Factoring and RSA (Rivest-Shamir-Adleman) Encryption

Peter Young

(Dated: April 26, 2020)

I. THE RSA ALGORITHM AND AN EXAMPLE

Shor's algorithm is used to factor large integers much more efficiently than any known classical algorithm. Factoring is not just of interest to mathematicians, however, because the difficulty of factoring is at the heart of the popular RSA method of encrypting sensitive information sent via the internet (or some other public channel). While RSA is not the only method use to encrypt information, my understanding is that some version of Shor's algorithm can be used to crack other encryption methods such as Diffie-Hellman. RSA stands for the names of its inventors, Rivest, Shamir and Adleman.

This handout is a LaTeX copy of a *Mathematica* notebook, the original of which is available at `https://young.physics.ucsc.edu/150/rsa.nb`. In it, the RSA algorithm is implemented, parameters are chosen, and random messages are generated. These are encrypted, the encrypted messages are decrypted, and a check is made that the original message is recovered. It you have *Mathematica* you can run the notebook version and verify that the RSA algorithm works.

Suppose that Bob wants to receive a message from Alice on the internet (a public channel). Anything sent on a public channel can be intercepted by others. How can Bob and Alice agree on a coding scheme and then send each other coded messages which can be decoded by the other person but not by anyone "sniffing" on the internet? This has to be accomplished by only sending messages down the public channel.

We will now describe the RSA encryption scheme for doing this. It uses a result of number theory which we will quote but not prove. To receive the message from Alice, Bob picks two large prime numbers p and q, and sends to Alice, on the public channel, their product

$$N = pq, \tag{1}$$

but not p and q separately. N is taken to be large enough, typically a few thousand bits, that it cannot be factored on a classical computer. You might ask how can one choose the large prime numbers p and q. If one selects a large integer N at random it can be shown that the probability that it is prime is about $1/\ln N$. Hence, even if N has, say, 400 digits (around 1000 bits) you only have to take test a few hundred to a thousand random integers to typically find a prime number.

But can one efficiently test if a number is prime? It turns out that one can, even though, if the number is found to be not prime, there is no known efficient classical algorithm to determine the prime factors. The website `http://mathworld.wolfram.com/PrimalityTest.html` explains how the test for primality is done in *Mathematica*.

Bob also sends a large "encoding number" c which has no factors in common with $(p-1)(q-1)$. If there are no factors in common then the greatest common divisor (GCD) is 1. The GCD of two integers is easily determined by Euclid's algorithm discussed at the end. According to Appendix J of Mermin[1], the probability that two large random integers have no common factors is greater that 1/2, so it is not difficult to find a suitable value for c. Hence the public key (available to everyone) is N and c. Since Bob knows both p and q, and hence $(p-1)(q-1)$, he can also determine the integer d such that

$$c\,d = 1(\text{mod } (p-1)(q-1)). \tag{2}$$

What is this mod function? The value of a mod b is the result after one subtracts (or adds) the appropriate multiple of b to a to get a value which lies in the range 0 to $b-1$. If a is positive, things are simple, one subtracts a multiple of b (possibly 0) so the mod function is just the remainder after integer division. Hence, for example, 9 mod 5 = 4 because 9/5 = 1 remainder 4. If a is negative one has to add a multiple of b, so, for example, (-13) mod 5 = 2 (since $-13 + (3 \times 5) = 2$).

The above equation, cd = 1 mod (something), looks strange at first. If c is an integer we normally think that its inverse should be a fraction. However, here d is also an integer, and the product of two integers *can* give 1 if we use modular arithmetic. For example if $c = 5$ and $d = 3$ then $cd = 15$, and cd mod 7 = 1 (since $15 = (7 \times 2) + 1$).

The algorithm for computing d in the equation above is efficient and an extension of Euclid's algorithm. Euclid's algorithm and the necessary extension are given at the end of this notebook and in Appendix J of Mermin[1]. It turns out that d is unique. Hence Alice, and anyone else sniffing on the public channel, knows N and c (but not p, and q, and hence not d). The private key (known only to Bob) is p and q (and hence d).

Alice breaks up her message into chunks of each containing a number of bits less than the number of bits of the integer N. Each chunk is then a binary number less than N. Let's denote by a the numerical value of one chunk.

a is the original message.

Using the values of N and c that Bob has sent, she computes

$$b = a^c \text{ (mod } N) \quad \text{the encoded message.} \tag{3}$$

The encoded message b is another large integer, and is sent down the public channel from Alice to Bob.

Bob knows not only c and N, but also the value of d. Here number theory kicks in and shows that the original (unencoded) message a is given by

$$a = b^d \pmod{N} \quad \text{(the original message is recovered).} \tag{4}$$

For a proof of this result see the book by Mermin[1]. Bob can compute it because he knows d, but anyone sniffing on the public channel does not. However, if a third person, traditionally called Eve, listening on the public channel, could factor N (which is sent down the public channel) into its factors p and q, she would then have $(p-1)(q-1)$ and, since c is also sent down the public channel, she could determine d where $cd = 1 \pmod{(p-1)(q-1)}$ using the algorithm given at the end and in Mermin's Appendix J. Hence she could find the original unencrypted message a from Eq. (4).

Let's do a simple example. We will take

$$p = 7,\ q = 13,\ \text{so } N = 91. \tag{5}$$

For the encoding integer we take $c = 11$, which has no factors in common with $(p-1)(q-1) = 6\times 12 = 72$. As shown at the end of this handout, using the extended Euclid algorithm also discussed in appendix J of Mermin[1], one finds that $d = 59$. (Let's verify this: $11 \times 59 = 649 = (9 \times 72) + 1$.) The *Mathematica* code below sets these values, checks that p and q are prime while N is not, and that $cd = 1 \pmod{(p-1)(q-1)}$. (Note: in *Mathematica* commands we use n rather than N because N has a special meaning in *Mathematica*.) The code then generates a message a by computing a random integer between 0 and $N-1$, and next computes the encoded message b from $b = a^c \pmod{N}$. It then computes $b^d \pmod{N}$ and checks that it gives back the original message a. If you have *Mathematica* you can run the code several times (each time a different random value for the message a will be generated) and see that the original message is always returned.

```
In[1]:=  p=7; q=13; c=11; d=59; n=p*q
Out[1]=  91
```

We check that p and q are prime. The Mathematica command PrimeQ[p] returns "True" if p is prime and "False" if it is not.

```
In[2]:=  PrimeQ[p]
Out[2]=  True
```

```
In[3]:=  PrimeQ[q]
Out[3]=  True
In[4]:=  PrimeQ[n]
Out[4]=  False
```

We check that $cd = 1 \bmod ((p-1)(q-1))$.

```
In[5]:=  Mod[c * d, (p-1)(q-1)]
Out[5]=  1
```

We generate a random message.

```
In[6]:= mess = Random[Integer, n - 1]
Out[6]=  51
```

We compute the encoded message.

```
In[7]:= encodedmess = Mod[mess^c, n]
Out[7]=  25
```

We decode the encoded message and check that we recover the original message.

```
In[8]:= recoveredmess = Mod[encodedmess^d, n]
Out[8]=  51
In[9]:= recoveredmess == mess
Out[9]=  True
```

Hence the message was successfully decoded.

A. The Euclidean Algorithm

We want to efficiently find the Greatest Common Divisor (GCD) of two integers. This is the largest factor that they have in common. As a simple example, the GCD of 24 and 9 is 3.

Suppose we want the GCD of two numbers a_0 and b_0 with $a_0 > b_0$. We proceed iteratively. At each stage, the new value of a is equal to the old value of b, and the new value of b is equal to the remainder when the old value of a is divided by the old value of b, i.e.

$$\begin{aligned} a_{n+1} &= b_n \\ b_{n+1} &= a_n - [a_n/b_n]b_n \quad \text{which is the same as } b_{n+1} = a_n \bmod b_n, \end{aligned} \tag{6}$$

where $[\cdots]$ means the integer part of the quantity in brackets. a_n and b_n decrease at successive iterations and maintain the inequality $a_n > b_n$. Also a_n and b_n have the same common factors as a_0 and b_0. Eventually we get to a stage where $b_{n+1} = 0$. This means that a_n is divisible by b_n so b_n is the greatest common divisor.

As an example we take $a_0 = 24, b_0 = 9$,

n	a_n	b_n
0	24	9
1	9	6
2	6	3
3	3	0

Hence the GCD of 24 and 9 is b_2 $(= 3)$.

B. Extension of the Euclidean Algorithm to find an inverse modulo an integer

Given a and c which have no common factors, and $a < c$, we want to find d where

$$c\,d = 1 \bmod a. \tag{7}$$

The greatest common divisor of c and a since, by assumption, they have no common factors. We go through the Euclid algorithm

$$\begin{aligned} a_{n+1} &= c_n \\ c_{n+1} &= a_n - [a_n/c_n]c_n \end{aligned} \tag{8}$$

until we get to the stage where $c_n = 1$, the greatest common divisor. One can then obtain d by working backwards through the iterations. This is best shown by an example. We take $p = 7, q = 13$, as in example above, so we have $a = (p-1)(q-1) = 72$ and hence we initialize $a_0 = 72$. We also take $c = 11$ (again as in the example) which has no factors in common with a, and so initialize $c_0 = 11$. Hence the Euclid algorithm proceeds as follows

n	a_n	c_n	
0	72	11	$a_0 = a, c_0 = c$ (the initial values)
1	11	6	$a_1 = c_0, c_1 = a_0 - 6c_0 = 6$
2	6	5	$a_2 = c_1, c_2 = a_1 - c_1 = 5$
3	5	1	$a_3 = c_2, c_3 = a_2 - c_2 = 1$ ($c_3 = 1$ so we stop).

Hence working backwards,

$$1 = a_2 - c_2 = c_1 - (a_1 - c_1) = 2c_1 - a_1 = 2(a_0 - 6c_0) - c_0 = 2a_0 - 13c_0 \ (= 2a - 13c). \qquad (9)$$

We want to take this (mod a). Now $2a \pmod a = 0$. Also $-13c$ is negative which is inconvenient. However, we can add to it $a\,c$ (which is zero (mod a)). Hence

$$2a - 13c \pmod a = -13c \pmod a = (-13 + a)c \pmod a = 59c \pmod a. \qquad (10)$$

Hence $d = 59$ as stated in the above example.

---

[1] N. D. Mermin, *Quantum Computer Science* (Cambridge University Press, Cambridge, 2007).