# Public Key Encryption

John O'Gorman
john@og.co.nz

9th April 2017

The following algorithm for public key encryption was discovered by Clifford Cocks working for the UK intelligence GCHQ in 1973 and subsequently patented in the USA (but nowhere else) by 3 Americans Rivest, Shamir, and Adleman. It is commonly known as the RSA algorithm. The patent expired in the USA in 2000. It is based on the following principles: modulo (or clock) arithmetic, the Chinese Remainder Theorem, and the Fermat/Euler Theorem.

## 1 RSA Algorithm

1. Generate 2 large prime numbers $p$ and $q$.
2. Let $n = pq$.
3. Let $m = (p-1)(q-1)$. $m$ is also known as $\phi(n)$ or the *totient* of $n$.
4. Choose a small integer $e$ which has no common factor with $m$ (except 1).
5. Find $d$ such that $de \bmod m = 1$. A practical way to do this is to systematically try increasing values of $k$ in the expression $\frac{1+km}{e}$ until it returns a whole number.
6. Publish $n$ and $e$ as the public key to encrypt messages letter by letter. You keep $d$ as your private key to decrypt messages encrypted with the public key $e$.
7. To encrypt plaintext $P$ use $C = P^e \bmod n$
8. To decrypt encrypted $C$ use $P = C^d \bmod n$

# 2 Explanation

## 2.1 Modulo Arithmetic

Imagine a clock with hour 0 instead of 12 so that the digits run from 0 to 11. This represents modulo 12 and constrains all values to the integers 0, 1,2, ... to 11. Add 1 to 11 and you get 0. If you start from 3 and add any number of 12s you go round the clock to 3. This can be expressed mathematically as $x = x + kn (\bmod n)$ for any $k$. When you divide in modulo arithmetic, you produce 2 integer values: a quotient and a remainder. The remainder when you divide $x$ by $n$ is expressed as $x$ mod $n$ in computer languages such as C, Pascal, and 4GL - other languages such as perl, python, sh, bash, etc, use the % operator instead of mod. e.g. $x \,\%\, n$

## 2.2 Factors, Primes, Coprimes, etc

If you Google for RSA you will encounter a lot of specialised mathematical terminology. This glossary should help you decipher the explanations.

**Factors** are numbers which divide equally into another number e.g. 2, 3, 4, and 6 are factors of 12 (and more trivially, so are 1 and 12)

**Prime numbers** have no factors except the trivially obvious themselves and 1. The first few primes are 1, 2, 3, 5, 7, 11. They get sparser as the value rises.

**Coprimes:** Two or more numbers are said to be coprime if they have no common factors (apart from 1). e.g. 7 and 8 are coprime but 8 and 10 are not (since they have 2 as a common factor). Note that a number does not have to prime to be coprime!

**Congruent:** The relationship: $ed = 1 \,(\bmod m)$ is sometimes expressed as $ed\, congruent\, 1 \,(\bmod m)$

**GCD** is the Greatest Common Divisor of 2 numbers. e.g. $GCD\,(12, 18)$ is 6. HCF (Highest Common Factor) is the same thing. In some sources you may find $GCD(e, \phi(n)) = 1$ meaning $e$ has no common factor with $m$ except 1.

**Totient:** The expression $m = (p - 1)\,(q - 1)$ where $n = pq$ can be called the totient of $n$, or $\phi\,(n)$, or $phi(n)$. It represents the number of values between 1 and $n$ which are coprime with n (i.e. have no common factors with $n$ apart from 1). It was studied by the Swiss mathematician Euler. Totient is from the Latin for *that many times.*

Why does the formula work?
There are $n$ values between 1 and $n$ but these include
$p$ multiples of $q$ $(q, 2q, 3q, ..pq)$ and similarly
$q$ multiples of $p$ $(p, 2p, 3p, ..qp)$.
As $pq$ appears in both lists of multiples of $p$ and $q$ as $pq$ and $qp$ , but we want to count it only once, the number of factors is
$p + q - 1$.
So the number of integers which are *not* factors of $n$ is $n - (p + q - 1)$ or $pq - p - q + 1$ which factorises into:
$(p - 1)(q - 1)$.[1]

**Diaphantine:** Polynomial equations confined to integers in modulo arithmetic such as the above are named after a 3rd century mathematician Diaphantus of Alexandria.

**Chinese Remainder Theorem** states that if $x = y \pmod p$ and $x = y \pmod q$ then $x = y \pmod{pq}$

**Fermat/Euler Theorem** states that if $p$ is prime and $x \neq 0 \pmod p$ then $x^{p-1} = 1 \pmod p$

## 3 Examples

From the command line in Linux or Mac OS X sustems, you can try out the RSA algorithm using small values for p, q, d, and e. It would be nice if we could use the shell's arithmetic evaluation to test the formulas but unfortunately even for smallish values of d and e we soon overflow the shells' integer storage. e.g. $(( 6**65 )) returns 0. So instead we pipe the expression 6^65 to the `bc` program as follows:
`echo "6^65" | bc`. `bc` has unlimited precision integer arithmetic. An example:

```
#!/bin/sh
p=7 q=19 n=$((p*q)) m=$(( (p-1)*(q-1) )) e=5 d=65
echo p=$p q=$q n=$n m="(p-1)(q-1)"=$m e=$e d=$d
P=6
echo Encrypting P=$P
C=$( echo "($P^$e)%$n" | bc )
echo "Sending ($P^$e)%$n" to bc
echo "C=(P^e)%n" = $C
echo Decrypting C=$C
```

[1] I owe this explanation to Professor Rachel Fewster of the University of Auckland.

```
nP=$( echo "($C^$d)%$n" | bc )
echo "Sending ($C^$d)%$n" to bc
echo "nP=(C^d)%n" = $nP
```

The output from the above:

```
p=7 q=19 n=133 m=(p-1)(q-1) = 108 e=5 d=65
Encrypting P=6
Sending (6^5)%133 to bc
C=(P^e)%n = 62
Decrypting C=62
Sending (62^65)%133 to bc
nP=(C^d)%n = 6
```

Note that `bc` treats ^ as the exponent operator and % as the mod operator.

The values of shell variables are accessed by prefixing the variable ID with a $ symbol.

$(...) is the shell mechanism for running a command and substituting its output in place of the $(...).

$(( expr )) is the shell mechanism for evaluating and arithmetic expression and substituting the returned value in place of the $(( ... )).

The quotes are used to protect the parentheses and ^ and * from interpretation by the shell. The quotes get stripped after the shell has parsed the quoted expression.

## 4 Comments

The RSA algorithm depends on the infeasibility of finding the prime factors of large integers. It is recommended that $n$ have a length of 2048 bits at least. Keys of lower magnitude are subject to persistent brute force attacks.

The open source application PGP (Pretty Good Privacy) and its GNU version GPG (GNU Privacy Guard) both have inplemented the RSA application for many years. Prior to 2000, PGP had to be downloaded from outside the USA to avoid American patent restrictions. Since 2000 it has been supplied with all distributions including US products such as RedHat Linux, Ubuntu, Debian, etc.