Encryption and Decryption with RSA Algorithm
Mathematics and the Computer
V55.0102 – Fall 1999

Dr. Peter Kramer, Instructor

November 8, 1999

These notes describe the spirit in which the RSA algorithm is used to encrypt and decrypt data. Details will differ in various applications, but I hope to convey the general idea.

We will first demonstrate the idea with an example (Section 1), and then concisely restate the steps involved in the form of a general recipe (Section 2). We end these notes with some explanation and motivation for some of the steps involved (Section 3).

1 Example Application of Public Key Cryptography Using RSA Algorithm

In our example, we will suppose that FBI Agent Scully wants to send a secret message to Agent Mulder over their cell phones equipped with encryption/decryption devices. As we discussed in class, the secret message itself is usually encrypted by a symmetric cipher. The key to this symmetric cipher, the *session key*, is transmitted securely through public key cryptography (using the RSA algorithm). Let's suppose that some silly outdated government regulation restricts the FBI to only using a shift substitution cipher to encode the main body of their secret messages (yeah, right, but bear with me.) But they do use the RSA algorithm to transmit the session key (which in the case of the shift substitution cipher, is just the shift value used.) Scully elects to use a shift value of 9 for her shift substitution enciphering of the main message, and she wishes to transmit the phrase "s=9" securely to Mulder via public key cryptography using the RSA algorithm. Note carefully that "s=9" is the *key to the shift substitution cipher*, and at the same time the *cleartext for the cipher based on the RSA algorithm*. We will from here on out focus on the use of the RSA algorithm, so we will just call the phrase "s=9" the cleartext.

The public key for the RSA algorithm consists of two numbers, n and e, and the private key consists of a single number d. For the RSA algorithm to work properly, the keys should have the following two properties:

- n should be the product of two distinct prime numbers p and q,
- the public and private keys should be related so that

$$ed = 1 \bmod \phi,$$

where $\phi = (p-1)(q-1)$. This equation means that the product of e and d should have a remainder of 1 when divided by ϕ.

We will in a later lecture describe how to choose RSA public and private keys with this property. But first we show how they are used. Since Scully is sending a secret message to Mulder, the message will be encrypted using Mulder's public key and decrypted using Mulder's private key. (Mulder has no objection to anyone sending him secret anonymous tips, but he certainly wouldn't want anyone to be able to decipher the secret messages sent to him!)

For our example, we will take Mulder's public key to be:

$$n = 111, \qquad e = 5$$

and Mulder's private key to be:

$$d = 29.$$

These keys satisfy the first RSA property listed above because $n = 111$ is the product of the prime numbers $p = 3$ and $q = 37$. The keys satisfy the second RSA property because $\phi = 2 \times 36 = 72$, and

$$ed = 5 \times 29 = 145 = 1 \bmod 72 = 1 \bmod \phi.$$

In truth, the numbers in this example are much too small to be of practical value; they'd be easy to crack. In practice, the keys used in practice involve numbers with hundreds of digits.

Now that we have the keys, we show how they can be used to encrypt and decrypt the cleartext data string "s=9".

1.1 RSA Encryption

1.1.1 Step E1: Convert cleartext data string to a cleartext numerical string

If the data string were just numbers (as with a digital signature), we could dispense with this step. But in general, as in our example, we may wish to use RSA encryption

on a cleartext which involves letters, numbers, and punctuation. One way to convert such data to a purely numerical format is to use the ASCII codes (see page 11). As the (standard) ASCII codes range up to 127, we can use 3 decimal digits to represent each character. Leading zeros are used to pad the ASCII codes which have fewer than 3 digits:

Character	ASCII Code (Decimal)
s	115
=	061
9	057

Once Scully's machine has computed the numerical representation of each character in the cleartext data string, she just concatenates them together. So the cleartext numerical string is:

$$115061057$$

In truth, Scully's machine would use the *binary* representation of the ASCII codes to make a binary string (made up of only ones and zeros) to represent the cleartext data. I have, however, decided to be merciful to you and explain the RSA algorithm using purely decimal numbers. Working in binary is conceptually the same.

1.1.2 Step E2: Chop up cleartext numerical string into cleartext blocks

Scully's machine counts the number of digits in n, and chooses L_{clear} to be one less than this. In the present example, $n = 111$ is a three digit number, so $L_{\text{clear}} = 2$.

Now Scully's machine chops up the cleartext numerical string into cleartext blocks of length L_{clear}.

$$11 \quad 50 \quad 61 \quad 05 \quad 7$$

The last block will sometimes have fewer than L_{clear} digits; in this case we can introduce additional zeros to the right to fill it out to a full L_{clear} digits:

$$11 \quad 50 \quad 61 \quad 05 \quad 70$$

Each of these blocks represent a decimal number which will be encrypted separately. Notice that whenever L_{clear} is not a multiple of 3, these cleartext blocks do not directly correspond to the ASCII representation of the original characters; this scrambling is good for destroying patterns which could help code-breakers.

1.1.3 Step E3: Encrypt each cleartext block

In general, if m is the value of the cleartext block of a message, then the value c of the ciphertext block is given by the following *RSA public key function*:

$$c = E(m) = m^e \bmod n,$$

where n and e are numbers supplied with the public key. In our particular example, the RSA public key function using Mulder's public key is:

$$c = E_M(m) = m^5 \bmod 111.$$

(The subscript M stands for Mulder's public key).

With sufficient effort, Scully's machine computes:

$$101 = 11^5 \bmod 111$$
$$35 = 50^5 \bmod 111$$
$$76 = 61^5 \bmod 111$$
$$17 = 05^5 \bmod 111$$
$$49 = 70^5 \bmod 111$$

This looks daunting, and indeed would be quite arduous by either pencil and paper or even by electronic calculator if the numbers were realistically large. The computer can however do these calculations easily *so long as it is programmed intelligently.* We will come back to this point later. For now, just assume that the encryption device has a program which can indeed figure out what the remainder of 61^5 is when divided by 111. You have such a program available in lab.

Note that in general, the results of this RSA encryption step can have up to L_{cipher} digits, where L_{cipher} is the number of digits which n has. (So in particular, $L_{\text{cipher}} = L_{\text{clear}} + 1$ always). Each enciphered block should have the same number of digits, so Scully introduces leading zeros to make each result have L_{cipher} digits.

In our example, $L_{\text{cipher}} = 3$, and so Scully's machine adds leading zeros where necessary to the ciphertext blocks:

101 035 076 017 049

1.1.4 Step E4: Concatenate ciphertext blocks together to form ciphertext string

101035076017049

We can take this as the encrypted string which will be securely transmitted. Now what will Mulder's machine do when it receives this jumble of numbers?

1.2 RSA Decryption

Part of the decryption process will involve undoing the systematic encryption steps, as with ordinary symmetric ciphers. But Step E3 is hard to reverse, except with Mulder's private key (see Step D2). It is only this part of the ciphering/deciphering process which is asymmetric.

1.2.1 Step D1: Break the ciphertext string into ciphertext blocks

Mulder's machine first breaks up the ciphertext string into blocks of L_{cipher} digits, where L_{cipher} is the number of digits in the number n, which is part of his public key. This just undoes Step E4:

$$101 \qquad 035 \qquad 076 \qquad 017 \qquad 049$$

1.2.2 Step D2: Decrypt each ciphertext block

The general algorithm for the RSA decryption of a ciphertext block value c of a message to get the cleartext block value m is described by the following *RSA private key function*:

$$m = D(c) = c^d \bmod n$$

where the value of d is obtained from the private key and n is obtained from the public key. In particular, no one can do this decryption step unless they either have the private key, or somehow manage to guess it through clever cryptanalysis.

But, whether Mulder types in the private key or the private key is securely built in, his decryption device (and hopefully his alone!) has access to the private key value $d = 29$. And of course Mulder's machine has the public key value n as well. So Mulder's particular RSA private key function is:

$$m = D_M(c) = c^{29} \bmod 111.$$

Applying this to the ciphertext blocks from Step 1, Mulder's machine computes the values of the cleartext blocks. The cleartext blocks should all have no more than $L_{\text{clear}} = L_{\text{cipher}} - 1$ digits, otherwise something has gone wrong. In the present

example, $L_{\text{clear}} = 2$, and indeed all the results have two digits or less:

$$\begin{aligned} 11 &= 101^{29} \bmod 111 \\ 50 &= 035^{29} \bmod 111 \\ 61 &= 076^{29} \bmod 111 \\ 5 &= 017^{29} \bmod 111 \\ 70 &= 049^{29} \bmod 111 \end{aligned}$$

Leading zeros are introduced if necessary to pad all these cleartext numerical values to a full L_{clear} digits:

$$11 \quad 50 \quad 61 \quad 05 \quad 70$$

1.2.3 Step D3: Concatenate cleartext blocks together to form cleartext numerical string

This just undoes Step E2:

$$1150610570$$

1.2.4 Step D4: Convert numerical cleartext string into cleartext character string

This reverses Step E1, and would be omitted if the original cleartext message were known to be purely numerical. Split the numerical cleartext string into blocks of three, discarding any trailing blocks or partial blocks of pure zeros:

$$115 \quad 061 \quad 057$$

These cleartext blocks are just the ASCII codes for the cleartext characters; consulting the table of ASCII codes, we decipher the cleartext message:

$$s = 9$$

This is the session key to the symmetric cipher which was used to encrypt the main body of the message which Scully sent to Mulder.

2 Recipe for RSA Encryption and Decryption

Here we concisely provide the general specification for how to use the RSA algorithm for encryption or decryption of a message. Recall that, in practice, the message encrypted or decrypted by RSA is usually either a session key or a digital signature, rather than the main body of the data being transmitted.

The variables used in RSA encryption and decryption are:

- The public key values e and n,
- the private key value d,
- the length of cleartext blocks, L_{clear}, which is deduced from the public key as one less than the number of digits in n,
- the length of ciphertext blocks, L_{cipher}, which is deduced from the public key as the number of digits in n

One thing to keep in mind is that, for sending secret messages (as in the above example), the public key function E is used by the sender for encryption and the private key function D is used by the recipient for decryption. To transmit a digital signature for verification, however, the roles are reversed: the private key function D is used by the sender for encryption and the public key function E is used by the recipient for decryption.

2.1 RSA Encryption

2.1.1 Step E1: Convert data string to a numerical string

This step may be omitted when the data is already purely numerical as, for example, in the case of a digital signature.

Otherwise, convert each character in the message into a three digit ASCII code (using decimal numbers). Introduce leading zeros if necessary to pad the ASCII codes to a full 3 digits. Concatenate these numerical ASCII codes together to form a cleartext string of numbers, which should be exactly three times as long as the original cleartext string of characters.

2.1.2 Step E2: Chop up cleartext numerical string into cleartext blocks

Chop up the cleartext numerical string into blocks of length L_{clear}. If the last block has fewer than L_{clear} digits, then add zeros to the right to fill it up to a full L_{clear}

digits.

2.1.3 Step E3: Encipher each cleartext block

If m is the value of the cleartext block, then the value c of the ciphertext block is given as follows:

1. For sending a secret message or session key, use the RSA public key function:
$$c = E(m) = m^e \bmod n.$$
2. For sending a digital signature, use the RSA private key function:
$$c = D(m) = m^d \bmod n.$$

In either case, each ciphertext block should have L_{cipher} digits. Fill up any ciphertext blocks with fewer digits by introducing leading zeros.

Unless n is a small number, you probably will want to do this calculation using the computer program supplied to you in lab.

2.1.4 Step E4: Concatenate ciphertext blocks together to form ciphertext string

This step is self-explanatory.

2.2 RSA Decryption

2.2.1 Step D1: Break the ciphertext string into ciphertext blocks

Break up the ciphertext string into blocks of L_{cipher} digits. All blocks should be complete if you are decrypting using the right key. This step undoes Step E4.

2.2.2 Step D2: Decrypt each ciphertext block

If c is the value of the ciphertext block, then the value m of the cleartext block is given as follows:

1. When receiving a secret message or session key, use the RSA private key function:
$$m = D(c) = c^d \bmod n.$$

2. For verifying a digital signature which has been received, use the RSA public key function:
$$m = E(c) = c^e \bmod n.$$

Each cleartext block should have L_{clear} digits. For blocks with fewer digits, introduce leading zeros to fill them up to a full L_{clear} digits.

This step undoes Step E3.

2.2.3 Step D3: Concatenate cleartext blocks together to form cleartext numerical string

This step is self-explanatory, and undoes Step E2.

2.2.4 Step D4: Convert numerical cleartext string into cleartext character string

This step may be omitted when the original cleartext data is known to be numerical as, for example, in the case of a digital signature.

Break the numerical cleartext string into blocks of three, dropping any trailing blocks or partial blocks of pure zeros. Then convert each three-digit block to the character with the corresponding ASCII code.

3 Some Answered Questions

3.1 Why do we choose L_{clear} and L_{cipher} the way we did?

A vital component of any cipher is that distinct cleartext blocks be enciphered as distinct ciphertext blocks. Otherwise, the enciphered message would be impossible to decipher. The RSA public key function $E(m)$ and private key function $D(m)$ are each known to map the set of numbers $0, 1, 2, \ldots, n-1$ in a one-to-one way onto the set of numbers $0, 1, 2, \ldots, n-1$. That is each number from 0 to $n-1$ is mapped to a different number also from 0 to $n-1$. But 0 and n are both mapped by the RSA functions to the same value (as are any pair of numbers differing by a multiple of n). Therefore, we should only use the RSA public key and private key functions to encrypt cleartext blocks which are guaranteed to have a numerical value less than n. We can do this by making sure these blocks have fewer digits than n does, hence we choose L_{clear} to be one fewer than the number of digits in n.

Another consequence of the fact that the RSA private key and public key functions map $0, 1, 2, \ldots, n-1$ in a one-to-one way onto the set of numbers $0, 1, 2, \ldots, n-1$ is that the ciphertext blocks can be as large as $n-1$, meaning they can have as many digits as n does. Therefore, we must allocate L_{cipher} digits for each ciphertext block, where L_{cipher} is the number of digits in n.

3.2 Why are the RSA public key and private key functions and key values denoted by E's and D's, respectively?

For sending secrets, whether it be messages or session keys, the sender uses the public key to encrypt and the receiver uses the private key to decrypt. For this use, therefore, it makes sense to associate the RSA public key with the letter "E" for encrypt and the RSA private key with the letter "D" for decrypt.

Digital signatures work in the reversed way, but the choice of E and D notation was probably fixed due to its use in the standard sending of secret messages. Digital signatures may be thought of as a clever secondary use of RSA cryptography with encryption and decryption "done backwards."