Introduction

RSA is a public key cryptosystem for both encryption and authentication; it was given by three scientists viz. Ron Rivest, Adi Shamir and Leonard Adleman. This algorithm is much secure than any other algorithm. The latest key size used for this encryption technique is 512 bits to 2048 bits.
With the advent of computers it has become possible to perform computations at teraflop speeds so such algorithms could easily be cracked. But RSA encryption uses the concept of two large prime numbers, such that, their product could not be easily factorized. Let us see how this algorithm works and understand its implementation using Java Script.

Mathematical Background

Modular Arithmetic

RSA uses modular arithmetic. This is similar to conventional arithmetic, but only uses positive integers that are less than a chosen value, called the modulus. Addition, subtraction and multiplication work like regular maths, but there is no division. You can use any value for the modulus; the diagram uses 13, so counting goes 0, 1, 2, ..., 11, 12, 0, 1, 2 ... The notation used for expressions involving modular arithmetic is:

x = y (mod m)

Which reads as "*x* is equivalent to *y*, modulo *m*". What this means is that *x* and *y* leave the same remainder when divided by *m*. For example, *7 = 23 (mod 8)* and *22 = 13 (mod 9)*. The following statement is a basic principle of modular arithmetic:

a + kp = a (mod p)

You can visualize this on the diagram - each time you add *p* you go round the circle, back to where you started. It doesn't matter where you start, how big the circle is, or how many times you do it, it's always true.

Primality and Coprimality

- A number is prime if the only numbers that exactly divide it are 1 and itself. e.g. 17 is prime, but 15 isn't, because it's divisible by 3 and 5.
- A pair of numbers are coprime if the largest number that exactly divides both of them is 1. The numbers themselves don't have to be prime. e.g. 8 and 9 are coprime, but 8 and 10 are not, because they're both divisible by 2.
- If you have a pair of distinct prime numbers, they will always be coprime to each other.

Chinese Remainder Theorem

This theorem provides a way to combine two modular equations that use different moduli.

Theorem <> < (mod
%TD_OPT%>x="y">

<> <> < q and p %TD_OPT%>with>

⇒ x = y (mod pq)

<>

Proof <> < (mod %TD_OPT%>x="y">

⇒ x = y + kp

⇒ x - y = kp

⇒ p divides (x - y)

<> by a similar route, q divides (x - y)

<> as p and q are coprime, pq divides (x - y)

⇒ x - y = l(pq)

⇒ x = y (mod pq)

Fermat/Euler Theorem

This theorem is a surprising identity that relates the exponent to the modulus.

Theorem

<> $^{p-1}$ = 1 (mod p)
if p is prime and x ≠0 (mod p)

<>

Proof <> < ... 2, 1, numbers of Q, set the %TD_OPT%>consider>

<>

as p is prime, these numbers are coprime to p

<> 0 is not coprime to p

Q includes all the numbers in (mod p) coprime to p

<>

<> now consider the set U, obtained by multiplying each element of Q by x (mod p)

<> < to coprime are Q element each x %TD_OPT%>both>

each element of U is coprime to p

<>

<> < by prove we which distinct, is U %TD_OPT%>also,>

<> < not elements two assuming %TD_OPT%>start>

<> $_i$ = xQ_j (mod p) with i ≠j

$Q_i = Q_j$ (mod p) as x ≠0

<> < distinct, is elements a this so %TD_OPT%>but>

elements of U are distinct

<>

<> < like just p, that p) in all uses %TD_OPT%>so,>

U is a permutation of Q

$U_1.U_2 ... U_{p-1} = Q_1.Q_2 ... Q_{p-1} \pmod p$

$xQ_1.xQ_2 ... xQ_{p-1} = Q_1.Q_2 ... Q_{p-1} \pmod p$

<> $_1.Q_2 ... Q_{p-1}$

$x^{p-1} = 1 \pmod p$

Using the code

This implementation of RSA uses 32 bit key. There are two files: *input.htm* and *output.htm*. The code for the *input.htm* file is as follows:

```
<html>

<head>
<title>Input</title>

<script language="JavaScript">
<!-- hide from old browsers
function gcd (a, b)
{
  var r;
  while (b>0)
  {
    r=a%b;
    a=b;
    b=r;
  }
  return a;
}

function rel_prime(phi)
{
  var rel=5;

  while (gcd(phi,rel)!=1)
    rel++;
  return rel;
}
```

```
function power(a, b)
{
   var temp=1, i;
   for(i=1;i<=b;i++)
      temp*=a;
    return temp;
}

function encrypt(N, e, M)
{
   var r,i=0,prod=1,rem_mod=0;
   while (e>0)
   {
      r=e % 2;
      if (i++==0)
         rem_mod=M % N;
      else
         rem_mod=power(rem_mod,2) % N;
      if (r==1)
      {
         prod*=rem_mod;
         prod=prod % N;
      }
      e=parseInt(e/2);
   }
   return prod;
}

function calculate_d(phi,e)
{
   var x,y,x1,x2,y1,y2,temp,r,orig_phi;
   orig_phi=phi;
   x2=1;x1=0;y2=0;y1=1;
   while (e>0)
   {
      temp=parseInt(phi/e);
      r=phi-temp*e;
      x=x2-temp*x1;
      y=y2-temp*y1;
      phi=e;e=r;
      x2=x1;x1=x;
      y2=y1;y1=y;
```

```
    if (phi==1)
    {
      y2+=orig_phi;
      break;
    }
  }
  return y2;
}

function decrypt(c, d, N)
{
  var r,i=0,prod=1,rem_mod=0;
  while (d>0)
  {
    r=d % 2;
    if (i++==0)
      rem_mod=c % N;
    else
      rem_mod=power(rem_mod,2) % N;
    if (r==1)
    {
      prod*=rem_mod;
      prod=prod % N;
    }
    d=parseInt(d/2);
  }
  return prod;
}



function openNew()
{
  var subWindow=window.open(
"Output.htm", "Obj","HEIGHT=400,WIDTH=600,SCROLLBARS=YES");
  var p=parseInt(document.Input.p.value);
  var q=parseInt(document.Input.q.value);
  var M=parseInt(document.Input.M.value);
  var N=p * q;
  var phi=(p-1)*(q-1);
  var e=rel_prime(phi);
  var c=encrypt(N,e,M);
  var d=calculate_d(phi,e);
  subWindow.document.Output.N.value=N;
```

```
  subWindow.document.Output.phi.value=phi;
  subWindow.document.Output.e.value=e;
  subWindow.document.Output.c.value=c;
  subWindow.document.Output.d.value=d;
  subWindow.document.Output.M.value=decrypt(c,d,N);
}

// end scripting here -->
</script>

</head>

<body>

<p><font size="6">Input Form</font></p>
<hr>
<form name="Input">
<table border="0" width="100%" height="109">
 <tr>
  <td width="24%" height="23">
    <font color="#0000FF">Enter P</font></td>
  <td width="76%" height="23">
     <input type="text" name="p" size="20"></td>
 </tr>
 <tr>
  <td width="24%" height="23"><font color="#0000FF">
       Enter Q</font></td>
  <td width="76%" height="23">
     <input type="text" name="q" size="20"></td>
 </tr>
 <tr>
  <td width="24%" height="20">
      <font color="#0000FF">Enter any Number ( M )</font></td>
  <td width="76%" height="20"><input type="text" name="M" size="20">
    <font size="1" color="#FF0000">(1-1000)</font></td>
 </tr>
 <tr>
  <td width="24%" height="19"><input type="button"
    value="Submit" name="Submit" onClick="openNew()"></td>
  <td width="76%" height="19"><input type="reset"
     value="Reset" name="Reset"></td>
 </tr>
</table>
```

```
</form>
<p> </p>

</body>

</html>
```

The code for the *output.htm* file is as follows:

```
<html>

<head>
<title>Output</title>

</head>

<body>

<p><font size="6">Output Form</font></p>
<hr>
<p><font color="#FF0000">1.    N = p * q
</font></p>
<p><font color="#FF0000">2.   
phi = ( p - 1 ) * ( q - 1
)    </font></p>
<p><font color="#FF0000">3.    GCD
( phi , e ) = 1</font></p>
<p><font color="#FF0000">4.   
Encrypted Text ( c ) = M<sup>e</sup>
* ( mod N )</font></p>
<p><font color="#FF0000">5.   
e * d =  1 * ( mod phi )</font></p>
<p><font color="#FF0000">6.   
Decrypted Text = c<sup>d</sup> * (
mod N )</font></p>
<form name="Output">
<table border="0" width="100%">
  <tr>
    <td width="22%"><font color="#0000FF">N
      </font></td>
    <td width="78%"><input type="text" name="N"
      size="20"></td>
  </tr>
  <tr>
```

```
    <td width="22%"><font color="#0000FF">Phi</font></td>
    <td width="78%"><input type="text"
      name="phi" size="20"></td>
  </tr>
  <tr>
    <td width="22%"><font color="#0000FF">e
      </font></td>
    <td width="78%">
  <input type="text" name="e" size="20"></td>
  </tr>
  <tr>
    <td width="22%"><font color="#0000FF">Encrypted Text
     </font></td>
    <td width="78%">
    <input type="text" name="c" size="20"></td>
  </tr>
  <tr>
    <td width="22%"><font color="#0000FF">d
      </font></td>
    <td width="78%"><input type="text" name="d" size="20">
      </td>
  </tr>
  <tr>
    <td width="22%"><font color="#0000FF">
      Decrypted Text</font></td>
    <td width="78%"><input type="text" name="M" size="20"></td>
  </tr>
  <tr>
    <td width="22%">
     <input type="button" value="Close" name="Close"
    onClick="window.close()"></td>
    <td width="78%"> </td>
  </tr>
</table>
</form>
<p> </p>

</body>

</html>
```

Now, after creating the files you can run the *input.htm* file from your browser and provide the necessary values. Clicking the Submit button will open the *output.htm* file with the necessary outputs.

Compatibility of Code

The code has been tested on IE 4.0 and above as well as on Netscape Navigator.

Working of RSA Algorithm

Suppose that B wants to send a message to A. A and B have exchanged their public keys. Let us try to understand how this works:

1. Person A selects two prime numbers. Say p = 53 and q = 61.
2. Person A calculates p * q = 3233. This is the public key which he sends to B.
3. Person A calculates the value of e such that GCD ((p – 1) * (q – 1), e) = 1. This is also send to B.
4. Suppose person B wants to send message M = 999 to A.
5. Person B encrypts the message, $c = M^e \pmod N = 999^7 \pmod{3233} = 3026$.
6. Person B sends c to person A.
7. Person A decodes c = 3026. Firstly, he finds d such that e * d = 1 (mod ((p – 1) * (q - 1)).
8. This equation is solved using Extended Euclidean Algorithm. Hence d = 1783.
9. Secondly, person A decodes the encrypted message c using: $c^d \pmod N = 3026^{1783} \pmod{3233} = 999$.

Points of Interest

1. The factors of the public key N, that is, p and q should be large enough so that its not easy to factorize N.
2. In general, the order of the primes should be 160 (512 bits) digits to 640 (2048 bits) digits.
3. No algorithm is available that could factorize a number of the mentioned order in reasonable amount of time.
4. So the RSA algorithm is defended by the non-availability of such algorithms.

Conclusion

1. One has to use brute-force to factorize N.
2. The algorithms to factorize N have a running time exponential with respect to the length of N.
3. Still the existence of a faster algorithm, to factorize N, is very remote.

Further Suggestions

1. Possible attacks on RSA
2. Algorithm to find whether a number is prime or not (less time complexity algorithm).
3. Largest prime numbers in use.

I would suggest that the readers should try to work on these topics so as to learn more about the RSA encryption scheme. I am having the necessary content, but I don't want to complicate the things write now.

Implementations of RSA

- S.S.L. (Secure Sockets Layer)
- Firewalls
- ATM machines
- Digital Signatures
- Certificates