CIS511
Introduction to the Theory of Computation
Public Key Cryptography and RSA

Jean Gallier

March 6, 2013

Chapter 1

Public Key Cryptography; The RSA System

1.1 The RSA System

Ever since written communication was used, people have been interested in trying to conceal the content of their messages from their adversaries.

This has led to the development of techniques of secret communication, a science known as *cryptography*.

The basic situation is that one party, A, say *Albert*, wants to send a message to another party, J, say *Julia*.

However, there is a danger that some ill-intentioned third party, *Machiavelli*, may intercept the message and learn things that he is not supposed to know about and as a result, do evil things.

The original message, understandable to all parties, is known as the *plain text*.

To protect the content of the message, Albert *encrypts* his message.

When Julia receives the encrypted message, she must *decrypt* it in order to be able to read it.

Both Albert and Julia *share* some information that Machiavelli *does not* have, a *key*. Without a key, Machiavelli, is incapable of decrypting the message and thus, to do harm.

There are many schemes for generating keys to encrypt and decrypt messages.

We are going to describe a method involving *public and private keys* known as the *RSA Cryptosystem*, named after its inventors, Ronald Rivest, Adi Shamir and Leonard Adleman (1978), based on ideas by Diffie and Hellman (1976).

The RSA system is widely used in practice, for example in SSL (Secure Socket Layer), which in turn is used in https (secure http).

Any time you visit a “secure site” on the internet (to read e-mail or to order a merchandise), your computer generates a public key and a private key for you and uses them to make sure that your credit card number and other personal data remains secret.

Interestingly, although one might think that the mathematics behind such a scheme are very advanced and complicated, this is not so. In fact, little more than *the ability to divide integers* is needed!

Euclid, Fermat and Euler (together) had basically all the tools to come up with RSA!

The first step is to convert the plain text of characters into an *integer*.

This can be done easily by assigning distinct integers to the distinct characters, for example by converting each character to its ASCII code. From now on, we will assume that this conversion has been performed.

How can two people communicate securely without knowing each other and without meeting each other?

On way to do this is to use pairs of *keys*:

1. *public keys*
2. *private (or secret) keys*

Public keys are known to all participants but private keys are *secret* and only known to their owner.

Public key cryptography works as follows:

Each participant, A, has two keys, (K_A^e, K_A^d), where

K_A^e is A's *public key* (e for *encrypt*) and
K_A^d is A's *private (secret) key* (d for *decrypt*).

The public key, K_A^e, is published and known to everybody but the private key, K_A^d, is *only known to A* and should be kept secret to ensure the security of the method.

For every user, A, the public key, K_A^e, gives access to an *encryption function*, E_A, and the private key, K_A^d, gives access to a *decryption function*, D_A.

The function, E_A, is public and known to everybody (who knows the public key, K_A^e).

The function, D_A, is private and only known to A.

The functions E_A and D_A are designed so that they are mutual inverses, which means that

$$D_A \circ E_A = \mathrm{id} \qquad \text{and} \qquad E_A \circ D_A = \mathrm{id}.$$

Furthermore, the encryption function, E_A, should be easy to compute but the decryption function, D_A, should be *very hard* to compute, *unless one has the secret key,* K_A^d.

Functions such as D_A are *trapdoor one-way functions*.

Encrypting a message

If Julia wants to send a message, x, (the *plain text*) to Albert (A), she proceeds as follows:

Julia looks up Albert's *public key*, K_A^e, then she gets the *encryption function*, E_A, and she uses it to compute

$$y = E_A(x),$$

which she sends to Albert. The message, y, is the encrypted message and it is known as the *cyphertext*.

Decrypting a message

In order to decrypt a message, y, that Julia sent him, Albert uses his *private key*, K_A^d, to get its *decryption function*, D_A, then he computes

$$x = D_A(y).$$

The next and more subtle step is to use *modular arithmetic* to design public and private keys together with encryption and decryption functions.

We pick a (large) positive integer, m, and perform arithmetic modulo m. Let us explain this step in more details.

Recall that for all $a, b \in \mathbb{Z}$, we write

$$a \equiv b \pmod{m}$$

iff $a - b = km$, for some $k \in \mathbb{Z}$, and we say that
a and b are congruent modulo m.

It is easy to check that congruence is an equivalence relation but it also satisfies the following properties:

Proposition 1.1.1 *For any positive integer, m, for all $a_1, a_2, b_1, b_2 \in \mathbb{Z}$, the following properties hold: If $a_1 \equiv b_1 \pmod{m}$ and $a_2 \equiv b_2 \pmod{m}$, then*

(1) $a_1 + a_2 \equiv b_1 + b_2 \pmod{m}$

(2) $a_1 - a_2 \equiv b_1 - b_2 \pmod{m}$

(3) $a_1 a_2 \equiv b_1 b_2 \pmod{m}$.

Proof. We only check (3), leaving (1) and (2) as easy exercises. Since $a_1 \equiv b_1 \pmod{m}$ and $a_2 \equiv b_2 \pmod{m}$, we have $a_1 = b_1 + k_1 m$ and $a_2 = b_2 + k_2 m$, for some $k_1, k_2 \in \mathbb{Z}$, and so

$$\begin{aligned} a_1 a_2 - b_1 b_2 &= a_1(a_2 - b_2) + (a_1 - b_1) b_2 \\ &= (a_1 k_2 + k_1 b_2) m, \end{aligned}$$

as claimed. $\square$

Proposition 1.1.1 allows us to define addition subtraction and multiplication on equivalence classes modulo m.

If we denote by $\mathbb{Z}/m\mathbb{Z}$ the set of equivalence classes modulo m and if we write $\overline{a}$ for the equivalence class of a, then we define

$$\begin{aligned} \overline{a} + \overline{b} &= \overline{a+b} \\ \overline{a} - \overline{b} &= \overline{a-b} \\ \overline{a}\overline{b} &= \overline{ab}. \end{aligned}$$

The above make sense because $\overline{a+b}$ does not depend on the representatives chosen in the equivalence classes $\overline{a}$ and $\overline{b}$, and similarly for $\overline{a-b}$ and $\overline{ab}$.

Of course, each equivalence class, $\overline{a}$, contains a unique representative from the set of remainders,
$\{0, 1, \ldots, m-1\}$, modulo m, so the above operations are completely determined by $m \times m$ tables.

Using the arithmetic operations of $\mathbb{Z}/m\mathbb{Z}$ is called *modular arithmetic*.

Modular arithmetic is *clock arithmetic*, with a clock whose dial is subdivided into m equal sectors!

For an arbitrary m, the set $\mathbb{Z}/m\mathbb{Z}$ is an algebraic structure known as a *ring*.

Addition and subtraction behave as in $\mathbb{Z}$ but multiplication is stranger. For example, when $m = 6$,

$$\begin{aligned} 2 \cdot 3 &= 0 \\ 3 \cdot 4 &= 0, \end{aligned}$$

since $2 \cdot 3 = 6 \equiv 0 \pmod{6}$, and $3 \cdot 4 = 12 \equiv 0 \pmod{6}$.

Therefore, it is not true that every nonzero element has a multiplicative inverse.

However, *every nonzero integer, a, has a multiplicative inverse iff* $\gcd(a, m) = 1$.

For example,

$$5 \cdot 5 = 1,$$

since $5 \cdot 5 = 25 \equiv 1 \pmod{6}$.

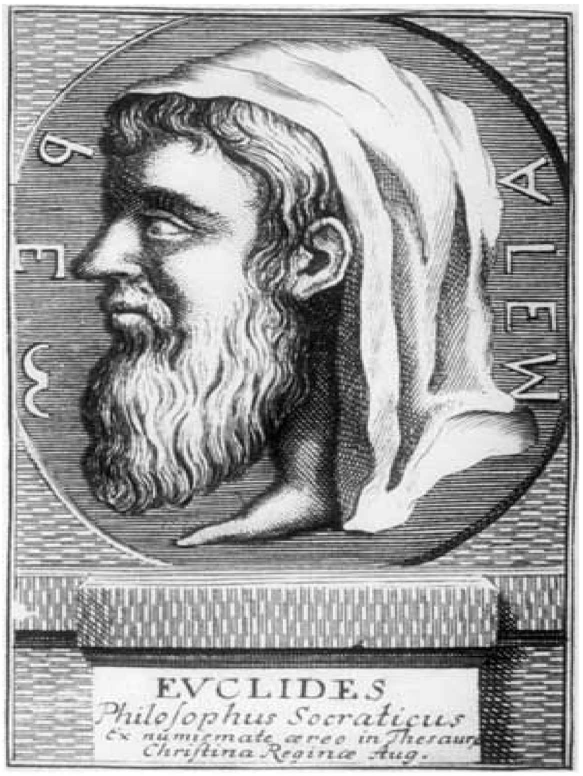

Figure 1.1: Euclid of Alexandria (about 325 BC, about 265 BC)

In order to prove the above claim, let us review *Euclidean division* and *greatest common divisors* (*gcd's*).

First, recall that Euclidean division applies to all integers, even if they are negative.

Theorem 1.1.2 *(Euclidean Division Lemma for $\mathbb{Z}$) Given any two integers, $a, b \in \mathbb{Z}$, with $b \neq 0$, there is some unique integer, $q \in \mathbb{Z}$ (the quotient), and some unique natural number, $r \in \mathbb{N}$ (the remainder or residue), so that*

$$a = bq + r \quad \textit{with} \quad 0 \leq r < |b|.$$

For example,
$-12 = 5 \cdot (-3) + 3$,
$12 = 5 \cdot 2 + 2$,
$200 = 5 \cdot 40 + 0$, and
$42823 = 6409 \times 6 + 4369$.

The remainder, r, in the Euclidean division, $a = bq + r$, of a by b, is usually denoted $a \bmod b$.

Given any two integers, $a, b \in \mathbb{Z}$, we say that *b divides a* and that *a is a multiple of b* iff $a = bc$, for some $c \in \mathbb{Z}$.

Note that we allow $b = 0$, but in this case, we must also have $a = 0$.

Using Theorem 1.1.2, it is easy to prove the following result:

Proposition 1.1.3 *For any two integers, $a, b \in \mathbb{Z}$, there is a unique natural number, $d \in \mathbb{N}$, and some integers, $u, v \in \mathbb{Z}$, so that d divides both a and b and*

$$ua + vb = d.$$

(The above is called the Bézout identity.) Furthermore, $d = 0$ iff $a = 0$ and $b = 0$.

Observe that any integer that divides both a and b also divides d. This motivates the following definition:

Definition 1.1.4 Given any two integers, $a, b \in \mathbb{Z}$, an integer, $d \in \mathbb{Z}$, is a *greatest common divisor of a and b* (for short, a *gcd of a and b*) if d divides a and b and, for any integer, $h \in \mathbb{Z}$, if h divides a and b, then h divides d. We say that a and b are *relatively prime* if 1 is a gcd of a and b.

It is easy to see that $\gcd(0, 0) = 0$.

If $a \neq 0$ or $b \neq 0$, there is a *unique positive integer*, d, which is a gcd of a and b. We call it *the* gcd of a and b and denote it $\gcd(a, b)$.

For example, $\gcd(20, 8) = 4$, $\gcd(1000, 50) = 50$, $\gcd(42823, 6409) = 17$, and $\gcd(5, 16) = 1$.

Here is a very useful characterization of the gcd:

Figure 1.2: Etienne Bézout (1730-1783)

Proposition 1.1.5 *(Bézout) Given any two integers, $a, b \in \mathbb{Z}$, a natural number, $d \in \mathbb{N}$, is the greatest common divisor of a and b iff d divides a and b and if there are some integers, $u, v \in \mathbb{Z}$, so that*

$$ua + vb = d. \tag{Bezout Identity}$$

In particular, a and b are relatively prime iff there are some integers, $u, v \in \mathbb{Z}$, so that

$$ua + vb = 1. \tag{Bezout Identity}$$

Proposition 1.1.5 shows why, given any positive integer, m, a nonzero integer, a, has a multiplicative inverse modulo m iff $\gcd(a, m) = 1$.

First, assume that $\gcd(a, m) = 1$ and let $b = m$ in the Bezout identity. Then, there exist $u, v \in \mathbb{Z}$ so that

$$ua + vm = 1,$$

that is $ua = 1 - vm$, which means that

$$ua \equiv 1 \pmod{m},$$

that is, *u is an inverse of a modulo m*.

Conversely, if u is an inverse of a modulo m, we have $ua \equiv 1 \pmod{m}$, which implies that

$$ua - km = 1,$$

for some $k \in \mathbb{Z}$ and by the Bezout identity, $\gcd(a, m) = 1$.

Also, note that if $m \geq 2$ and $\gcd(a, m) = 1$, then a has a *unique inverse (modulo m)*, $u \in \mathbb{Z}$, such that $1 \leq u \leq m - 1$.

Indeed, if $\gcd(a, m) = 1$, then we know that there are some $u, v \in \mathbb{Z}$ so that

$$ua + vm = 1.$$

Using the Euclidean division, write

$$u = mq + r,$$

with $0 \leq r < m$.

Actually, $r \neq 0$, since otherwise, m would divide both ua and vm and thus, it would divide 1, which is impossible since $m \geq 2$.

Then, we have

$$1 = ua + vm = (mq + r)a + vm = ra + (qa + v)m,$$

with $1 \leq r < m$, as claimed.

The gcd of two natural numbers can be found using a method involving Euclidean division and so can the numbers u and v

This method is based on the following simple observation:

Proposition 1.1.6 *If a, b are any two positive integers with $a \geq b$, then for every $k \in \mathbb{Z}$,*

$$\gcd(a, b) = \gcd(b, a - kb).$$

In particular,

$$\gcd(a, b) = \gcd(b, a - b) = \gcd(b, a + b),$$

and if $a = bq + r$ is the result of performing the Euclidean division of a by b, with $0 \leq r < b$, then

$$\gcd(a, b) = \gcd(b, r).$$

Euclidean Algorithm for Finding the gcd.

The input consists of two natural numbers, m, n, with $(m, n) \neq (0, 0)$.

```
begin
  $a := m$; $b := n$;
  if $a < b$ then
    $t := b$; $b := a$; $a := t$; (swap $a$ and $b$)
  while $b \neq 0$ do
    $r := a \bmod b$; (divide $a$ by $b$; get remainder $r$)
    $a := b$; $b := r$
  endwhile;
  $\gcd(m, n) := a$
end
```

There is an extension of the above algorithm that computes a pair of integers, (u, v), so that

$$um + vn = \gcd(m, n).$$

This extension is the *Extended Euclidean Algorithm*.

As a consequence of all this, *when m is a prime number, every nonzero element not divisible by m has a multiplicative inverse*.

In this case, $\mathbb{Z}/m\mathbb{Z}$ is more like $\mathbb{Q}$, it is a finite *field*.

However, note that in $\mathbb{Z}/m\mathbb{Z}$ we have

$$\underbrace{1 + 1 + \cdots + 1}_{m \text{ times}} = 0$$

(since $m \equiv 0 \pmod{m}$), a phenomenom that does not happen in $\mathbb{Q}$ (or $\mathbb{R}$).

The RSA methods uses modular arithmetic.

One of the main ingredients of public key cryptography is that one should use an encryption function, $f\colon \mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$, which is *easy to compute* ($i.e$, can be computed efficiently) but such that its inverse, f^{-1}, is *practically impossible to compute* unless one has *special additional information*.

Such functions are usually referred to as *trapdoor one-way functions*.

Remarkably, exponentiation modulo m, that is, the function, $x \mapsto x^e \bmod m$, is a trapdoor one-way function for suitably chosen m and e.

Thus, we claim that

(1) Computing $x^e \bmod m$ can be done *efficiently*

(2) Finding x such that

$$x^e \equiv y \pmod{m}$$

with $0 \leq x, y \leq m - 1$, is *hard*, unless one has extra information about m.

The function that finds an $e^{\rm th}$ root modulo m is sometimes called a *discrete logarithm*.

We will explain shortly how to compute $x^e \bmod m$ efficiently using the *square and multiply* method also known as *repeated squaring*.

As to the second claim, actually, *no proof has been given yet that this function is a one-way function* but, so far, this has not been refuted either.

Now, what's the trick to make it a trapdoor function?

What we do is to pick two *distinct large prime numbers*, p and q (say over 200 decimal digits), which are "sufficiently random" and we let

$$m = pq.$$

Next, we pick a random, e, with $1 < e < (p-1)(q-1)$, relatively prime to $(p-1)(q-1)$.

Since $\gcd(e, (p-1)(q-1)) = 1$, we know that there is some d, with $1 < d < (p-1)(q-1)$, such that

$$ed \equiv 1 \pmod{(p-1)(q-1)}.$$

Then, we claim that to find x such that

$$x^e \equiv y \pmod{m},$$

we compute $y^d \bmod m$, and this can be done easily, as we claimed earlier.

The reason why the above "works" is that

$$x^{ed} \equiv x \pmod{m}, \qquad (*)$$

for all $x \in \mathbb{Z}$, which we will prove later.

Setting up RSA

In, summary to set up RSA for Albert (A) to receive encrypted messages, perform the following steps:

1. Albert generates two distinct large and sufficiently random primes, p_A and q_A. They are kept secret.
2. Albert computes $m_A = p_A q_A$. This number called the *modulus* will be made public.
3. Albert picks at random some e_A, with $1 < e_A < (p_A - 1)(q_A - 1)$, so that $\gcd(e_A, (p_A - 1)(q_A - 1)) = 1$. The number e_A is called the *encryption key* and it will also be public.
4. Albert computes the inverse, $d_A = e_A^{-1}$ modulo m_A, of e_A. This number is kept secret. The pair, (d_A, m_A), is Albert's *private key* and d_A is called the *decryption key*.
5. Albert publishes the pair, (e_A, m_A), as his *public key*.

Encrypting a message

Now, if Julia wants to send a message, x, to Albert, she proceeds as follows: First, she splits x into chunks, $x_1, \ldots, x_k$, each of length at most $m_A - 1$, if necessary (again, I assume that x has been converted to an integer in a preliminary step).

Then, she looks up Albert's public key, (e_A, m_A), and she computes

$$y_i = E_A(x_i) = x_i^{e_A} \bmod m_A,$$

for $i = 1, \ldots, k$.

Finally, she sends the sequence $y_1, \ldots, y_k$ to Albert. This encrypted message is known as the *cyphertext*.

The function E_A is Albert's *encryption function*.

Decrypting a message

In order to decrypt the message, $y_1, \ldots, y_k$, that Julia sent him, Albert uses his private key, (d_A, m_A), to compute each

$$x_i = D_A(y_i) = y_i^{d_A} \bmod m_A,$$

and this yields the sequence $x_1, \ldots, x_k$.

The function D_A is Albert's *decryption function*.

Similarly, in order for Julia to receive encrypted messages, she must set her own public key, (e_J, m_J), and private key, (d_J, m_J), by picking two distinct primes p_J and q_J and e_J, as explained earlier.

The beauty of the scheme is that the sender only needs to know the public key of the recipient to send a message but an eavesdropper is unable to decrypt the encoded message unless he somehow gets his hands on the secret key of the receiver.

Let us give a concrete illustration of the RSA scheme using an example borrowed from Silverman (Chapter 18).

We will write messages using only the 26 upper-case letters A, B, ..., Z, encoded as the integers A $= 11$, B $= 12, \ldots$, Z $= 36$.

It would be more convenient to have assigned a number to represent a blank space but to keep things as simple as possible we will not do that.

Say Albert pick the two primes, $p_A = 12553$ and $q_A = 13007$, so that $m_A = p_A q_A = 163,276,871$ and $(p_A - 1)(q_A - 1) = 163,251,312$.

Albert also pick $e_A = 79921$, relatively prime to $(p_A - 1)(q_A - 1)$ and then finds the inverse, d_A, of e_A modulo $(p_A - 1)(q_A - 1)$ using the extended Euclidean Algorithm (more details will be given in Section 1.3) which turns out to be $d_A = 145,604,785$.

One can check that

$$145,604,785 \cdot 79921 - 71282 \cdot 163,251,312 = 1,$$

which confirms that d_A is indeed the inverse of e_A modulo $163,251,312$.

Now, assume that Albert receives the following message, broken in chunks of at most 9 digits, since $m_A = 163,276,871$ has 9 nine digits:

$$145387828 \quad 47164891 \quad 152020614 \quad 27279275 \quad 35356191.$$

Albert decrypts the above messages using his private key, (d_A, m_A), where $d_A = 145,604,785$, using the repeated squaring method (described in Section 1.3) and finds that

$$\begin{aligned}
145387828^{145,604,785} &\equiv 30182523 \pmod{163,276,871} \\
47164891^{145,604,785} &\equiv 26292524 \pmod{163,276,871} \\
152020614^{145,604,785} &\equiv 19291924 \pmod{163,276,871} \\
27279275^{145,604,785} &\equiv 30282531 \pmod{163,276,871} \\
35356191^{145,604,785} &\equiv 122215 \pmod{163,276,871}
\end{aligned}$$

which yields the message

30182523 26292524 19291924 30282531 122215,

and finally, translating each two digits numeric code to its corresponding character, to the message

T H O M P S O N I S I N T R O U B L E

or, in more readable format

Thompson is in trouble

It would be instructive to encrypt the decoded message

30182523 26292524 19291924 30282531 122215

using the public key, $e_A = 79921$. If everything goes well, we should get our original message

145387828 47164891 152020614 27279275 35356191

back!

Let us now explain in more detail how the RSA system works and why it is correct.

1.2 Correctness of The RSA System

We begin by proving the correctness of the inversion formula $(*)$.

For this, we need a classical result known as *Fermat's Little Theorem*.

This result was first stated by Fermat in 1640 but apparently no proof was published at the time and the first known proof was given by Leibnitz (1646-1716).

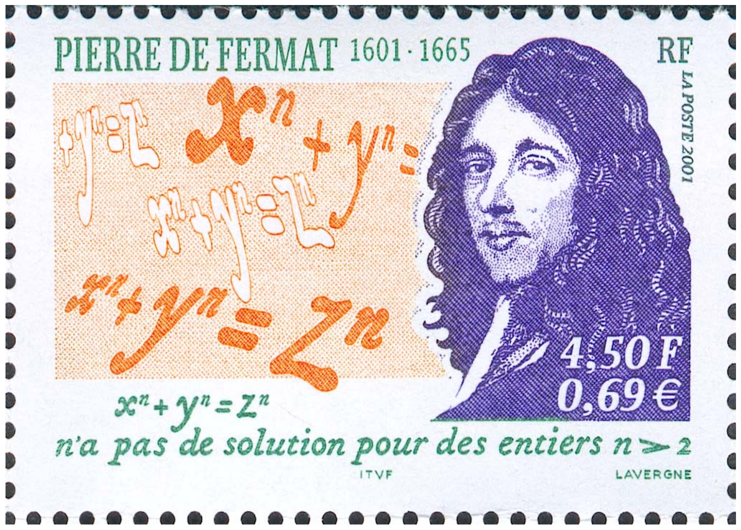

Figure 1.3: Pierre de Fermat (1601-1665) and Leonhard Euler (1707-1783)

Euler also proved Fermat's Little Theorem and generalized it. A different proof was given by Ivory in 1806.

This proof has the advantage that it can be easily generalized to Euler's version (1760) of Fermat's Little Theorem.

Theorem 1.2.1 *(Fermat's Little Theorem) If p is any prime number, then the following two equivalent properties hold:*

(1) For every integer, $a \in \mathbb{Z}$, if a is not divisible by p, then we have

$$a^{p-1} \equiv 1 \pmod{p}.$$

(2) For every integer, $a \in \mathbb{Z}$, we have

$$a^{p} \equiv a \pmod{p}.$$

Proof. See the notes on the web site for CIS511.

It is now easy to establish the correctness of RSA.

Proposition 1.2.2 *For any two distinct prime numbers, p and q, if e and d are any two positive integers such that,*

1. $1 < e, d < (p-1)(q-1)$,
2. $ed \equiv 1 \pmod{(p-1)(q-1)}$,

then for every $x \in \mathbb{Z}$, we have

$$x^{ed} \equiv x \pmod{pq}.$$

Proof. See the notes on the web site for CIS511.

Remark: Of course, Proposition 1.2.2 holds if we allow $e = d = 1$, but this not interesting for encryption.

The number $(p-1)(q-1)$ turns out to be the number of positive integers less than pq which are relatively prime to pq.

For any arbitrary positive integer, m, the number of positive integers less than m which are relatively prime to m is given by the *Euler ϕ function* (or *Euler totient*), denoted ϕ.

Fermat's Little Theorem can be generalized to what is known as *Euler's Formula*:

For every integer, a, if $\gcd(a, m) = 1$, then

$$a^{\phi(m)} \equiv 1 \pmod{m}.$$

Since $\phi(pq) = (p-1)(q-1)$, when $\gcd(x, \phi(pq)) = 1$, Proposition 1.2.2 follows from Euler's Formula.

However, that argument does not show that Proposition 1.2.2 holds when $\gcd(x, \phi(pq)) > 1$ and a special argument is required in this case.

It can be shown that if we replace pq by a positive integer, m, that is square-free (does not contain a square factor) and if we assume that e and d are chosen so that
$1 < e, d < \phi(m)$ and $ed \equiv 1 \pmod{\phi(m)}$, then

$$x^{ed} \equiv x \pmod{m}$$

for all $x \in \mathbb{Z}$.

We see no great advantage in using this fancier argument and this is why we used the more elementary proof based on Fermat's Little Theorem.

Proposition 1.2.2 immediately implies that the decrypting and encryting RSA functions D_A and E_A are mutual inverses, for any A.

Furthermore, E_A is easy to compute but, without extra information, namely, the trapdoor d_A, it is practically impossible to compute $D_A = E_A^{-1}$.

That D_A is hard to compute without a trapdoor is related to the fact that *factoring a large number, such as m_A, into its factors, p_A and q_A, is hard*.

Today, it is practically impossible to factor numbers over 300 decimal digits long.

Although no proof has been given so far, it is believed that factoring will remain a hard problem.

So, even if in the next few years it becomes possible to factor 300 digits numbers, it will still be impossible to factor 400 digits numbers.

RSA has the peculiar property that it depends both on the fact that primality testing is easy but that factoring is hard. What a stroke of genius!

1.3 Algorithms for Computing Powers and Inverses modulo m

First, we explain how to compute $x^n \bmod m$ efficiently, where $n \geq 1$. Let us first consider computing the n^{th} power, x^n, of some positive integer.

The idea is to look at the *parity* of n and to proceed recursively.

If n is *even*, say $n = 2k$, then

$$x^n = x^{2k} = (x^k)^2,$$

so, compute x^k recursively and then square the result.

If n is *odd*, say $n = 2k + 1$, then

$$x^n = x^{2k+1} = (x^k)^2 \cdot x,$$

so, compute x^k recursively, square it, and multiply the result by x.

What this suggests is to write $n \geq 1$ in binary, say

$$n = b_\ell \cdot 2^\ell + b_{\ell-1} \cdot 2^{\ell-1} + \cdots + b_1 \cdot 2^1 + b_0,$$

where $b_i \in \{0, 1\}$ with $b_\ell = 1$ or, if we let
$J = \{j \mid b_j = 1\}$, as

$$n = \sum_{j \in J} 2^j.$$

Then, we have

$$x^n \equiv x^{\sum_{j \in J} 2^j} = \prod_{j \in J} x^{2^j} \bmod m.$$

This suggests computing the residues, r_j, such that

$$x^{2^j} \equiv r_j \pmod{m},$$

because then,

$$x^n \equiv \prod_{j \in J} x_j \pmod{m},$$

where we can compute this latter product modulo m two terms at a time.

For example, say we want to compute $999^{179} \bmod 1763$.

First, we observe that

$$179 = 2^7 + 2^5 + 2^4 + 2^1 + 1,$$

and we compute the powers modulo 1763:

$$\begin{aligned}
999^{2^1} &\equiv 143 \pmod{1763}\\
999^{2^2} &\equiv 143^2 \equiv 1056 \pmod{1763}\\
999^{2^3} &\equiv 1056^2 \equiv 920 \pmod{1763}\\
999^{2^4} &\equiv 920^2 \equiv 160 \pmod{1763}\\
999^{2^5} &\equiv 160^2 \equiv 918 \pmod{1763}\\
999^{2^6} &\equiv 918^2 \equiv 10 \pmod{1763}\\
999^{2^7} &\equiv 10^2 \equiv 100 \pmod{1763}.
\end{aligned}$$

Consequently,

$$\begin{aligned}
999^{179} &\equiv 999 \cdot 143 \cdot 160 \cdot 918 \cdot 100 \pmod{1763}\\
&\equiv 54 \cdot 160 \cdot 918 \cdot 100 \pmod{1763}\\
&\equiv 1588 \cdot 918 \cdot 100 \pmod{1763}\\
&\equiv 1546 \cdot 100 \pmod{1763}\\
&\equiv 1219 \pmod{1763},
\end{aligned}$$

and we find that

$$999^{179} \equiv 1219 \pmod{1763}.$$

Of course, it would be impossible to exponentiate 999^{179} first and then reduce modulo 1763.

As we can see, the number of multiplications needed is $O(\log_2 n)$, which is quite good.

The above method can be implemented without actually converting n to base 2.

If n is even, say $n = 2k$, then $n/2 = k$ and if n is odd, say $n = 2k + 1$, then $(n - 1)/2 = k$, so we have a way of dropping the unit digit in the binary expansion of n and shifting the remaining digits one place to the right without explicitly computing this binary expansion.

Here is an algorithm for computing $x^n \bmod m$, with $n \geq 1$, using the *repeated squaring* method.

An Algorithm to Compute $x^n \bmod m$ Using Repeated Squaring

```
begin
  $u := 1$; $a := x$;
  while $n > 1$ do
    if even($n$) then $e := 0$ else $e := 1$;
    if $e = 1$ then $u := a \cdot u$ mod $m$;
    $a := a^2$ mod $m$; $n := (n - e)/2$
  endwhile;
  $u := a \cdot u$ mod $m$
end
```

The final value of u is the result. The reason why the algorithm is correct is that after j rounds through the while loop, $a = x^{2^j} \bmod m$ and

$$u = \prod_{i \in J \mid i < j} x^{2^i} \bmod m,$$

with this product interpreted as 1 when $j = 0$.

Observe that the while loop is only executed $n-1$ times to avoid squaring once more unnecessarily and the last multiplication $a \cdot u$ is performed outside of the while loop.

Also, if we delete the reductions modulo m, the above algorithm is a fast method for computing the $n^{\rm th}$ power of an integer x and the time speed-up of not performing the last squaring step is more significant.

Let us now consider the problem of computing efficiently the inverse of an integer, a, modulo m, provided that $\gcd(a, m) = 1$.

We mentioned in Section 1.1 how the Extended Euclidean Algorithm can be used to find some integers, x, y, such that

$$ax + by = \gcd(a, b),$$

where a and b are any two positive integers.

In our situation, $a = m$ and $b = a$ and we only need to find y (we would like a positive integer).

When using the Euclidean algorithm for computing $\gcd(m, a)$, with $2 \leq a < m$, we compute the following sequence of quotients and remainders:

$$\begin{aligned}
m &= aq_1 + r_1 \\
a &= r_1q_2 + r_2 \\
r_1 &= r_2q_3 + r_3 \\
\vdots &\quad \vdots \\
r_{k-1} &= r_kq_{k+1} + r_{k+1} \\
\vdots &\quad \vdots \\
r_{n-3} &= r_{n-2}q_{n-1} + r_{n-1} \\
r_{n-2} &= r_{n-1}q_n + 0,
\end{aligned}$$

with $n \geq 3$, $0 < r_1 < b$, $q_k \geq 1$, for $k = 1, \ldots, n$, and $0 < r_{k+1} < r_k$, for $k = 1, \ldots, n-2$. Observe that $r_n = 0$.

If $n = 2$, we just have two divisions,

$$\begin{aligned}
m &= aq_1 + r_1 \\
a &= r_1q_2 + 0
\end{aligned}$$

with $0 < r_1 < b$, $q_1, q_2 \geq 1$ and $r_2 = 0$.

Thus, it is convenient to set $r_{-1} = m$ and $r_0 = a$.

It can be shown that if we set

$$\begin{aligned} x_{-1} &= 1 \\ y_{-1} &= 0 \\ x_0 &= 0 \\ y_0 &= 1 \\ x_{i+1} &= x_{i-1} - x_i q_{i+1} \\ y_{i+1} &= y_{i-1} - y_i q_{i+1}, \end{aligned}$$

for $i = 0, \ldots, n-2$, then

$$m x_{n-1} + a y_{n-1} = \gcd(m, a) = r_{n-1},$$

and so, if $\gcd(m, a) = 1$, then $r_{n-1} = 1$ and we have

$$a y_{n-1} \equiv 1 \pmod{m}.$$

Now, y_{n-1} may be greater than m or negative but we already know how to deal with that.

This suggests reducing modulo m during the recurrence and we are led to the following recurrence:

$$\begin{aligned} y_{-1} &= 0 \\ y_0 &= 1 \\ z_{i+1} &= y_{i-1} - y_i q_{i+1} \\ y_{i+1} &= z_{i+1} \bmod m \quad \text{if} \quad z_{i+1} \geq 0 \\ y_{i+1} &= m - ((-z_{i+1}) \bmod m) \quad \text{if} \quad z_{i+1} < 0, \end{aligned}$$

for $i = 0, \ldots, n-2$.

It is easy to prove by induction that

$$ay_i \equiv r_i \pmod{m}$$

for $i = 0, \ldots, n-1$ and thus, if $\gcd(a, m) > 1$, then a does not have an inverse modulo m, else

$$ay_{n-1} \equiv 1 \pmod{m}$$

and y_{n-1} is the inverse of a modulo m such that
$1 \leq y_{n-1} < m$, as desired.

Note that we also get $y_0 = 1$ when $a = 1$.

An Algorithm for Computing the Inverse of a modulo m (where $1 \leq a < m$ and $\gcd(a, m) = 1$).

```
begin
  $y := 0$; $v := 1$; $g := m$; $r := a$;
  $pr := r$; $q := \lfloor g/pr \rfloor$; $r := g - pr\, q$; (divide $g$ by $pr$)
  if $r = 0$ then
    $y := 1$; $g := pr$
  else
    $r = pr$;
    while $r \neq 0$ do
      $pr := r$; $pv := v$;
      $q := \lfloor g/pr \rfloor$; $r := g - pr\, q$; (divide $g$ by $pr$)
      $v := y - pv\, q$;
      if $v < 0$ then
        $v := m - ((-v) \bmod m)$
      else
        $v = v \bmod m$
      endif
      $g := pr$; $y := pv$
    endwhile;
  endif;
  inverse($a$) $:= y$
end
```

For example, we used the above algorithm to find that $d_A = 145,604,785$ is the inverse of $e_A = 79921$ modulo $(p_A - 1)(q_A - 1) = 163,251,312$.

The remaining issues are how to choose large random prime numbers, p, q, and how to find a random number, e, which is relatively prime to $(p-1)(q-1)$.

For this, we rely on a deep result of number theory known as the *Prime Number Theorem*.

1.4 Finding Large Primes; Signatures; Safety of RSA

Roughly speaking, the Prime Number Theorem ensures that the density of primes is high enough to guarantee that there are many primes with a large specified number of digits.

The relevant function is the *prime counting function, $\pi(n)$*.

Definition 1.4.1 The *prime counting function*, π, is the function defined so that

$$\pi(n) = \text{number of prime numbers, } p, \text{ such that } p \leq n,$$

for every natural number, $n \in \mathbb{N}$.

Obviously, $\pi(0) = \pi(1) = 0$. We have $\pi(10) = 4$ since the primes no greater than 10 are $2, 3, 5, 7$ and $\pi(20) = 8$ since the primes no greater than 20 are
$2, 3, 5, 7, 11, 13, 17, 19$.

The growth of the function π was studied by Legendre, Gauss, Tschebycheff and Riemann between 1808 and 1859.

By then, it was conjectured that

$$\pi(n) \sim \frac{n}{\ln(n)},$$

for n large, which means that

$$\lim_{n \mapsto \infty} \pi(n) \Bigg/ \frac{n}{\ln(n)} = 1.$$

However, a rigorous proof had to wait until 1896. Indeed, in 1896, Jacques Hadamard and Charles de la Vallée-Poussin independendly gave a proof of this "most wanted theorem," using methods from complex analysis.

These proofs are difficult and although more elementary proofs were given later, in particular by Erdös and Selberg (1949), those proofs are still quite hard.

Thus, we content ourselves with a statement of the theorem.

Theorem 1.4.2 *(Prime Number Theorem) For n large, the number of primes, $\pi(n)$, no larger than n is approximately equal to $n/\ln(n)$, which means that*

$$\lim_{n\mapsto\infty} \pi(n) \bigg/ \frac{n}{\ln(n)} = 1.$$

For a rather detailed account of the history of the Prime Number Theorem (for short, *PNT*), we refer the reader to Ribenboim (Chapter 4).

As an illustration of the use of the PNT, we can estimate the number of primes with 200 decimal digits.

Indeed this is the difference of the number of primes up to 10^{200} minus the number of primes up to 10^{199}, which is approximately

$$\frac{10^{200}}{200 \ln 10} - \frac{10^{199}}{199 \ln 10} \approx 1.95 \cdot 10^{197}.$$

Thus, we see that there is a huge number of primes with 200 decimal digits.

Since the number of natural numbers with 200 digits is $10^{200} - 10^{199} = 9 \cdot 10^{199}$, the proportion of 200 digits numbers that are prime is

$$\frac{1.95 \cdot 10^{197}}{9 \cdot 10^{199}} \approx \frac{1}{460}.$$

Consequently, among the natural numbers with 200 digits, roughly one in every 460 is a prime!

Beware that the above argument is not entirely rigorous because the Prime Number Theorem only yields an approximation of $\pi(n)$ but sharper estimates can be used to say how large n should be to guarantee a prescribed error on the probability, say 1%.

The implication of the above fact is that if we wish to find a random prime with 200 digits, we pick at random some natural number with 200 digits and test whether it is prime.

If this number is not prime, then we discard it and try again, and so on. On the average, after 460 trials, a prime should pop up!

This leads us the question: *How do test for primality*?

Primality testing has also been studied for a long time.

Remarkably, *Fermat's Little Theorem yields a test for non-primality*.

Indeed, if $p > 1$ fails to divide $a^{p-1} - 1$ for some natural number, a, where $2 \leq a \leq p-1$, then p cannot be a prime.

The simplest a to try is $a = 2$. From a practical point of view, we can compute $a^{p-1} \bmod p$ using the method of repeated squaring and check whether the remainder is 1.

But what if p fails the Fermat test?

Unfortunately, there are natural numbers, p, such that p divides $2^{p-1} - 1$ and yet, p is composite.

For example $p = 341 = 11 \cdot 31$ is such a number.

Actually, 2^{340} being quite big, how do we check that $2^{340} - 1$ is divisible by 341?

We just have to show that $2^{340} - 1$ is divisible by 11 and by 31. We can use Fermat's Little Theorem!

Since 11 is prime, we know that 11 divides $2^{10} - 1$. But,

$$2^{340} - 1 = (2^{10})^{34} - 1 = (2^{10} - 1)((2^{10})^{33} + (2^{10})^{32} + \cdots + 1),$$

so $2^{340} - 1$ is also divisible by 11.

As to divisibility by 31, observe that $31 = 2^5 - 1$, and

$$2^{340} - 1 = (2^5)^{68} - 1 = (2^5 - 1)((2^5)^{67} + (2^5)^{66} + \cdots + 1),$$

so $2^{340} - 1$ is also divisible by 31.

A number, p, which is not a prime but behaves like a prime in the sense that p divides $2^{p-1} - 1$, is called a *pseudo-prime*.

Unfortunately, the Fermat test gives a “false positive” for pseudo-primes.

Rather than simply testing whether $2^{p-1} - 1$ is divisible by p, we can also try whether $3^{p-1} - 1$ is divisible by p and whether $5^{p-1} - 1$ is divisible by p, *etc*.

Unfortunately, there are composite natural numbers, p, such that p divides $a^{p-1} - 1$, for all positive natural numbers, a, with $\gcd(a, p) = 1$.

Such numbers are known as *Carmichael numbers*.

The smallest Carmichael number is $p = 561 = 3 \cdot 11 \cdot 17$.

The reader should try proving that, in fact, $a^{560} - 1$ is divisible by 561 for every positive natural number, a, such that $\gcd(a, 561) = 1$, using the technique that we used to prove that 341 divides $2^{340} - 1$.

It turns out that there are infinitely many Carmichael numbers.

Again, for a thorough introduction to primality testing, pseudo-primes, Carmichael numbers and more, we highly recommend Ribenboim (Chapter 2).

An excellent (but more terse) account is also given in Koblitz (Chapter V).

Still, what do we do about the problem of false positives?

The key is to switch to *probabilistic methods*.

Indeed, if we can design a method which is guaranteed to give a false positive with probablity less than 0.5, then we can *repeat* this test for randomly chosen a's and reduce the probability of false positive considerably.

For example, if we repeat the experiment 100 times, the probability of false positive is less than $2^{-100} < 10^{-30}$.

This is probably *less than the probability of hardware failure*!

Various probabilistic methods for primality testing have been designed.

One of them is the *Miller-Rabin test*, another the *APR test*, yet another the *Solovay-Strassen test*.

Since 2002, it is known that *primality testing can be done in polynomial time*.

This result, due to Agrawal, Kayal and Saxena and known as the *AKS test* solved a long-standing problem, see Dietzfelbinger and Crandall and Pomerance (Chapter 4).

Remarkably, Agrawal and Kayal worked on this problem for their senior project in order to complete their bachelor's degree!

It remains to be seen whether this test is really practical for very large numbers.

A very important point to make is that these primality testing methods *do not* provide a factorization of m when m is composite.

This is actually a crucial ingredient for the security of the RSA scheme.

So far, it appears (and it is hoped!) that *factoring* an integer is a much harder problem than testing for primality and all known methods are incapable of factoring natural numbers with over 300 decimal digits (it would take centuries).

For a comprehensive exposition of the subject of primality-testing, we refer the reader to Crandall and Pomerance (Chapter 4) and again, to Ribenboim (Chapter 2) and Koblitz (Chapter V).

Going back to the RSA method, we now have ways of finding the large random primes p and q by picking at random some 200 digits numbers and testing for primality.

Rivest, Shamir and Adleman also recommend to pick p and q so that they differ by a few decimal digits, that both $p-1$ and $q-1$ should contain large prime factors and that $\gcd(p-1, q-1)$ should be small.

The public key, e, relatively prime to $(p-1)(q-1)$ can also be found by a similar method:

Pick at random a number, $e < (p-1)(q-1)$, which is large enough (say, greater than $\max\{p, q\}$) and test whether $\gcd(e, (p-1)(q-1)) = 1$, which can be done quickly using the Extended Euclidean Algorithm.

If not, discard e and try another number, *etc*.

It is easy to see that such an e will be found in no more trials than it takes to find a prime.

If Albert receives a message coming from Julia, how can he be sure that this message does not come from an imposter?

Just because the message is signed “Julia” does not mean that it comes from Julia; it could have been sent by someone else pretending to be Julia, since all that is needed to send a message to Albert is Albert’s public key, which is known to everybody.

This leads us to the issue of *signatures*.

There are various schemes for adding a signature to an encrypted message to ensure that the sender of a message is really who he/she claims to be (with a high degree of confidence).

The trick is to make use of the keys of the sender.

We propose two scenarios:

1. The sender, Julia, encrypts the message, x, to be sent with *her own private key*, (d_J, m_J), creating the message $D_J(x) = y_1$.

   Then, Julia adds her signature, “Julia”, at the end of the message y_1, encrypts the message “y_1 Julia” using *Albert’s public key*, (e_A, m_A), creating the message $y_2 = E_A(y_1\,\text{Julia})$, and finally sends the message y_2 to Albert.

   When Albert receives the encrypted message, y_2, claiming to come from *Julia*, first he decrypts the message using *his private key*, (d_A, m_A).

   He will see an encrypted message, $D_A(y_2) = y_1\,\text{Julia}$, with the legible signature, *Julia*.

   He will then delete the signature from this message and decrypts the message y_1 using *Julia’s public key*, (e_J, m_J), getting $x = E_J(y_1)$.

Albert will know whether someone else faked this message if the result is garbage. Indeed, only Julia could have encrypted the original message, x, with her private key, which is only known to her.

An eavesdropper who is pretending to be Julia would not know Julia's private key and so, would not have encrypted the original message to be sent using Julia's secret key.

2. The sender, Julia, first adds her signature, “Julia”, to the message, x, to be sent and then, she encrypts the message “x Julia” with *Albert’s public key*, (e_A, m_A), creating the message $y_1 = E_A(x \text{ Julia})$.

   Julia also encrypts the original message, x, using *her private key*, (d_J, m_J) creating the message $y_2 = D_J(x)$, and finally she sends the pair of messages (y_1, y_2).

   When Albert receives a pair of messages, (y_1, y_2), claimed to have been sent by Julia, first Albert decrypts, y_1 using *his private key*, (d_A, m_A), getting the message $D_A(y_1) = x \text{ Julia}$.

   Albert finds the signature, Julia, and then decrypts y_2 using *Julia’s public key*, (e_J, m_J), getting the message $x' = E_J(y_2)$.

   If $x = x'$, then Albert has serious assurance that the sender is indeed Julia and not an imposter.

The last topic that we would like to discuss is the *security* of the RSA scheme.

This is a difficult issue and many researchers have worked on it. As we remarked earlier, the security of RSA hinges on the fact that factoring is hard.

It has been shown that if one has a method for breaking the RSA scheme (namely, to find the secret key, d), then there is a probabilistic method for finding the factors, p and q, of $m = pq$ (see Koblitz, Chapter IV, Section 2, or Menezes van Oorschot and Vanstone's, Section 8.2.2).

If p and q are chosen to be large enough, factoring $m = pq$ will be practically impossible and so, it is unlikely that RSA can be cracked.

However, there may be other attacks and, at the present, there is no proof that RSA is fully secure.

Observe that since $m = pq$ is known to everybody, if somehow one can learn $N = (p-1)(q-1)$, then p and q can be recovered. Indeed

$$N = (p-1)(q-1) = pq - (p+q) + 1 = m - (p+q) + 1$$

and so,

$$\begin{aligned} pq &= m \\ p+q &= m - N + 1, \end{aligned}$$

and p and q are the roots of the quadratic equation

$$X^2 - (m - N + 1)X + m = 0.$$

Thus, a line of attack is to try to find the value of $(p-1)(q-1)$.

For more on the security of RSA, see Menezes van Oorschot and Vanstone's Handbook.