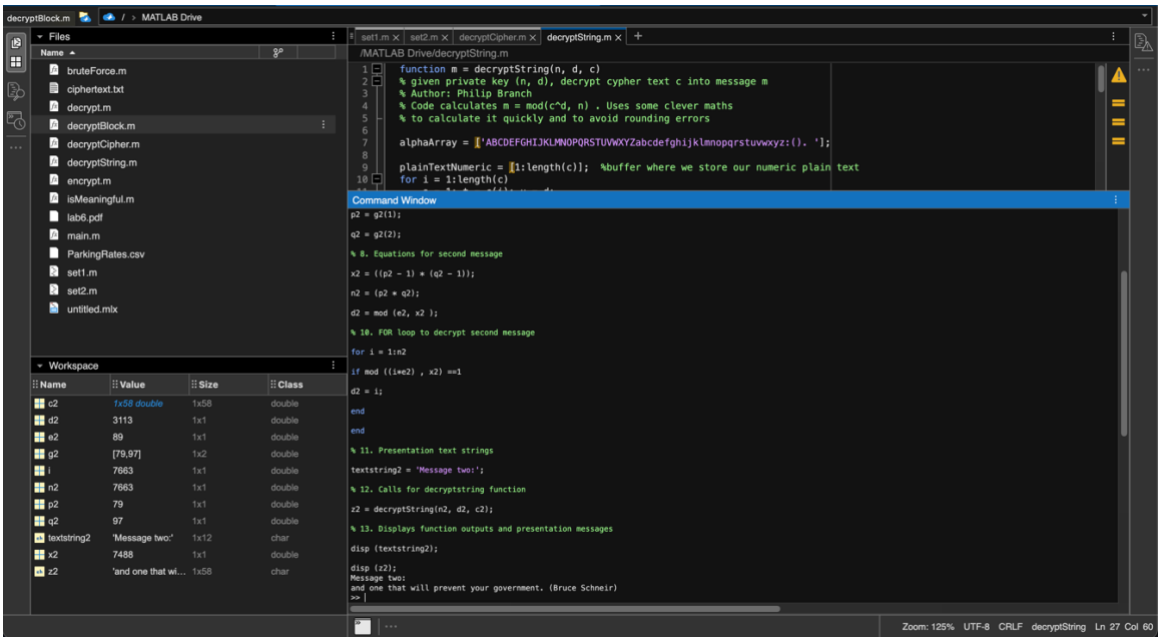


Figure 10: Execution of cipher c_2