$$
\begin{aligned}
916 \bmod 2^5 - 1 &= 1110010100_2 \pmod{2^5 - 1} \\
&= 10100_2 + 11100_2 \pmod{2^5 - 1} \\
&= 110000_2 \pmod{2^5 - 1} \\
&= 10000_2 + 1_2 \pmod{2^5 - 1} \\
&= 10001_2 \pmod{2^5 - 1} \\
&= 10001_2 \\
&= 17.
\end{aligned}
$$

The Lucas–Lehmer test applied to $N = 127 = 2^7 - 1$ yields the following steps, if we denote $S_k \bmod 2^p - 1$ by r_k.

$r_0 = 4$,
$r_1 = 4^2 - 2 = 14 \pmod{127}$; that is, $r_1 = 14$.
$r_2 = 14^2 - 2 = 194 \pmod{127}$; that is, $r_2 = 67$.
$r_3 = 67^2 - 2 = 4487 \pmod{127}$; that is, $r_3 = 42$.
$r_4 = 42^2 - 2 = 1762 \pmod{127}$; that is, $r_4 = 111$.
$r_5 = 111^2 - 2 = 12319 \pmod{127}$; that is, $r_5 = 0$.

As $r_5 = 0$, the Lucas–Lehmer test confirms that $N = 127 = 2^7 - 1$ is indeed prime.

5.9 Public Key Cryptography; 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). We highly recommend reading the orginal paper by Rivest, Shamir, and Adleman [14]. It is beautifully written and easy to follow. A very clear, but concise exposition can also be found in Koblitz [9]. An encyclopedic coverage of cryptography can be found in Menezes, van Oorschot, and Vanstone's *Handbook* [11].

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 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 remain secret. Interestingly, although one might think that the mathematics behind such a scheme is very advanced and complicated, this is not so. In fact, little more than the material of Section 5.4 is needed. Therefore, in this section, we are going to explain the basics of 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 assume that this conversion has been performed.

The next and more subtle step is to use modular arithmetic. We pick a (large) positive integer m and perform arithmetic modulo m. Let us explain this step in more detail.

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*. We already know that congruence is an equivalence relation but it also satisfies the following properties.

Proposition 5.24. *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. Because $a_1 \equiv b_1 \pmod{m}$ 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

$$a_1 a_2 = (b_1 + k_1 m)(b_2 + k_2 m) = b_1 b_2 + (b_1 k_2 + k_1 b_2 + k_1 m k_2)m,$$

which means that $a_1 a_2 \equiv b_1 b_2 \pmod{m}$. A more elegant proof consists in observing that

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

as claimed. $\square$

Proposition 5.24 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*.

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 $m = 6$,

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

inasmuch as $2 \cdot 3 = 6 \equiv 0 \pmod{6}$, and $3 \cdot 4 = 12 \equiv 0 \pmod{6}$. Therefore, it is not true that every nonzero element has a multiplicative inverse. However, we know from Section 5.4 that a nonzero integer a has a multiplicative inverse iff $\gcd(a, m) = 1$ (use the Bézout identity). For example,

$$5 \cdot 5 = 1,$$

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

As a consequence, 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

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

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

The RSA method 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 the following.

(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 eth root modulo m is sometimes called a *discrete logarithm*.

We 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 $(p-1)(q-1)$.

Because $\gcd(e,(p-1)(q-1)) = 1$, we know from the discussion just before Theorem 5.10 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 simply compute $y^d \bmod m$, and this can be done easily, as we claimed earlier. The reason why the above "works" is that

$$x^{ed} \equiv x \pmod{m}, \tag{$*$}$$

for all $x \in \mathbb{Z}$, which we 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 [15] (Chapter 18). We 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 do not do that.

Say Albert picks 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 picks $e_A = 79921$, 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 are given in Section 5.11) 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 nine digits, because $m_A = 163{,}276{,}871$ has nine digits.

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

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 5.11) and finds that

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

$$35356191^{145,604,785} \equiv 122215 \pmod{163,276,871}$$

which yields the message

30182523 26292524 19291924 30282531 122215,

and finally, translating each two-digit 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.

5.10 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). This is basically the proof suggested in Problem 5.14. A different proof was given by Ivory in 1806 and this is the proof that w give here. It has the advantage that it can be easily generalized to Euler's version (1760) of Fermat's little theorem.

Theorem 5.14. *(Fermat's Little Theorem) If p is any prime number, then the following two equivalent properties hold.*

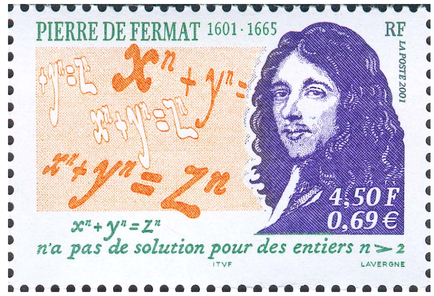

Fig. 5.17 Pierre de Fermat, 1601–1665

(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. (1) Consider the integers

$$a,\, 2a,\, 3a,\, \ldots, (p-1)a$$

and let

$$r_1, r_2, r_3, \ldots, r_{p-1}$$

be the sequence of remainders of the division of the numbers in the first sequence by p. Because $\gcd(a,p) = 1$, none of the numbers in the first sequence is divisible by p, so $1 \leq r_i \leq p-1$, for $i = 1, \ldots, p-1$. We claim that these remainders are all distinct. If not, then say $r_i = r_j$, with $1 \leq i < j \leq p-1$. But then, because

$$ai \equiv r_i \pmod{p}$$

and

$$aj \equiv r_j \pmod{p},$$

we deduce that

$$aj - ai \equiv r_j - r_i \pmod{p},$$

and because $r_i = r_j$, we get,

$$a(j-i) \equiv 0 \pmod{p}.$$

This means that p divides $a(j-i)$, but $\gcd(a,p)=1$ so, by Euclid's proposition (Proposition 5.9), p must divide $j-i$. However $1 \leq j-i < p-1$, so we get a contradiction and the remainders are indeed all distinct.

There are $p-1$ distinct remainders and they are all nonzero, therefore we must have

$$\{r_1, r_2, \ldots, r_{p-1}\} = \{1, 2, \ldots, p-1\}.$$

Using Property (3) of congruences (see Proposition 5.24), we get

$$a \cdot 2a \cdot 3a \cdots (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p};$$

that is,

$$(a^{p-1}-1) \cdot (p-1)! \equiv 0 \pmod{p}.$$

Again, p divides $(a^{p-1}-1) \cdot (p-1)!$, but because p is relatively prime to $(p-1)!$, it must divide $a^{p-1}-1$, as claimed.

(2) If $\gcd(a,p)=1$, we proved in (1) that

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

from which we get

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

because $a \equiv a \pmod{p}$. If a is divisible by p, then $a \equiv 0 \pmod{p}$, which implies $a^p \equiv 0 \pmod{p}$, and thus, that

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

Therefore, (2) holds for all $a \in \mathbb{Z}$ and we just proved that (1) implies (2). Finally, if (2) holds and if $\gcd(a,p)=1$, as p divides $a^p - a = a(a^{p-1}-1)$, it must divide $a^{p-1}-1$, which shows that (1) holds and so, (2) implies (1). $\square$

It is now easy to establish the correctness of RSA.

Proposition 5.25. *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. Because p and q are two distinct prime numbers, by Euclid's proposition it is enough to prove that both p and q divide $x^{ed}-x$. We show that $x^{ed}-x$ is divisible by p, the proof of divisibility by q being similar.

By condition (2), we have

$$ed = 1 + (p-1)(q-1)k,$$

with $k \geq 1$, inasmuch as $1 < e, d < (p-1)(q-1)$. Thus, if we write $h = (q-1)k$, we have $h \geq 1$ and

$$\begin{aligned}
x^{ed} - x &\equiv x^{1+(p-1)h} - x \pmod{p} \\
&\equiv x((x^{p-1})^h - 1) \pmod{p} \\
&\equiv x(x^{p-1} - 1)((x^{p-1})^{h-1} + (x^{p-1})^{h-2} + \cdots + 1) \pmod{p} \\
&\equiv (x^p - x)((x^{p-1})^{h-1} + (x^{p-1})^{h-2} + \cdots + 1) \pmod{p} \\
&\equiv 0 \pmod{p},
\end{aligned}$$

because $x^p - x \equiv 0 \pmod{p}$, by Fermat's little theorem. $\square$

Remark: Of course, Proposition 5.25 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 that are relatively prime to pq. For any arbitrary positive integer, m, the number of positive integers less than m that are relatively prime to m is given by the *Euler ϕ function* (or *Euler totient*), denoted ϕ (see Problems 5.23 and 5.27 or Niven, Zuckerman, and Montgomery [12], Section 2.1, for basic properties of ϕ).

Fermat's little theorem can be generalized to what is known as *Euler's formula* (see Problem 5.23): For every integer a, if $\gcd(a,m) = 1$, then

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

Because $\phi(pq) = (p-1)(q-1)$, when $\gcd(x, \phi(pq)) = 1$, Proposition 5.25 follows from Euler's formula. However, that argument does not show that Proposition 5.25 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 $x \in \mathbb{Z}$ (see Niven, Zuckerman, and Montgomery [12], Section 2.5, Problem 4).

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 5.25 immediately implies that the decrypting and encrypting 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-digit numbers, it will still be impossible to factor 400-digit 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!

5.11 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 nth 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,

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

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** $u := a \cdot u \bmod m$;
    $a := a^2 \bmod m$; $n := (n - e)/2$
  endwhile;
  $u := a \cdot u \bmod m$
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 nth power of an integer x and the time speed-up of not performing the last squaring step is more significant. We leave the details of the proof that the above algorithm is correct as an exercise.

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 5.4 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. The details are worked out in Problem 5.18 and another version is explored in Problem 5.19. 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.

$$\begin{aligned}
m &= aq_1 + r_1 \\
a &= r_1 q_2 + r_2 \\
r_1 &= r_2 q_3 + r_3 \\
&\vdots \\
r_{k-1} &= r_k q_{k+1} + r_{k+1} \\
&\vdots \\
r_{n-3} &= r_{n-2} q_{n-1} + r_{n-1} \\
r_{n-2} &= r_{n-1} q_n + 0,
\end{aligned}$$

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 have just two divisions,

$$\begin{aligned}
m &= aq_1 + r_1 \\
a &= r_1 q_2 + 0,
\end{aligned}$$

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 $r_0 = a$.

In Problem 5.18, it is 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 from the discussion just before Theorem 5.10. 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

$$a y_i \equiv r_i \pmod{m}$$

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

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

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 $a = 1$.

We leave this proof as an exercise (see Problem 5.58). Here is an algorithm obtained by adapting the algorithm given in Problem 5.18.

An Algorithm for Computing the Inverse of a Modulo m

Given any natural number a with $1 \leq a < m$ and $\gcd(a, m) = 1$, the following algorithm returns the inverse of a modulo m as y.

```
begin
  y := 0; v := 1; g := m; r := a;
  pr := r; q := ⌊g/pr⌋; r := g − pr q; (divide g by pr, to get g = pr q + r)
  if r = 0 then
    y := 1; g := pr
  else
    r = pr;
    while r ≠ 0 do
      pr := r; pv := v;
      q := ⌊g/pr⌋; r := g − pr q; (divide g by pr, to get g = pr q + r)
      v := y − pv q;
      if v < 0 then
        v := m − ((−v) mod m)
      else
        v = v mod 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*.

5.12 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 5.14. 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 $\pi(10) = 4$ because the primes no greater than 10 are $2, 3, 5, 7$ and $\pi(20) = 8$ because 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, Chebyshev, 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

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

However, a rigorous proof was not found until 1896. Indeed, in 1896, Jacques

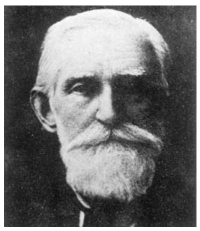

Fig. 5.18 Pafnuty Lvovich Chebyshev, 1821–1894 (left), Jacques Salomon Hadamard, 1865–1963 (middle), and Charles Jean de la Vallée Poussin, 1866–1962 (right)

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.

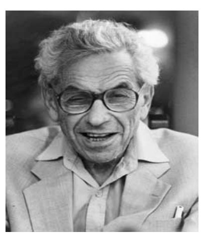


Fig. 5.19 Paul Erdös, 1913–1996 (left), Atle Selberg, 1917–2007 (right)

Theorem 5.15. *(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*

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

For a rather detailed account of the history of the prime number theorem (for short, *PNT*), we refer the reader to Ribenboim [13] (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. The number of natural numbers with 200 digits is $10^{200} - 10^{199} = 9 \cdot 10^{199}$, thus the proportion of 200-digit 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 we test for primality?

Primality testing has also been studied for a long time. Remarkably, Fermat's little theorem yields a test for nonprimality. 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. Because 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 that 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, and so on.

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 $2^{340}-1$.

Fig. 5.20 Robert Daniel Carmichael, 1879–1967

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 [13] (Chapter 2). An excellent (but more terse) account is also given in Koblitz [9] (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 that is guaranteed to give a false positive with probablity less than 0.5, then we can repeat this test for randomly chosen as 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, and yet another the Solovay–Strassen test. Since 2002, it has been known that primality testing can be done in polynomial time. This result is due to Agrawal, Kayal, and Saxena and known as the AKS test solved a long-standing problem; see Dietzfelbinger [4] and Crandall and Pomerance [2] (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 [2] (Chapter 4) and again, to Ribenboim [13] (Chapter 2) and Koblitz [9] (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-digit 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, and so on. It is easy to see that such an e will be found in no more trials than it takes to find a prime; see Lovász, Pelikán, and Vesztergombi [10] (Chapter 15), which contains one of the simplest and clearest presentations of RSA that we know of. Koblitz [9] (Chapter IV) also provides some details on this topic as well as Menezes, van Oorschot, and Vanstone's *Handbook* [11].

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, inasmuch as 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 or she claims to be (with a high degree of confidence). The trick is to make use of the the sender's keys. 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, $D_A(y_2)=y_1$ Julia, with the legible signature, *Julia*. He will then delete the signature from this message and decrypt 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 pri-

vate 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), claiming 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 [9], Chapter IV, Section 2, or Menezes, van Oorschot, and Vanstone [11], 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 present, there is no proof that RSA is fully secure.

Observe that because $m = pq$ 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* [11].

5.13 Distributive Lattices, Boolean Algebras, Heyting Algebras

If we go back to one of our favorite examples of a lattice, namely, the power set 2^X of some set X, we observe that it is more than a lattice. For example, if we look at Figure 5.6, we can check that the two identities D1 and D2 stated in the next definition hold.