

1-02-328: Communication and E-Commerce Security

Recitations 7–8: RSA

Michael J. May

April 6–27, 2011

In this recitation we will work over some examples using the RSA encryption algorithm and write some code in C# which performs the encryptions and decryptions for us.

1 Hand Example of RSA

1. Entity A chooses primes $p = 47, q = 71$
2. He calculates $n = p \times q = 3337$
3. ϕ is then $\phi = (p - 1) * (q - 1) = 3220$
4. A chooses the public exponent to be relatively prime to 3220: $e = 79$
5. A 's public key is therefore $(n = 3337, e = 79)$
6. A finds the inverse for e in n : $d = 79^{-1} \pmod{3220} = 1019$
 - This is because $79 \times 1019 = 80501$
 - Taking modulo 3320: $3220 \times 25 + 1$, so $80501 \pmod{3220} = 1$
7. A 's private key is $(n = 3337, d = 1019)$

To do encryption:

1. B produces a message $m = 688232687966683$
2. B must break it into chunks smaller than n (otherwise we get ambiguity in the results since it's modulo n)
3. Break into chunks < 3337
688 232 687 966 683
4. Performing encryption on the first block:

$$\begin{aligned} & E((3337, 79), 688) \\ &= 688^{79} \pmod{3337} \\ &= 1570 \end{aligned}$$

To do decryption:

1. A receives the first block 1570 from B

2. A calculates using d :

$$\begin{aligned} & D((3337, 1019), 1570) \\ &= 1570^{1019} \pmod{3337} \\ &= 688 \end{aligned}$$

thus recovering the first block

3. Each subsequent block is processed in turn

2 Simple Examples of RSA

We begin with a hand-based example of an RSA encryption and decryption. The example is intentionally small so that we can do the calculations easily by hand.

1. Entity A chooses the primes $p = 2357$, $q = 2551$, and computes

$$n = pq = 6012707$$

and

$$\phi = (p - 1)(q - 1) = 6007800$$

2. A chooses $e = 3674911$ and, using the extended Euclidean algorithm, finds

$$d = 422191$$

such that

$$ed \equiv 1 \pmod{\phi}$$

3. A 's public key is the pair $(n = 6012707; e = 3674911)$, while A 's private key is $d = 422191$.

To do encryption:

1. To encrypt a message $m = 5234673$, B uses an algorithm for modular exponentiation to compute

$$\begin{aligned} c &= m^e \pmod{n} \\ &= 5234673^{3674911} \pmod{6012707} \\ &= 3650502 \end{aligned}$$

and sends this to A .

To do decryption:

1. A computes

$$\begin{aligned} &c^d \pmod{n} \\ &= 3650502^{422191} \pmod{6012707} \\ &= 5234673 \end{aligned}$$

which is the original message

3 Homomorphic Property of RSA

Let's go back to the hand example of RSA from section 1. Consider two message that is 1 block long: 75 and 39.

Let's do the calculations for 75:

$$\begin{aligned} & E((3337, 79), 75) \\ &= 75^{79} \pmod{3337} \\ &= 2213 \end{aligned}$$

For 39:

$$\begin{aligned} & E((3337, 79), 39) \\ &= 39^{79} \pmod{3337} \\ &= 2739 \end{aligned}$$

If we now calculate $75 \times 39 = 2925$:

$$\begin{aligned} & E((3337, 79), 2925) \\ &= 2925^{79} \pmod{3337} \\ &= 1415 \end{aligned}$$

However, we can also note that:

$$\begin{aligned} 2213 \times 2739 &= 6061407 \\ 6061407 \pmod{3337} &= 1415 \end{aligned}$$

Which means that an attacker can multiply together two (perhaps unknown) messages and produce a ciphertext which corresponds precisely to the multiplication of the plain texts.

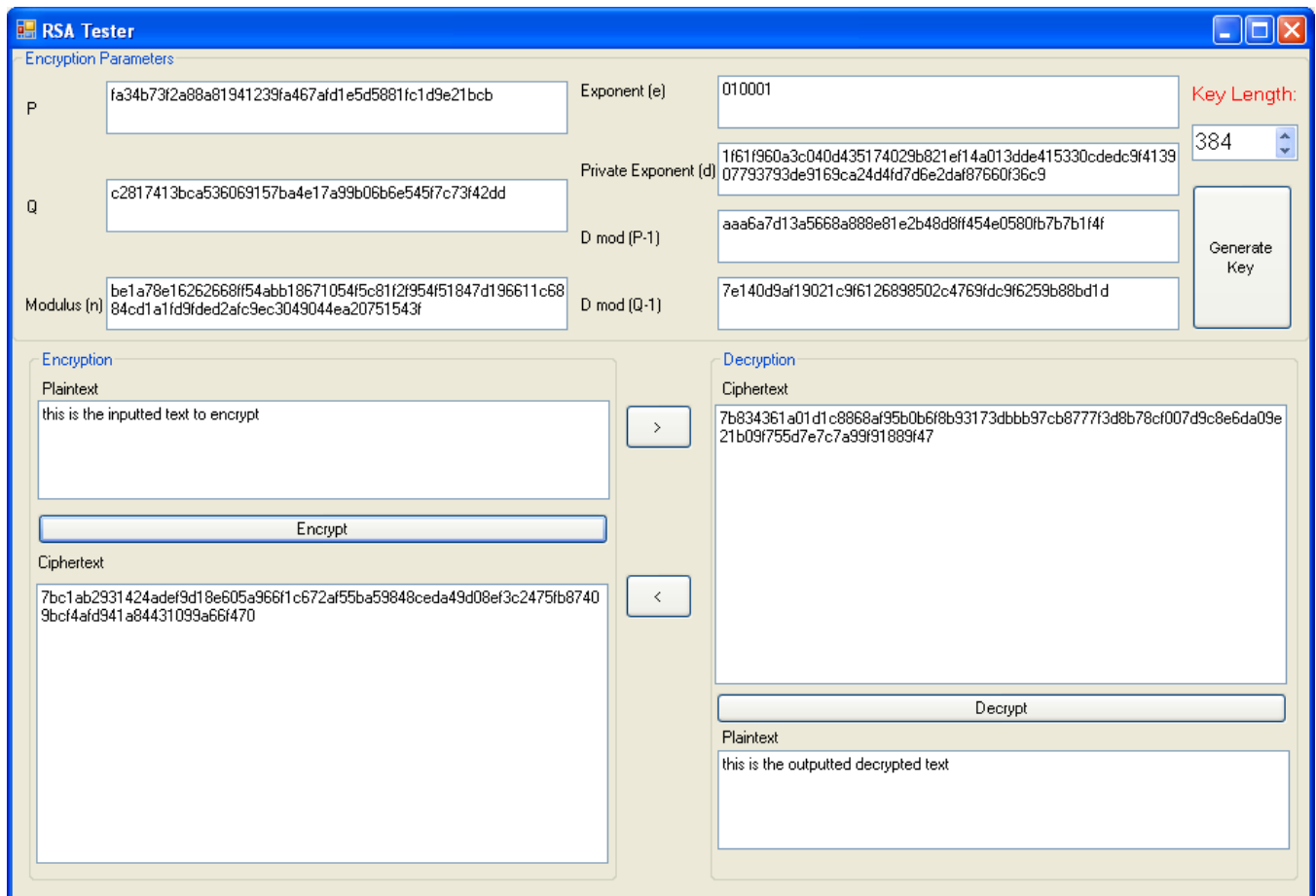
This property follows directly from the multiplicative properties of modular arithmetic:

$$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}$$

4 Using RSA in C#

In this part of the recitation I will show you a program which lets you perform RSA encryptions on plain text messages. All of the work for the algorithm takes place under the hood, so there isn't a lot to see here. Still, it's important for you to recognize the classes that offer the RSA cryptographic service for later use in this class (and outside).

4.1 User Interface



Notes on the GUI:

1. The fields in the "Encryption Parameters" groupbox come from the RSAParameters object. It's read only here.
2. The button labeled ">" copies the contents of the "Ciphertext" textbox from the "Encryption" groupbox to the "Decryption" groupbox.
3. The button labeled "<" copies the contents of the "Plaintext" textbox from the "Decryption" groupbox to the "Encryption" groupbox.
4. The numeric selector allows you to change the key size for the encryption and decryption. In .NET's implementation, the minimum key size is 384 bits and the maximum is 16,384 bits. You can choose any number in between so long as its a multiple of 8 (the numeric selector enforces this).

4.2 The RSAParameters Object

The fields shown in the “Encryption Parameters” groupbox are the fields in the RSAParameters object. As you may recognize from class, they are sufficient information to encrypt and decrypt messages using the RSA algorithm.

4.3 The RSACryptoServiceProvider Object

The RSACryptoServiceProvider is the interface to C#'s default implementation of the RSA algorithm. It can take as a parameter a default key size or existing RSAParameters.

Creating the object with a default key size of 512 bytes is done as follows:

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(512);
```

When you create the object in this manner, it automatically generates RSA key parameters for you randomly. You can access the parameters as an RSAParameters object using the Export function:

```
RSAParameters par = rsa.ExportParameters(true);
```

Each parameter can be accessed from the RSAParameters object. They are stored as `byte[]` arrays and so can be printed using a simple function shown below in Section ??.

5 What to do

Use the code and tool provided to perform the following tasks:

5.1 Fill in the Code

Fill in the functions `btnEncrypt_Click` for encryption and `btnDecrypt_Click` for decryption using the class `RSACryptoServiceProvider`. Encryption and Decryption are very simple using it. Encryption takes a byte array and produces a byte array as output:

```
RSA.Encrypt(DataToEncrypt, true);
```

The second parameter indicates whether the algorithm should use a non-default padding algorithm. Decryption is just as simple:

```
RSA.Decrypt(DataToDecrypt, true)
```

Since the output is a byte array, you can use the “ShowHex” function provided to easily show the array in the “Ciphertext” textbox. To read the output from the textbox into a byte array, you can use the “ReadHex” function provided.

Like DES, the `RSACryptoServiceProvider` works only over `byte[]` arrays, we must convert all data into byte arrays before processing. To convert text to bytes, start by using the `ASCII Encoding` object in C# (`Encoding.ASCII`). Use the `GetBytes()` and `GetString()` functions to convert strings to and from bytes.

5.2 Experiment 1: Encryptions

1. Generate a new 512 bit key. Enter the text: “It was a picture of a boa constrictor” in the plain text box. Encrypt it using the Encrypt button. Note the result.
2. Press the Encrypt button again. What happens? (Why?)
3. Use the “>” button to copy the cipher text over to the right side. Press the Decrypt button. What is the result?

4. Press the Encrypt button again. Use the “>” button to copy the cipher text over to the right side. Press the Decrypt button. What is the result?

What is going on here?

5.3 Experiment 2: Text Length

1. Generate a new 512 bit key. Enter the text: “It was a picture of a boa constrictor” in the plain text box. Encrypt it using the Encrypt button.
2. Replace the text with: “It was a picture of a boa constrictor digesting an elephant.”. Encrypt it using the Encrypt button. What happens? Why?
3. Figure out the maximum length text that you can successfully encrypt.
4. Generate a new 1,024 bit key. Press Encrypt again to encrypt the text. What happens? Figure out the maximum length text you can successfully encrypt.

What can you tell about the way .NET limits the length of text inputs?

1. Go into the code and change the encoding algorithm you used in `btnEncrypt_Click` to read something like:

```
byte[] dataToEncrypt = Encoding.Unicode.GetBytes(tbPlainIn.Text);
```

2. Change the encoding algorithm you used in `btnDecrypt_Click` to read something like:

```
this.tbPlainOut.Text = Encoding.Unicode.GetString(decryptedData);
```

3. Recompile and run the application
4. Generate a new 512 bit key. Enter the text: “It was a picture of a boa constrictor” in the plain text box. Encrypt it using the Encrypt button. What happens? Why?

5.4 Experiment 3: Extending the RSA Testing Application

In the previous experiment you saw that .NET limits the length of the input text for encryption. Adapt the code provided to permit the encryption and decryption of unlimited length strings.

Note: You will need to do some work in the parsing functions as well.