IRAQI JOURNAL OF APPLIED PHYSICS
IJAP, Vol. 10, No. 4, October-December 2014, pp. 27-29

Alia K. Abdul Hassan

Department of Computer Science, University of Technology, Baghdad, Iraq

Reliable Implementation of Paillier Cryptosystem

Paillier cryptography has its security based on large key sizes. Such requirement causes problems at implementation stage; an overflow may be caused when using a large prime numbers; especially when computing exponentiation and the modules of division. In this paper two ways were suggested to overcome such problems and to provide reliable secure Paillier cryptography system. Paillier cryptosystem were implemented with these suggestions using visual Basic programming. The results of the suggested implementation showed that Paillier cryptosystem was executed with relatively long key size and with encryption time shorter than decryption time.

***Keywords*:** Paillier cryptosystem; Cryptography; Data security; Encryption
Received: 05/11/2014; **Revised**: 05/12/2014; **Accepted**: 12/12/2014

1. Introduction

Private communications, protecting passwords, secure payments and commerce are some schemes of security applications. Cryptography is another important aspect to information security in which confidentiality, data integrity and authentication are studied [1, 2]. Depending on complexity of mathematical algorithms, cryptosystems are divided into private-key cryptosystem and public-key cryptosystem. Both systems are controlled by keys [3]. Public-key cryptosystem uses 2 keys; a public key for encrypting the information and a private key for decrypting it. Both keys can be known, but the information will be very difficult to reveal [1, 3, 4]. The public-key cryptosystem relies on surpassing obstacles facing the mathematical difficulties "trap-door function". It is not easy to adopt this cryptosystem, where some challenges such as computation power, memory requirement and execution time are arising [5]. A probabilistic asymmetric cryptosystem for public key cryptography called Paillier cryptosystem has been introduced in 1999 [6]. In Composite Residuosity (CR) it is assumed that the computation of the n^{th} residue classes is not easy [1,6,8]. According to this assumption, one can compute the encryption of m_1+m_2 if the encryption of m_1 and m_2 and the public-key are given. With this property, the scheme was proposed for designing secret sharing, private information retrieval, voting protocols, watermarking, threshold cryptosystems and server-aided polynomial evaluation [6,7]. Security in public key cryptography, however, is confronted with needed memory size and execution time: its presumed small set of number- [2,7], and Key sizes are much greater than those required for symmetric-key encryption. These insecurities can be passed by selecting large key sizes to ensure an extremely long time needed by any attacker before enabling him to break the message [4].

This paper focuses on the above requirements to avoid insecurities that cause problems at the implementation stage, since most programming languages may be affected when using a large prime numbers; especially when computing exponentiation and modules of division.

2. Related work

Some researchers have focused their works on the improvement of cryptosystems. In [7] Paillier was implemented in order to improving performance over the basic algorithms, with some ideas to find the algorithms with the best performance. The Chinese Remainder Theorem (CRT) is used to greatly increase speed in all algorithms, pre-computed values as well as making sure all algorithms are performing as few operations as possible. The final version of the cryptosystem implements the fastest one. In [5], parallel computation was used to improve the performance of Advanced Encryption Algorithm (AES). Parallel computations provide an efficient and reliable way to implement AES cryptography algorithm.

3. Paillier Cryptography Algorithm [7]

Below is the description of the Paillier algorithm and the steps that have been produced from:

a) Steps for Key Generation

1. Choose two large prime numbers p and q randomly and independently of each other.
2. Compute the modulus n; the product of two primes n = $p.q$ and $\lambda = lcm\ (p-1, q-1)$ (λ is Carmichael's function)
3. Select random integer g where $g \in Z_n^*2$ and g's order is a non-zero multiple of n (since g=(1+n) works and is easily calculated - this is the best

 ISSN (printed) 1813-2065, (online) 2309-1673 Printed in IRAQ

choice).
4. Ensure *n* divides the order of *g* by checking the existence of the following modular multiplicative inverse: μ=L ($g\lambda$ mod n^2) $^{-1}$ mod n, where function *L* is defined as (Lagrange function) L (u)=u-1\ n for u =1 mod n

- The public (encryption) key is (n, g).
- The private (decryption) key is (λ).

b) Steps for Encryption

1. Plaintext is m where m < n.
$r \in Z_{n^2}^*$
2. Find a random r
3. Let cipher text c = g^m . r^n mod n^2.

c) Steps for Decryption

1. The cipher text c < n^2.
2. Retrieve plaintext m=L(cλ mod n^2)/L(gλ mod n^2) mod n
Or in same expression m=L(cλ mod n^2). μ mod n
Looking at the form of the equation, one will see that the decryption will be done by first removing the random part r^n, then retrieving the exponent m mod n^2.

Example

a) Key Generation

1. p = 11, q = 17.
2. n = 187 , n^2=34969
3. g = (n + 1) = 188.
4. Calculate λ =l cm (17 − 1, 11 − 1) = 80.
5. Remember L(u) = u−1\n so we also calculate L (188^{80} mod 34969) = 80
80^{-1} = 180 mod 187=188. We get the public key (n, g) = (187, 188, 97) and the private key (p, q, λ) = (11, 17, 80).

Encryption step:

1. Plaintext m = 100.
2. r = 97 ∈ Z_n^*
3. Let cipher text c = 188^{100} · 97^{187} mod 34969 = 26118.

Decryption step:

1. The cipher text 26118 < 34969.
2. m = L(26118^{80} mod 34969)\ 80 mod 187
• m = 146\ 80 mod 187.
• m = 146 · 180 mod 187 = 100.

4. Paillier Cryptosystem Reliable Implementation

Reliable secure ***Paillier*** crypto-system can be performed by using two ways:

First: By using a set of Algorithms and functions, these are:

1- ***Algorithem-1:*** Calculate the modules (take *b, n* not equal to *0* and a is congruent of b modulo n where *b, n, a* are integer numbers and a is in the rang (n-1).

Algorithm 1: *gcd(a,n)*
g_0 =n;g_1=a; i=1;
while g_i<> 0 *do*
g_{i+1}= g_{i-1} *mod* g_i; i=i+1;
end while;
gcd=g_{i-1};
end

2-***Algorithem-2:*** Fast modular exponentiation (exploit the module function and convert the exponent to binary to give the output).

Algorithm 2*: inverse (a,n)[10]*
Begin "return x such that ax mod n =1, where 0<a<n"
g_0 =n; g_1=a; u_0=1;v_0=0;u_1=0;v_1=1;i=1;
While g_i<> 0 *do*
y=g_{i-1} *div* g_i; g_{i+1}=g_{i-1} –y* g_i ; u_{i+1}=u_{i-1}-y * u_i ;
v_{i+1}=v_{i-1} –y* v_i ;i=i+1;
end while;
x=vi-1;
if x >=o then Inverse=x; else Inverse =x+ n
End.

3-***Algorithem-3:*** Least common multiplicative (is Carmichael's function), and program to perform Euclid's algorithm for computing the greatest command divisor that is used to speed up the computation of *lcm* (p − 1, q − 1). This algorithm extended to compute inverse.

Algorithm 3: (*fast modular exponentiation* x^e *mod n).*This algorithm will compute the modular exponentiation [10]

$c = x^e \bmod n$

1. (*Pre-computation*) let
eβ-1 eβ-1...e1e0
be the binary representation of *e (i.e. e has β bit)*. For example, for *562=1000110010*, we have *β=10* and

1	0	0	0	1	1	0	0	1	0
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
e9	e8	e7	e6	e5	e4	e3	e2	e1	e0

2. (*Initialization*) set *c←1.*
3. (*Modular Exponentiation*) compute $c = x^e \bmod n$ in the following way:
for I from β-1 down to 0 do
c← c^2 mod n (squaring)
if ei=1 then
c← c . x mod n (multiplication)
4. *print c and terminate the algorithm*
end.

4- A function to generate large prime number (exploit the *rnd ()* built-in function in visual basic programming and multiply it by integer value and using that function to check whether the number prime or not).

Second: the other way is by pre-computing values in

 ISSN (printed) 1813-2065, (online) 2309-1673 Printed in IRAQ

key generation or the initialization of the scheme, instead of every time a chunk is encrypted or decrypted. Pre-computing values are:

1. Pre-compute the value r^n only once in initialization for each message passed.
2. Set g=(1+n). This is the simplest value and there seems to be no benefit of calculating something more complicated.
3. Pre-compute n^2, which is necessary in compute modules n^2.
4. Pre-compute $L(g^{\lambda} \bmod n^2)^{-1} \bmod n$ which needs to be used in decryption step .

5. Implementation, Results and Discussion

Paillier cryptosystem was implemented using Visual Basic programming language on PC (personal computer) type P4 with RAM 256 MB and 1.82 GHz Intel. First implementation was without the suggested improvements, and then with suggested improvement was conducted (section 4). First execution for Paillier cryptography by using the power and mod function that was built in the language of the program can perform only with very small prime numbers such as (3,5). With a large prime numbers the program got no result (overflow) as shown in table (1) and table (2).

Table (1) Encryption time

Message size / Key length	8182byte (8KB) Traditional implementation	8182byte (8KB) Reliable implementation
2N=24 bit	More than one second	1sec.
2N=32 bit	No result (over flow)	1 sec.
2N=40bit	No result (over flow)	1sec.
2N=48 bit	No result (over flow)	1sec.
2N=64 bit	No result (over flow)	1sec.

Table (2) Encryption time

Message size / Key length	8182byte (8KB) Traditional implementation	8182byte (8KB) Reliable implementation
2N=24 bit	More than one second	2sec.
2N=32 bit	No result (over flow)	2 sec.
2N=40bit	No result (over flow)	2sec.
2N=48 bit	No result (over flow)	2sec.
2N=64 bit	No result (over flow)	2sec.

This problem makes cryptosystem unreliable and may affect the security of the system. The second execution for Paillier cryptography proceeds with no problems (reliable implementation). Tables (1), shows the prime numbers that are used to generate modules with different key lengths; the encryption (E) was one second for message file size 8 KB (Kilo Byte). Table (2) shows the decryption (D) time is longer than the encryption time with same message and different key size due to the mathematical operation in the decryption; it executes more exponentiations operation and modules than in the encryption step.

6. Conclusion

In this paper, two schemes are suggested for the purpose of avoiding insecurity problem of large size of keys, and made a reliable implementation of Paillier cryptosystem. In the first scheme a set of algorithms and functions were used, and in the other a pre-computation of some values that are required for repeated operations was adopted. The cryptosystem was executed without those two suggestions and its result was compared with those when executing the cryptosystem with the suggested schemes. The comparison proved that the suggested schemes are reliable ways to implement the Paillier cryptosystem with relatively long key size and with encryption time less than decryption time. The decryption process takes time longer than that needed by the encryption process because the mathematical operation in the decryption step is longer.

References

[1] G.C., Kessler, **"An Overview of Cryptography"**, 1998.
[2] J. Alfred Menezes, C. Paul van Oorschot and A. Scott Vanstone, **"Handbook of Applied Cryptography"**, CRC Press, 1996.
[3] A.K. Farhan, "Enhancement of Public Key Cryptography Methods", MSc thesis, University of Technology, Baghdad, IRAQ, 2005
[4] W. Stallings, **"Cryptography and Network Security, Principle and Practice"**, Addison Wesley, 1999.
[5] M. Nagendra and M. Chandra Sekhar, Int. J. Software Eng. & its Appl., 8(2) (2014), 287-296.
[6] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes", in Advances in Cryptology EUROCRYPT 1999, pp. 223-238.
[7] L. Nguyen, R. Safavi-Naini and K. Kurosawa, J. Univ. Computer Sci., 11(6) (2005).
[8] K. Richardson, "Progress on Probabilistic Encryption Schemes", MSc thesis, Rochester Institute of Technology, July 7, 2006.
[9] Y.S. Yan, **"Number Theory for Computing"**, Springer-Verlag Berlin Heidelberg GmbH, (2nd ed.), February 2000.

 ISSN (printed) 1813-2065, (online) 2309-1673 Printed in IRAQ