

# RSA Cryptography

John Hutchinson

```
> restart;
```

## Translating Between Messages and Numbers

Here are two functions `message_to_number` and `number_to_message` which convert messages to numbers and conversely. The letters {a, b, ..., z} are replaced by {10, 11, ..., 35}, {A, B, ..., Z} by {36, 37, ..., 61}, {0, 1, ..., 9} by {62, 63, ..., 71}, and the characters { , . ; : - / "space" } by {72, 73, 74, 75, 76, 77, 78} respectively. (If the input number does not correspond to a message then an error message is given.)

*Ignore the code describing these two functions.*

```
> message_to_number := proc(message)
  local message_bytes, message_numlist, length_of_message, number_string
  :
  # We first use the ASCII coding
  message_bytes:= convert(message,bytes):
  # We next convert to a simpler but more restricted translation
  # as described above, using 2 digits rather than 3.
  message_numlist := map(proc(x) if (97<=x and x<=122) then x-87 elif
  (65<=x and x<=90) then x-29 elif (48<=x and x<=57) then x+14 elif x=44
  then 72 elif x=46 then 73 elif x=59 then 74 elif x=58 then 75 elif x=45
  then 76 elif x=47 then 77 elif x=32 then 78 end if end proc ,
  message_bytes ):
  length_of_message := nops(message_numlist):
  # The sequence of 2 digit numbers is next concatenated
  number_string := cat(seq(message_numlist[i],i=1..length_of_message)):
  # And finally turned into an integer
  convert(number_string,decimal,10):
end:
```

```
> number_to_message := proc(number)
  local message_length, number_string, message_string, message_numlist,
  message_bytes, i :
  message_length := length(number)/2 :
  # The number is first converted into a string of digits
  number_string := convert(number,string) :
  # And then into a vector of numbers between 10 and 99
  message_string := linalg[vector](message_length):
  for i from 1 to message_length do
  message_string[i] := convert(substring(number_string,2*i-1..2*i),
  decimal, 10);
  od:
```

```

# And then into a list of numbers
message_numlist := convert(message_string,list):
# And then into standard ASCII coding
message_bytes := map(proc(x) if (10<=x and x<=35) then x+87 elif (36<=x
and x<=61) then x+29 elif (62<=x and x<=71) then x-14 elif x=72 then 44
elif x=73 then 46 elif x=74 then 59 elif x=75 then 58 elif x=76 then 45
elif x=77 then 47 elif x=78 then 32 else "*" end if end proc ,
message_numlist) :
# And then into standard ASCII characters
convert(message_bytes,bytes) :
end:

```

```

> message := "Beware the Ides of March, 15/3/07";
      message := "Beware the Ides of March, 15/3/07"

```

```

> number := message_to_number(message);
      number := 371432102714782917147844131428782415784810271217727863677765776269

```

```

> message := number_to_message(number);
      message := "Beware the Ides of March, 15/3/07"

```

## [-] Finding Your Public "(n, e)" and Your Private Key "d"

Enter a random integer after ":=", before ";" and give it the name "pp".

With 100 digits it will take over 100 million 5 GHz-Opteron-CPU years with current technology to crack your code.

Each digit added (subtracted) increases (decreases) time to crack by a factor about 10.

With 150 digits, all the worlds current computers working in parallel for the lifetime of the universe will not crack your code.

```

> pp := 76398256649802838977659276645678008926455022098127;
      length = length(pp);
      pp := 76398256649802838977659276645678008926455022098127
      length = 50

```

Find the first prime after pp, name it p.

```

> p := nextprime(pp);
      p := 76398256649802838977659276645678008926455022098233

```

Enter a second random integer with about the same number of digits.

```

> qq:= 49290864532859690887375854988876987997867279656298;
      length = length(qq);
      qq := 49290864532859690887375854988876987997867279656298
      length = 50

```

Find the first prime after qq, name it q.

```

> q := nextprime(qq);

```

```
q := 49290864532859690887375854988876987997867279656387
```

Multiply **p** and **q** and call the result **n**.

```
> n := p*q;
n := 37657361190720787856100500965143810983948577931129350043359106376901286067071023:
6228667007999864171
```

We now begin the calculation to find the *encryption* exponent **e**, the second number in your public key.

```
> m := (p-1)*(q-1);
m := 37657361190720787856100500965143810983948577931128093152147279751602635715754677
1231742685698109552
```

Find **e** relatively prime to **m**.

We do this by successively testing 3, 4, 5, ... until we find an integer which has no factor in common with **m**.

The function **igcd** gives the (integer) greatest common divisor.

```
> e := 3 ; 'gcd' = igcd(m,e);
while igcd(m,e) <> 1 do
e := e+1; 'gcd' = igcd(m,e);
od;

e := 3
gcd = 3
e := 4
gcd = 4
e := 5
gcd = 1
```

Find the multiplicative inverse of **e mod m**.

```
> d := 1/e mod m;
d := 150629444762883151424402003860575243935794311724512372608589119006410542863018710:
8492697074279243821
```

You now have your *public key* (**n,e**) which you publish, and your *private key* **d**.

```
> '(n,e)' = (n,e); 'd' = d;
(n,e) =
(3765736119072078785610050096514381098394857793112935004335910637690128606707102:
26228667007999864171, 5)
d = 150629444762883151424402003860575243935794311724512372608589119006410542863018710:
492697074279243821
```

## How Anyone Uses Your Public Key to Encode a Secret Message Only You can Decode

If someone has a secret message for you they first enter it into their computer.

Only messages with small and capital letters, digits, { , . ; : - / "space" } work with this simple program.

```
> message := "The walk will be at 9am tomorrow." ;
```

```
message := "The walk will be at 9am tomorrow."
```

They then translate this secret message into a secret number **W** by replacing a by 10, b by 11, ... , A by 35, ... , etc. as before.

```
> W := message_to_number(message);  
W := 551714783210212078321821217811147810297871102278292422242727243273
```

**W** should be less than **n**.

If **W** is less than *half the length* of **n** then for additional security it can be padded out at the beginning by random junk numbers. We will not bother.

(Begin and end any junk with 99 and put random stuff in between which contains *no* 9's. Then you will know after later decoding where the junk begins and ends.)

```
> verify(W,n,less_than);  
verify(length(W),length(n)/2,greater_than) ;  
true  
true
```

They next find the coded number **C** from the secret number **W** by using your public key (**n,e**) to compute the remainder obtained after raising **W** to the power **e** and dividing by **n**.

```
> C := W&^e mod n;  
C := 31301782953327603158570946980826419160939893010885023606508546351779562174665333'  
5390945958659606911
```

Finally, they publish the coded number **C**.

## — How You Use Your Private Key to Decode the Coded Secret Message

You decode the coded number **C** by computing the remainder after raising **C** to the private decoding exponent **d** and dividing by the public number **n**. The result will be the original secret number **W**.

```
> DD := C&^d mod n;  
DD := 551714783210212078321821217811147810297871102278292422242727243273
```

The decoded number **DD** is translated back into the original secret message by replacing 10 by a, 11 by b, ... , 35 by A, ... etc. as previously discussed.

(First strip off junk, if any, at the beginning -- it will begin and end with 99 and have no 9's between.)

```
> original_message := number_to_message(DD);  
original_message := "The walk will be at 9am tomorrow."
```

