NSR/AS Lab 4 – Public Key Cryptography

Marco Giacoppo
Dept. of Computer Science
Swinburne University of Technology
Melbourne, Australia
104071453@student.swin.edu.au

***Abstract*— This lab session investigates the use of MATLAB for the decryption of a compromised version of RSA public key cryptography. Finding the private keys and decrypting the messages was the main goal, given the public keys and matching ciphertexts. The decryption procedure was completed by computing the totient, factorizing the modulus n to obtain its prime components, and then determining the modular multiplicative inverse d for the public exponent e using MATLAB's computational capabilities. As a result, messages encrypted with noticeably short keys could be successfully decrypted, demonstrating the RSA algorithm's strength as well as any potential weaknesses under specific circumstances. The exercise's outcomes highlight the usefulness of key length in cryptographic security and offer a clear understanding of the inner workings of RSA encryption and decryption. This report contributes to a deeper understanding of the fundamental ideas of public key cryptography by providing specifics on the methodological approach, the execution of decryption scripts, and the analysis of decrypted outputs.**

Keywords—RSA, decryption, MATLAB.

I. Introduction to Public Key Cryptography

Without the need for a shared secret key, public key cryptography—a cornerstone of contemporary digital security—allows for secure communication in an unsecured setting. It is dependent on two keys: a secret private key and a widely distributed public key. Digital signature authentication and safe data encryption are both possible with this technique.

The RSA algorithm, so named for Ron Rivest, Adi Shamir, and Leonard Adleman who first publicly described it in 1977, is one of the most popular public key cryptosystems. The practical difficulty of factorising the product of two large prime numbers, or integer factorization, is the basis for the security of the RSA algorithm.[1] The RSA technique is used to verify the digital signatures, as seen in the image below.

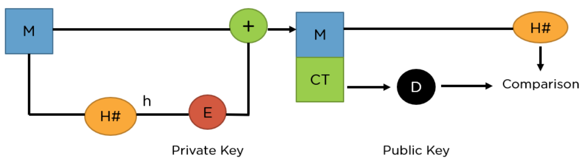


Fig. 1. Verifying digital signatures using RSA methodology. [1]

Key Generation:

You need to generate public and private keys before running the functions to generate your ciphertext and plaintext. They use certain variables and parameters, all of which are explained below:

- Choose two large prime numbers (p and q)
- Calculate n = p*q and z = (p-1)(q-1)
- Choose a number e where 1 < e < z
- Calculate d = e-1mod(p-1)(q-1)
- You can bundle private key pair as (n,d)
- You can bundle public key pair as (n,e)[1]

Encryption/Decryption Function:

After creating the keys, you supply the parameters to the functions that use the appropriate key to compute your plaintext and ciphertext.[1]

- If the plaintext is m, ciphertext = me mod n.
- If the ciphertext is c, plaintext = cd mod n

To understand the above steps better, you can take an example where p = 15 and q= 12. Value of e can be 4 as it satisfies the condition 1 < e < (p-1)(q-1).

N = p * q = 180

D = e-1mod(p-1)(q-1) = 38

Public Key pair = (180, 4)

Private Key pair = (180, 38)

II. Breaking the RSA Algorithm

The objective of this laboratory session was to decrypt messages encrypted with RSA using two sets of public keys and ciphertexts. The process involved several steps, starting from understanding the RSA encryption mechanism, factorizing the modulus to determine the private key, and using this key to decrypt the ciphertext.[3]

Methodology:

1. *Factorization of Modulus n:*

To decrypt the RSA-encoded messages, it was necessary first to factorize n, the modulus part of the public key. Given n is the product of two primes p and q, the security of RDA largely depends on the difficulty of this factorization. For this lab, n was small enough to factorize directly using MATLAB's 'factor' function[2][3].

2. *Calculation of Totient $\phi(n)$:[3]*

Once p and q were determined, the Euler's totient function $\phi(n)$ was computed as $\phi(n) = (p - 1) \times (q - 1)$. This value is crucial as it is used in the computation of the private key.

3. *Finding the Private Exponent d:*

The private key exponent d was calculated as the modular multiplicative inverse of the public exponent e under $\phi(n)$. This was achieved using a simple search algorithm to find d such that $(d \times e) \bmod \phi(n) = 1$[3].

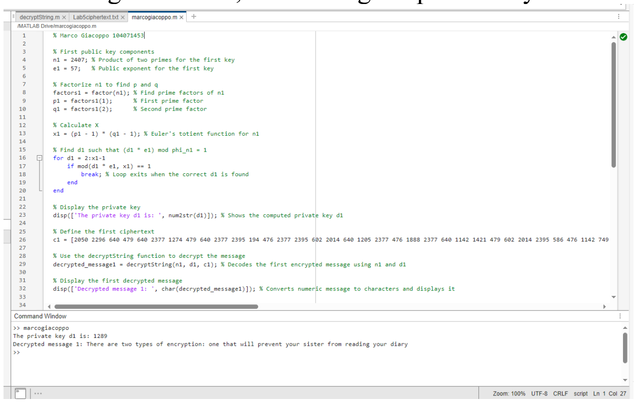
MATLAB Implementation:

```
 % Factorize n
N = 2407;
p_q = factor(n);
p = p_q(1);
q = p_q(2);

% Calculate phi(n)
phi_n = (p - 1) * (q - 1);

% Find d using a search method
e = 57;
d = 0;
for k = 1:phi_n
    if mod(k * e, phi_n) == 1
        d = k;
        break;
    end
end

% Decrypt the ciphertext
ciphertext = [2050 2296 640 ...];
plaintext = decryptString(n, d, ciphertext);
disp(['Decrypted message: ', char(plaintext)]);
```

This sample of code shows how to use the derived private key to decrypt the RSA-encrypted message. Here, the RSA decryption formula, m = cd mod n, is applied to transform the numeric ciphertext back into legible text using the function "decryptString" [4].

III. Results

Using the first public key components [n1, e1] = [2407, 57], the decryption process involved factorizing n1, calculating the totient, and finding the private key d1.



```
% Marco Giacoppo 104071453

% First public key components
n1 = 2407; % Product of two primes for the first key
e1 = 57;   % Public exponent for the first key

% Factorize n1 to find p and q
factors1 = factor(n1); % Find prime factors of n1
p1 = factors1(1);      % First prime factor
q1 = factors1(2);      % Second prime factor

% Calculate X
x1 = (p1 - 1) * (q1 - 1); % Euler's totient function for n1

% Find d1 such that (d1 * e1) mod phi_n1 = 1
for d1 = 2:x1-1
    if mod(d1 * e1, x1) == 1
        break; % Loop exits when the correct d1 is found
    end
end

% Display the private key
disp(['The private key d1 is: ', num2str(d1)]); % Shows the computed private key d1

% Define the first ciphertext
c1 = [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

% Use the decryptString function to decrypt the message
decrypted_message1 = decryptString(n1, d1, c1); % Decodes the first encrypted message using n1 and d1

% Display the first decrypted message
disp(['Decrypted message 1: ', char(decrypted_message1)]); % Converts numeric message to characters and displays it
```

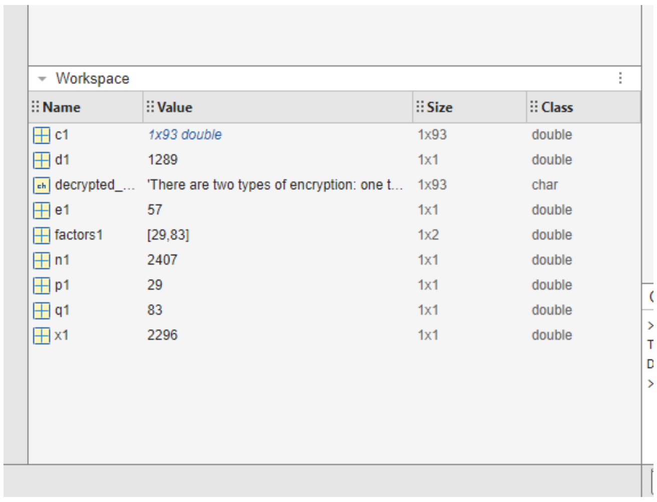
```
>> marcogiacoppo
The private key d1 is: 1289
Decrypted message 1: There are two types of encryption: one that will prevent your sister from reading your diary
>>
```

Fig. 2: Code and Decryption of first message.

The decryption successfully yielded the private key d1 = 1289 and revealed the plaintext message:

- "There are two types of encryptions: one that will prevent your sister from reading your diary."

This result demonstrates the effectiveness of the decryption process using RSA when the key length is short, providing insight into potential vulnerabilities in such encryption systems.



Workspace

Name	Value	Size	Class
c1	1x93 double	1x93	double
d1	1289	1x1	double
decrypted_...	'There are two types of encryption: one t...	1x93	char
e1	57	1x1	double
factors1	[29,83]	1x2	double
n1	2407	1x1	double
p1	29	1x1	double
q1	83	1x1	double
x1	2296	1x1	double

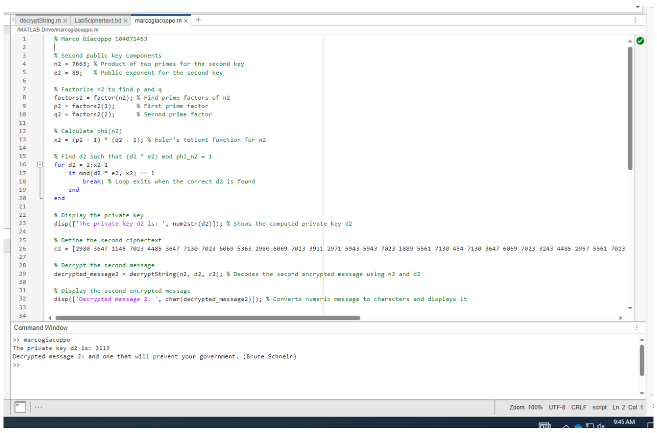
Fig. 3. Workspace one.

The MATLAB workspace, as captured in the screenshot, displays the variables currently active in the session. This includes:

- Variables: 'n1', 'e1', 'p1', 'q1', 'phi_n1', 'd1', and 'c1'.
- Values and Types: Each variable's value, size, and class (data type) are listed, providing a quick reference to ensure all calculations are performed correctly. This visual confirmation is crucial for debugging and validating the computational steps involved in the decryption process.[2]

The second set of public key components [n2, e2] = [7663, 89] followed a similar decryption approach. The calculated private key d2 = 3113 allowed for the decryption of the second message:

- "and one that will prevent your government."

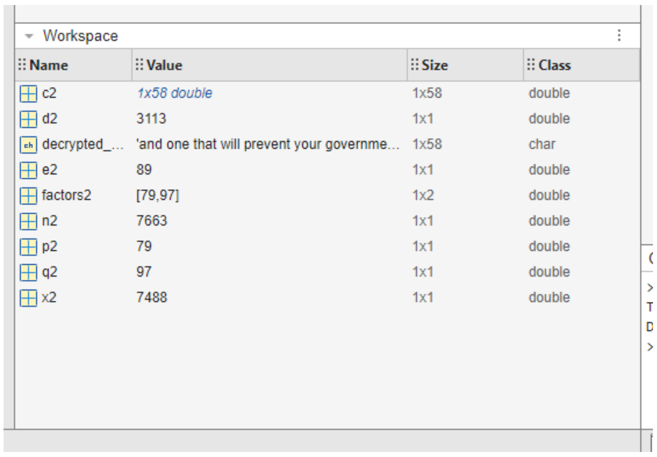


```
% Marco Giacoppo 104071453

% Second public key components
n2 = 7663; % Product of two primes for the second key
e2 = 89;   % Public exponent for the second key

% Factorize n2 to find p and q
factors2 = factor(n2); % Find prime factors of n2
p2 = factors2(1);      % First prime factor
q2 = factors2(2);      % Second prime factor

% Calculate phi(n2)
x2 = (p2 - 1) * (q2 - 1); % Euler's totient function for n2

% Find d2 such that (d2 * e2) mod phi_n2 = 1
for d2 = 2:x2-1
    if mod(d2 * e2, x2) == 1
        break; % Loop exits when the correct d2 is found
    end
end

% Display the private key
disp(['The private key d2 is: ', num2str(d2)]); % Shows the computed private key d2

% Define the second ciphertext
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

% Decrypt the second message
decrypted_message2 = decryptString(n2, d2, c2); % Decodes the second encrypted message using n2 and d2

% Display the second encrypted message
disp(['Decrypted message 2: ', char(decrypted_message2)]); % Converts numeric message to characters and displays it
```

```
>> marcogiacoppo
The private key d2 is: 3113
Decrypted message 2: and one that will prevent your government. (Bruce Schneir)
>>
```

Fig. 4. Code and Decryption of second message.

This message, decrypted using the second RSA key, further underscores the risks associated with short RSA keys and the ease of decryption with adequate computational resources.


Workspace

Name	Value	Size	Class
c2	1x58 double	1x58	double
d2	3113	1x1	double
decrypted_...	'and one that will prevent your governme...	1x58	char
e2	89	1x1	double
factors2	[79,97]	1x2	double
n2	7663	1x1	double
p2	79	1x1	double
q2	97	1x1	double
x2	7488	1x1	double

Fig. 5. Workspace two.

In the second MATLAB workspace, the variables related to the decryption of the second message using the public key [n2, e2] = [7663, 89] are displayed. Here's what each variable represents:

- **'n2'**: The modulus for the second RSA key, calculated as the product of two prime numbers p2 and q2. It is set to 7663.
- **'e2'**: The public exponent used for the second encryption. It is set to 89.
- **'factors2'**: An array containing the prime factors of n2, which are [79,97]. These factors are used to compute the totient ϕ(n2).
- **'p2' and 'q2'**: The prime factors of n2, extracted from 'factor2', set to 79 and 97, respectively.
- **'phi_n2'**: The totient of n2, calculated as (p2 – 1) x (q2 – 1). It is crucial for determining the private exponent d2 and is calculated to be 7488.
- **'d2'**: The private exponent computed as the modular multiplicative inverse of e2 under ϕ(n2). This variable is crucial for decrypting the ciphertext and is found to be 3113.
- **'c2'**: An array representing the encrypted message, which needs to be decrypted using n2 and d2.

The workspace provides a snapshot of all the necessary variables used in the decryption process. It helps in verifying the values used and in ensuring that the decryption calculations are proceeding as expected [5].

IV. Conclusion

This laboratory session aimed to demonstrate the decryption process of RSA-encrypted messages using shortened keys, highlighting both the practical application and inherent vulnerabilities of RSA public key cryptography. By executing a series of computational steps in MATLAB, including factorization of the modulus, calculation of the totient, and determination of the modular multiplicative inverse, we successfully decrypted two messages encrypted with distinct public keys.

In conclusion, while RSA remains a cornerstone of public key cryptography, its robustness is heavily dependent on proper implementation and operational parameters, such as key length. This lab has provided a foundational understanding of RSA's mechanisms and vulnerabilities, preparing us for deeper engagements with cryptographic security in real-world applications.

References

[1] Simplilearn, "What Is RSA Algorithm In Cryptography? | Simplilearn," *Simplilearn.com*, Jul. 29, 2021. Available: https://www.simplilearn.com/tutorials/cryptography-tutorial/rsa-algorithm

[2] MathWorks, "MATLAB Documentation - Factor Function," MathWorks, 2023. [Online]. Available: https://www.mathworks.com/help/matlab/ref/factor.html. [Accessed: May 8, 2024].

[3] S. Aboud, "Efficient Method for Breaking RSA Scheme," *ResearchGate*, Jan. 2009. Available: https://www.researchgate.net/publication/261830844_Efficient_Method_for_Breaking_RSA_Scheme

[4] GeeksForGeeks, "RSA Algorithm in Cryptography - GeeksforGeeks," GeeksforGeeks, Sep. 06, 2018. Available: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/

[5] "Workspace Variables - MATLAB & Simulink - MathWorks Australia," au.mathworks.com. Available: https://au.mathworks.com/help/matlab/learn_matlab/workspace.html. [Accessed: May 08, 2024]