On Using The First Variant of Dependent RSA Encryption Scheme to Secure Text: A Tutorial

D Rachmawati¹, M A Budiman¹

¹Departemen Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Kampus USU, Medan 20155, Indonesia

Email: dian.rachmawati@usu.ac.id and mandrib@usu.ac.id

Abstract. The Rivest-Shamir-Adleman (RSA) public key cryptosystem has been extensively used to secure digital communication since more than forty years. The security of RSA relies on the hardness of factoring a big integer into its two big prime factors: the bigger the integer, the secure the cryptosystem. However, the RSA cryptosystem is proven to be not semantically secure. Some variants of Dependent RSA encryption schemes have been introduced by Pointcheval to overcome this problem. This study shows how the first variant of Dependent RSA (DRSA-1) encryption scheme is used to secure text. All algorithms have been implemented in the Python programming language version 2.7.15.

1. Introduction

In digital communication, securing confidential messages is a sine qua non condition. Modern cryptography is the art of using tactical action and the science of using mathematical methods to secure messages. A cryptographic algorithm used to encrypt a message into a concealed form is called a cipher. In general, a cipher consists of one encryption function, one decryption function, and one secret key. The decryption function is merely the inverse of the encryption function, mathematically [1]. If the same key is used in both the encryption and decryption processes, the cipher is said to belong to the class of symmetric key algorithm or symmetric cryptography.

Prior to 1976, all available ciphers had belonged to the class of symmetric key algorithm. All symmetric key algorithms have at least one main problem: the secret keys have to be distributed to all communicating parties. The distribution of secret key has to be done in a secure channel. A secure channel can be quite expensive to built and quite difficult to manage [2].

In 1976, the Diffie-Hellman key exchange protocol [3] (or should rather be called “Diffie-Hellman-Merkle key exchange protocol” as suggested by Hellman in [4]) was published by Whitfield Diffie and Martin Hellman to overcome the problem of exchanging secret keys. In this protocol, a new term called “public key” was introduced. A public key is a key which is made available to all communicating parties and used to encrypt messages. In order to decrypt encrypted messages, a private (secret) key is used. Since different keys are used for encryption and decryption processes, encryption algorithms that implement Diffie-Hellman-Merkle key exchange protocol are said to belong to the class of asymmetric key algorithm or asymmetric cryptography.

The RSA cryptosystem [5], which was published in 1978, was the first encryption algorithm to use Diffie-Hellman-Merkle’s idea of asymmetric cryptography. Until now, the RSA has been the most
widely used asymmetric key algorithm. The security of RSA depends on the hardness of factoring a big integer into its two big prime factors: the bigger the integer, the securer the RSA cryptosystem.

The RSA, however, has a problem of being not semantically secure [6] [7]. Semantically secure, a term brought forward by Goldwasser and Micali [8] [9], is a condition where only very little information regarding the original text can be practically pulled out by an adversary from the ciphertext (concealed text). To overcome this problem, Pointcheval [6] has proposed a variant of RSA cryptosystem, namely Dependent RSA (DRSA) encryption scheme.

The DRSA encryption scheme has a number of variants. One of those variants is DRSA-1, a scheme that is proven to be semantically secure against the presentation of adaptive chosen-ciphertext attacks [6]. This paper shows a step-by-step tutorial on how to secure text with DRSA-1.

2. Methods
In this section we explain the DRSA-1 algorithm. The DRSA-1 algorithm consists of three phases: key generation, encryption, and decryption. Suppose there are two parties that are willing to communicate. One party is the sender of the information, called “Alice”, and the other is the recipient, called “Bob”. The three phases of the encryption scheme [6] are described as follows.

2.1 The key generation phase
In the phase of key generation, Bob does these six steps:
1. Using some primality tests such as Fermat’s little theorem, Bob generates two large distinct prime numbers, \( p \) and \( q \). Bob keeps the values of \( p \) and \( q \) as his private keys.
2. Bob calculates \( n = pq \). Bob publish the value of \( n \) as his public key. Anyone who wants to send a confidential message to Bob should obtain this value.
3. Bob calculates \( \Phi(n) = (p – 1)(q – 1) \). Bob keeps the value of \( \Phi(n) \).
4. Bob chooses one random value of \( e \), such that \( e \) and \( \Phi(n) \) are relatively prime. Relatively prime means that \( \text{gcd}(\Phi(n), e) = 1 \). The value of \( e \) is also Bob public key and, therefore, should be published.
5. Bob computes \( d \), where \( d \equiv e^{-1} \pmod{\Phi(n)} \). The value of \( d \) should be kept private.
6. Bob chooses one hash function \( h \) to be used in the scheme. The hash function can be a simple one, but a cryptographically secure hash function is preferable. The hash function acts as control padding and should use two parameters as an input. The chosen hash function should be known anyone who wants to send Bob a confidential message.

2.2 The encryption phase
In the phase of encryption, Alice does these seven steps:
1. Alice chooses \( m \), the message to be encrypted.
2. Alice obtains the values of \( e \) and \( n \), which are Bob’s public key, and also obtain the hash function used by Bob.
3. Alice chooses a random value of \( k \), where \( k \) can be any integer between \( 1 \) and \( n – 1 \).
4. Alice calculates \( A = k^e \mod n \).
5. Alice calculates \( B = m \times (k + 1)^e \mod n \).
6. Alice calculates \( H = h(m, k) \).
7. Alice sends the ciphertext \( C = (A, B, H) \) to Bob.

2.3 The decryption phase
In the phase of decryption, Bob does these … steps:
1. Bob gets the ciphertext \( C = (A, B, H) \) from Alice.
2. Bob calculates \( k = A^d \mod n \).
3. Bob calculates \( m = B / (k + 1)^e \mod n \).
4. Bob checks whether \( H = h(m, k) \).
3. Discussions

In this section we discuss how to use the DRSA-1 algorithm to secure text. Consider a scenario that a sender (Alice) intends to send a message to a recipient (Bob). Suppose the message is a letter “A”. Bob and Alice will use the DRSA-1 algorithm to secure the message and to achieve semantical security. The computation of the three phases of the DRSA-1 encryption scheme [6] is illustrated as follows. All computations have been done in the Python programming language version 2.7.15.

3.1 Key generation (Bob)

Using some primality tests such as Fermat’s little theorem or Agrawal-Kayal-Saxena algorithm, Bob generates two large prime numbers, \(p\) and \(q\). Both are chosen from random 30-digit integers.

\[
p = 894860479296468251010778381253
\]
\[
q = 927232533298178416549126807541
\]

Bob calculates \(n = pq\) and publish the value of \(n\) as his public key.

\[
n = p \times q = 894860479296468251010778381253 \times 927232533298178416549126807541 = 829743749166486395087817424422221936851876214126919653428873
\]

Bob calculates \(\Phi(n) = (p – 1)(q – 1)\) and keeps the value of \(\Phi(n)\) to himself.

\[
\Phi(n) = (p – 1) \times (q – 1) = (894860479296468251010778381253 – 1) \times (927232533298178416549126807541 – 1) = 829743749166486395087817424420399843839281567459359748240080
\]

Bob chooses one random value of \(e\), so that \(gcd(\Phi(n), e) = 1\). Bob publishes the value of \(e\) as his public key. To compute the \(gcd\) (the greatest common divisor) function, see [10].

\[
e = 481562733369321329725315214636406979333550049489386650041523
\]

Test: \(gcd(\Phi(n), e) = \)

\[
gcd(829743749166486395087817424420399843839281567459359748240080, 481562733369321329725315214636406979333550049489386650041523) = 1 \text{ (OK)}
\]

Bob computes \(d\), where \(d = e^{-1} \pmod {\Phi(n)}\) and keeps the value of \(d\) to himself. The value of \(d\) can be computed using the extended Euclidean algorithm (see [11]).

\[
d = 1894135429061850153372721259756758526427959439977493983329467
\]

Bob chooses a simple hash function \(h\) that has two input parameters and lets anybody who wants to communicate with him to use that hash function as control padding.

\[
h(m, k) = m^k \mod 100
\]

3.2 Encryption (Alice)

Alice intends to encrypt a text message which consists of one letter “A”. Alice looks up the value of “A” in her preferred encoding table, such as ASCII. In the ASCII table, the value of “A” is 65, and this should be the value of \(m\).
Alice obtains Bob’s public key, which are \( n \) and \( e \).

\[
\begin{align*}
 n &= 829743749166486395087817424422221936851876214126919653428873 \\
 e &= 481562733369321329725315214636406979333550049489386650041523
\end{align*}
\]

Alice chooses a random value of \( k \) from an integer between 1 and \( n - 1 \).

\[
k = 6013644342956060970094811175915561437988342009712609676717674
\]

Alice calculates A.

\[
A = k^e \mod n = 829743749166486395087817424422221936851876214126919653428873^{481562733369321329725315214636406979333550049489386650041523} \mod 829743749166486395087817424422221936851876214126919653428873 = 3376905564305541731089687519718876201560459589599382253046897
\]

Alice calculates B.

\[
B = m \times (k + 1)^e \mod n = 65 \times (6013644342956060970094811175915561437988342009712609676717674 + 1) \mod 829743749166486395087817424422221936851876214126919653428873 = 107136975025660754978402798396109763219415037568602799185438
\]

Alice calculates H.

\[
H = h(m, k) = m^k \mod 100 = 65^{6013644342956060970094811175915561437988342009712609676717674} \mod 100 = 25
\]

Alice then sends the ciphertext \( C = (A, B, H) \) to Bob.

### 3.3 Decryption (Bob)

Bob obtains the ciphertext \( C = (A, B, H) \) from Alice.

\[
\begin{align*}
A &= 3376905564305541731089687519718876201560459589599382253046897 \\
B &= 107136975025660754978402798396109763219415037568602799185438 \\
H &= 25
\end{align*}
\]

Bob calculates k.

\[
k = A^d \mod n = 3376905564305541731089687519718876201560459589599382253046897^{184135442368155553372722897676585264379549097749398125947} \mod 829743749166486395087817424422221936851876214126919653428873 = 6013644342956060970094811175915561437988342009712609676717674
\]
Bob calculates $m$.

$$m = B / (k + 1) \mod n = 107136975025607549784027983961097632194150375568602799185438 / (60136443429560097090491175915561437988342009712609676717674 + 1) \mod 8297437491664839508787817424422221936851876214126919653428873 = 65$$

Bob checks whether $H = h(m, k)$

$$h(m, k) = m^k \mod 100 = 65 \mod 100 = 65 \mod 100 = 25$$

Since $H = h(m, k)$, Bob concludes that the control padding has been satisfied. Bob then looks up the ASCII table for $m = 65$ and he gets the letter “A” which is the original text that Alice wants him to read.

4. Conclusion
We have presented the tutorial on how to use the first variant of Dependent RSA (DRSA-1) encryption scheme to secure a simple text. The DRSA-1 algorithm has been implemented in the Python programming version 2.7.15. This tutorial uses two 30-digit prime numbers to construct 60-digit public key $n$. However, in practice, we suggest using longer digits of $n$, for example 500-digit, in order to preserve higher security.

5. References
[1] Rachmawati D and Budiman M A 2017 An implementation of the H-Rabin algorithm in the Shamir three-pass protocol 2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro–Mechanical System, and Information Technology (ICACOMIT)
[2] Wang Y, Wang J and Chen X 2016 Secure searchable encryption: a survey Journal of Communications and Information Networks 1 52–65
[3] Diffie W and Hellman M 1976 New directions in cryptography IEEE Transactions on Information Theory 22 6 pp 644-654
[4] Hellman M 2002 An overview of public key cryptography IEEE Communications Magazine 40 42–9
[5] 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
[6] Pointcheval D 1999 New Public Key Cryptosystems Based on the Dependent-RSA Problems Advances in Cryptology — EUROCRYPT ’99 Lecture Notes in Computer Science 239–54
[7] Padyhe S 2006 On DRSA public key cryptosystem International Arab Journal of Information Technology 3 4 pp 334-336
[8] Goldwasser S and Micali S 1982 Probabilistic encryption & how to play mental poker keeping secret all partial information Proceedings of the fourteenth annual ACM symposium on Theory of computing - STOC 82
[9] Goldwasser S and Micali S 1984 Probabilistic encryption Journal of Computer and System Sciences 28 270–99
[10] Rosenthal D, Rosenthal D and Rosenthal P 2014 The Euclidean Algorithm and Applications Undergraduate Texts in Mathematics A Readable Introduction to Real Mathematics 47–60
[11] Iliev A, Kyurkchiev N and Rahnev A 2018 A Note on Adaptation of the Knuth’s Extended Euclidean Algorithm for Computing Multiplicative Inverse International Journal of Pure and Applied Mathematics 118 2 pp 281-290
Acknowledgments
We gratefully acknowledge that this study is funded by Universitas Sumatera Utara. The support is under the research grant Talenta USU of Year 2019 Contract Number: 4167/UN5.1.R/PPM/2019 Date: 1st April 2019.