Public Key Encryption

(RSA algorithm)

The objective is for a business to send two numbers—A and N—to your computer for it to encrypt your credit card number a and send the encrypted information x back to them. Anyone intercepting the encryption numbers A and N will not be able to use them to decrypt the information x you've sent. But the business has a number B it keeps secret to decrypt your information.

We've seen a process that acts on a number a to give back the same number a:

$$a^n \bmod pq = a \bmod pq$$

where a is the information to be sent,

$n = m\cdot(p-1)(q-1)/2 + 1$,

p and q are primes,

and m = some integer.

The public key encryption scheme is to divide the operation a^n into two parts. If we factor n into two parts $n = A\cdot B$, then $a^n = a^{A\cdot B} = (a^A)^B$, then we can raise a to the power n in two steps:

Let $N = pq$.

Encryption: $x = a^A \bmod N$

Decryption: $y = x^B \bmod N$

$= (a^A)^B \bmod N$

$= a^{A\cdot B} \bmod N$

$= a^n \bmod N$

$= a \bmod N$

$= a$ (if $a < N$)

To assure that $a < N$, the information to be encrypted must be split into numbers a that are less than N.

Example

$a = 116$ (information to be sent)

$p = 11$, $q = 13$ (primes)

$N = pq = 143$ (one part of both the encryption and decryption keys)

$m = 8$ (integer chosen so n is factorable)

$n = m\cdot(p-1)(q-1)/2 + 1 = 8\cdot(10)(12)/2 + 1 = 481$

$n = 13\cdot 37$ (factorization)

$A = 13$ (the other part of the encryption key)

$B = 37$ (the other part of the decryption key)

$x = a^A \bmod N = 116^{13} \bmod 143 = 51$ (encrypted information)

$y = x^B \bmod N = 51^{37} \bmod 143 = 116$ (decrypted information, returning the original a)

Suppose we want to send the message, "Help! I'm hurt." Each letter or symbol can be represented by 7 bits using the ASCII code:

H	e	l	p	!	(space)	I	'	m	(space)	h	u	r	t	.
072	101	108	112	033	032	073	039	109	032	104	117	114	116	046

Since $N = 143$ is greater than the largest ASCII code (127), the encryption can be done symbol by symbol, each a equaling its ASCII code. Using $A = 13$ and $N = 143$, we encrypt the series of a's to produce a series of x's:

007 140 069 008 033 032 112 117 109 032 026 013 075 051 085

Since the x's can be as large as $N - 1 = 142$, eight bits must be used to represent each x. The encrypted message sent in binary looks like this:

00000111|10001100|01000101|00001000|00100001|00100000|01001001|00100111|
01101101|00100000|00011010|00001101|01001011|00110011|01010101

(The vertical lines demarcate the x's.) At the receiving end, the bits are grouped by eight, and the xs are decrypted using B and N to recover the the ys (which are the original message of as.)

Security

If a hacker intercepts the encryption keys N and A and the encrypted message $x = a^A \bmod N$, why can't he just reverse the process to find a? It would be easy if mod N were not part of the process. For example, take $a = 127$ and $A = 13$, and suppose $x = a^A = 223587938856003706253977356 7$. Then a simple A-root algorithm could reverse the process to find $a = 127$. But $x = a^A$ **mod N**, yielding $x = 62$. The mod N operation has removed too much information, and the process can't be reversed. The hacker's only hope is to build a table of $x = a^A \bmod N$ for all possible a's and then look up the a corresponding to each x. It would be easy to build such a table for the simple example above because $N = 143$ is so small; there are only 142 entries to generate in the table. But if N is on the order of 10^{25}, it would take the age of the universe to generate the table.

There is another reason to make N large. The hacker can easily factor the small $N = 143$ into $p = 11$ and $q = 13$, try m's until he finds one for which $n = m \cdot (p - 1)(q - 1)/2 + 1$ has $A = 13$ as a factor (such as $m = 8$ gives $n = 481 = 13 \times 37$). Then $B = N/A = 481/13 = 37$. He now has the N and B to decrypt the message x. So the success of the scheme depends on N being so large that p and q can't be recovered by factorization in less than 10 years of computing time. It is recommended that N be on the order of 10^{616} (which requires 2048 bits to express it).

Practical Encryption-Decryption Computation

Encryption requires raising a large number a (such as 10^{500}) to a large power A (such as 10^{150}) and taking mod N. But a^A is a tremendous number that exceeds the memory capacity of all the computers in the world. In practice, the operation a^A is broken down into a number of steps which each result in a number small enough for a computer to handle. After each step (addition or multiplication) the result is reduced in size by the operation mod N. This operation has no effect on the final encryption because the last step is to take mod N. The following examples illustrate this.

Addition

Suppose we want to find mod 100 of the sum of two numbers:

$$(5360783 + 298864) \bmod 100 = 5659647 \bmod 100 = 47.$$

(Note that in the decimal number system, mod 100 is just taking the last two digits.)

If we take mod 100 before the addition (as well as after), the result is unaffected:

$$(5360783 \bmod 100 + 298864 \bmod 100) \bmod 100$$
$$= (83 + 64) \bmod 100 = 147 \bmod 100 = 47.$$

Multiplication

$$(5360783 \times 298864) \bmod 100 = 1602145050512 \bmod 100 = 12$$

and

$$(5360783 \bmod 100 \times 298864 \bmod 100) \bmod 100$$
$$= (83 \times 64) \bmod 100 = 5312 \bmod 100 = 12.$$

Powers

To take advantage of using mod N after each multiplication, we can implement a^A by multiplying A number of a's together:

$a^A = a \times a \times a \times a \times \cdots \times a.$ (There are $A - 1$ multiplications.)

Then

$$a^A \bmod N = (a \times a \times a \times a \times \cdots \times a) \bmod N$$
$$= [(\cdots [([([(a \times a) \bmod N] \times a) \bmod N] \times a) \bmod N] \times \cdots \times a) \bmod N] \bmod N$$

The numbers are now small enough that the computer can handle them. But the number of multiplications is $A - 1$, which can be as large as 10^{150}, and this would take much longer than the age of the universe. But computers use a more efficient algorithm to calculate the power a^A than simply multiplying A number of a's together.

Consider first the problem of *multiplying* two numbers by hand. To multiply 376×235 we could add 376 to itself 235 times. But it's quicker to use "long multiplication," which uses the fact that $235 = 2 \cdot 100 + 3 \cdot 10 + 5$. Then

$$376 \times 235 = 376 \times 235 = 376 \times (2 \cdot 100 + 3 \cdot 10 + 5)$$
$$= 37 \times 2 \cdot 100 + 376 \times 3 \cdot 10 + 376 \times 5$$
$$= 752 \cdot 100 + 1128 \cdot 10 + 1880$$

Now, multiplying by 10 is simply shifting to the left one place, and multiplying by 100 is shifting to the left *two* places.

$$\begin{array}{r} 376 \\ \times 235 \\ \hline 1880 \\ 1128 \\ \underline{752\ \ .} \\ 88360 \end{array}$$

In a similar way we can calculate a *power* by using the algorithm

$$376^{235} = 376^{2\cdot 100 + 3\cdot 10 + 5} = (376^{2\cdot 100})(376^{3\cdot 10})(376^5)$$
$$= (376^{100})^2\,(376^{10})^3\,(376)^5$$

In the binary number system, which computers use, the algorithm is even simpler; we can get rid of the powers 2, 3, and 5 in the above equation.

$$376_{10} = 101111000_2 \qquad 235_{10} = 11101011_2$$

$$376^{235}\,(\text{base } 10) = 101111000^{11101011}\ \ (\text{base } 2)$$
$$= 101111000^{10000000 + 1000000 + 100000 + 0 + 1000 + 0 + 10 + 1}$$
$$= (101111000^{10000000})\cdot(101111000^{1000000})\cdot(\,101111000^{100000})\cdot$$
$$(101111000^{1000})\cdot(101111000^{10})\cdot(101111000) \qquad \text{(Eq.1)}$$

Note that binary 101111000^{10} is 101111000 squared, and 101111000^{100} is 101111000^{10} squared, etc. The factors in Eq.1 are calculated sequentially:

$$\left.\begin{array}{l} 101111000 = 101111000 \\ 101111000^{10} = 101111000\times 101111000 \\ 101111000^{100} = 101111000^{10}\times 101111000^{10} \\ 101111000^{1000} = 101111000^{100}\times 101111000^{100} \\ 101111000^{10000} = 101111000^{1000}\times 101111000^{1000} \\ 101111000^{100000} = 101111000^{10000}\times 101111000^{10000} \\ 101111000^{1000000} = 101111000^{100000}\times 101111000^{100000} \\ 101111000^{10000000} = 101111000^{1000000}\times 101111000^{1000000} \end{array}\right\} \quad \text{(Eqs.2)}$$

mod *N*

Now we want to calculate 376^{235} **mod *N***, where $N = 437$. Then the factors in Eq.1 (given in Eqs.2) can be kept small by taking mod N before their multiplication:

$101111000 \bmod N = 376_{10}$
$101111000^{10} \bmod N = (376_{10}\times 376_{10}) \bmod N = 141376_{10} \bmod N = 225_{10}$
$101111000^{100} \bmod N = (225_{10}\times 225_{10}) \bmod N = 50625_{10} \bmod N = 370_{10}$
$101111000^{1000} \bmod N = (370_{10}\times 370_{10}) \bmod N = 136900_{10} \bmod N = 119_{10}$
$101111000^{10000} \bmod N = (119_{10}\times 119_{10}) \bmod N = 14161_{10} \bmod N = 177_{10}$
$101111000^{100000} \bmod N = (177_{10}\times 177_{10}) \bmod N = 31329_{10} \bmod N = 302_{10}$
$101111000^{1000000} \bmod N = (302_{10}\times 302_{10}) \bmod N = 91204_{10} \bmod N = 308_{10}$
$101111000^{10000000} \bmod N = (308_{10}\times 308_{10}) \bmod N = 94864_{10} \bmod N = 35_{10}$

Then the mod N operation on Eq.1 gives

$376^{235} \bmod N = (35 \cdot 308 \cdot 302 \cdot 119 \cdot 225 \cdot 376) \bmod N$

The five products here are also kept small by taking mod N after each multiplication:

$$\begin{aligned}
376^{235} \bmod N &= ([([([([(35 \cdot 308) \bmod N] \cdot 302) \bmod N] \cdot 119) \bmod N] \cdot 225) \bmod N] \cdot 376) \bmod N \\
&= ([([([([10780 \bmod N] \cdot 302) \bmod N] \cdot 119) \bmod N] \cdot 225) \bmod N] \cdot 376) \bmod N \\
&= ([([([(292 \cdot 302) \bmod N] \cdot 119) \bmod N] \cdot 225) \bmod N] \cdot 376) \bmod N \\
&= ([([([88184 \bmod N] \cdot 119) \bmod N] \cdot 225) \bmod N] \cdot 376) \bmod N \\
&= ([([(347 \cdot 119) \bmod N] \cdot 225) \bmod N] \cdot 376) \bmod N \\
&= ([([41293 \bmod N] \cdot 225) \bmod N] \cdot 376) \bmod N \\
&= ([(215 \cdot 225) \bmod N] \cdot 376) \bmod N \\
&= ([48375 \bmod N] \cdot 376) \bmod N \\
&= (305 \cdot 376) \bmod N \\
&= 114680 \bmod N \\
&= 186
\end{aligned}$$

The A, B, and N used here are to small to for good security. Suppose (more realistically) that $A = 10^{150} = 1.222 \times 2^{498}$, which has 499 bits. Then there would be less than 500 multiplications in Eq.1 (not all the bits in A are 1). As well, there would be 498 multiplications for Eqs.2. But handling this number of multiplications is no problem for a computer.