# How RSA Works

**By Martin Rupp & Petr Smirnov**
SCIENTIFIC AND COMPUTER DEVELOPMENT SCD LTD

---

## Introduction

In this article, we will explain the functioning of the well-known cryptographic algorithm called RSA (Rivest-Shamir-Adleman) .

## What is RSA

RSA is the root of all asymmetric key cryptographic algorithms. It is asymmetric because the decryption key is different from the encryption key.

RSA was first published in 1977.

RSA uses basic properties of **modular arithmetic** , that is to say the arithmetic operations in a finite congruence ring , namely the ring of integers modulo n, where n is an integer > 0.

The ring of integers modulo n is usually noted $Z/nZ$. In such ring it is possible to find a couple of integers (e,d) such that:$(m^e)^d = m$for any integer m such that n > m > 0. Concretely this means that $(m^e)^d - m$will always be a multiple of n.

In fact there are many different couples of integers (e,d).

For instance, if we take n=10, if e=3 and d=3 then:

| $m$ | $m^9$ | $m^9 - m$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 512 | 510 |
| 3 | 19683 | 19680 |
| 4 | 262144 | 262140 |
| 5 | 1953125 | 1953120 |
| 6 | 10077696 | 10077690 |
| 7 | 40353607 | 40353600 |
| 8 | 134217728 | 134217720 |
| 9 | 387420489 | 387420480 |

The combination of (e,d,n) is the RSA cryptosystem. (e,d) is the keypair of the cryptosystem.

In our example, the number m<10 will be ciphered by the operation $m \rightarrow m^3$ and will be deciphered by the same operation.

| $m$(clear) | $m^3$(ciphered) |
|---|---|
| 1 | **1** |
| 2 | **8** |
| 3 | 27=**7** |
| 4 | 64=**4** |
| 5 | 3125=**5** |
| 6 | 216=**6** |

| | |
|---|---|
| 7 | 343=**3** |
| 8 | 512=**2** |
| 9 | 729=**9** |

In our example m is ciphered with e and deciphered with d. Since $(m^e)^d = m$ in the ring $Z/nZ$, this makes sense, indeed ciphering followed by deciphering returns the clear text *m*.

We have defined the RSA cryptosystem! The whole security of the RSA lies in the fact that the only knowledge of the encrypted data , e.g $m^e$ together with n, is not enough to find “easily” the other part, e.g *d*.

Given a message *m* represented by a number < n, the ciphering operation is:

$$m \to (m)^e$$

and the deciphering operation is:

$$m' \to (m')^d$$

In this example, *e* is the public key and *d* is the private key. The couple *(e,d)* is called an RSA key pair.

Concretely, the value of *n* will be a “big” number. The number which can be encrypted therefore can represent a 32 bit or 64 bits value.

Next we need to explain

1) How to generate the key, eg finding keypairs *(e,d)* for a value of *n*
2) Why RSA is a secure cryptosystem

# Key Pair generation

## Some results in modular arithmetic

The RSA system relies on “basic” results of modular arithmetic such as the Lagrange Theorem.

Lagrange Theorem states that, for a number x > 0, coprime with n, $x^{\varphi(n)} \equiv 1(n)$ where $\varphi$ is the Euler function.

The Euler function of a number *q* is defined as the amount of numbers which are relatively prime with it.

$$\varphi(n) = \sum_{\Delta(s,n)=1} s$$

$\Delta(s,n) = 1$ means that *s* and *n* are relatively prime with each other (coprime) , e.g they share no common factors but 1.

For instance we list the following values of $\varphi$:

| $n$ | $\varphi(n)$ |
|---|---|
| 5 | {1,2,3,4}=4 |
| 10 | {1,3,7,9}=4 |
| 11 | {1,2,3,4,5,6,7,8,9,10}=10 |
| 28 | {1,3,5,9,11,13,15,17,19,23,25,27}=12 |

There are a lot of properties for the Euler function but the one we are interested in is the following:

$\varphi(pq) = (p-1)(q-1)$ for two distinct prime numbers *p* and *q*.

This means that, if $n = pq$ *we have* $m^{(p-1)(q-1)} \equiv 1\ (n)$

The original RSA algorithm uses Euler function, The next versions of RSA are using the Charmichael function $\lambda(n)$. Here we will use Euler $\varphi$ function.

## Key generation

Our problem is to find two numbers *e* and *d*, such that $m^{ed} \equiv m\ (n).$
For this it is enough to find *e* and *d* such that $ed \equiv 1\ (p-1)(q-1).$

Indeed, in such case
$m^{ed} = m^{1+k(p-1)(q-1)} = m((m^{(p-1)(q-1)})^{k}) \equiv m\ (n).$

For this, we choose *e* to be coprime with *(p-1)(q-1)* , e.g having no common factor but 1.

To generate the private key, we then need to find d such that:

$$ed = 1\ (modulo\ (p-1)(q-1))$$

*e.g d is the* ***modular inverse of e*** *modulo (p-1)(q-1)).*

Finding such a number is always possible, because *e* is coprime with *(p-1)(q-1)* and so it has an inverse in the ring $Z/((p-1)(q-1))Z$.

# Security of the system

To break the RSA system, we must find the value of the private key d , knowing the modulus n and the exponent e (and the ciphered message $m^e$ )

For this we must solve:

“Knowing (e,n), find d such that *ed=(p-1)(q-1) (modulo (p-1)(q-1))*“

If we know the two primes p and q, the problem is trivial, so the problem is equivalent to:

“Knowing a number n which is the product of two primes p and q, find p and q”

The security of the RSA cryptosystem lies in the fact that it is a very difficult problem and the best way to find p and q is to apply the  general number field sieve (GNFS) algorithm but that algorithm has sub-exponential time complexity, e,g range between polynomial time and exponential time.

Even sub-exponential, the time needed for factorization of large integers, using the GNFS, is at the moment too important, and can typically reach hundreds of years, even using the best computational power.

Cracking the RSA-768, e.g factoring a 232 digit number, was achieved in 2009 using hundreds of extremely optimized machines. The RSA laboratories, owner of the RSA patent, are offering prizes to anyone who can factor RSA numbers.

As of 2019, the RSA-2048 which has 617 decimal digits is considered to be very secure,

The fact that, currently, it is not (publically) known about any integer factoring algorithm with polynomial time complexity (e.g which could be used with the actual machines) does not mean such algorithms do not exist but for the moment, RSA is considered to be a secure cryptosystem.

RSA is slow, because it requires modular arithmetic computations and is not usually used to cipher data, but to cipher keys or to encrypt hashes of data.

# Example

We need to  choose two prime numbers p and q . There are several methods to generate such primesm usually this involves primality testing.

For instance p=5023 and q=64037.

We have *n*=*pq*=321657851 and *(p1)(q-1)*=5022*64036=321588792.

We need to choose e such that *e* is coprime with 321588792.

It is enough to choose e as a  prime number < 321588792. For instance e=113.
Only left is to find *d* as the modular inverse of *e* modulo 321588792, e.g. such that *ed-1* is a multiple of 321588792. It is not hard to find such a number, it is *d*=105298985.

Our algorithm will then encode m by the operation

$$Cipher = m^{113}$$

And decipher by the operation:

$$Decipher = Cipher^{105298985}$$

RSA encrypts numbers but any digital data - including text - can be converted into an integer into a wide number of ways.

Here we can encrypt numbers up to 321,657,851.

We code letters in the alphabet by a number 0-25 , period and space by the value 26 and 27.

We consider a text made of n letters (represented by numerical values (0-27) $x_1 \ldots x_n$.

We form N as $N = \sum_{i=1}^{n} x_i 28^i$. e,g N is the number represented by $[x_1 \ldots x_n]$ in base-28.

If N < 321,657,851, we can use our RSA system to encrypt it. We can encode only up to 5 letters. But we can split any text into groups of 5 letters.

For instance the text:

"TONIGHT FIVE MORNING.ATTACK RIVER KWAI."

Can be split into groups of words of 5 letters (eventually padded with spaces) then ciphered by the RSA system.

| Clear base28 | Clear word | Cipher base 28 | Ciphered word |
|---|---|---|---|
| 11996214 | TONIG | 110409650 | GLRQRW |
| 4740996 | HT FI | 91722354 | FJGIWK |
| 13017102 | VE MO | 263389813 | PIOMLB |
| 10741170 | RNING | 261279660 | PFCIYM |
| 15996484 | .ATTA | 317852682 | SNDMJO |
| 1470484 | CK RI | 116157134 | GU LRC |
| 13009678 | VER K | 252680772 | OTCQZI |
| 13529459 | WAI. | 195108971 | LJL NP |

Here is the program (PERL) used for ciphering and deciphering:

```
sub text2base28($)
{

my $word= shift @_;
my @letters= split //,$word;

my $N=0;
my $n=0;

      foreach my $letter(@letters)
      {

            if ($letter =~ /[A-Z]/)
            {
                        #65-90
                         $n= ord ($letter) -65;

            }
             elsif($letter eq ".")
            {
            $n=26;
            }
             elsif($letter eq " ")
            {
            $n=27;
            }

            $N=28*($N)+$n;
      }
return $N;
}



sub base282Text($)
{

my $N=shift @_;
```

```
my $word="";

my @letters= split //,$word;


my $n=0;

      for(my $i=5;$i>-1;$i--)
      {

      my $N1=$N;

      for(my $j=0;$j<$i;$j++)
      {

      $N1=$N1/28;
      }

      $n=int $N1;

      if(($n==0)&&($i ==5 ))
      {


      }
      elsif($n<26)
      {
      my $c=chr ($n+65);


      $word=$word.$c;
      }
      elsif($n==26)
      {
      $word=$word.'.';
      }
      elsif($n==27)
```

```
      {
      $word=$word.' ';
      }
      else
      {
      die("$N: incorrect word at i=$i, value=$n \n");
      }

      my $N1=1;

      for(my $j=0;$j<$i;$j++)
      {

      $N1=$N1*28;
      }

      $N=$N-$n*$N1;

      }

return $word;

}


sub RSACipherDecipher($$)
{

my $n=321657851;
my $m= shift @_;
my $x= shift @_;

my $res=1;

#m^e/m^d

for(my $i=0;$i<$x;$i++)
{
```

```
$res=$res * $m;
$res=$res % $n;

}

$res=$res % $n;

return $res;

}

my $e= 113;
my $d=105298985;


my @texts=(
"TONIG",
"HT FI",
"VE MO",
"RNING",
".ATTA",
"CK RI",
"VER K",
"WAI. "
);

foreach my $text (@texts)
{
my $n1=text2base28($text);

my $W0=base282Text($n1);

#print "clear=$n1\n$W0\n";


my $n2= RSACipherDecipher($n1,$e);
```

```
my $W=base282Text($n2);

print "\n---------\n";
print "cipher=$n2\t\t ($W)";
print "\n---------\n";

my $n3= RSACipherDecipher($n2,$d);

my $W2=base282Text($n3);

print "\n---------\n";
print "decipher=$n3\t\t ($W2)";
print "\n---------\n";

}
```

```
---------
cipher=110409650                (GLRQRW)
---------

---------
decipher=11996214               (TONIG)
---------

---------
cipher=91722354                 (FJGIWK)
---------

---------
decipher=4740996                (HT FI)
---------

---------
cipher=263389813                (PIOMLB)
---------

---------
decipher=13017102               (VE MO)
---------

---------
cipher=261279660                (PFCIYM)
---------

---------
decipher=10741170               (RNING)
---------

---------
cipher=317852682                (SNDMJO)
```

```
---------
decipher=15996484               (.ATTA)
---------

---------
cipher=116157134                (GU LRC)
---------

---------
decipher=1470484                (CK RI)
---------

---------
cipher=252680772                (OTCQZI)
---------

---------
decipher=13009678               (VER K)
---------

---------
cipher=195108971                (LJL NP)
---------

---------
decipher=13529459               (WAI. )
---------
```

The program will run very slowly compared to a symmetric cipher. That is why RSA is not supposed to be used for long inputs.

**Conclusion: We aimed at presenting the way RSA works and allow the reader to understand what's hidden "under the hood" of that well-known ciphering algorithm. There is a large literature about the topic of RSA, integers factorizations etc… and we just aimed at presenting the basic theory behind RSA.**