On using Euler’s Factorization Algorithm to Factor RSA Modulus

M A Budiman¹, M Zarlis¹, O S Sitompul¹, and H Mawengkang¹

¹Graduate Program of Computer Science, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia

Email: mandrib@usu.ac.id

Abstract. The RSA public key cryptosystem was among the first algorithms to implement the Diffie-Hellman key exchange protocol. At the core of RSA’s security is the problem of factoring its modulus, a very large integer, into its prime factors. In this study, we show a step-by-step tutorial on how to factor the RSA modulus using Euler’s factorization algorithm, an algorithm that belongs to the class of exact algorithms. The Euler’s factorization algorithm is implemented in Python programming language. In this experiment, we also record the relation between the length of the RSA moduli and its factorization time. As a result, this study shows that the Euler’s factorization algorithm can be used to factor small modulus of RSA, the correlation between the factoring time and the size of RSA modulus is directly proportional, and better selection of some Euler’s parameters may lead to lower factoring time.

1. Introduction

In November 1976, Whitfield Diffie and Martin Edward Hellman [1] published a new paradigm in cryptography that solved key distribution problem. Key distribution is a problem in symmetric key cryptography that any parties that wish to communicate must exchange secret keys before the communication can take place. The secret keys must be exchanged in secure manners, and, therefore, an eyeball-to-eyeball appointment is often arranged to exchange the keys. Since the appointment is conducted non-digitally (i.e., in person), the price of exchanging the keys is expensive. Diffie and Hellman resolve this problem by making the encryption key to be public, so there is no need to exchange any secret keys, and, therefore, no eyeball-to-eyeball appointment is needed. Until now, the Diffie-Hellman protocol has been researched extensively, e.g., [2][3].

The Rivest-Shamir-Adleman (RSA) [4] was among the first algorithms to use Diffie-Hellman protocol in practice. Until now, the RSA is believed to be the most extensively used public key algorithm due to its simplicity. This algorithm relies its security on the fact that it is very hard to factor its modulus, a very big integer, into its prime factors. Some studies (e.g., [5][6][7]) have been conducted to measure how (hard it is) to factor the RSA modulus using different kinds of algorithms. In this study, we elaborate a tutorial on how the Euler’s factorization algorithm can be used to factor the RSA modulus and, as a result, may cause the RSA cryptosystem (with a small modulus) to be compromised. The Euler’s factorization algorithm is a classic integer algorithm that pertains to the class of exact algorithm. This study also records the correlation between the factoring time and the size of the RSA modulus.
2. Methods
In this section we illustrate how to do the RSA computation and how the Euler’s factorization can be used to factor the RSA modulus.

2.1 RSA
As a public key cryptosystem, RSA has three main stages, which are: key generation, encryption, and decryption. The stage of key generation is conducted by the recipient of the message. The recipient will do as follows [8] [9].
1. With a chosen prime generator, for example Fermat’s Little Theorem or Agrawal-Kayal-Saxena algorithm, two large prime numbers, \( p \) and \( q \), are generated.
2. Calculate the RSA modulus, \( n = p \times q \).
3. Calculate the Euler’s totient function, \( \Phi(n) = (p–1) \times (q–1) \).
4. Select \( e \) (the RSA encryption key) randomly, such that \( \gcd(e, \Phi(n)) = 1 \) and \( 1 < e < \Phi(n) \).
5. Calculate \( d \) (the RSA decryption key), such that \( d \equiv e^{-1} \pmod{\Phi(n)} \).
6. Keep the value of \( p, q, d, \) and \( \Phi(n) \) so that no one will ever know these values except the recipient.
7. Publish the value of \( e \) and \( n \) by electronic means, so that anyone who wants to communicate with the recipient will be able to use these values.

The stage of encryption is conducted by the sender of the message. The sender will do the following steps [8] [9]:
1. Find electronically the published value of \( e \) and \( n \) from the recipient.
2. Prepare \( m \), the message to be encrypted.
3. Encrypt the message into ciphertext using formula: \( c = m^e \mod n \).
4. Send electronically the ciphertext \( c \) to the recipient.

The stage of decryption is conducted by the recipient of the message. The recipient will do the following steps [8] [9]:
1. Receive electronically the ciphertext \( c \) from the recipient.
2. Decrypt the ciphertext into the original message using formula: \( m = c^d \mod n \).

2.2 Factoring the RSA modulus with Euler’s factorization algorithm
From subsection 2.1, one may see that the strength of RSA depends on the size of the modulus \( n \). The modulus \( n \) and the encryption key \( e \) are always made public, so that anybody including a cryptanalyst may know their values. In contrast, the values of \( p \) and \( q \) that were used to construct \( n \) is made private, so nobody including a cryptanalyst can directly know their values.

Knowing that \( n = p \times q \), the cryptanalyst may try to use Euler’s algorithm to factor \( n \). The cryptanalyst will do the following steps [10] [11]:
1. Input \( n \), the RSA modulus.
2. Choose two random integers, \( a \) and \( b \).
3. Check if there exist two pairs of integers \( (x_1, y_1) \) and \( (x_2, y_2) \) so that: \( n = ax_1^2 + by_1^2 = ax_2^2 + by_2^2 \).
4. If no such two pairs exist, then go to step 2 to choose different values of \( a \) and \( b \).
5. If such two pairs exist, then \( n = p \times q \), where \( p = gcd(x_1 \times y_2 - x_2 \times y_1, n) \) and \( q = gcd(x_1 \times y_2 + x_2 \times y_1, n) \).

Since the values of \( p \) and \( q \) have been known, the cryptanalyst may be able to recover every ciphertext into original plaintext, and, therefore, the RSA cryptosystem is compromised.
3. Results and Discussions

In this section, firstly, we will illustrate how the RSA cryptosystem works using a toy example. Secondly, using a working example, we will elaborate how a cryptanalyst may attack that RSA cryptosystem using Euler’s factorization algorithm. Thirdly, we will present a graphical relation between the size of the modulus and time to factor the modulus. The RSA cryptosystem and the Euler’s factorization algorithm are implemented in Python programming language version 2.7.17 on a Wing 7 Integrated Development Environment on a 1.4 GHz dual-core Intel Core i5 processor with 4 GB of onboard memory.

3.1 Securing message with RSA cryptosystem

Suppose a sender named Alice wants to send a simple message which is character “B” to a recipient called Bob using the RSA cryptosystem. Firstly, Bob will generate his public and private keys as follows:

1. He generates two primes using his chosen primality test: \( p = 821 \) and \( q = 661 \).
2. He computes the RSA modulus: \( n = p \times q = 821 \times 661 = 542681 \).
3. He computes the Euler’s totient function, \( \Phi(n) = (p - 1) \times (q - 1) = (821 - 1) \times (661 - 1) = 541200 \).
4. He randomly selects \( e = 509417 \) (the RSA encryption key) which satisfies \( gcd(e, \Phi(n)) = 1 \) and \( 1 < e < \Phi(n) \).
5. He computes \( d = 107753 \) which satisfies \( d \equiv e^{-1} \pmod{\Phi(n)} \).
6. He keeps the value of \( (p, q, d, \Phi(n)) = (821, 661, 107753, 541200) \) as his private key.
7. He publishes the value of \( (n, e) = (542681, 509417) \) as his public key to the internet.

Secondly, Alice will do the encryption process as follows:

1. She looks up Bob’s public key and obtains \( (n, e) = (542681, 509417) \).
2. She prepares her message “B”, finds out in the ASCII table that the value of “B” is 66, and, therefore, sets \( m = 66 \).
3. She encrypts the message into ciphertext: \( c = m^e \mod n = 66^{509417} \mod 542681 = 437007 \).
4. She sends electronically the ciphertext \( c = 437007 \) to Bob, the recipient.
5. Thirdly, Bob will do the decryption process as follows:
   1. From Alice, he gets the ciphertext \( c = 437007 \).
   2. Using his private key, he recovers the message: \( m = c^d \mod n = 437007^{107753} \mod 542681 = 66 \), which is the character “B”.

3.2 Attack on the RSA cryptosystem using Euler’s factorization algorithm

Suppose that while Alice is sending her ciphertext to Bob, a cryptanalyst called Carol intercepts the communication. From this interception, Carol knows that \( c = 437007 \). The values of \( (n, e) = (542681, 509417) \) are also known to her, since they are public anyway. Using Euler’s factorization’s algorithm, Carol will do the following steps:

1. Carol lets \( n = 542681 \).
2. Randomly, she chooses \( a = 1 \) and \( b = 4 \).
3. She checks if there exist two pairs of integers \( (x_A, y_1) \) and \( (x_B, y_2) \) so that: \( 542681 = x_A^2 + 4y_1^2 = x_B^2 + 4y_2^2 \). The potential values of \( y^2 \) run from 0 to approximately \( 542681^2 / 4 \approx 135670 \). Thus, she needs to check all the perfect squares from 0, 1, 4, 9, 16, 25, ..., 135424 as the values of \( y^2 \) is 368; it is the nearest perfect squares to 135670). By checking these potential values of \( y^2 \), she obtains that two pairs of \( x \) and \( y \), which are \( (x_1, y_1) = (541, 250) \) and \( (x_2, y_2) = (709, 100) \) satisfy the equation \( 542681 = x_1^2 + 4y_1^2 = x_2^2 + 4y_2^2 \), since \( 542681 = 541^2 + 4 \times 250^2 = 709^2 + 4 \times 100^2 \).
4. She computes \( p = gcd(x_1 \times y_2 - x_2 \times y_1, n) = gcd(541 \times 100 - 709 \times 250, 542681) = 821 \) and \( q = gcd(x_1 \times y_2 + x_2 \times y_1, n) = gcd(541 \times 100 + 709 \times 250, 542681) = 661 \). She checks that \( p \times q = 821 \times 661 = 542681 = n \).
3. Since Carol already knows the values of $p$ and $q$, she can easily compute the totient function, $\Phi(n) = (p-1) \times (q-1) = (821-1) \times (661-1) = 541200$. She computes $d \equiv e^{-1} \pmod{\Phi(n)}$ and gets $d = 107753$. Since the private key has been exposed to Carol, the RSA cryptosystem has been compromised. Therefore, Carol can decrypt Alice’s message: $m = c^d \mod{n} = 437007^{107753} \mod{542681} = 66$, which is the character “B”.

3.3 The relation between the RSA modulus and the time to factor the modulus with Euler’s factorization algorithm

In subsection 3.2, we have shown that once a cryptanalyst has factored the RSA modulus $n$ into $p$ and $q$, she can compute the private key, and the RSA cryptosystem is compromised. The remaining question is how much time the Euler’s algorithm needs to factor the RSA modulus. This question is answered in Table 1.

| N           | a | b | iterations | p     | q     | time to factor (seconds) |
|-------------|---|---|------------|-------|-------|--------------------------|
| 21          | 5 | 1 | 3          | 7     | 3     | 0.000410                 |
| 65          | 5 | 2 | 12         | 5     | 13    | 0.000535                 |
| 493         | 7 | 12| 7           | 29    | 17    | 0.000549                 |
| 14863       | 7 | 15| 6           | 167   | 89    | 0.005031                 |
| 47029       | 13| 6 | 43          | 131   | 359   | 0.078584                 |
| 216673      | 1 | 7 | 6           | 389   | 557   | 0.090570                 |
| 542681      | 1 | 4 | 4           | 821   | 661   | 0.177549                 |
| 2707553     | 13| 1 | 22          | 2411  | 1123  | 2.172594                 |
| 9156919     | 1 | 3 | 1           | 3331  | 2749  | 1.334612                 |
| 18970619    | 3 | 14| 50          | 5101  | 3719  | 34.024452                |
| 100604177   | 2 | 9 | 3139        | 9001  | 11177 | 47.656650                |
| 18970619    | 2 | 3 | 50          | 5101  | 3719  | 34.024452                |
| 512637743   | 5 | 2 | 11          | 55259 | 9277  | 277.314941               |

From Table 1, one may conclude that the time to factor the RSA modulus $n$ tends to increase as value of $n$ increases. It is hypothesized that the values of $a$ and $b$, which were generated randomly in this study, may have relations with $n$. Better choices of $a$ and $b$, may lead to the decrease of factoring time. However, the analysis of how to get “better” values of $a$ and $b$ that results in lower iterations and lower factoring time is beyond the scope of this study.
Figure 1. The relation between the RSA modulus $n$ and the time to factor with Euler’s algorithm

Figure 1 shows the graphical relation between the RSA modulus $n$ and the time to factor. It confirms that the time to factor the RSA modulus $n$ with Euler’s factorization algorithm tends to increase with bigger values of $n$.

4. Conclusions

Summing up, it has been shown that:

1. The Euler’s factorization algorithm can be used to factor small modulus of RSA. In this study, the largest value of RSA modulus being factor is $n = 512637743$ (9 digits), which is 111101000110101101101111 in binary (29 bits), which has been factored in 277.31 seconds. However, in practice, the RSA usually has a 1024-bit modulus or more, so, understandably, it may still be computationally infeasible to factor this very large RSA modulus with Euler’s factorization algorithm.

2. The correlation between the time to factor the RSA modulus using Euler’s factorization algorithm and the size of modulus $n$ has a tendency to be directly proportional.

3. The values of $a$ and $b$, the two Euler’s initial parameters, may have relations with $n$. It is hypothesized that appropriate selection of $a$ and $b$ may lower the factoring time. Therefore, studies on how to select these values appropriately are encouraged.
References

[1] Diffie W and Hellman M 1976 New directions in cryptography *IEEE Transactions on Information Theory* 22 6 pp 644-654

[2] Galbraith S, Massimo J and Paterson K G 2019 Safety in Numbers: On the Need for Robust Diffie-Hellman Parameter Validation *Public-Key Cryptography – PKC 2019 Lecture Notes in Computer Science* 379–407

[3] Liskov M D, Guttman J D, Ramsdell J D, Rowe P D and Thayer F J 2019 Enrich-by-Need Protocol Analysis for Diffie-Hellman *Foundations of Security, Protocols, and Equational Reasoning Lecture Notes in Computer Science* 135–55

[4] Rivest R L, Shamir A and Adleman L 1978 A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM* 21 2 pp 120-126

[5] Aggarwal D and Maurer U 2016 Breaking RSA Generically Is Equivalent to Factoring *IEEE Transactions on Information Theory* 62 6251–9

[6] Akchiche O and Khadir O 2016 Factoring multi-power RSA moduli with primes sharing least or most significant bits Groups *Complexity Cryptology* 8

[7] Budiman M A and Rachmawati D 2017 On factoring RSA modulus using random-restart hill-climbing algorithm and Pollard’s rho algorithm *Journal of Physics: Conference Series* 943 012057

[8] Klima R E and Sigmon N P 2019 *Cryptology: classical and modern* (Boca Raton: CRC Press, Taylor & Francis Group)

[9] Mollin R 2019 *RSA and Public-Key Cryptography* (CRC Press)

[10] Batten L M 2013 *Public key cryptography: applications and attacks* (Hoboken, NJ: Chichester)

[11] Zhang X, Li M, Jiang Y and Sun Y 2019 A Review of the Factorization Problem of Large Integers *International Conference on Artificial Intelligence and Security* pp 202-213 Springer Cham