LAY NETWORKS

MCA – I I I Semister

CS-13

RSA Algorithm

MCA(6)

By Kartik Shah

You can send your contributions to contribute@laynetworks.com

The RSA Algorithm

Encryption is the act of encoding text so that others not privy to the decryption mechanism (the "key") cannot understand the content of the text. Encryption has long been the domain of spies and diplomats, but recently it has moved into the public eye with the concern of the protection of electronic transmissions and digitally stored data. Standard encryption methods usually have two basic flaws: (1) A secure channel must be established at some point so that the sender may exchange the decoding key with the receiver; and (2) There is no guarantee who sent a given message. Public key encryption has rapidly grown in popularity (and controversy, see, for example, discussions of the Clipper chip on the archives given below) because it offers a very secure encryption method that addresses these concerns.

In a classic cryptosystem in order to make sure that nobody, except the intended recipient, deciphers the message, the people involved had to strive to keep the key secret. In a public-key cryptosystem. The public key cryptography solves one of the most vexing problems of all prior cryptography: the necessity of establishing a secure channel for the exchange of the key.

The RSA algorithm, named for its creators Ron Rivest, Adi Shamir, and Leonard Adleman, is currently one of the favorite public key encryption methods. Here is the algorithm:

1. Choose two (in practice, large 100 digit) prime numbers ***p*** and ***q*** and let ***n*** = ***pq***.
2. Let P_i be the block of (plain) text to be encrypted. Actually P_i is the numerical equivalent of the text which may either be single letters or blocks of letters, just as long as $P_i < (p-1)(q-1) = \phi(n)$.
3. Choose a random value ***E*** (usually small) such that ***E*** is relatively prime to $\phi(n)$. Then the encrypted text is calculated from

$$C_i \equiv P_i^E \bmod(n).$$

   The pair of values (***n***,***E***) act as the public key.

4. To decode the ciphertext, we need to find an exponent ***D***, which is known only to the person decoding the message, such that

$$DE \equiv 1 \bmod((p-1)(q-1)).$$

   Note that $\phi(n) = \phi(pq) = (p-1)(q-1)$. Then we may calculate

$$C_i^D = (P_i^E)^D = P_i^{DE} \equiv P_i \bmod(n).$$

This step is based on the following result:

$$(a^x)^y = a^{xy} \equiv a^z \bmod(n),$$

where $z \equiv xy \bmod(\phi(n))$. Show that this result is true.

(Since $xy \equiv z \bmod(n)$, then $xy = m\phi(n) + z$ for some integer ***m***. Thus, applying Euler's theorem we have
$a^{xy} = a^{m\phi(n)+z} = (a^{\phi(n)})^m a^z \equiv (1)^m a^z = a^z \bmod(n).$).

By Euler's theorem

$$E^{\phi(\phi(n))} \equiv 1 \bmod(\phi(n))$$

provided ***E*** and $\phi(n)$ are relatively prime, which is true by the choice of ***E***. So we obtain

$$DE \equiv 1 \bmod(\phi(n)),$$

$$DE \equiv E^{\phi(\phi(n))} \bmod(\phi(n)),$$

$$D \equiv E^{\phi(\phi(n))-1} \bmod(\phi(n)).$$

Therefore, we have an equation that can be used to find the "key" exponent ***D***. The central result of the RSA algorithm is that this equation is computationally solvable in polynomial time (actually using the Euclidean Algorithm) whereas solving by factoring ***n*** requires exponential computational time. [Note however that this last statement has never actually been proven but only demonstrated given today's algorithms. Should someone discover an algorithm that factors integers in polynomial time, the RSA and similar algorithms could be rendered useless for secure communications.] Central to these calculations is knowing the factorization of ***n***, since we will need to calculate both $\phi(n)$ and $\phi(\phi(n))$.

Example

Suppose we wish to encode the plaintext message $P_i = 3$ (that is, under our encoding some letter has been assigned the numerical value 3) subject to our

choices of ***p***=11, ***q***=17 (thus, ***n***=187) and ***E***=7 (note that 7 is relatively prime to 187.) Then the ciphertext C_i is given by

$C_i = 3^7 = 2187 \equiv 130 \bmod(187)$.

Thus the receiver must decode the message $C_i = 130$. To decode this "message" the receiver must calculate the exponent ***D***. [Note that in this example the factorization of ***n*** is relatively easy, so someone could break the code by factoring ***n*** and calculating ***D***. However, in practice, we could choose ***n*** large so that only we would (theoretically) know the factorization.]

Since $n = 11 \cdot 17$, then $\phi(n) = 10 \cdot 16 = 160$, and $\phi(\phi(n)) = \phi(160) = 64$ [WARNING! WARNING! Will Robinson.] Thus we obtain

$$D = E^{\phi(\phi(n))-1} = 7^{63} \bmod(160).$$

Example: Calculate 7^{63} mod(160).

Why was there a warning in the previous example? If you have been closely examining what has taken place in the RSA algorithm you may have noticed that although we know the factorization of ***n*** (since we choose the prime factors ***p*** and ***q***) and hence $\phi(n) = (p-1)(q-1)$, we may not have an easy time determing $\phi(\phi(n)) = \phi((p-1)(q-1))$, which requires us to know all the factors of what could be a very large number. This seems to contradict the polynomial time needed to solve for the key. The solution is (and is a key -- unintentional pun -- element of the RSA algorithm) that the formula for ***D***, although concise, is *not* the way the solution is found in practice. The actual method of solution (which *does* require polynomial time computation) is based on the Euclidean algorithm.

Returning to our previous example, recall that we want to solve

$$DE \equiv 1 \bmod(160),$$

$$7D \equiv 1 \bmod(160).$$

By our choice of ***E***, 7 and 160 are relatively prime, and thus

$$160 = 7(22) + 6$$

$$7 = 1(6) + 1$$

using the Euclidean algorithm. Working in reverse gives

$$1 = 7 - 1(6)$$

$$1 = 7 - 1(160 - 22(7))$$

$$1 = 23(7) - 1(160)$$

$$1 = 23(7) + (-1)(160).$$

In algebraic terms, we say we have written 1 as a *linear combination* of 7 and 160. Since 160 is the modulus, we have

$$(23)(7) \equiv 1 \bmod(160).$$

Hence ***D***=23! Thus the real key to the solution of ***D*** is knowing $\phi(n)$which requires the knowledge of the factorization of ***n*** since $\phi(n) = (p-1)(q-1)$.

A Simple explanation of RSA Algorithm in view to computer :

Let *p* and *q* be distinct large primes and let *n* be their product. Assume that we also computed two integers, *d* (for decryption) and *e* (for encryption) such that

d * e ≡ 1 (mod ø(n))

where *ø(n)* is the number of positive integers smaller than *n* that have no factor except 1 in common with *n*

The integers *n* and *e* are made public, while *p*, *q*, and *d* are kept secret.

Let *m* be the message to be sent, where m is a positive integer less than and relativley prime to *n*. A plaintext message is easily converted to a number by using either the alphabet position of each letter (a=01, b=02, ..., z=26) or using the standard ASCII table. If necessary (so that *m*<*n*), the message can be broken into several blocks.

The encoder computes and sends the number

m' = *m*^*e* mod *n*

To decode, we simply compute

e^*d* mod *n*

Now, since both *n* and *e* are public, the question arises: can we compute from them *d*? The answer: it is possible, if n is factored into

prime numbers.

The security of RSA depends on the fact that it takes an impractical amount of time to factor large numbers.