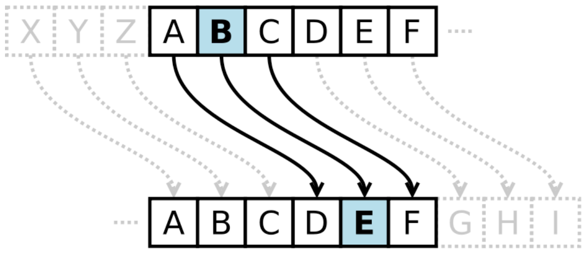
Cryptography & Cryptanalysis

Soumya C. Kambhampati
Science
Period 5
12/17/09

As long as we have been able to communicate, there has been a need for secrecy and encryption. It started simple with shift ciphers, moved on to Vigenère ciphers, and reached the modern-day Public/Private cryptography. Throughout time, there has always been a long never-ending battle between cryptographers and cryptanalysts. In this project, I hope to better understand how this works, how one can write, “The quick brown fox jumps over the lazy dog,” and end up with an unintelligible stream of gibberish: “WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ”, and have it converted it back into the former. I would also like to implement different ciphers in Javascript. I got my information from the site “Crypto Club: Cryptography, the mathematics of secret codes: The Project for Children” [1], a project sponsored by National Science Foundation, and “Codes, Ciphers, & Codebreaking” [2]. I am also reading “The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography” [3] to get additional details. Please note that I have not learned about all of the ciphers: the ones I have a *complete* understanding of are in this paper.

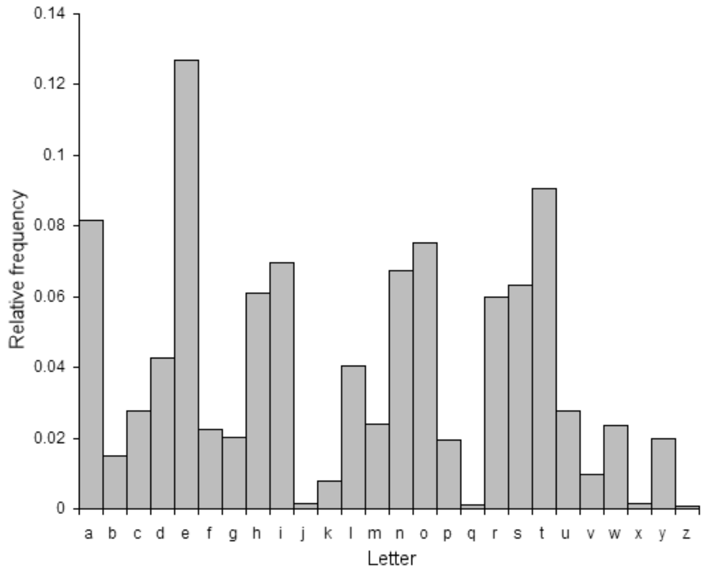
Ciphers & How to Decode Them

The example above is, actually one of the easier ciphers, known as the shift cipher. To encode, one simply moves each letter down fixed positions in the alphabet, as shown in the figure on the right. This can also be



http://en.wikipedia.org/wiki/File:Caesar3.svg

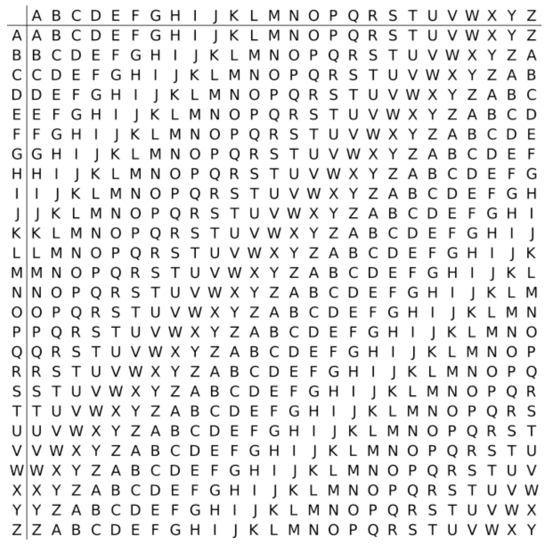
expressed algebraically using *modular arithmetic*. If you think of A-Z in terms of their positions 0-25, and **s** is the shift, then the code for a plain text letter **x** will simply be **(x + s) mod 26**. Here **u mod v** is just the remainder when you divide **u** by **v**. To decipher ciphertext **y**, you do **(y - s) mod 26**.



http://en.wikipedia.org/wiki/File:English-slf.png

To the crack the cipher, you do something called frequency analysis. Some letters in the alphabet are used more than others in text, as is shown on the left. For example the vowel "E" and the consonant "T" are used more frequently in English than "J". The cryptanalyst will match this with the frequency of letters in the message and attempt to fill in obvious letters. Then, the cryptanalyst will guess letters that will make the code into plausible plaintext.

The Vigenère cipher is a more complex version of shift cipher. It uses two things to encode plaintext: a keyword, and a Vigenère square, like the one on the left. It takes 26 alphabets, each one moving one to the left. To encode, you take a keyword and repeat it until it is matching in length with the plaintext. Assuming once again that letters A-Z are numbers 0-25, the code **Ci** for each plain text letter **Pi** with the corresponding keyword letter Ki is simply **Ci = (Pi+Ki) mod 26**.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

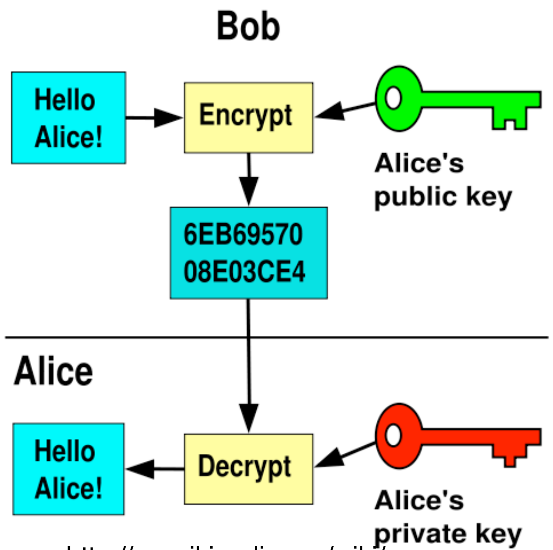
http://en.wikipedia.org/wiki/File:Vigenère_square.svg

This modulus addition can also be done in terms of the Vigenère square shown on the right. For each letter, you find the row with the letter in your keyword and the column with the plaintext letter. On the letter on which the row and column intersect is the ciphertext. For example, suppose you want to encrypt the word "HI" with the keyword "MOM". For the first letter, "H", you would get the first letter in the keyword, "M", and find the "M" row. Then, you would find the column for "H". The encrypted letter would be "T". The entire ciphertext would be "TW".

Cracking the Vigenère cipher is much harder than a simple shift cipher, because it is polysyllabic. That means that the same letters in plaintext can be represented by different letters in ciphertext. So, this renders frequency analysis useless, earning it the nickname “*le chiffre indéchiffrable”*, (French for “the unbreakable cipher”). But, there is one inherent flaw in the Vigenère cipher: if the keyword is guessed, you can quickly unravel the plaintext message. The problem is that it is nearly impossible to guess a keyword. So, one must us the Kasiski test to crack the cipher. The test relies upon the plaintext having repeated words, like “An” that have been encrypted with the same part of the keyword. If they are, you should find repeated segments of ciphertext. Measure the letter distance between the beginning of one segment and the letter before the beginning of the other segment. Find all of the factors of the distance. These are the possible key lengths. Then, repeat this process on other repeated segments of ciphertext. Continue doing this until you narrow the key length down to one number. You put the ciphertext in as many columns as letters in the keyword, with each column corresponding to a letter in the keyword. This effectively breaks the Vigenère cipher into a shift cipher, which can be solved using frequency analysis.

One major shortcoming of both the Vigenère cipher and the Shift cipher was the need for both parties (people who are transmitting encrypted messages) to know the keyword for the message. If the keyword was intercepted, the eavesdropper would know exactly what the message said. NSA and other government agencies used their bottomless budget to deliver the keyword to the recipient via a trustworthy person. This did not make much sense as this ‘trustworthy’ person might sell this keyword to the highest bidder, and if he was trustworthy, it made no sense to send the message separately. To solve this problem, NSA would send ‘pads’ of keywords which were to be used in a certain order, but this meant that a single shady keyword deliverer could mean a serious security breach. Additionally, normal companies could not use this method because it was too expensive. There needed to be a cipher that could be used to exchange messages without the recipient knowing the mailer's keyword.

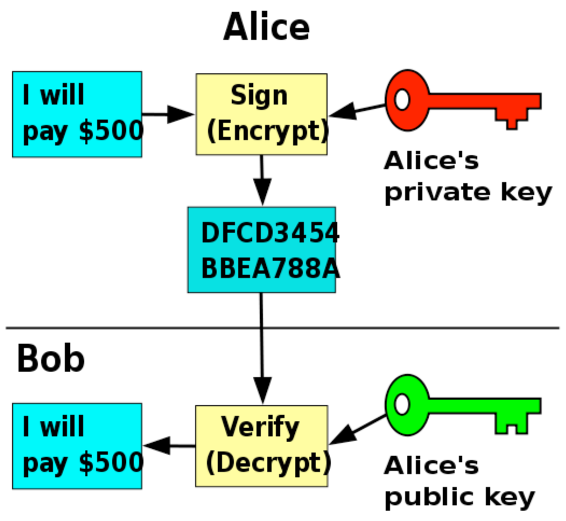
Public-key cryptography is an asymmetric algorithm, so it, unlike symmetrical algorithms, does not require a secure initial exchange of keywords. It can be easily described by this analogy; Bob wants to sent a message to Alice, but does not want anyone to read it, so Bob puts a padlock on the box that contains the message, and sends it to Alice. Alice puts her own padlock onto the box, and sends it back to Bob. Bob uses his key to take off his padlock, and sends it back to Alice. Alice simply has to unlock the box with her key, and then can read the message, without ever receiving a key from Bob.



http://en.wikipedia.org/wiki/File:Public_key_encryption.svg

This entire process is compressed into a single exchange of data. In public-key cryptography, you have a public and private key. Your public key can be seen by anyone, and your private key is secret; only you know can know it. If Bob wants to send a message to Alice, his message with Alice's public key using algorithms that are non-reversible (1+x=2 is reversible because you can find what x is, 1) to prevent a malicious eavesdropper, Eve, from cracking the message. Bob then sends the ciphertext to Alice. Using her private key, Alice is able to decrypt the received message, and read it without Eve being able to decode it.

One issue with public-key cryptography is that Eve can create a public key in Alice’s name, and if Bob uses that public key instead of Alice’s real public key, Alice will be unable to read the message, and Eve will easily be able to decrypt the false public key. To prevent this from happening, certification authorities (trusted 3rd parties) will validate a single public key as Alice’s public key, so Eve cannot trick Bob into using a false public key.



http://en.wikipedia.org/wiki/File:Public_key_signing.svg

Need

On the web, encryption is used constantly to protect private and important data from being read by 3rd parties. There needs to be a program that will easily protect private data in a series of different ways depending on need (private data will be encoded with harder encryptions than unimportant data).

Design Criteria

I will create a website, either from scratch or by expanding my previous website. Inside the website, I will create a program that encodes text using Vigenere, Shift, and Public-Key encryption methods. My website will additionally explain what the program is doing, so that the user understands the basics of what the ciphers are.

Information about the program

	Design Criteria	Now
Functionality	Encrypts text with different methods	Encrypts and Decrypts text with Vigenere, Shift, and RSA.
Capacity	Can work with any kind of text	Can work with any kind of text
UI	Basic	Basic
Customizability	None	Can edit source code if necessary
Speed	Fast	Fast (8-224 kb size) (19-80 ms download time) (Safari)
Communicating with misc Programs	None	Communicates with default mail program
Error Handling	None	Puts a NaN or Undefined when error occurs.
Programming Language	Javascript	Javascript with other elements (CSS, HTML) for the GUI
Portability	Works on all major browsers	Works on all major browsers except IE, with which there is limited compatibility
Language Support	English alphabet only	English alphabet only

Conclusions

In the website, I have added one modern cipher, RSA to my website. (Last year I had implemented ancient ciphers, Shift and Vigenere.) I debugged my GUI code, making it more lean and making it load faster (I did things such as fix tag placements and start/end tags). I was unable to create text boxes to enter text like I was hoping for, and instead used the same prompt boxes as before (entering text using boxes required the text to be sent back to the server, and I was doing strictly client-side (the user does all of the computations) programming). I was able to successfully do large computations in very little time by using Fermat's Little Theorem (exponentiation by successive squaring). I additionally changed my website's layout to make room for RSA. All in all, I believe that this project was a success, as I successfully implemented RSA and smoothly transitioned my website to have space for RSA.

Bibliography

[1] Crypto Club: Cryptography, the mathematics of secret codes: The project for Children. University of Illinois, Chicago. http://cryptoclub.math.uic.edu/indexmain.html

[2] Codes, Ciphers, & Codebreaking. Greg Goebel. http://www.vectorsite.net/ttcode.html

[3] The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography. Simon Singh. Anchor. 2000.

JavaScript Code

(This contains some unnecessary code)

```
<script type="text/javascript">
var alphabet = new Array
("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s",
"t","u","v","w","x","y","z");

function num2let(num)
//converts a number into a letter
{
//document.write("num2let " + num);
return(alphabet[num%26]);
}

function let2num(let)
//converts a letter into a number
{
// document.write("let2num " + let);
 for (i=0; i<26; i++)
   {
    if (alphabet[i]==let)
return(i);
}
return(1000);
}
function shift(let,s)
/*this converts a letter into a number, adds the shift, and converts
it back*/
{
// document.write("shift  " + let + " " + s);
   return(num2let(let2num(let)+s));

}


function vigenere()
{
   var ptext = prompt("Plaintext", "Type your plaintext here");
     var ktext = prompt("Keyword", "Type your keyword here");
     var ctext = "";

ij=0
for(ij=0;ij<ptext.length;ij++)
 {
  next=ptext.charAt(ij);
  if (let2num(next)==1000)  //checking if next is a letter
   {ctext=ctext+next;}
   else
   {
  nextshift=ij%ktext.length; //corresponding position from keytext
  nextkeylet=ktext.charAt(nextshift);
  nextkeyshift=let2num(nextkeylet);
  ctext=ctext+shift(next,nextkeyshift);
  //ctext.write(shift(next,nextkeyshift));
}
```

```
}
confirm("The Ciphertext is "+ctext);
codemail(ctext);


}
function vigenere_decode()
{
   var ptext = prompt("Ciphertext", "Type your ciphertext here");
     var ktext = prompt("Keyword", "Type your keyword here");
     var ctext = "";

ij=0
for(ij=0;ij<ptext.length;ij++)
 {
  next=ptext.charAt(ij);
  if (let2num(next)==1000)  //checking if next is a letter
   {ctext=ctext+next;}
   else
   {
  nextshift=ij%ktext.length; //corresponding position from keytext
  nextkeylet=ktext.charAt(nextshift);
  nextkeyshift=let2num(nextkeylet);
  nextkeyshift=26-nextkeyshift;
  ctext=ctext+shift(next,nextkeyshift);
  //ctext.write(shift(next,nextkeyshift));
}
}
confirm("The Plaintext is "+ctext);
codemail(ctext);

//document.write("</b><br>");
}

function vigenere_ktext_transfer()
{
 var ctext = prompt("Plaintext", "Type your plaintext here");
     var ktext = prompt("Keyword", "Type your keyword here");
     var ptext = "";
ij=0
for(ij=0;ij<ptext.length;ij++)

 {
  next=ptext.charAt(ij);
  if (let2num(next)==1000)  //checking if next is a letter
   {ctext=ctext+next;}
   else
   {
  nextshift=ij%ktext.length; //corresponding position from keytext
  nextkeylet=ktext.charAt(nextshift);
  nextkeyshift=let2num(nextkeylet);
  ctext=ctext+shift(next,nextkeyshift);
  //ctext.write(shift(next,nextkeyshift));
}
{
confirm("The Ciphertext is "+ctext);
codemail(ctext);
parent.location.href='mailto:'+who+'?subject='+what+'&body='+message+'';
confirm("When you have sent the mail, wait until the person replies, then
press OK");
}
}
{
var ptext1 = prompt("Ciphertext", "Type your new ciphertext here");
     var ktext1 = prompt("Keyword", "Type your keyword here");
     var ctext1 = "";
{
  next=ptext1.charAt(ij);
  if (let2num(next)==1000)  //checking if next is a letter
   {ctext1=ctext1+next;}
   else
   {
  nextshift=ij%ktext.length; //corresponding position from keytext
  nextkeylet=ktext1.charAt(nextshift);
  nextkeyshift=let2num(nextkeylet);
  nextkeyshift=26-nextkeyshift;
  ctext=ctext+shift(next,nextkeyshift);
  //ctext.write(shift(next,nextkeyshift));
}
}
confirm("The Plaintext is "+ctext1);
codemail(ctext1);
confirm("Once you have sent the mail, you have completed your in the keyword
transfer process! Good Job!")
}
}



function shift_cipher()
{
   var ptext = prompt("Plaintext", "Type your plaintext here");
     var s = prompt("Shift", "Type your shift number here");
     var ctext="";

s=parseInt(s);
//document.write("The Ciphertext is <b> ");

ij=0;
for(ij=0;ij<ptext.length;ij++)
 {
  next=ptext.charAt(ij);
  if(let2num(next)==1000)
{
 {ctext=ctext+next;}
}
else
{
ctext=ctext+shift(next,s);
}
}
confirm("The Ciphertext is "+ctext);
codemail(ctext);
}

function shift_cipher_decode()
```

```
{
   var ptext = prompt("Ciphertext", "Type your ciphertext here");
     var s = prompt("Shift", "Type your shift key here");
     var ctext="";

s=parseInt(s);
s=26-s
//document.write("The Plaintext is <b> ");

ij=0;
for(ij=0;ij<ptext.length;ij++)
 {
  next=ptext.charAt(ij);
  if (let2num(next)==1000)  //checking if next is a letter
     {ctext=ctext+next;}
  else
    {ctext=ctext+shift(next,s);}
}

confirm("The Plaintext is "+ctext);
}

function shift_cipher_crack()
{
 var ptext = prompt("Ciphertext", "Type your ciphertext here");
     var ctext="";

for (s=0;s<26;s++)
{
    ij=0;
    ctext="";
    for(ij=0;ij<ptext.length;ij++)
    {
     next=ptext.charAt(ij);
  if (let2num(next)==1000)  //checking if next is a letter
     {ctext=ctext+next;}
  else
     {ctext=ctext+shift(next,s);}
    }
{
document.write(ctext+"<p>");
}
}}

function implement_mod_x()
{
     var b = prompt("Enter the base");
     var x = prompt("Enter the exponent");
     var m = prompt("Enter the mod");
     alert(mod_exp(b, x, m));
}

function fact(n)
{
     if (n == 0)
     {
          return 1
     }
     else
     {
          return n * fact(n-1)
     }
}

function choose(n, m)
{
     return fact(n) / (fact(m) * fact(n-m));
}

function implement_choose()
{
     var n = prompt("Number of things you have");
     var m = prompt("Number of things you want to choose");
     alert("You can do it " + choose(n, m) + " ways");
}

function fib(n)
{
     if (n == 1 || n == 2)
     {
          return 1;
     }
     else
     {
          return fib(n - 1) + fib(n - 2);
     }
}

function fib_whole(n)
{
     var a = 0;
     var b = 1;
     while (b < n)
     {
          return b;
          a, b = b, a + b;
     }
}

function implement_fib()
{
     var n = prompt("Enter a number");
     alert("The "+ n + "the fibbonaci number is " + fib(n) + ". The golden
ratio is " + (fib(n) / fib(n-1)));
     alert("The fibonnaci series up to " + n + " will the displayed now");
     document.write(fib_whole(n));
}

function mod_exp(base, exp, mod)
{
     // Square by bases
     if (exp == 1)
     {
          return base % mod;
     }
     else if (exp % 2 == 0)
     {
          var z = mod_exp(base, exp/2, mod);
          return z * z % mod;
     }
     else
     {
          return base * mod_exp(base, exp-1, mod) % mod;
     }
}

function gen_primes(max)
{
     // Make a global array
     primes = new Array();
     primes[0] = 1;
     primes[1] = 2;

     // Generate primes
     var i = 0;
     for (i = 3;i < max;i++)
     {
          if (check_prime(i) == 1)
          {
               primes.push(i);
          }
     }
}

function rand_prime()
{
     return primes[Math.floor(Math.random() * primes.length)]
}

function gen_priv_pub()
{
     gen_primes(1000); // Find all of the primes from 1-1000
     var p = rand_prime(); // Find a random prime from the array primes (2)
     var q = rand_prime();
     while (p == q)
     {
          q = rand_prime();
     }
     var n = p * q; // Compute the modulus
     var tot = (p - 1) * (q - 1); // Compute the totient
     var e = rand_prime() // Compute the public key exponent
     while (e >= tot)
     {
          e = rand_prime()
     }
     var d = solve_pvt_exp(tot, e);
     priv_pub = [p, q, n, e, d];
     alert(priv_pub.join(", "));
}

function set_key_cookie(array, c_name)
{
     document.cookie = c_name + "=" + escape(array);
}
```

```
function get_key_cookie(c_name)
{
     if (document.cookie.length>0)
     {
          c_start=document.cookie.indexOf(c_name + "=");
          if (c_start!=-1)
     {
          c_start=c_start + c_name.length + 1;
          c_end=document.cookie.indexOf(";",c_start);
          if (c_end==-1) c_end=document.cookie.length;
          return unescape(document.cookie.substring(c_start,c_end));
     }
     }
return "";
}

function check_key_cookie()
{
     var keys = get_key_cookie(keys);
     if (keys != [] && keys != null && keys != "")
     {
          alert(keys.join(", "));
     }
     else
     {
          gen_priv_pub();
          set_key_cookie[priv_pub, keys];
     }
}

function encode_message()
{
     var ptext = prompt("Enter the plain text here");
     var i = 0;
     var enc = new Array();
     for (i = 0;i < (ptext.length);i++)
     {
          var let = ptext.charAt(i);
          if(let2num(let) !== 1000)
          {
               enc.push(encode_char(let2num(let)));
          }
     }
     alert("The encoded message (cipher text) is " + enc.join(";"));
     return enc;
}

function encode_char(message)
{
     while (message >= priv_pub[2])
     {
          message = message - 1;
     }
     pad = 5
     return mod_exp(message + pad, priv_pub[3], priv_pub[2]);
}
```

```
function decode_message()
{
     var ctext = prompt("Enter the cipher text here");
     var ctext_array = ctext.split(";");
     var i = 0;
     var dec = new Array();
     for (i = 0;i < (ctext_array.length);i++)
     {
          var num = ctext_array[i];
          dec.push(decode_char(num));
     }
     alert("The original message is " + dec.join());
     return dec;
}

function decode_char(message)
{
     var ptext = mod_exp(message, priv_pub[4], priv_pub[2]) - pad;
     return num2let(ptext);
}

function check_prime(j)
{
     var i = 0;
     for (i = 0;i < primes.length;i++)
     {
          if (j % primes[i + 1] == 0)
          {
               return 0; // not prime
          }
     }
     return 1; // prime
}

function gcd(a, b)
{
     if (a > b)
     {
          a, b = b, a;
     }
     if (a <= 0)
     {
          return null;
     }
     if (a == 1 || b - a == 0)
     {
          return a;
     }

     return gcd(b-a, a);
}

function solve_pvt_exp(tot, e) // Solve the private exponent (d) using a
given equation
{
     var k = 1
     while ((k * tot + 1) % e !== 0)
     {
          k++
     }
     return ((k * tot + 1) / e)
}

function gcd_implement()
{
     alert(gcd(3, 5));
}

function codemail(message)
{
  if (confirm("Do you want to mail it to a friend?")==true)
   {
         who=prompt("Enter a friend's email address: ","budugu2z@gmail.com");
        what=prompt("Enter the subject: ","I have a coded message for you!");

      if (confirm("Are you sure you want to mail "+who+" with the subject of
"+what+"?")==true)
    {
          parent.location.href='mailto:'+who+'?subject='+what+'&body='+message
+'';
}
}
}
</script>
```