Presenter's information

Weihu Hong, Ph.D.
Department of Mathematics
Clayton College & State University
5900 North Lee Street, Morrow, GA 30260
Phone: 770-961-3620
Fax: 770-960-5135
Email: weihuhong@mail.clayton.edu

Title

RSA Cryptosystem and Its Applications

Description

RSA cryptosystem is one of the well-known public key cryptosystems in the world. It can be used to secure communications by using the public keys as well as to encrypt electronic files by keeping the public key private. We will discuss how it works and how to make it work.

Abstract

RSA cryptosystem is perhaps one of the most interesting applications of elementary number theory. It is based on Fermat's Little Theorem and the Chinese Remainder Theorem. In order to visualize the process in classroom, we construct a useful implementation of the cryptosystem by creating an invertible process: converting each file to a string of digits, then slice and dice the string into small sub-strings, cipher each of the sub-strings, concatenate them into a string, and then convert the result into a string of characters as an encrypted file. To decipher the file, we simply reverse the process. The implementation can be applied to encrypt electronic files such as student records, email, and other classified documents.

RSA Cryptosystem and Its applications

Weihu Hong
Department of Mathematics, Clayton College & State University, Morrow, GA 30260, USA

Abstract

RSA cryptosystem is perhaps one of the most interesting applications of elementary number theory. It is based on Fermat's Little Theorem and the Chinese Remainder Theorem. In order to visualize the process in classroom, we construct a useful implementation of the cryptosystem by creating an invertible process: converting each file to a string of digits, then slice and dice the string into small sub-strings, cipher each of the sub-strings, concatenate them into a string, and then convert the result into a string of characters as an encrypted file. To decipher the file, we simply reverse the process. The implementation can be applied to encrypt electronic files such as student records, email, and other classified documents.

Keywords: RSA Cryptosystem, encrypt, decipher, plaintext, private key, public key.

1. Introduction In this information age, we are facing many challenges. One of the most serious ones is the increase in identity theft. In the following, we introduce the RSA cryptosystem and demonstrate a way to construct an implementation. The result can be applied to encrypt electronic files such as student records, email, and other documents.

2. The Process A plaintext file can be viewed as a long string of characters. The process is as shown in the figure 1. To encrypt a plaintext file f, the function T converts f into a long string of digits, namely, T(f). The function E encrypts the file, namely, E(T(f)). The inverse function of T converts the encrypted file into a text file, namely, T^{-1}(E(T(f)). To decrypt the file, it reverses the procedure. The function T converts the file T^{-1}(E(T(f)) back to the file E(T(f)). Then the decryption function E^{-1} converts the file E(T(f)) back to the file T(f). Finally the inverse function of T converts the file T(f) back to the original plaintext file f.

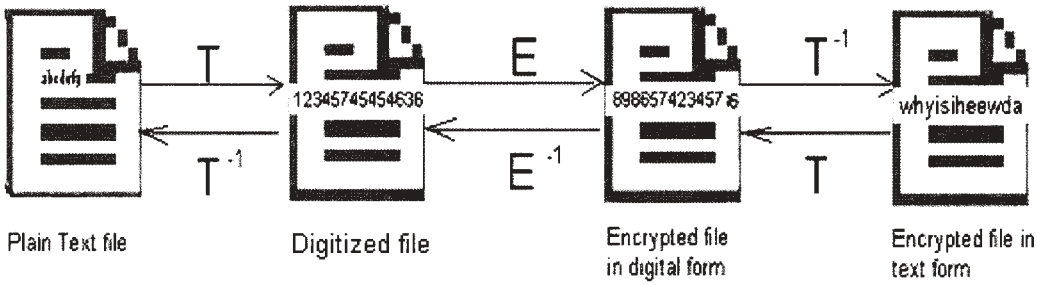

Figure1 an invertible process

In order to insure the security of an encryption, one has to find a good encryption algorithm E such that its inverse E^{-1} is not easily broken. There are many encryption algorithms available such as Diffie-Hellman Key Exchange, Massey-Omura Encryption, ElGamal Public Key Encryption[6], RSA Public Key Encryption, and Rijndael Encryption Algorithm[2],[4],[5]. Since RSA Public Key Encryption relies on elementary number theory, it has been a hot topic within mathematical community. In the following, we show why it works and how we can implement it as a private key cryptosystem.

3. RSA Cryptosystem In 1976, three researchers at MIT, Ronald Rivest, Adi Shamir, and Leonard Adleman, introduced a public key cryptosystem known as the RSA cryptosystem (from the initials of its inventors)[3]. The RSA cryptosystem is based on modular exponentiation. The modulus is the product of two large primes $n = p \cdot q$.

Each individual has an encryption key consisting of the modulus n and an exponent e that is relatively prime to the number (p-1)(q-1). We must produce a usable key. First two large primes must be found. This can be done quickly on a computer using probabilistic primality tests. We must also in sure that the product of these primes cannot be factored in a reasonable length of time. This is why decryption cannot be done efficiently without a separate decryption key.

The encryption function E is given by the following formula

$$C = E(M) = M^e \bmod n$$

Where M is the integer representing a block of characters. The decryption function E^{-1} is the inverse of the encryption function E given by

$$M = E^{-1}(C) = C^d \bmod n$$

Where C is the integer representing a block of encrypted message, and d is the decryption exponent. The **public** key is (e, n), which can be published like a phone number. The **private** key is (d, n), which must be kept in a safe place.

Note: the encryption exponent e and the decryption exponent d are inverses of each other modulo (p-1)(q-1), namely, $de \equiv 1 \bmod((p-1)(q-1))$. In order to find the decryption exponent d one has to factor the integer n, which is very hard if the integer n is very large.

To understand how the RSA cryptosystem works, let us look at the following example.

Example 1. Let $p = 149, q = 151$, then $n = p \cdot q = 22499$, and $e = 43$. Note that $\gcd(e,(p-1)(q-1)) = \gcd(43,148 \cdot 150) = 1$. Since $d = 12907$ satisfies the formula $de \equiv 1 \bmod((p-1)(q-1))$, it can be the decryption exponent. If the message is "MEET ME", then we might translate each of the characters including the space in the message into its equivalent ASCII code and then make it into a string of length that is equal to the number of digits of n (that is 5 in this case) by padding enough zeros in front as necessary. We concatenate these strings of equal length into one big string in the natural order. We obtain, for instance using the built-in function Convert.ToInt32(s) in C#, the string 00077000690006900084000320007700069.
We encrypt each sub string of length 5 using the encryption function E and write each result as a string of length 5 by padding enough zeros in front whenever it is necessary, that is,

$$E(00077) = 77^{43} \bmod 22499 = 12018$$

$$E(00069) = 69^{43} \bmod 22499 = 01088$$

$$E(00084) = 84^{43} \bmod 22499 = 15597$$

$$E(00032) = 32^{43} \bmod 22499 = 14266$$

Therefore by concatenating them in the natural order, the encrypted message is 120180108801088155971422612018010 88. We convert this to a text string: □pp□□□p, where the symbol □ stands for a special character. To decrypt the message, first we convert the text string □pp□□□p back to 120180108801088155971422612018010 88. We then decrypt each sub string of length 5 using the function E^{-1} and write each result as a string of 5 by padding enough zeros in front as necessary, that is,

$$E^{-1}(12018) = 12018^{12907} \bmod 22499 = 00077$$

$$E^{-1}(01088) = 1088^{12907} \bmod 22499 = 00069$$

$$E^{-1}(15597) = 15597^{12907} \bmod 22499 = 00084$$

$$E^{-1}(14226) = 14266^{12907} \bmod 22499 = 00032$$

Thus, by concatenating them in the natural order, the message is 000770006900069000840003200077000 69. We translate each sub string of length 5 into character by using the built-in function Convert.ToChar(s) in C# and recover the original message "MEET ME". Since we make each sub string of equal length, it is easy to loop. It is obvious that the length has to be at most the number of digits of n.

The reason this works is that it is a result of the Fermat's Little Theorem, the Chinese Remainder Theorem, and the congruence theory. We state both theorems in the following. You can find a proof in any textbook [1] of elementary number theory.

Fermat's Little Theorem. If p is a prime and $\gcd(p,a)=1$, then $a^{p-1} \equiv 1 (\mathrm{mod}\, p)$.

Chinese Remainder Theorem. Let $n_1, n_2, \ldots, n_r$ be positive integers such that $\gcd(n_i, n_j) = 1$ for $i \neq j$. Then the system of linear congruences

$$\begin{aligned} x &\equiv a_1 (\mathrm{mod}\, n_1), \\ x &\equiv a_2 (\mathrm{mod}\, n_2), \\ &\vdots \\ x &\equiv a_r (\mathrm{mod}\, n_r), \end{aligned}$$

has a simultaneous solution, which is unique modulo the integer $n_1 n_2 \cdots n_r$.

We are ready to show why RSA system works as a cryptosystem.

RSA Theorem. Let p and q be different primes and e be a positive integer such that $\gcd(e,r)=1$, and d be a positive integer such that $de \equiv 1 \mathrm{mod}(r)$, where $r=(p-1)(q-1)$. Let M be a whole number such that $M < pq = n$. If $C = E(M) := M^e \bmod n$, then $M = E^{-1}(C) = C^d \bmod n$.

Proof: Since $de \equiv 1 \mathrm{mod}((p-1)(q-1))$, there is an integer k such that $de = 1 + k(p-1)(q-1)$. It follows that

$$C^d \equiv \left(M^e\right)^d = M^{ed} = M^{1+k(p-1)(q-1)} (\mathrm{mod}\, n).$$

First, we consider the case that $\gcd(M,p) = \gcd(M,q) = 1$. It follows from **Fermat's Little Theorem** that $M^{p-1} \equiv 1 (\mathrm{mod}\, p)$ and $M^{q-1} \equiv 1 (\mathrm{mod}\, q)$. Thus,

$$C^d \equiv \left(M^e\right)^d = M^{ed} = M^{1+k(p-1)(q-1)} \equiv M \cdot \left(M^{p-1}\right)^{k(q-1)} \equiv M \cdot 1 \equiv M (\mathrm{mod}\, p)$$

and

$$C^d \equiv \left(M^e\right)^d = M^{ed} = M^{1+k(p-1)(q-1)} \equiv M \cdot \left(M^{q-1}\right)^{k(p-1)} \equiv M \cdot 1 \equiv M (\mathrm{mod}\, q)$$

Since $\gcd(p,q)=1$, it follows from the **Chinese Remainder Theorem** that $C^d \equiv M (\mathrm{mod}\, pq)$.

Now let us consider the case that *either* $\gcd(M,p) > 1$ *or* $\gcd(M,q) > 1$. Since $M < pq$, it cannot be both. Without loss of generality, let us assume that $\gcd(M,p) > 1$ *and* $\gcd(M,q) = 1$. As before, it follows from **Fermat's Little Theorem** that $C^d \equiv M (\mathrm{mod}\, q)$. Since $\gcd(M,p) > 1$, M must be a multiple of p. Thus, $M \equiv 0 (\mathrm{mod}\, p)$. It follows that

$$C^d \equiv \left(M^e\right)^d = M^{ed} = M^{1+k(p-1)(q-1)} \equiv M \cdot \left(M^{p-1}\right)^{k(q-1)} \equiv M \cdot 0 \equiv M (\mathrm{mod}\, p)$$

that is, $C^d \equiv M \pmod{p}$. Therefore, it follows again from the **Chinese Remainder Theorem** that $C^d \equiv M \pmod{pq}$. Hence, $M = C^d \bmod n$. The proof is complete.

When the RSA cryptosystem is used in communication, for instance, between Alice and Bob, it works as follows. When Bob wants to send a file to Alice, he finds Alice's public key (*e*, *n*) in the directory. He encrypts the file *M* by using the formula $C = E(M) = M^e \bmod n$. He sends the encrypted file *C* to Alice. When Alice gets the file, she deciphers the file using her private key (*d*, *n*) and the formula $M = E^{-1}(C) = C^d \bmod n$.

When the RSA cryptosystem is used privately, the public key becomes private. In order to simplify the management of keys, it is necessary to build a procedure that automatically recovers the decryption key (*d*, *n*) once users enter the encryption key (*e*, *n*). Therefore, users need only remember the encryption key (*e*, *n*). We can implement a RSA cryptosystem to protect electronic files such as student records, email, and other documents.

4. An Implementation As mentioned above, we are going to keep the public key (e,n) private. Therefore we deal with small prime numbers only. Thus, we need only remember the key (e,n) and let the system to solve the other key (d,n) before decryption. To avoid any technical issues of programming, we are not going to show any actual code but focusing on the ideas. The implementation is shown as in the figure 2.

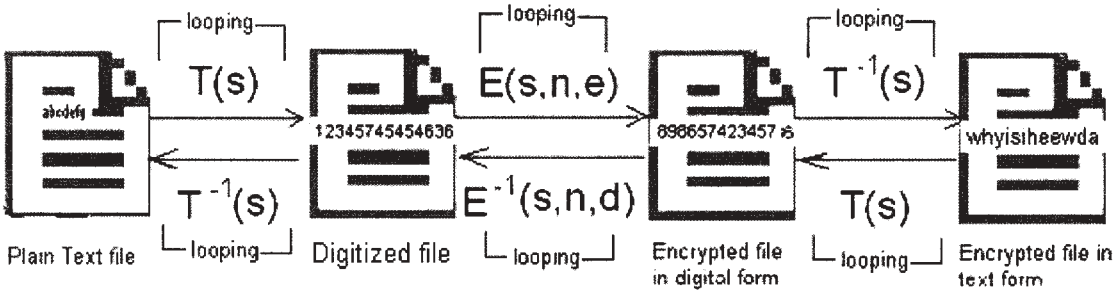

Figure2 an implementation process

To implement $T(s)$, we simply apply a built-in conversion function that converts each character to its correspondent ASCII code, for instance, the built-in function in C# is Convert.ToInt32(s).

To implement the inverse function $T^{-1}(s)$, we apply a built-in conversion function that converts each integer to its correspondent character, for instance, the built-in function in C# is Convert.ToChar(s).

To implement the encryption function $E(s,n,e)$, we implement the formula $C = E(M) = M^e \bmod n$ by using a built-in modulo function (for instance % is the function in C#) and looping through each integer bounded by e. We should, however, never calculate the power M^e directly or it might cause a data overflow.

To implement the decryption function $E^{-1}(s,n,d)$, we implement the formula $M = E^{-1}(C) = C^d \bmod n$ similarly. Since we keep the key (e,n) private, we need implement a function $GetKey2(n,e)$ that is used to find the other key (d,n) for decryption. To this end, we need build another function $Factor(n)$ that will factor the integer $n = pq$, which is realistic for small

integers n, and use the factors to solve the equation $de \equiv 1 \bmod((p-1)(q-1))$ for d, which is the decryption exponent. To make sure the validity of a key provided by users, we need build another function
$CheckKey(n,e)$, which is a Boolean type function. Furthermore, we need build another function $GetNewKey()$ that will allow us to make a new key (e,n).

To apply the process, we may use a visual programming tool such as Java and C# to create a user interface, and assemble all the components to make it work.

5. Conclusion RSA cryptosystem is perhaps one of the most beautiful applications of pure mathematics. An implementation of such cryptosystem can be applied to encrypt an electronic file such as student records, email, and classified documents. Since we keep both keys private, the security of an encryption depends on the key (e,n). To enhance the security, we might add some random strings to an encrypted file and remove them before decryption. Since it requires significant amount of calculations, it will take a lot of time to encrypt and decrypt a large file. Therefore, in practice, it works well for small files. If you look for a faster cryptosystem, you might read [**2**], [**4**], [**5**], and [**6**] for solutions.

Acknowledgments. The author thanks Dr. Tony Giovannitti for his many suggestions that improved this note.

References

[1] Burton, David M., 1989, Elementary Number Theory, the 2nd Ed., WCB.

[2] Susan Landau, 2004, Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard, the American Mathematical Monthly, Volume 111, Number 2.

[3] Kenneth H. Rosen, 2002, Discrete Mathematics and Its Applications, the 5th Ed., WCB/McGraw-Hill.

[4] Robinson et all, 2002, Professional C#, the 2nd Ed., Wrox.

[5] Bruce Schneier, 1996, Applied Cryptography, Protocols, Algorithms, and Source Code in C, the 2nd Ed., Wiley.

[6] Lawrence C. Washington, Elliptic Curves, Number Theory and Cryptography, 2003, Chapman & Hall / CRC.