Comparison of Okamoto-Uchiyama and RSA Cryptosystems

Kevin Sendjaja / 13517023[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13517023@std.stei.itb.ac.id

***Abstract*—The Okamoto-Uchiyama cryptosystem is one among many public-key algorithms which relies on the difficulty of integer factorization for its security, much like the better-known RSA cryptosystem. While relying on the same base principle, both algorithms use different ways in computing both the private and public keys, as well as the encryption and decryption method. As such, a comparison between both cryptosystems should lead to a better understanding regarding the properties, as well as the strong and weak points of both cryptosystems.**

***Keywords*—Okamoto-Uchiyama, RSA, public key algorithms, cryptosystem**

I. INTRODUCTION

Cryptography has been a major part in security, especially in modern times where technology and communications have greatly improved since several centuries prior. Many forms of communication over the net, as well as crucial data transfer, are vulnerable to threats from an unauthorized third party, such as wiretapping, data manipulation, and more. Which is why, security measures such as data encryption as well as digital signatures, which implements the principles of cryptography, are crucial to ensure that transferred data remains ineligible to read by unauthorized party and to prove that the data was not tampered in any kind of method.

Among the many forms of cryptography, public key cryptography was one that is widely used by the public. Compared to the usual symmetric key cryptography, it uses two different keys for encryption and decryption process, making it relatively more secure as the private key is not shared with anyone else, albeit not without its own set of drawbacks. The RSA algorithm is one of the most famous and widely used in real-life applications.

One other example of the public-key algorithms is the Okamoto-Uchiyama algorithm. While being in the same group as RSA which relies on integer factorization problem, they use different methods in generating the keys, as well as the encryption and decryption processes. While not as popular as the RSA, it can still be used for the same functionality. This paper aims to analyze the properties of the Okamoto-Uchiyama cryptosystem, while being compared to the RSA cryptosystem, measuring both outputs using similar metrics in hopes that it will bring better understanding regarding the nature of the two cryptosystems, as well as their advantages and drawbacks.

II. THEORIES

A. Cryptography

The term cryptography originated from the word "*cryptós*" and "*gráphein*" in Greek language, which could be translated as secret writing. Cryptography refers to the technique used to maintain the security of a message. The main processes in cryptography revolves around encryption, which encode a message into a ciphertext to make it difficult to read, and decryption, which decode a ciphertext into the original message so it may be read by the receiver. The term message that is used in cryptography actually have a lot of varieties, from simple text messages, to a more complex form such as images, audio files, and videos.

Cryptography has long history that could be traced back into the ancient Egyptian period. A lot of cryptography techniques have been created since then. The "classic' cryptography techniques mostly revolve around substitution and transposition methods to encrypt the message into ciphertext. The message itself was mostly limited to basic alphabets, and still uses paper to write down the message or the ciphertext.

The advent of technology brought a lot of major improvements to the cryptography techniques. The 'modern' cryptography techniques still retain the classic substitution and transposition principles, while adding new techniques such as the XOR function, since the messages are mostly processed in bit or byte forms. The processing power of modern computers also allow the encryption and decryption processes to be executed more quickly, while allowing messages of larger sizes to be processed as well. However, the modern techniques still have a lot of flaws, which motivates researchers to develop better methods to counter those flaws, developing cryptography further as a whole.

B. Public Key Cryptography

Prior to 1970s, the techniques used in cryptography were classified as the symmetric key cryptography, which used a key to encrypt a message, while the same key is later used to decrypt the ciphertext into the original message. As such, the key plays a major part in the algorithm, but at the same time, should the key ever fall into the hands of an unauthorized party, the ciphertext would be vulnerable to many kinds of attack, which might reveal the message as a whole. Because of that, ensuring

that the key is known only to the sender and receiver is crucial, which usually requires a private channel which might be costly.

The concept of public key cryptography was introduced in 1976, by researchers Whitfield Diffie and Martin E. Hellman, in the paper "*New Directions in Cryptography*". The idea revolves around using two different keys for encryption and decryption processes. One is the public key, which as the name implies, is available to the general public and can be used by anyone. The other one is the private key, which is kept secret by the owner. The encryption process would have the sender encrypt the message using the receiver's public key. The receiver would then receive the ciphertext and decrypt it into the original message using his own private key. The idea could be reversed as well, which is used as the principle for digital signatures. The sender would sign a document using his own private key before sending it. The receiver would then verify the signature using the sender's public key, to ensure that the document was not tampered in any way, while at the same time verify that the sender was indeed the one who sent the document.

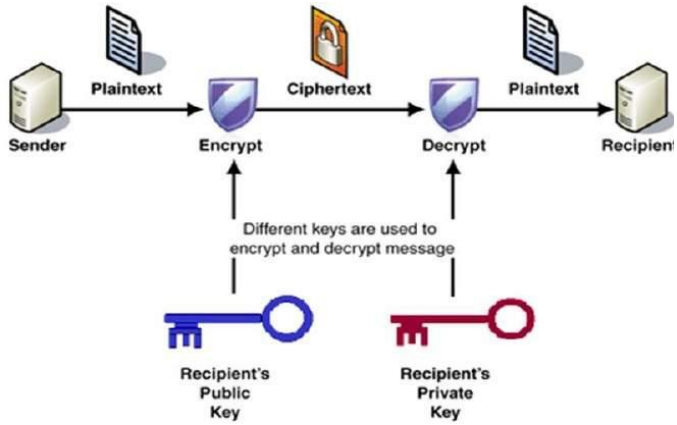

Image 1. Public key cryptography process
Source:(https://www.tutorialspoint.com/cryptography/public_key_encryption.htm)

The security aspect of the public key cryptography lies in the fact that it is difficult to derive the private key from the public key. Public key algorithms use classic calculation problems which took a tremendous amount of time and computing resources to generate the public and private keys, such as the integer factorization problem and the discrete logarithmic problem. The larger the number that is used in the calculation, the harder it is to figure out the keys without sufficient info, which is why most algorithms use large numbers to make sure it is harder to break.

The public key algorithm answers to the problem in sending keys through a private channel. Since the private keys will remain a secret from the general public, there is no reason to send any keys because anyone can access the public key. At the same time, the same keys can be used repeatedly in numerous occasions, while in symmetric key algorithm, the keys need to be replaced often as a safety measure should a key managed to be intercepted by an unauthorized party. However, the encryption and decryption processes done using public key algorithms tend to have longer execution time than symmetric key algorithms, due to the calculation of very large numbers. The messages are generally converted into numbers during the encryption process, and when calculated together with the key, would produce a large number that might be even bigger than the original message, increasing the size of the resulting ciphertext. As such, most public key algorithms are instead used to encrypt the key for the symmetric key algorithms, which the receiver would his own private key to decrypt the symmetric key which is later used to decrypt the ciphertext into the original message.

C. Rivest-Shamir-Adleman (RSA) Algorithm

The Rivest-Shamir-Adleman algorithm, also known as the RSA algorithm, was one of the most well-known public key algorithms. It was named after its founders, Ronald Rivest, Adi Shamir, and Len Adleman, and was first introduced in 1976. The algorithm is based on the prime integer factorization problem.

The method to generate the public and private keys for encryption and decryption processes are as follows:

1. Select two prime numbers p and q. It is preferred for both p and q to be quite large to make it harder to decipher.
2. Calculate $n = pq$
3. Calculate $\phi(n)$, or the totient of n. $\phi(n)$ refers to the number of integers which are less than n and are relatively prime to n. It can be calculated using the formula:
$$\phi(n) = \phi(p)\phi(q) = (\mathrm{p}-1)(\mathrm{q}-1) \quad (1)$$
4. Choose a random integer e such that $1 < e < \phi(n)$ and e must be relatively prime to $\phi(n)$
5. Calculate an integer d using the following formula:
$$ed \equiv 1(mod\ \phi(n)) \quad (2)$$
The formula could be calculated using the Euclidean algorithm, or using the following formula:
$$d = \frac{1 + k\phi(n)}{e} \quad (3)$$
where k is an integer 1, 2, 3, … which would result in d being an integer.
6. The public key is the pair (e, n), while the private key is the pair (d, n). n is called the modulus while e and d are called the exponents.

Suppose that the sender wishes to encrypt a message, then he/she must follow the following steps:

1. The message must be separated into blocks $m_1, m_2, ..., m_i$ such that $0 \leq i < n - 1$.
2. For each message block m_i, by using the receiver's public key (e, n), calculate the ciphertext block c_i using the following formula:
$$c_i = m_i^e \ mod\ n \quad (4)$$

To decrypt the ciphertext back into the original message, the receiver must follow the following steps:

1. Define the ciphertext blocks $c_1, c_2, \ldots$
2. For each ciphertext block c_i, by using the receiver's private key (d, n), calculate the message block m_i using the following formula:
$$m_i = c_i^d \ mod\ n \quad (5)$$

Generally, the private key will remain secure as long as n

remains large enough to prevent brute force attacks. The moment p and q can be figured out, then d can also be derived, allowing the attacker to be able to decrypt the ciphertext. As such, it is advised that the numbers should be large enough in size, at least 100 digits for both p and q.

D. Okamoto-Uchiyama Cryptosystem

The Okamoto-Uchiyama cryptosystem was named after its founders Tatsuaki Okamoto and Shigenori Uchiyama, and was introduced in 1998. The algorithm satisfies the property of homomorphic encryption, where it is possible to perform calculations on the encrypted data without the need to decrypt the data first. It also uses a probabilistic property using a random constant when encrypting each part of the message.

The method to generate the public and private keys follows these set of steps:

1. Select two prime numbers p and q. Both p and q must have the same length k-bits.
2. Calculate $n = p^2q$
3. Select a random generator g such that $1 < g < n - 1$, and $g^{p-1} \not\equiv 1 \ mod \ p^2$. Haryanto et al [2] further simplifies this equation and the value g is valid as long as it satisfies:
$$g^{\phi(p^2)} = g^{p(p-1)} \ mod \ p^2 = 1 \quad (6)$$
4. Calculate $h = g^n \ (mod \ n)$
5. The public key is (n, g, h), while the private key is the pair (p, q)

The encryption process will use a random constant which will lead to a probabilistic cryptosystem. The encryption steps are as follows:

1. Ensure message m satisfies $0 < m < 2^{k-1}$
2. Select random integer $r \in \mathbb{Z}/n\mathbb{Z}$
3. Generate ciphertext c using the formula:
$$c = g^m h^r \ (mod \ n) \quad (7)$$

The decryption process will follow these steps:

1. Define function L(x) as:
$$L(x) = \frac{x-1}{p} \quad (8)$$
2. From ciphertext c, derive the message m using the formula:
$$m = \frac{L(c^{p-1} \ mod \ p^2)}{L(g^{p-1} \ mod \ p^2)} \ (mod \ p) \quad (9)$$

The cryptosystem satisfies the following homomorphic encryption, that is for any two plaintexts m_0 and m_1 where $m_0 + m_1 < p$,

$$E(m_0, r_0) * E(m_1, r_1) = E(m_0 + m_1, r_2) \quad (10)$$

where $E(m, r)$ represents the encryption process of message m with random constant r. However, as Haryanto et al [2] stated, the equation is only satisfiable as long as r_2 remains dependent on both r_0 and r_1, for example, by process of addition. In the event where r_0, r_1, and r_2 are completely independent from each other, then the equation would be proven false.

III. TESTING

A. Hardware Specifications

The testing is done using the author's personal laptop. The specifications are as follows:

Operating System	Windows 10 Home Single Language
Processor	AMD Ryzen 5 4600H with Radeon Graphics CPU @ 3.00 GHz
Architecture	64-bit OS
Memory	8.00 GB

Table 1. List of hardware specifications used for the testing period

B. Algorithms

Both the RSA and the Okamoto-Uchiyama algorithms are implemented using the programming language Python, with additional functions used from the library libnum, pycryptodomex, and hashlib. The functions used in the Okamoto-Uchiyama algorithm are as follows:

```
1.  def gen_key(k):
2.      p = getPrime(k)
3.      q = getPrime(k)
4.      n = p**2 * q
5.      while True:
6.          g = getRandomRange(1, n-1)
7.          if pow(g, p * (p-1), p**2) ==
    1:
8.              break
9.      r = getRandomRange(1, n-1)
10.     h = pow(r, n, n)
11.     return (n, g, h,p,q)
12.
13. def string_to_int(string):
14.     temp = ""
15.     for i in range(len(string)):
16.         if(string[i] != ' '):
17.             rep = ord(string[i]) - 38
18.         else:
19.             rep = 99
20.         temp = temp + str(rep)
21.
22.     return int(temp)
23.
24. def int_to_string(num):
25.     numstring = str(num)
26.     string = ""
27.     arr = [numstring[i:i+2] for i in
    range(0, len(numstring), 2)]
28.     for element in arr:
```

```
        temp = int(element)
        if(temp != 99):
            string = string + chr(temp + 38)
        else:
            string = string + ' '

    return string


def encrypt(m, n, g, h):
    num = str(string_to_int(m))
    arr = [num[i:i+2] for i in range(0, len(num), 2)]
    c = []
    for element in arr:
        r = getRandomRange(1, n)
        temp = (pow(g, int(element), n) * pow(h, r, n)) % n
        c.append(temp)
    return c

def L(x, p):
    return (x-1)//p

def divide(x, y, p):
    return x*(libnum.invmod(y, p))

def decrypt(c,p, g):
    message = ""
    for element in c:
        cp = pow(int(element), p-1, p**2)
        gp = pow(g, p-1, p**2)

        x = divide(L(cp, p), L(gp, p), p) % p
        message = message + int_to_string(x)
    return message
```

Code 1. Functions for the Okamoto-Uchiyama algorithm

It is to be noted that this implementation doesn't cover all possible characters for the encryption process, as the integer representation for each character is modified so it will consist of exactly two digits. In return for allowing all letters and numbers to be encrypted, characters with ASCII decimal value below 48 will cause an error during the decryption process, with the exception of the character SPACE which is converted so it will be represented using the value 99.

The functions used in the RSA algorithm are as follows:

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def generate_keypair(k):
    p = getPrime(k)
    q = getPrime(k)
    n = p * q

    toitent = (p-1) * (q-1)
    e = getRandomRange(1, toitent)

    g = gcd(e, toitent)
    while g != 1:
        e = getRandomRange(1, toitent)
        g = gcd(e, toitent)

    d = libnum.invmod(e, toitent)

    return ((e, n), (d, n))

def encrypt(public, plaintext):
    e, n = public
    cipher = [pow(ord(char),e,n) for char in plaintext]
    return cipher

def decrypt(private, ciphertext):
    d, n = private
    plain = [chr(pow(char,d,n)) for char in ciphertext]
    return ''.join(plain)
```

Code 2. Functions for the RSA algorithm

C. Testing Parameters

The parameters that will be tested and compared are as follows:

1. Key generation time with varying key sizes
2. Ciphertext size compared to the original message size with varying message
3. Encryption time with varying key and message sizes
4. Decryption time with varying key and message sizes

IV. TESTING RESULTS AND ANALYSIS

A. Testing Results

The results of the testing processes are as follows:

1. Key generation

p and q size (bits)	Generation time (s)	
	RSA	Okamoto-Uchiyama
64	0.003997325897216797	0.002995729446411133
128	0.006988763809204 1016	0.016010046005249023
256	0.013997077941894531	0.037018775939941406
512	0.103999853134155527	0.12614917755126953
1024	0.4341142177581787	0.35590696334838867
2048	1.7667217254638672	1.9954702854156494

2. Ciphertext size

For this test, the prime integers p and q will have the size of 64 bits each. The ciphertext is in text form, so each digit is counted as 1 byte.

Plaintext size (bytes)	Ciphertext size (bytes)	
	RSA	Okamoto-Uchiyama
10	386	576
100	3839	5757
1000	38367	57775
10000	383404	572367
100000	3826852	5732172
1000000	38498965	57673554

3. Encryption time

Plaintext size (bytes)	p and q size (bits)	Encryption time (s)	
		RSA	Okamoto-Uchiyama
10	128	0.002014398574 8291016	0.003980159759 521484
10	256	0.005997419357 299805	0.016995429992 67578
10	512	0.030988454818 725586	0.103999376296 99707
100	128	0.012000322341 918945	0.034011125564 575195
100	256	0.054015874862 6709	0.162000417709 35059
100	512	0.284131050109 8633	1.062177658081 0547
1000	128	0.102071046829 22363	0.330856800079 3457
1000	256	0.509296894073 4863	1.624149560928 3447
1000	512	2.833071708679 199	10.26911187171 936

4. Decryption time

Plaintext size (bytes)	p and q size (bits)	Decryption time (s)	
		RSA	Okamoto-Uchiyama
10	128	0.001996755599 975586	0.002001047134 399414
10	256	0.006000518798 828125	0.007975101470 947266
10	512	0.034020185470 581055	0.037991523742 67578
100	128	0.012655258178 710938	0.017999887466 430664
100	256	0.061014413833 618164	0.071998357772 82715
100	512	0.343044757843 0176	0.366112232208 25195
1000	128	0.122325658798 21777	0.179152727127 0752
1000	256	0.606773853302 002	0.708589076995 8496
1000	512	3.450299739837 6465	3.742982625961 3037

B. Analysis

From the test results, it can be seen that the key generation times for both algorithms are relatively similar with only slight difference between them. Both algorithms have some similar calculations within the key generating process, although the random generator part might cause some differences due to the random nature which might need longer time to generate a number that satisfies the rule.

While both algorithms generate ciphertext with at least 10 times larger size than the original plaintext, the ciphertext size from the Okamoto-Uchiyama algorithm are generally larger than the RSA. This is most likely due to the fact that the algorithm works with $n = p^2q$ compared of RSA's $n = pq$. The resulting n would be larger in size, and when is used to generate the random constant and during the modulo function, the resulting ciphertext would be generally have larger size as well.

The encryption and decryption times are slightly different for both algorithms. In RSA's case, the decryption time is usually faster until some point where the encryption time becomes faster, although both times are relatively similar with not no significant difference. This is probably because the encryption and decryption calculations for RSA are almost identical to each

other, so it depends on how large the value of the exponents, which are e and d. In Okamoto-Uchiyama's case, the decryption time is always faster than the encryption time during the testing period. This is due to the fact that the encryption process works using exponents m and r, both are heavily influenced by the length of the message as well as the length of the primes. On the other hand, the decryption process uses only p for the exponent, which would make the calculation faster than the encryption. That being said, due to the nature of the exponent calculation, increasing the variables of message length and primes' length would cause significant rise in both encryption and decryption times once they have passed a certain value.

V. CONCLUSION

From the test results, it can be concluded that the RSA cryptosystem is generally faster than the Okamoto-Uchiyama cryptosystem, both from the key generation process, as well as the encryption and decryption processes. On the other hand, Okamoto-Uchiyama provides larger ciphertext from the encryption process, which might be preferable to prevent known ciphertext attacks. With homomorphic and probabilistic encryption properties, it could be said that the algorithm is safer compared to the RSA, especially if there are necessities to do calculations on the data without decrypting it first. But in the end, both algorithms have their own strengths and weaknesses, and it can't be said that one is better than the other in all aspects. So, it goes back to which one is more suitable depending on the prerequisites of the solution.

VI. ACKNOWLEDGMENT

The author would first express his thanks to God, for without His blessing and guidance, it wouldn't be possible to complete this paper. The author also would like to express his utmost gratitude to Dr. Ir. Rinaldi Munir, MT. as the lecturer for the course IF4020 Kriptografi, who had given knowledge related to the topic as well as guidance throughout the semester. Lastly, the author would like to thank all of the author's friends who had given moral support until this paper has been concluded.

REFERENCES

[1] Okamoto T., Uchiyama S. (1998) A new public-key cryptosystem as secure as factoring. In: Nyberg K. (eds) Advances in Cryptology — EUROCRYPT'98. EUROCRYPT 1998. Lecture Notes in Computer Science, vol 1403. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0054135.

[2] Haryanto, L., et al (2019). On the Okamoto-Uchiyama cryptosystem: (A brief essay on basic mathematics applied in cryptography). Journal of Physics: Conference Series. 1341. 042013. 10.1088/1742-6596/1341/4/042013..

[3] Munir, Rinaldi. 2020. Algoritma RSA. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Algoritma-RSA-2020.pdf (diakses tanggal 19 Desember 2020)

[4] Munir, Rinaldi. 2020. Kriptografi Kunci-Publik. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/Kriptografi-Kunci-Publik-2020.pdf (diakses tanggal 19 Desember 2020)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 21 Desember 2020

Kevin Sendjaja / 13517023