

## 2. Quantum Attacks on IFP-Based Cryptosystems

*If you don't work on important problems, it's not likely that you'll do important work.*

RICHARD HAMMING (1915–1998)  
The 1968 Turing Award Recipient

In this chapter we shall first study the integer factorization problem (IFP) and the classical solutions to IFP, then we shall discuss the IFP-based cryptography whose security relies on the infeasibility of the IFP problem, and finally, we shall introduce some quantum algorithms for attacking both IFP and IFP-based cryptography.

### 2.1 IFP and Classical Solutions to IFP

#### Fundamental Theorem of Arithmetic

In mathematics, there are many fundamental theorems such as fundamental theorem of geometry, fundamental theorem of algebra, and fundamental theorem of calculus. The fundamental theorem of arithmetic (FTA) may be regarded as the first and most important fundamental theorem in mathematics, stating as follows.

**Theorem 2.1 (FTA).** Any positive integer  $n > 1$  can be written uniquely as the following standard prime factorization form:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}, \quad (2.1)$$

where  $p_1 < p_2 < \cdots < p_k$  are primes and  $\alpha_1, \alpha_2, \dots, \alpha_k$  are positive integers.

### Integer Factorization Problem

The idea of FTA can be traced to Euclid's *Elements* [25], but it was first clearly stated and proved by Gauss [29] in his *Disquisitiones*. According to FTA, any positive integer can be uniquely written as its prime decomposition form, say, for example,

$$12345678987654321 = 3^4 \cdot 37^2 \cdot 333667^2.$$

So, we can define the prime factorization problem (PFP) as follows:

$$\text{PFP} \stackrel{\text{def}}{=} \begin{cases} \text{Input :} & n \in \mathbb{Z}_{>1} \text{ and } n \notin \text{Primes} \\ \text{Output :} & n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k} \end{cases} \quad (2.2)$$

The solution to PFP is actually involved in the solutions of two other problems: the primality testing problem (PTP) and the IFP, which can be described as follows:

$$\text{PTP} \stackrel{\text{def}}{=} \begin{cases} \text{Input :} & n \in \mathbb{Z}_{>1} \\ \text{Output :} & \begin{cases} \text{Yes,} & n \in \text{Primes} \\ \text{No,} & \text{Otherwise} \end{cases} \end{cases} \quad (2.3)$$

and

$$\text{IFP} \stackrel{\text{def}}{=} \begin{cases} \text{Input :} & n \in \mathbb{Z}_{>1} \text{ and } n \notin \text{Primes} \\ \text{Output :} & 1 < f < n \text{ (} f \text{ is a nontrivial factor of } n \text{)}. \end{cases} \quad (2.4)$$

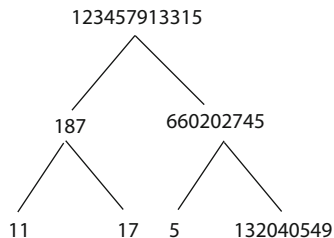
So, to solve PFP, one just needs to recursively execute the following two algorithms:

1. Algorithm for PTP
2. Algorithm for IFP

That is,

$$\text{PFP} \stackrel{\text{def}}{=} \overset{\Omega}{\text{PTP}} \oplus \overset{\Omega}{\text{IFP}}.$$

For example, if we wish to factor the integer 123457913315, the recursive process may be shown in Fig. 2.1. Since PTP can be solved easily in polynomial time [3], we shall only concentrate on the solutions to IFP.



**Figure 2.1.** Prime factorization of 123457913315

## Methods for Integer Factorization

There are many methods and algorithms for factoring a large integer. If we are concerned with the determinism of the algorithms, then there are two types of factoring algorithms:

1. Deterministic factoring algorithms
2. Probabilistic factoring algorithms

However, if we are more concerned with the form and the property of the integers to be factored, then there are two types factoring methods or algorithms:

1. General-purpose factoring algorithms: The running time depends mainly on the size of  $N$ , the number to be factored, and is not strongly dependent on the size of the factor  $p$  found. Examples are:
  - (a) *Lehman's method* [44], which has a rigorous worst-case running time bound  $\mathcal{O}(n^{1/3+\epsilon})$ .
  - (b) *Euler's factoring method* [49], which has deterministic running time  $\mathcal{O}(n^{1/3+\epsilon})$ .
  - (c) *Shanks' SQUARE FOrm Factorization method* [68] SQUFOF, which has expected running time  $\mathcal{O}(n^{1/4})$ .
  - (d) *The FFT-based factoring methods of Pollard and Strassen* [58, 75] which have deterministic running time  $\mathcal{O}(n^{1/4+\epsilon})$ .
  - (e) *The lattice-based factoring methods of Coppersmith* [18], which has deterministic running time  $\mathcal{O}(n^{1/4+\epsilon})$ .
  - (f) *Shanks' class group method* [67], which has running time  $\mathcal{O}(n^{1/5+\epsilon})$ , assuming the extended Riemann's hypothesis (ERH).
  - (g) *Continued FRACtion (CFRAC) method* [55], which under plausible assumptions has expected running time

$$\mathcal{O}\left(\exp\left(c\sqrt{\log n \log \log n}\right)\right) = \mathcal{O}\left(n^{c\sqrt{\log \log n / \log n}}\right),$$

where  $c$  is a constant (depending on the details of the algorithm); usually  $c = \sqrt{2} \approx 1.414213562$ .

- (h) *Quadratic sieve/Multiple polynomial quadratic sieve (QS/MPQS)* [60], which under plausible assumptions has expected running time

$$\mathcal{O}\left(\exp\left(c\sqrt{\log n \log \log n}\right)\right) = \mathcal{O}\left(n^{c\sqrt{\log \log n / \log n}}\right),$$

where  $c$  is a constant (depending on the details of the algorithm); usually  $c = \frac{3}{2\sqrt{2}} \approx 1.060660172$ .

- (i) *Number field sieve (NFS)* [46], which under plausible assumptions has the expected running time

$$\mathcal{O}\left(\exp\left(c\sqrt[3]{\log n} \sqrt[3]{(\log \log n)^2}\right)\right),$$

where  $c = (64/9)^{1/3} \approx 1.922999427$  if GNFS (a general version of NFS) is used to factor an arbitrary integer  $n$ , whereas  $c = (32/9)^{1/3} \approx 1.526285657$  if SNFS (a special version of NFS) is used to factor a special integer  $n$  such as  $n = r^e \pm s$ , where  $r$  and  $s$  are small,  $r > 1$ , and  $e$  is large. This is substantially and asymptotically faster than any other currently known factoring method.

2. Special purpose factoring algorithms: The running time depends mainly on the size of  $p$  (the factor found) of  $n$ . (We can assume that  $p \leq \sqrt{n}$ .) Examples are:

- (a) *Trial division* [41], which has running time  $\mathcal{O}(p(\log n)^2)$ .
- (b) *Pollard's  $\rho$ -method* [10, 59] (also known as Pollard's "*rho*" algorithm), which under plausible assumptions has expected running time  $\mathcal{O}(p^{1/2}(\log n)^2)$ .
- (c) *Pollard's  $p-1$  method* [58], which runs in  $\mathcal{O}(B \log B(\log n)^2)$ , where  $B$  is the smooth bound; larger values of  $B$  make it run more slowly, but are more likely to produce a factor of  $n$ .
- (d) *Lenstra's elliptic curve method (ECM)* [47], which under plausible assumptions has expected running time

$$\mathcal{O}\left(\exp\left(c\sqrt{\log p \log \log p}\right) \cdot (\log n)^2\right),$$

where  $c \approx 2$  is a constant (depending on the details of the algorithm). The term  $\mathcal{O}((\log n)^2)$  is for the cost of performing arithmetic operations on numbers which are  $\mathcal{O}(\log n)$  or  $\mathcal{O}((\log n)^2)$  bits long; the second can be theoretically replaced by  $\mathcal{O}((\log n)^{1+\epsilon})$  for any  $\epsilon > 0$ .

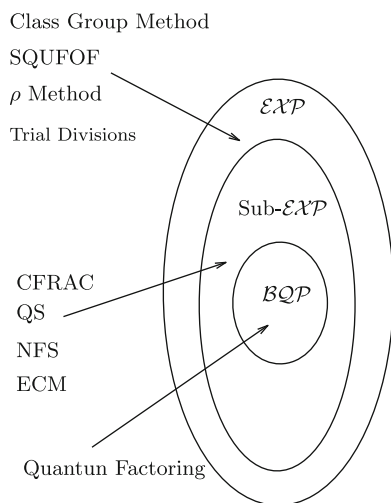
Note that there is a quantum factoring algorithm, first proposed by Shor [70], which can run in polynomial time

$$\mathcal{O}((\log n)^{2+\epsilon}).$$

However, this quantum algorithm requires to be run on a quantum computer, which is not available at present.

In practice, algorithms in both categories are important. It is sometimes very difficult to say whether one method is better than another, but it is generally worth attempting to find small factors with algorithms in the second class before using the algorithms in the first class. That is, we could first try the *trial division algorithm*, then use some other method such as NFS. This fact shows that the trial division method is still useful for integer factorization, even though it is simple. In this chapter we shall introduce some most the useful and widely used factoring algorithms.

From a computational complexity point of view, the IFP is an infeasible (intractable) problem, since there is no polynomial-time algorithm for solving it; all the existing algorithms for IFP run in subexponential-time or above (see Fig. 2.2). Note that there is a quantum algorithm proposed by Shor [70]



**Figure 2.2.** Algorithms/methods for IFP

for IFP that can be run in polynomial time, but it needs to be run on a practical quantum computer which does not exist at present.

## NFS Factoring

A fundamental idea of many modern general-purpose algorithms for factoring  $n$  is to find a suitable pair  $(x, y)$  such that

$$x^2 \equiv y^2 \pmod{n} \text{ but } x \not\equiv \pm y \pmod{n}$$

then there is a good chance to factor  $n$ :

$$\text{Prob}(\gcd(x \pm y, n) = (f_1, f_2), 1 < f_1, f_2 < n) > \frac{1}{2}.$$

In practice, the asymptotically fastest general-purpose factoring algorithm is the NFS, and it runs in expected subexponential-time:

$$\mathcal{O}(\exp(c(\log n)^{1/3}(\log \log n)^{2/3})).$$

**Definition 2.1.** A complex number  $\alpha$  is an *algebraic number* if it is a root of a polynomial

$$f(x) = a_0x^k + a_1x^{k-1} + a_2x^{k-2} \cdots + a_k = 0 \quad (2.5)$$

where  $a_0, a_1, a_2, \dots, a_k \in \mathbb{Q}$  and  $a_0 \neq 0$ . If  $f(x)$  is irreducible over  $\mathbb{Q}$  and  $a_0 \neq 0$ , then  $k$  is the degree of  $x$ .

**Example 2.1.** Two examples of algebraic numbers are as follows:

- 1 Rational numbers, which are the algebraic numbers of degree 1
- 2  $\sqrt{2}$ , which is of degree 2 because we can take  $f(x) = x^2 - 2 = 0$  ( $\sqrt{2}$  is irrational)

Any complex number that is not algebraic is said to be *transcendental* such as  $\pi$  and  $e$ .

**Definition 2.2.** A complex number  $\beta$  is an *algebraic integer* if it is a root of a monic polynomial

$$x^k + b_1x^{k-1} + b_2x^{k-2} \cdots + b_k = 0 \quad (2.6)$$

where  $b_0, b_1, b_2, \dots, b_k \in \mathbb{Z}$ .

**Remark 2.1.** A quadratic integer is an algebraic integer satisfying a monic quadratic equation with integer coefficients. A cubic integer is an algebraic integer satisfying a monic cubic equation with integer coefficients.

**Example 2.2.** Some examples of algebraic integers are as follows:

- 1 Ordinary (rational) integers, which are the algebraic integers of degree 1. That is, they satisfy the monic equations  $x - a = 0$  for  $a \in \mathbb{Z}$ .
- 2  $\sqrt[3]{2}$  and  $\sqrt[5]{3}$ , because they satisfy the monic equations  $x^3 - 2 = 0$  and  $x^5 - 3 = 0$ , respectively.
- 3  $(-1 + \sqrt{-3})/2$ , because it satisfies  $x^2 + x + 1 = 0$ .
- 4 Gaussian integer  $a + b\sqrt{-1}$ , with  $a, b \in \mathbb{Z}$ .

Clearly, every algebraic integer is an algebraic number, but the converse is not true.

**Proposition 2.1.** A rational number  $r \in \mathbb{Q}$  is an algebraic integer if and only if  $r \in \mathbb{Z}$ .

**Proof.** If  $r \in \mathbb{Z}$ , then  $r$  is a root of  $x - r = 0$ . Thus,  $r$  is an algebraic integer. Now, suppose that  $r \in \mathbb{Q}$  and  $r$  is an algebraic integer (i.e.,  $r = c/d$  is a root of (2.6), where  $c, d \in \mathbb{Z}$ ; we may assume  $\gcd(c, d) = 1$ ). Substituting  $c/d$  into (2.6) and multiplying both sides by  $d^n$ , we get

$$c^k + b_1 c^{k-1} d + b_2 c^{k-2} d^2 \cdots + b_k d^k = 0.$$

It follows that  $d \mid c^k$  and  $d \mid c$  (since  $\gcd(c, d) = 1$ ). Again, since  $\gcd(c, d) = 1$ , it follows that  $d = \pm 1$ . Hence,  $r = c/d \in \mathbb{Z}$ . It follows, for example, that  $2/5$  is an algebraic number but not an algebraic integer.  $\square$

**Remark 2.2.** The elements of  $\mathbb{Z}$  are the only rational numbers that are algebraic integers. We shall refer to the elements of  $\mathbb{Z}$  as *rational integers* when we need to distinguish them from other algebraic integers that are not rational. For example,  $\sqrt{2}$  is an algebraic integer but not a rational integer.

The most interesting results concerned with the algebraic numbers and algebraic integers are the following theorem.

**Theorem 2.2.** The set of algebraic numbers forms a field, and the set of algebraic integers forms a ring.

**Proof.** See pp 67–68 of Ireland and Rosen [38].  $\square$

**Lemma 2.1.** Let  $f(x)$  be an irreducible monic polynomial of degree  $d$  over integers and  $m$  an integer such that  $f(m) \equiv 0 \pmod{n}$ . Let  $\alpha$  be a complex root of  $f(x)$  and  $\mathbb{Z}[\alpha]$  the set of all polynomials in  $\alpha$  with integer coefficients. Then there exists a unique mapping  $\Phi : \mathbb{Z}[\alpha] \mapsto \mathbb{Z}_n$  satisfying:

- 1  $\Phi(ab) = \Phi(a)\Phi(b), \quad \forall a, b \in \mathbb{Z}[\alpha].$
- 2  $\Phi(a + b) = \Phi(a) + \Phi(b), \quad \forall a, b \in \mathbb{Z}[\alpha].$
- 3  $\Phi(za) = z\Phi(a), \quad \forall a \in \mathbb{Z}[\alpha], z \in \mathbb{Z}.$
- 4  $\Phi(1) = 1.$
- 5  $\Phi(\alpha) = m \pmod{n}.$

Now, we are in a position to introduce the NFS. Note that there are two main types of NFS: NFS (general NFS) for general numbers and SNFS (special NFS) for numbers with special forms. The idea, however, behind the GNFS and SNFS is the same:

1. Find a monic irreducible polynomial  $f(x)$  of degree  $d$  in  $\mathbb{Z}[x]$  and an integer  $m$  such that  $f(m) \equiv 0 \pmod{n}$ .

2. Let  $\alpha \in \mathbb{C}$  be an algebraic number that is the root of  $f(x)$ , and denote the set of polynomials in  $\alpha$  with integer coefficients as  $\mathbb{Z}[\alpha]$ .
3. Define the mapping (ring homomorphism):  $\Phi : \mathbb{Z}[\alpha] \mapsto \mathbb{Z}_n$  via  $\Phi(\alpha) = m$  which ensures that for any  $f(x) \in \mathbb{Z}[x]$ , we have  $\Phi(f(\alpha)) \equiv f(m) \pmod{n}$ .
4. Find a finite set  $U$  of coprime integers  $(a, b)$  such that

$$\prod_{(a,b) \in U} (a - b\alpha) = \beta^2, \quad \prod_{(a,b) \in U} (a - bm) = y^2$$

for  $\beta \in \mathbb{Z}[\alpha]$  and  $y \in \mathbb{Z}$ . Let  $x = \Phi(\beta)$ . Then

$$\begin{aligned} x^2 &\equiv \Phi(\beta)\Phi(\beta) \\ &\equiv \Phi(\beta^2) \\ &\equiv \Phi\left(\prod_{(a,b) \in U} (a - b\alpha)\right) \\ &\equiv \prod_{(a,b) \in U} \Phi(a - b\alpha) \\ &\equiv \prod_{(a,b) \in U} (a - bm) \\ &\equiv y^2 \pmod{n} \end{aligned}$$

which is of the required form of the factoring congruence, and hopefully, a factor of  $n$  can be found by calculating  $\gcd(x \pm y, n)$ .

There are many ways to implement the above idea, all of which follow the same pattern as we discussed previously in CFRAC and QS/MPQS: By a sieving process, one first tries to find congruences modulo  $n$  by working over a factor base, and then do a Gaussian elimination over  $\mathbb{Z}/2\mathbb{Z}$  to obtain a congruence of squares  $x^2 \equiv y^2 \pmod{n}$ . We give in the following a brief description of the NFS algorithm [54].

**Algorithm 2.1.** Given an odd positive integer  $n$ , the NFS algorithm has the following four main steps in factoring  $n$ :

- [1] (Polynomials Selection) Select two irreducible polynomials  $f(x)$  and  $g(x)$  with small integer coefficients for which there exists an integer  $m$  such that

$$f(m) \equiv g(m) \equiv 0 \pmod{n} \tag{2.7}$$

The polynomials should not have a common factor over  $\mathbb{Q}$ .



- [2] (Sieving) Let  $\alpha$  be a complex root of  $f$  and  $\beta$  a complex root of  $g$ . Find pairs  $(a, b)$  with  $\gcd(a, b) = 1$  such that the integral norms of  $a - b\alpha$  and  $a - b\beta$

$$N(a - b\alpha) = b^{\deg(f)} f(a/b), \quad N(a - b\beta) = b^{\deg(g)} g(a/b) \quad (2.8)$$

are smooth with respect to a chosen factor base. (The principal ideals  $a - b\alpha$  and  $a - b\beta$  factor into products of prime ideals in the number field  $\mathbb{Q}(\alpha)$  and  $\mathbb{Q}(\beta)$ , respectively.)

- [3] (Linear Algebra) Use techniques of linear algebra to find a set  $U = \{a_i, b_i\}$  of indices such that the two products

$$\prod_U (a_i - b_i\alpha), \quad \prod_U (a_i - b_i\beta) \quad (2.9)$$

are both squares of products of prime ideals.

- [4] (Square root) Use the set  $S$  in (2.9) to find algebraic numbers  $\alpha' \in \mathbb{Q}(\alpha)$  and  $\beta' \in \mathbb{Q}(\beta)$  such that

$$(\alpha')^2 = \prod_U (a_i - b_i\alpha), \quad (\beta')^2 = \prod_U (a_i - b_i\beta) \quad (2.10)$$

Define  $\Phi_\alpha : \mathbb{Q}(\alpha) \rightarrow \mathbb{Z}_n$  and  $\Phi_\beta : \mathbb{Q}(\beta) \rightarrow \mathbb{Z}_n$  via  $\Phi_\alpha(\alpha) = \Phi_\beta(\beta) = m$ , where  $m$  is the common root of both  $f$  and  $g$ . Then

$$\begin{aligned} x^2 &\equiv \Phi_\alpha(\alpha') \Phi_\alpha(\alpha') \\ &\equiv \Phi_\alpha((\alpha')^2) \\ &\equiv \Phi_\alpha\left(\prod_{i \in U} (a_i - b_i\alpha)\right) \\ &\equiv \prod_U \Phi_\alpha(a_i - b_i\alpha) \\ &\equiv \prod_U (a_i - b_i m) \\ &\equiv \Phi_\beta(\beta')^2 \\ &\equiv y^2 \pmod{n} \end{aligned}$$

which is of the required form of the factoring congruence, and hopefully, a factor of  $N$  can be found by calculating  $\gcd(x \pm y, n)$ .

**Example 2.3.** We first give a rather simple NFS factoring example. Let  $n = 14885 = 5 \cdot 13 \cdot 229 = 122^2 + 1$ . So we put  $f(x) = x^2 + 1$  and  $m = 122$ , such that

$$f(x) \equiv f(m) \equiv 0 \pmod{n}.$$

If we choose  $|a|, |b| \leq 50$ , then we can easily find (by sieving) that

$(a, b)$	$\text{Norm}(a + bi)$	$a + bm$
$\vdots$	$\vdots$	$\vdots$
$(-49, 49)$	$4802 = 2 \cdot 7^4$	$5929 = 7^2 \cdot 11^2$
$\vdots$	$\vdots$	$\vdots$
$(-41, 1)$	$1682 = 2 \cdot 29^2$	$81 = 3^4$
$\vdots$	$\vdots$	$\vdots$

(Readers should be able to find many such pairs of  $(a_i, b_i)$  in the interval that are smooth up to, e.g., 29.) So, we have

$$\begin{aligned}
 (49 + 49i)(-41 + i) &= (49 - 21i)^2, \\
 f(49 - 21i) &= 49 - 21m \\
 &= 49 - 21 \cdot 122 \\
 &= -2513 \rightarrow x, \\
 5929 \cdot 81 &= (2^2 \cdot 7 \cdot 11)^2 \\
 &= 693^2 \\
 &\rightarrow y = 693.
 \end{aligned}$$

Thus,

$$\begin{aligned}
 \gcd(x \pm y, n) &= \gcd(-2513 \pm 693, 14885) \\
 &= (65, 229).
 \end{aligned}$$

In the same way, if we wish to factor  $n = 84101 = 290^2 + 1$ , then we let  $m = 290$  and  $f(x) = x^2 + 1$  so that

$$f(x) \equiv f(m) \equiv 0 \pmod{n}.$$

We tabulate the sieving process as follows:

$(a, b)$	$\text{Norm}(a + bi)$	$a + bm$
$\vdots$	$\vdots$	$\vdots$
$-50, 1$	$2501 = 41 \cdot 61$	$240 = 2^4 \cdot 3 \cdot 5$
$\vdots$	$\vdots$	$\vdots$
$-50, 3$	$2509 = 13 \cdot 193$	$820 = 2^2 \cdot 5 \cdot 41$
$\vdots$	$\vdots$	$\vdots$
$-49, 43$	$4250 = 2 \cdot 5^3 \cdot 17$	$12421 = 12421$
$\vdots$	$\vdots$	$\vdots$
$-38, 1$	$1445 = 5 \cdot 17^2$	$252 = 2^2 \cdot 3^2 \cdot 7$
$\vdots$	$\vdots$	$\vdots$
$-22, 19$	$845 = 5 \cdot 13^2$	$5488 = 2^4 \cdot 7^3$
$\vdots$	$\vdots$	$\vdots$
$-118, 11$	$14045 = 5 \cdot 53^2$	$3072 = 2^{10} \cdot 3$
$\vdots$	$\vdots$	$\vdots$
$218, 59$	$51005 = 5 \cdot 101^2$	$17328 = 2^4 \cdot 3 \cdot 19^2$
$\vdots$	$\vdots$	$\vdots$

Clearly,  $-38 + i$  and  $-22 + 19i$  can produce a product square, since

$$\begin{aligned}
(-38 + i)(-22 + 19i) &= (31 - 12i)^2, \\
f(31 - 12i) &= 31 - 12m \\
&= -3449 \rightarrow x, \\
252 \cdot 5488 &= (2^3 \cdot 3 \cdot 7^2)^2 \\
&= 1176^2, \\
&\rightarrow y = 1176, \\
\gcd(x \pm y, n) &= \gcd(-3449 \pm 1176, 84101) \\
&= (2273, 37).
\end{aligned}$$

In fact,  $84101 = 2273 \times 37$ . Note that  $-118 + 11i$  and  $218 + 59i$  can also produce a product square, since

$$\begin{aligned}
(-118 + 11i)(218 + 59i) &= (14 - 163i)^2, \\
f(14 - 163i) &= 14 - 163m \\
&= -47256 \rightarrow x, \\
3071 \cdot 173288 &= (2^7 \cdot 3 \cdot 19)^2 \\
&= 7296^2, \\
&\rightarrow y = 7296, \\
\gcd(x \pm y, n) &= \gcd(-47256 \pm 7296, 84101) \\
&= (37, 2273).
\end{aligned}$$

**Example 2.4.** Next, we present a little bit more complicated example. Use NFS to factor  $n = 1098413$ . First, notice that  $n = 1098413 = 12 \cdot 45^3 + 17^3$ , which is in a special form and can be factored by using SNFS.

- [1] (Polynomials Selection) Select the two irreducible polynomials  $f(x)$  and  $g(x)$  and the integer  $m$  as follows:

$$\begin{aligned}
m &= \frac{17}{45}, \\
f(x) = x^3 + 12 &\implies f(m) = \left(\frac{17}{45}\right)^3 + 12 \equiv 0 \pmod{n}, \\
g(x) = 45x - 17 &\implies g(m) = 45\left(\frac{17}{45}\right) - 17 \equiv 0 \pmod{n}.
\end{aligned}$$

- [2] (Sieving) Suppose after sieving, we get  $U = \{a_i, b_i\}$  as follows:

$$U = \{(6, -1), (3, 2), (-7, 3), (1, 3), (-2, 5), (-3, 8), (9, 10)\}.$$

That is, the chosen polynomial that produces a product square can be constructed as follows (as an exercise, readers may wish to choose some other polynomial which can also produce a product square):

$$\prod_U (a_i + b_i x) = (6-x)(3+2x)(-7+3x)(1+3x)(-2+5x)(-3+8x)(9+10x).$$

Let  $\alpha = \sqrt[3]{-12}$  and  $\beta = \frac{17}{45}$ . Then

$$\begin{aligned}
\prod_U (a - b\alpha) &= 7400772 + 1138236\alpha - 10549\alpha^2 \\
&= (2694 + 213\alpha - 28\alpha^2)^2 \\
&= \left( \frac{5610203}{2025} \right) \\
&= 270729^2, \\
\prod_U (a - b\beta) &= \frac{2^8 \cdot 11^2 \cdot 13^2 \cdot 23^2}{3^{12} \cdot 5^4} \\
&= \left( \frac{52624}{18225} \right)^2 \\
&= 875539^2.
\end{aligned}$$

So, we get the required square of congruence:

$$270729^2 \equiv 875539^2 \pmod{1098413}.$$

Thus,

$$\gcd(270729 \pm 875539, 1098413) = (563, 1951).$$

That is,

$$1098413 = 563 \cdot 1951.$$

**Example 2.5.** We give some large factoring examples using NFS.

1 SNFS examples: One of the largest numbers factored by SNFS is

$$n = (12^{167} + 1)/13 = p_{75} \times p_{105}$$

It was announced by P. Montgomery, S. Cavallar, and H. te Riele at CWI in Amsterdam on 3 September 1997. They used the polynomials  $f(x) = x^5 - 144$  and  $g(x) = 12^{33}x + 1$  with common root  $m \equiv 12^{134} \pmod{n}$ . The factor base bound was 4.8 million for  $f$  and 12 million for  $g$ . Both large prime bounds were 150 million, with two large primes allowed on each side. They sieved over  $|a| \leq 8.4$  million and  $0 < b \leq 2.5$  million. The sieving lasted 10.3 calendar days; 85 SGI machines at CWI contributed a combined 13027719 relations in 560 machine-days. It took 1.6 more calendar days to process the data. This processing included 16 CPU-hours on a Cray C90 at SARA in Amsterdam to process a  $1969262 \times 1986500$  matrix with 57942503 nonzero entries. The other large number factorized by using SNFS is the 9th Fermat number:

$$F_9 = 2^{2^9} + 1 = 2^{512} + 1 = 2424833 \cdot p_{49} \cdot p_{99},$$

a number with 155 digits; it was completely factored in April 1990. The most wanted factoring number of special form at present is the 12th Fermat number  $F_{12} = 2^{2^{12}} + 1$ ; we only know its partial prime factorization:

$$F_{12} = 114689 \cdot 26017793 \cdot 63766529 \cdot 190274191361 \cdot 1256132134125569 \cdot c_{1187}$$

and we want to find the prime factors of the remaining 1187-digit composite.

## 2 GNFS examples:

RSA – 130 (130 digits)

$$\begin{aligned} &= 18070820886874048059516561644059055662781025167694013491 \\ &\quad 70127021450056662540244048387341127590812303371781887966 \\ &\quad 563182013214880557 \\ &= 396859994595974542901611261628837 \\ &\quad 86067576449112810064832555157243 \\ &\quad \times \\ &\quad 455344986467359721884036868972744 \\ &\quad 08864356301263205069600999044599. \end{aligned}$$

RSA – 140 (140 digits)

$$\begin{aligned} &= 2129024631825875754749788201627151749780670396327721627 \\ &\quad 8233383215381949984056495911366573853021918316783107387 \\ &\quad 995317230889569230873441936471 \\ &= 33987174230284385545301236276138758 \\ &\quad 35633986495969597423490929302771479 \\ &\quad \times \\ &\quad 62642001874012850961516549482644422 \\ &\quad 19302037178623509019111660653946049. \end{aligned}$$

RSA – 155 (512 digits)

$$\begin{aligned} &= 1094173864157052742180970732204035761200373294544920599 \\ &\quad 0913842131476349984288934784717997257891267332497625752 \\ &\quad 899781833797076537244027146743531593354333897 \\ &= 102639592829741105772054196573991675900 \\ &\quad 716567808038066803341933521790711307779 \\ &\quad \times \\ &\quad 106603488380168454820927220360012878679 \\ &\quad 207958575989291522270608237193062808643. \end{aligned}$$

RSA – 576 (576 bits, 174 digits)

$$\begin{aligned}
 &= 18819881292060796383869723946165043980716356337941738 \\
 &\quad 27007633564229888597152346654853190606065047430453173 \\
 &\quad 88011303396716199692321205734031879550656996221305168 \\
 &\quad 759307650257059 \\
 &= 3980750864240649373971255005503864911990643 \\
 &\quad 62342526708406385189575946388957261768583317 \\
 &\quad \times \\
 &\quad 4727721461074353025362230719730482246329146 \\
 &\quad 95302097116459852171130520711256363590397527.
 \end{aligned}$$

RSA – 640 (193 digits, 640 bits)

$$\begin{aligned}
 &= 31074182404900437213507500358885679300373460228427275 \\
 &\quad 45720161948823206440518081504556346829671723286782437 \\
 &\quad 91627283803341547107310 \\
 &= 163473364580925384844313388386509085984178367003 \\
 &\quad 3092312181110852389333100104508151212118167511579 \\
 &\quad \times \\
 &= 190087128166482211312685157393541397547189678996 \\
 &\quad 8515493666638539088027103802104498957191261465571.
 \end{aligned}$$

RSA – 663 (200 digits, 663 bits)

$$\begin{aligned}
 &= 27997833911221327870829467638722601621070446786955428 \\
 &\quad 53756000992932612840010760934567105295536085606182235 \\
 &\quad 19109513657886371059544820065767750985805576135790987 \\
 &\quad 34950144178863178946295187237869221823983 \\
 &= 35324619344027701212726049781984643686711974001976 \\
 &\quad 25023649303468776121253679423200058547956528088349 \\
 &\quad \times \\
 &\quad 79258699544783330333470858414800596877379758573642 \\
 &\quad 19960734330341455767872818152135381409304740185467.
 \end{aligned}$$

RSA – 704 (704 bits, 212 digits)

$$\begin{aligned}
&= 7403756347956171282804679609742957314259318888923128 \\
&\quad 9084936232638972765034028266276891996419625117843995 \\
&\quad 8943305021275853701189680982867331732731089309005525 \\
&\quad 0511687706329907239638078671008609696253793465056379 \\
&\quad 6359 \\
&= 90912135295978188784406583026004374858926 \\
&\quad 08310328358720428512168960411528640933367 \\
&\quad 824950788367956756806141 \\
&\quad \times \\
&\quad 81438592591100452657278091262844293358778 \\
&\quad 99002167627883200914172429324360133004116 \\
&\quad 702003240828777970252499.
\end{aligned}$$

RSA – 768 (768 bits, 232 digits)

$$\begin{aligned}
&= 123018668453011775513049495838496272077285356959533 \\
&\quad 479219732245215172640050726365751874520219978646938 \\
&\quad 995647494277406384592519255732630345373154826850791 \\
&\quad 702612214291346167042921431160222124047927473779408 \\
&\quad 0665351419597459856902143413 \\
&= 334780716989568987860441698482126908177047 \\
&\quad 949837137685689124313889828837938780022876 \\
&\quad 14711652531743087737814467999489 \\
&\quad \times \\
&\quad 36746043666799590428244633799627952632279 \\
&\quad 15816434308764267603228381573966651127923 \\
&\quad 3373417143396810270092798736308917.
\end{aligned}$$

**Remark 2.3.** Prior to the NFS, all modern factoring methods had an expected running time of at best

$$\mathcal{O}\left(\exp\left((c + o(1))\sqrt{\log n \log \log n}\right)\right).$$

For example, Dixon’s random square method has the expected running time

$$\mathcal{O}\left(\exp\left((\sqrt{2} + o(1))\sqrt{\log n \log \log n}\right)\right),$$



whereas the MPQS takes time

$$\mathcal{O}\left(\exp\left((1+o(1))\sqrt{\log\log n/\log n}\right)\right).$$

Because of the Canfield–Erdős–Pomerance theorem, some people even believed that this could not be improved, except maybe for the term  $(c+o(1))$ , but the invention of the NFS has changed this belief.

**Conjecture 2.1 (Complexity of NFS).** Under some reasonable heuristic assumptions, the NFS method can factor an integer  $N$  in time

$$\mathcal{O}\left(\exp\left((c+o(1))\sqrt[3]{\log n}\sqrt[3]{(\log\log n)^2}\right)\right) \quad (2.11)$$

where  $c = (64/9)^{1/3} \approx 1.922999427$  if GNFS is used to factor an arbitrary integer  $N$ , whereas  $c = (32/9)^{1/3} \approx 1.526285657$  if SNFS is used to factor a special integer  $N$ .

## $\rho$ -Factoring Method

Although NFS is the fastest method of factoring at present, other methods are also useful, one of the particular method is the  $\rho$ -factoring method [59]; surprisingly it is the method that is applicable for all the three infeasible problems, IFP, DLP, and ECDLP discussed in this book.

$\rho$  uses an iteration of the form

$$\left. \begin{aligned} x_0 &= \text{random}(0, n-1), \\ x_i &\equiv f(x_{i-1}) \pmod{n}, \quad i = 1, 2, 3, \dots \end{aligned} \right\} \quad (2.12)$$

where  $x_0$  is a random starting value,  $n$  is the number to be factored, and  $f \in \mathbb{Z}[x]$  is a polynomial with integer coefficients; usually, we just simply choose  $f(x) = x^2 \pm a$  with  $a \neq -2, 0$ . If  $p$  is prime, then the sequence  $\{x_i \bmod p\}_{i>0}$  must eventually repeat. Let  $f(x) = x^2 + 1, x_0 = 0, p = 563$ . Then we get the sequence  $\{x_i \bmod p\}_{i>0}$  as follows (Fig. 2.3):

$$\begin{aligned} x_0 &= 0, \\ x_1 &= x_0^2 + 1 = 1, \\ x_2 &= x_1^2 + 1 = 2, \\ x_3 &= x_2^2 + 1 = 5, \\ x_4 &= x_3^2 + 1 = 26, \\ x_5 &= x_4^2 + 1 = 114, \\ x_6 &= x_5^2 + 1 = 48, \end{aligned}$$

$$x_7 = x_6^2 + 1 = 53,$$

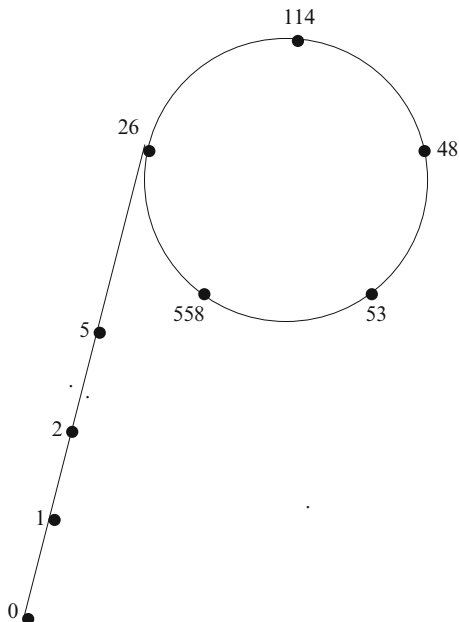
$$x_8 = x_7^2 + 1 = 558,$$

$$x_9 = x_8^2 + 1 = 26.$$

That is,

$$0, 1, 2, 5, \overline{26, 114, 48, 53, 558}.$$

This sequence symbols a diagram, looks like the Greek letter  $\rho$ : As an exercise,



**Figure 2.3.**  $\rho$  cycle modulo 563 using  $f(x) = x^2 + 1$  and  $x_0 = 0$

readers may wish to find the  $\rho$  cycle modulo 1951 using  $f(x) = x^2 + 1$  and  $x_0 = 0$ . Of course, to factor  $n$ , we do not know its prime factors before hand, but we can simply modulo  $n$  (justified by the Chinese Remainder Theorem). For example, to factor  $n = 1098413 = 563 \cdot 1951$ , we perform (all modulo 1098413):

$x_0 = \mathbf{0},$	$y_i = x_{2i}$	$\gcd(x_i - y_i, n)$
$x_1 = x_0^2 + 1 = \mathbf{1},$		
$x_2 = x_1^2 + 1 = \mathbf{2},$	$y_1 = x_2 = 2$	$\gcd(1 - 2, n) = 1$
$x_3 = x_2^2 + 1 = \mathbf{5},$		
$x_4 = x_3^2 + 1 = \mathbf{26},$	$y_2 = x_4 = 26$	$\gcd(2 - 26, n) = 1$

$$x_5 = x_4^2 + 1 = 677 \\ \equiv \mathbf{114},$$

$$x_6 = x_5^2 + 1 = 458330 \\ \equiv \mathbf{48}, \quad y_3 = x_6 = 458330 \quad \gcd(5 - 458330, n) = 1$$

$$x_7 = x_6^2 + 1 = 394716 \\ \equiv \mathbf{53},$$

$$x_8 = x_7^2 + 1 = 722324 \\ \equiv \mathbf{558}, \quad y_4 = x_8 = 722324 \quad \gcd(26 - 722324, n) = 1$$

$$x_9 = x_8^2 + 1 = 293912 \\ \equiv \mathbf{26}$$

$$x_{10} = x_9^2 + 1 = 671773 \\ \equiv \mathbf{114} \quad y_5 = x_{10} = 671773 \quad \gcd(677 - 671773, n) = \underline{\mathbf{563}}.$$

The following algorithm is an improved version of Brent [10] over Pollard's original  $\rho$ -method.

**Algorithm 2.2 (Brent–Pollard's  $\rho$ -method).** Let  $n$  be a composite integer greater than 1. This algorithm tries to find a nontrivial factor  $d$  of  $n$ , which is small compared with  $\sqrt{n}$ . Suppose the polynomial to use is  $f(x) = x^2 + 1$ .

[1] (Initialization) Choose a seed, say  $x_0 = 2$ , a generating function, say  $f(x) = x^2 + 1 \pmod{n}$ . Choose also a value for  $t$  not much bigger than  $\sqrt{d}$ , perhaps  $t < 100\sqrt{d}$ .

[2] (Iteration and Computation) Compute  $x_i$  and  $y_i$  in the following way:

$$\begin{aligned} x_1 &= f(x_0), \\ x_2 &= f(f(x_0)) = f(x_1), \\ x_3 &= f(f(f(x_0))) = f(f(x_1)) = f(x_2), \\ &\vdots \\ x_i &= f(x_{i-1}). \\ \\ y_1 &= x_2 = f(x_1) = f(f(x_0)) = f(f(y_0)), \\ y_2 &= x_4 = f(x_3) = f(f(x_2)) = f(f(y_1)), \\ y_3 &= x_6 = f(x_5) = f(f(x_4)) = f(f(y_2)), \\ &\vdots \\ y_i &= x_{2i} = f(f(y_{i-1})). \end{aligned}$$

and simultaneously compare  $x_i$  and  $y_i$  by computing  $d = \gcd(x_i - y_i, n)$ .

- [3] (Factor Found?) If  $1 < d < n$ , then  $d$  is a nontrivial factor of  $n$ , print  $d$ , and go to Step [5].
- [4] (Another Search?) If  $x_i = y_i \pmod{n}$  for some  $i$  or  $i \geq \sqrt{t}$ , then go to Step [1] to choose a new seed and a new generator and repeat.
- [5] (Exit) Terminate the algorithm.

The  $\rho$  algorithm has the conjectured complexity:

**Conjecture 2.2 (Complexity of the  $\rho$ -method).** Let  $p$  be a prime dividing  $n$  and  $p = \mathcal{O}(\sqrt{p})$ , then the  $\rho$ -algorithm has expected running time

$$\mathcal{O}(\sqrt{p}) = \mathcal{O}(\sqrt{p}(\log n)^2) = \mathcal{O}(n^{1/4}(\log n)^2) \quad (2.13)$$

to find the prime factor  $p$  of  $n$ .

**Remark 2.4.** The  $\rho$ -method is an improvement over trial division, because in trial division,  $\mathcal{O}(p) = \mathcal{O}(n^{1/4})$  divisions is needed to find a small factor  $p$  of  $n$ . But of course, one disadvantage of the  $\rho$ -algorithm is that its running time is only a conjectured expected value, not a rigorous bound.

## Exercises and Problems for Sect. 2.1

1. Explain why general-purpose factoring algorithms are slower than special purpose factoring algorithms, or why the special numbers are easy to factor than general numbers.
2. Show that:
  - (a) Addition of two  $\log n$  bit integers can be performed in  $\mathcal{O}(\log n)$  bit operations.
  - (b) Multiplication of two  $\log n$  bit integers can be performed in  $\mathcal{O}((\log n)^{1+\epsilon})$  bit operations.
3. Show that:
  - (a) Assume the ERH, there is deterministic algorithm that factors  $n$  in  $\mathcal{O}(n^{1/5+\epsilon})$  steps.
  - (b) FFT (fast Fourier transform) can be utilized to factor an integer  $n$  in  $\mathcal{O}(n^{1/4+\epsilon})$  steps.
  - (c) Give two deterministic algorithms that factor integer  $n$  in  $\mathcal{O}(n^{1/3+\epsilon})$  steps.
4. Show that if  $\mathcal{P} = \mathcal{NP}$ , then IFP  $\in \mathcal{P}$ .
5. Prove or disprove that IFP  $\in \mathcal{NP}$ -complete.

6. Extend the NFS to FFS (function field sieve). Give a complete description of the FFS for factoring large integers.
7. Let  $x_i = f(x_{i-1})$ ,  $i = 1, 2, 3, \dots$ . Let also  $t, u > 0$  be the smallest numbers in the sequence  $x_{t+i} = x_{t+u+i}$ ,  $i = 0, 1, 2, \dots$ , where  $t$  and  $u$  are called the lengths of the  $\rho$  tail and cycle, respectively. Give an efficient algorithm to determine  $t$  and  $u$  exactly, and analyze the running time of your algorithm.
8. Find the prime factorization of the following RSA numbers, each of these numbers has two prime factors:

(a) RSA-896 (270 digits, 896 bits)

4120234369866595438555313653325759481798116998443279828454556  
2643387644556524842619809887042316184187926142024718886949256  
0931776375033421130982397485150944909106910269861031862704114  
8808669705649029036536588674337317208131041051908642547932826  
01391257624033946373269391

(b) RSA-1024 (309 digits, 1024 bits)

1350664108659952233496032162788059699388814756056670275244851  
4385152651060485953383394028715057190944179820728216447155137  
3680419703964191743046496589274256239341020864383202110372958  
7257623585096431105640735015081875106765946292055636855294752  
1350085287941637732853390610975054433499981115005697723689092  
7563

(c) RSA-1536 (463 digits, 1536 bits)

1847699703211741474306835620200164403018549338663410171471785  
7749106516967111612498593376843054357445856160615445717940522  
2971773252466096064694607124962372044202226975675668737842756  
2389508764678440933285157496578843415088475528298186726451339  
8633649319080846719904318743812833635027954702826532978029349  
1615581188104984490831954500984839377522725705257859194499387  
0073695755688436933812779613089230392569695253261620823676490  
316036551371447913932347169566988069

(d) RSA-2048 (617 digits, 2048 bits)

2519590847565789349402718324004839857142928212620403202777713  
7836043662020707595556264018525880784406918290641249515082189  
2985591491761845028084891200728449926873928072877767359714183  
4727026189637501497182469116507761337985909570009733045974880  
8428401797429100642458691817195118746121515172654632282216869  
9875491824224336372590851418654620435767984233871847744479207  
3993423658482382428119816381501067481045166037730605620161967  
6256133844143603833904414952634432190114657544454178424020924  
6165157233507787077498171257724679629263863563732899121548314

3816789988504044536402352738195137863656439121201039712282212  
0720357

9. Try to complete the following prime factorization of the smallest unfactored (not completely factored) Fermat numbers:

$$F_{12} = 114689 \cdot 26017793 \cdot 63766529 \cdot 190274191361 \cdot \\ 1256132134125569 \cdot c_{1187}$$

$$F_{13} = 2710954639361 \cdot 2663848877152141313 \cdot 36031098445229199 \cdot \\ 319546020820551643220672513 \cdot c_{2391}$$

$$F_{14} = c_{4933}$$

$$F_{15} = 1214251009 \cdot 2327042503868417 \cdot \\ 168768817029516972383024127016961 \cdot c_{9808}$$

$$F_{16} = 825753601 \cdot 188981757975021318420037633 \cdot c_{19694}$$

$$F_{17} = 31065037602817 \cdot c_{39444}$$

$$F_{18} = 13631489 \cdot 81274690703860512587777 \cdot c_{78884}$$

$$F_{19} = 70525124609 \cdot 646730219521 \cdot c_{157804}$$

$$F_{20} = c_{315653}$$

$$F_{21} = 4485296422913 \cdot c_{631294}$$

$$F_{22} = c_{1262612}$$

$$F_{23} = 167772161 \cdot c_{2525215}$$

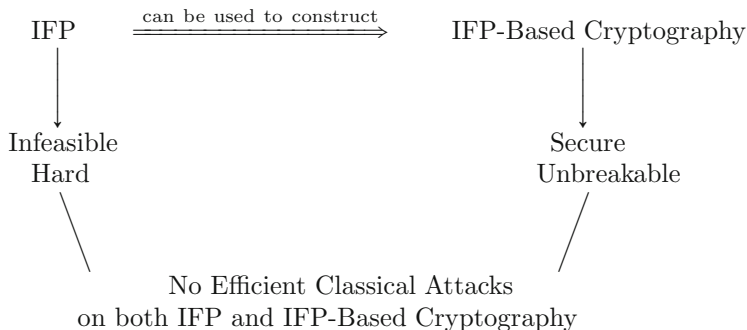
$$F_{24} = c_{5050446}$$

Basically, you are asked to factor the unfactored composite numbers, denoted by  $c_x$ , of the Fermat numbers. For example, in  $F_{12}$ ,  $c_{1187}$  is the unfactored 1187 digit composite.

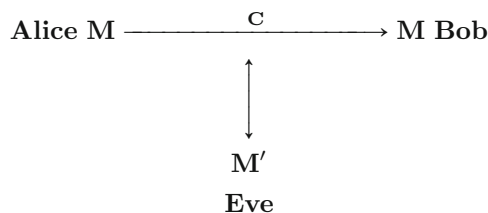
## 2.2 IFP-Based Cryptography

### Basic Idea of IFP-Based Cryptography

IFP-based cryptography is a class of cryptographic systems whose security relies on the intractability of the IFP problem:



Typical cryptographic systems in this class include RSA [64], Rabin [62], and Goldwasser–Micali probabilistic encryption [32] and Goldwasser–Micali–Rackoff zero-knowledge interactive proof [33]. We shall first give an account of the RSA cryptographic system. In a general cryptographic setting, we assume Alice wishes to send a ciphertext  $C$  of the plaintext  $M$  to Bob (or vice versa), Eve, the eavesdropper, wishes to understand the communication between Alice and Bob:



### RSA Cryptography

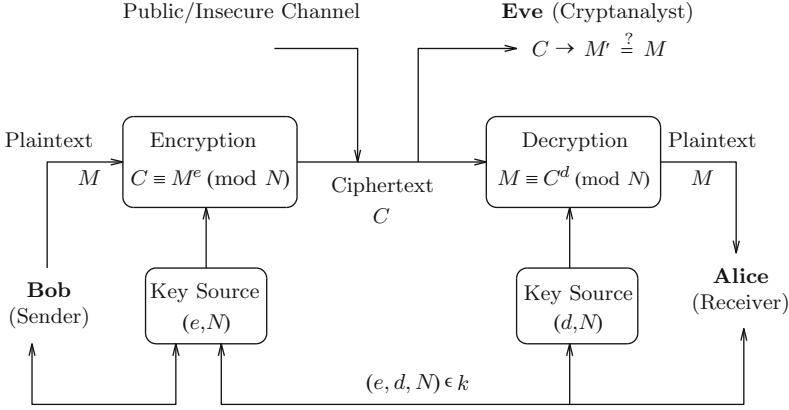
RSA is the most famous, first practical, widely used, and still unbreakable public-key cryptography, for which its three inventors, Rivest, Shamir, and Adleman, received the 2002 Turing Award. The security of RSA relies completely on the infeasibility of the IFP problem.

**Definition 2.3.** The *RSA public-key cryptosystem* may be formally defined as follows (Depicted in Fig. 2.4):

$$\text{RSA} = (\mathcal{M}, \mathcal{C}, \mathcal{K}, M, C, e, d, N, E, D) \quad (2.14)$$

where:

1.  $\mathcal{M}$  is the set of plaintexts, called the plaintext space.
2.  $\mathcal{C}$  is the set of ciphertexts, called the ciphertext space.
3.  $\mathcal{K}$  is the set of keys, called the key space.
4.  $M \in \mathcal{M}$  is a piece of particular plaintext.



**Figure 2.4.** RSA public-key cryptography

5.  $C \in \mathcal{C}$  is a piece of particular ciphertext.
6.  $N = pq$  is the modulus with  $p, q$  prime numbers, usually each with at least 100 digits.
7.  $\{(e, N), (d, N)\} \in \mathcal{K}$  with  $e \neq d$  are the encryption and encryption keys, respectively, satisfying

$$ed \equiv 1 \pmod{\phi(N)} \quad (2.15)$$

where  $\phi(N) = (p-1)(q-1)$  is the Euler  $\phi$ -function and defined by  $\phi(N) = \#(\mathbb{Z}_N^*)$ , the number of elements in the multiplicative group  $\mathbb{Z}_N^*$ .

8.  $E$  is the encryption function

$$E_{e,N} : M \mapsto C$$

That is,  $M \in \mathcal{M}$  maps to  $C \in \mathcal{C}$ , using the public-key  $(e, N)$ , such that

$$C \equiv M^e \pmod{N}. \quad (2.16)$$

9.  $D$  is the decryption function

$$D_{d,N} : C \mapsto M$$

That is,  $C \in \mathcal{C}$  maps to  $M \in \mathcal{M}$ , using the private-key  $(d, N)$ , such that

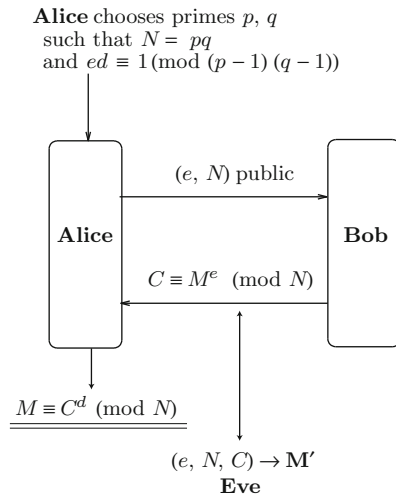
$$M \equiv C^d \equiv (M^e)^d \pmod{N}. \quad (2.17)$$

The idea of RSA can be best depicted in Fig. 2.5.

**Theorem 2.3 (The Correctness of RSA).** Let  $M, C, N, e, d$  be plaintext, ciphertext, encryption exponent, decryption exponent, and modulus, respectively. Then

$$(M^e)^d \equiv M \pmod{N}.$$





**Figure 2.5.** RSA encryption and decryption

**Proof.** Notice first that

$$\begin{aligned}
 C^d &\equiv (M^e)^d \pmod{N} && (\text{since } C \equiv M^e \pmod{N}) \\
 &\equiv M^{1+k\phi(N)} \pmod{N} && (\text{since } ed \equiv 1 \pmod{\phi(N)}) \\
 &\equiv M \cdot M^{k\phi(N)} \pmod{N} \\
 &\equiv M \cdot (M^{\phi(N)})^k \pmod{N} \\
 &\equiv M \cdot (1)^k \pmod{N} && (\text{by Euler's Theorem } a^{\phi(n)} \equiv 1 \pmod{N}) \\
 &\equiv M
 \end{aligned}$$

The result thus follows.  $\square$

Both encryption  $C \equiv M^e \pmod{N}$  and decryption  $M \equiv C^d \pmod{N}$  of RSA can be implemented in polynomial time by the fast exponentiation method. For example, the RSA encryption can be implemented as follows:

**Algorithm 2.3.** Given  $(e, M, N)$ , this algorithm finds  $C \equiv M^e \pmod{N}$ , or given  $(d, C, N)$ , finds  $M \equiv C^d \pmod{N}$  in time polynomial in  $\log e$  or  $\log d$ , respectively.

**Encryption:**

Given  $(e, M, N)$  to find  $C$   
 Set  $C \leftarrow 1$   
 While  $e \geq 1$  do  
   if  $e \bmod 2 = 1$   
     then  $C \leftarrow C \cdot M \bmod N$   
    $M \leftarrow M^2 \bmod N$   
    $e \leftarrow \lfloor e/2 \rfloor$   
 Print  $C$

**Decryption:**

Given  $(d, C, N)$  to find  $M$   
 Set  $M \leftarrow 1$   
 While  $d \geq 1$  do  
   if  $d \bmod 2 = 1$   
     then  $M \leftarrow M \cdot C \bmod N$   
    $C \leftarrow C^2 \bmod N$   
    $d \leftarrow \lfloor d/2 \rfloor$   
 Print  $M$

**Remark 2.5.** For the decryption process in RSA, as the authorized user knows  $d$  and hence knows  $p$  and  $q$ , thus instead of directly working on  $M \equiv C^d \pmod{N}$ , he can speed-up the computation by working on the following two congruences:

$$\begin{aligned} M_p &\equiv C^d \equiv C^{d \bmod p-1} \pmod{p} \\ M_q &\equiv C^d \equiv C^{d \bmod q-1} \pmod{q} \end{aligned}$$

and then use the Chinese Remainder Theorem to get

$$M \equiv M_p \cdot q \cdot q^{-1} \bmod p + M_q \cdot p \cdot p^{-1} \bmod q \pmod{N}. \quad (2.18)$$

The Chinese Remainder Theorem is a two-edged sword. On the one hand, it provides a good way to speed-up the computation/performance of the RSA decryption, which can even be easily implemented by a low-cost cryptochip [34]. On the other hand, it may introduce some serious security problems vulnerable to some side-channel attacks, particularly the random fault attacks;

**Example 2.6.** Let the letter-digit encoding be as follows:

$$\text{space} = 00, A = 01, B = 02, \dots, Z = 26.$$

(We will use this digital representation of letters throughout the book.) Let also

$$\begin{aligned} e &= 9007, \\ M &= 200805001301070903002315180419000118050019172105011309\_ \\ &\quad 190800151919090618010705, \\ N &= 114381625757888867669235779976146612010218296721242362\_ \\ &\quad 562561842935706935245733897830597123563958705058989075\_ \\ &\quad 147599290026879543541. \end{aligned}$$

Then the encryption can be done by using Algorithm 2.3:

$$\begin{aligned} C &\equiv M^e \\ &\equiv 968696137546220614771409222543558829057599911245743198\_ \\ &\quad 746951209308162982251457083569314766228839896280133919\_ \\ &\quad 90551829945157815154 \pmod{N}. \end{aligned}$$

For the decryption, since the two prime factors  $p$  and  $q$  of  $N$  are known to the authorized person who does the decryption:

$$\begin{aligned} p &= 34905295108476509491478496199038981334177646384933878\_ \\ &\quad 43990820577 \\ q &= 32769132993266709549961988190834461413177642967992942\_ \\ &\quad 539798288533 \end{aligned}$$

then

$$\begin{aligned}
 d &\equiv 1/e \\
 &\equiv 106698614368578024442868771328920154780709906633937862\_ \\
 &\equiv 801226224496631063125911774470873340168597462306553968\_ \\
 &\equiv 544513277109053606095 \pmod{(p-1)(q-1)}.
 \end{aligned}$$

Thus, the original plaintext  $M$  can be recovered either directly by using Algorithm 2.3 or indirectly by a combined use of Algorithm 2.3 and the Chinese Remainder Theorem (2.18):

$$\begin{aligned}
 M &\equiv C^d \\
 &= 200805001301070903002315180419000118050019172105011309\_ \\
 &\quad 190800151919090618010705 \pmod{N}
 \end{aligned}$$

which is “THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE.”

**Remark 2.6.** Prior to RSA, Pohlig and Hellman in 1978 [57] proposed a secret-key cryptography based on arithmetic modulo  $p$ , rather than  $N = pq$ . The Pohlig–Hellman system works as follows: Let  $M$  and  $C$  be the plain and cipher texts, respectively. Choose a prime  $p$ , usually with more than 200 digits, and a secret encryption key  $e$  such that  $e \in \mathbb{Z}^+$  and  $e \leq p-2$ . Compute  $d \equiv 1/e \pmod{(p-1)}$ .  $(e, p)$  and of course  $d$  must be kept as a secret.

[1] **Encryption:**

$$C \equiv M^e \pmod{p}. \quad (2.19)$$

This process is easy for the authorized user:

$$\{M, e, p\} \xrightarrow[\text{easy}]{\text{find}} \{C \equiv M^e \pmod{p}\}. \quad (2.20)$$

[2] **Decryption:**

$$M \equiv C^d \pmod{p}. \quad (2.21)$$

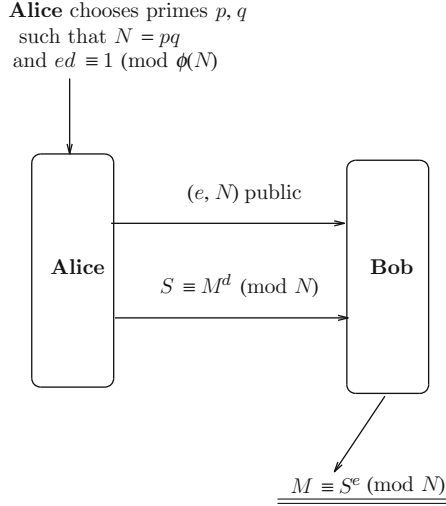
For the authorized user who knows  $(e, p)$ , this process is easy, since  $d$  can be easily computed from  $e$ .

[3] **Cryptanalysis:** The security of this system is based on the infeasibility of the discrete logarithm problem. For example, for a cryptanalyst who does not know  $e$  or  $d$  would have to compute:

$$e \equiv \log_M C \pmod{p}.$$

**Remark 2.7.** One of the most important features of RSA encryption is that it can also be used for digital signatures. Let  $M$  be a document to be signed,

and  $N = pq$  with  $p, q$  primes,  $(e, d)$  the public and private exponents as in RSA encryption scheme. Then the processes of RSA signature signing and signature verification are just the same as that of the decryption and encryption; that is, use  $d$  for signature signing and  $e$  signature verification as follows (see also Fig. 2.6):



**Figure 2.6.** RSA digital signature

[1] **Signature signing:**

$$S \equiv M^d \pmod{N} \quad (2.22)$$

The signing process can only be done by the authorized person who has the private exponent  $d$ .

[2] **Signature verification:**

$$M \equiv S^e \pmod{N} \quad (2.23)$$

This verification process can be done by anyone since  $(e, N)$  is public. Of course, RSA encryption and RSA signature can be used together to obtain a signed encrypted document to be sent over an insecure network.

## RSA Problem and RSA Assumption

As can be seen from the previous section, the whole idea of the RSA encryption and decryption is as follows:

$$\left. \begin{aligned} C &\equiv M^e \pmod{N}, \\ M &\equiv C^d \pmod{N} \end{aligned} \right\} \quad (2.24)$$

where

$$\left. \begin{aligned} ed &\equiv 1 \pmod{\phi(N)} \\ N &= pq \text{ with } p, q \in \text{Primes.} \end{aligned} \right\} \quad (2.25)$$

Thus, the *RSA function* can be defined by

$$f_{\text{RSA}} : M \mapsto M^e \bmod N. \quad (2.26)$$

The *inverse of the RSA function* is then defined by

$$f_{\text{RSA}}^{-1} : M^e \mapsto M \bmod N. \quad (2.27)$$

Clearly, the RSA function is a *one-way trap-door function*, with

$$\{d, p, q, \phi(N)\} \quad (2.28)$$

the *RSA trap-door information*. For security purposes, this set of information must be kept as a secret and should never be disclosed in anyway even in part. Now, suppose that Bob sends  $C$  to Alice, but Eve intercepts it and wants to understand it. Since Eve only has  $(e, N, C)$  and does not have any piece of the trap-door information in (2.28), then it should be infeasible/intractable for her to recover  $M$  from  $C$ :

$$\{e, N, C \equiv M^e \pmod{N}\} \xrightarrow{\text{hard}} \{M \equiv C^d \pmod{N}\}. \quad (2.29)$$

On the other hand, for Alice, since she knows  $d$ , which implies that she knows all the pieces of trap-door information in (2.28), since

$$\{d\} \xleftrightarrow{\mathcal{P}} \{p\} \xleftrightarrow{\mathcal{P}} \{q\} \xleftrightarrow{\mathcal{P}} \{\phi(N)\} \quad (2.30)$$

Thus, it is easy for Alice to recover  $M$  from  $C$ :

$$\{N, C \equiv M^e \pmod{N}\} \xrightarrow[\text{easy}]{\{d, p, q, \phi(N)\}} \{M \equiv C^d \pmod{N}\}. \quad (2.31)$$

Why is it hard for Eve to recover  $M$  from  $C$ ? This is because Eve is facing a hard computational problem, namely, the *RSA problem* [65]:

**The RSA problem:** Given the RSA public-key  $(e, N)$  and the RSA ciphertext  $C$ , find the corresponding RSA plaintext  $M$ . That is,

$$\{e, N, C\} \longrightarrow \{M\}.$$

It is conjectured although it has never been proved or disproved that:

**The RSA conjecture:** Given the RSA public-key  $(e, N)$  and the RSA ciphertext  $C$ , it is hard to find the corresponding RSA plaintext  $M$ . That is,

$$\{e, N, C\} \xrightarrow{\text{hard}} \{M\}.$$

But how hard is it for Alice to recover  $M$  from  $C$ ? This is another version of the RSA conjecture, often called the *RSA assumption*, which again has never been proved or disproved:

**The RSA assumption:** Given the RSA public-key  $(e, N)$  and the RSA ciphertext  $C$ , then finding  $M$  is as hard as factoring the RSA modulus  $N$ . That is,

$$\text{IFP}(N) \iff \text{RSA}(M)$$

provided that  $N$  is sufficiently large and randomly generated, and  $M$  and  $C$  are random integers between 0 and  $N - 1$ . More precisely, it is conjectured (or assumed) that

$$\text{IFP}(N) \stackrel{\mathcal{P}}{\iff} \text{RSA}(M).$$

That is, if  $N$  can be factorized in polynomial time, then  $M$  can be recovered from  $C$  in polynomial time. In other words, cryptanalyzing RSA must be as difficult as solving the IFP problem. But the problem is, as we discussed previously, that no one knows whether or not IFP can be solved in polynomial time, so RSA is only assumed to be secure, not proved to be secure:

$$\text{IFP}(N) \text{ is hard} \longrightarrow \text{RSA}(M) \text{ is secure.}$$

The real situation is that

$$\begin{aligned} \text{IFP}(N) &\stackrel{\vee}{\implies} \text{RSA}(M), \\ \text{IFP}(N) &\stackrel{?}{\impliedby} \text{RSA}(M). \end{aligned}$$

Now, we can return to answer the question of how hard is it for Alice to recover  $M$  from  $C$ ? By the RSA assumption, cryptanalyzing  $C$  is as hard as factoring  $N$ . The fastest known integer factorization algorithm, the NFS, runs in time

$$\mathcal{O}(\exp(c(\log N)^{1/3}(\log \log N)^{2/3}))$$

where  $c = (64/9)^{1/3}$  if a general version of NFS, GNFS, is used for factoring an arbitrary integer  $N$  whereas  $c = (32/9)^{1/3}$  if a special version of NFS, SNFS, is used for factoring a special form of integer  $N$ . As in RSA, the modulus  $N = pq$  is often chosen to be a large general composite integer  $N = pq$  with  $p$  and  $q$  of the same bit size, which makes SNFS not useful. This means that RSA cannot be broken in polynomial time but in subexponential-time, which makes RSA secure, again, by assumption. Thus, readers should note that the RSA problem is *assumed* to be *hard*, and the RSA cryptosystem is *conjectured* to be *secure*.

In the RSA cryptosystem, it is assumed that the cryptanalyst, Eve:

1. Knows the public-key  $\{e, N\}$  with  $N = pq$  and also the ciphertext  $C$
2. Does not know any one piece of the trap-door information  $\{p, q, \phi(N), d\}$
3. Wants to know  $\{M\}$

That is,

$$\{e, N, C \equiv M^e \pmod{N}\} \xrightarrow{\text{Eve wants to find}} \{M\}.$$

Obviously, there are several ways to recover  $M$  from  $C$  (i.e., to break the RSA system):

1. Factor  $N$  to get  $\{p, q\}$  so as to compute

$$M \equiv C^{1/e \pmod{(p-1)(q-1)}} \pmod{N}.$$

2. Find  $\phi(N)$  so as to compute

$$M \equiv C^{1/e \pmod{\phi(N)}} \pmod{N}.$$

3. Find  $\text{order}(a, N)$ , the order of a random integer  $a \in [2, N-2]$  modulo  $N$ , then try to find

$$\{p, q\} = \gcd(a^{r/2} \pm 1, N) \text{ and } M \equiv C^{1/e \pmod{(p-1)(q-1)}} \pmod{N}.$$

4. Find  $\text{order}(C, N)$ , the order of  $C$  modulo  $N$ , so as to compute

$$M \equiv C^{1/e \pmod{\text{order}(C, N)}} \pmod{N}.$$

5. Compute  $\log_C M \pmod{N}$ , the discrete logarithm  $M$  to the base  $C$  modulo  $N$  in order to find

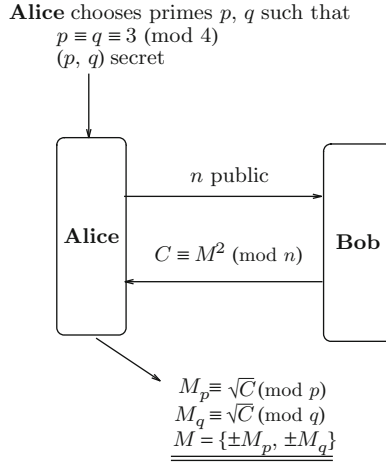
$$M \equiv C^{\log_C M \pmod{N}} \pmod{N}$$

## Rabin Cryptography

As can be seen from the previous sections, RSA uses  $M^e$  for encryption, with  $e \geq 3$  (3 is the smallest possible public exponent in RSA); in this way, we might call RSA encryption  $M^e$  encryption. In 1979, Michael Rabin [62] proposed a scheme based on  $M^2$  encryption, rather than the  $M^e$  for  $e \geq 3$  encryption used in RSA. A brief description of the Rabin cryptosystem is as follows (see also Fig. 2.7).

1. **Key generation:** Let  $n = pq$  with  $p, q$  odd primes satisfying

$$p \equiv q \equiv 3 \pmod{4}. \quad (2.32)$$



**Figure 2.7.** Rabin cryptosystem

## 2. Encryption:

$$C \equiv M^2 \pmod{n}. \quad (2.33)$$

3. **Decryption:** Use the Chinese Remainder Theorem to solve the system of congruences:

$$\begin{cases} M_p \equiv \sqrt{C} \pmod{p} \\ M_q \equiv \sqrt{C} \pmod{q} \end{cases} \quad (2.34)$$

to get the four solutions:  $\{\pm M_p, \pm M_q\}$ . The true plaintext  $M$  will be one of these four values.

4. **Cryptanalysis:** A cryptanalyst who can factor  $n$  can compute the four square roots of  $C$  modulo  $n$  and hence can recover  $M$  from  $C$ . Thus, breaking the Rabin system is equivalent to factoring  $n$ .

**Example 2.7.** Let  $M = 31$ .

- [1] **Key generation:** Let  $n = 11 \cdot 19$  be the public-key, but keep the prime factors  $p = 11$  and  $q = 19$  of  $n$  as a secret.
- [2] **Encryption:**

$$C \equiv 31^2 \equiv 125 \pmod{209}.$$

- [3] **Decryption:** Compute

$$\begin{cases} M_p \equiv \sqrt{125} \equiv \pm 2 \pmod{p} \\ M_q \equiv \sqrt{125} \equiv \pm 7 \pmod{q}. \end{cases}$$



Now, use the Chinese Remainder Theorem to solve

$$\begin{aligned} \begin{cases} M \equiv 2 \pmod{11} \\ M \equiv 7 \pmod{19} \end{cases} &\implies M = 178 \\ \begin{cases} M \equiv -2 \pmod{11} \\ M \equiv 7 \pmod{19} \end{cases} &\implies M = 64 \\ \begin{cases} M \equiv -2 \pmod{11} \\ M \equiv 7 \pmod{19} \end{cases} &\implies M = 145 \\ \begin{cases} M \equiv -2 \pmod{11} \\ M \equiv -7 \pmod{19} \end{cases} &\implies M = 31 \end{aligned}$$

The true plaintext  $M$  will be one of the above four values, and in fact,  $M = 31$  is the true value.

Unlike the RSA cryptosystem whose security was only conjectured to be equivalent to the intractability of IFP, the security of Rabin system and its variant such as Rabin–Williams system is proved to be equivalent to the intractability of IFP. First, notice that there is a fast algorithm to compute the square roots modulo  $N$  if  $n = pq$  is known. Consider the following quadratic congruence

$$x^2 \equiv y \pmod{p} \tag{2.35}$$

there are essentially three cases for the prime  $p$ :

- (1)  $p \equiv 3 \pmod{4}$ .
- (2)  $p \equiv 5 \pmod{8}$ .
- (3)  $p \equiv 1 \pmod{8}$ .

All three cases may be solved by the following process:

$$\left\{ \begin{array}{l} \text{if } p \equiv 3 \pmod{4}, \quad x \equiv \pm y^{\frac{p+1}{4}} \pmod{p}, \\ \text{if } p \equiv 5 \pmod{8}, \quad \left\{ \begin{array}{l} \text{if } y^{\frac{p+1}{4}} = 1, \quad x \equiv \pm y^{\frac{p+3}{8}} \pmod{p} \\ \text{if } y^{\frac{p+1}{4}} \neq 1, \quad x \equiv \pm 2y(4y)^{\frac{p-5}{8}} \pmod{p}. \end{array} \right. \end{array} \right. \tag{2.36}$$

**Algorithm 2.4 (Computing square roots modulo  $pq$ ).** Let  $n = pq$  with  $p$  and  $q$  odd prime and  $y \in \text{QR}_n$ . This algorithm will find all the four solutions in  $x$  to congruence  $x^2 \equiv y \pmod{pq}$  in time  $\mathcal{O}((\log p)^4)$ .

- [1] Use (2.36) to find a solution  $r$  to  $x^2 \equiv y \pmod{p}$ .

- [2] Use (2.36) to find a solution  $s$  to  $x^2 \equiv y \pmod{q}$ .
- [3] Use the Extended Euclid's algorithm to find integers  $c$  and  $d$  such that  $cp + dq = 1$ .
- [4] Compute  $x \equiv \pm(rdq \pm scp) \pmod{pq}$ .

On the other hand, if there exists an algorithm to find the four solutions in  $x$  to  $x^2 \equiv y \pmod{n}$ , then there exists an algorithm to find the prime factorization of  $n$ . The following is the algorithm.

**Algorithm 2.5 (Factoring via square roots).** This algorithm seeks to find a factor of  $n$  by using an existing square root finding algorithm (namely, Algorithm 2.4).

- [1] Choose at random an integer  $x$  such that  $\gcd(x, n) = 1$ , and compute  $x^2 \equiv a \pmod{n}$ .
- [2] Use Algorithm 2.4 to find four solutions in  $x$  to  $x^2 \equiv a \pmod{n}$ .
- [3] Choose one of the four solutions, say  $y$  such that  $y \not\equiv \pm x \pmod{n}$ , then compute  $\gcd(x \pm y, n)$ .
- [4] If  $\gcd(x \pm y, n)$  reveals  $p$  or  $q$ , then go to Step [5], or otherwise, go to Step [1].
- [5] Exit.

**Theorem 2.4.** Let  $N = pq$  with  $p, q$  odd prime. If there exists a polynomial-time algorithm  $A$  to factor  $n = pq$ , then there exists an algorithm  $B$  to find a solution to  $x^2 \equiv y \pmod{n}$ , for any  $y \in \text{QR}_N$ .

**Proof.** If there exists an algorithm  $A$  to factor  $n = pq$ , then there exists an algorithm (in fact, Algorithm 2.4), which determines  $x = \pm(rdq \pm scp) \pmod{pq}$ , as defined in Algorithm 2.4, for  $x^2 \equiv y \pmod{n}$ . Clearly, Algorithm 2.4 runs in polynomial time.  $\square$

**Theorem 2.5.** Let  $n = pq$  with  $p, q$  odd prime. If there exists a polynomial-time algorithm  $A$  to find a solution to  $x^2 \equiv a \pmod{n}$ , for any  $a \in \text{QR}_n$ , then there exists a probabilistic polynomial-time algorithm  $B$  to find a factor of  $n$ .

**Proof.** First, note that for  $n$  composite,  $x$  and  $y$  integer, if  $x^2 \equiv y^2 \pmod{n}$  but  $x \not\equiv \pm y \pmod{n}$ , then  $\gcd(x + y, n)$  are proper factors of  $n$ . If there exists an algorithm  $A$  to find a solution to  $x^2 \equiv a \pmod{n}$  for any  $a \in \text{QR}_n$ , then there exists an algorithm (in fact, Algorithm 2.5), which uses algorithm  $A$  to find four solutions in  $x$  to  $x^2 \equiv a \pmod{n}$  for a random  $x$  with  $\gcd(x, n) = 1$ . Select one of the solutions, say,  $y \not\equiv \pm x \pmod{n}$ , then by computing  $\gcd(x \pm y, n)$ , the probability of finding a factor of  $N$  will be  $\geq 1/2$ . If Algorithm 2.5 runs for  $k$  times and each time randomly chooses a different  $x$ , then the probability of not factoring  $n$  is  $\leq 1/2^k$ .  $\square$

So, finally, we have

**Theorem 2.6.** Factoring integers, computing the modular square roots, and breaking the Rabin cryptosystem are computationally equivalent. That is,

$$\text{IFP}(n) \stackrel{\mathcal{P}}{\Longleftrightarrow} \text{Rabin}(M). \quad (2.37)$$

## Residuosity-Based Cryptography

Recall that an integer  $a$  is a quadratic residue modulo  $n$ , denoted by  $a \in Q_n$ , if  $\gcd(a, n) = 1$  and there exists a solution  $x$  to the congruence  $x^2 \equiv a \pmod{n}$ , otherwise  $a$  is a quadratic non-residue modulo  $n$ , denoted by  $a \in \overline{Q}_n$ . The quadratic residuosity problem (QRP) may be stated as:

Given positive integers  $a$  and  $n$ , decide whether or not  $a \in Q_n$ .

It is believed that solving QRP is equivalent to computing the prime factorization of  $n$ , so it is computationally infeasible. If  $n$  is prime then

$$a \in Q_n \iff \left(\frac{a}{n}\right) = 1, \quad (2.38)$$

and if  $n$  is composite, then

$$a \in Q_n \implies \left(\frac{a}{n}\right) = 1, \quad (2.39)$$

but

$$a \in Q_n \stackrel{\times}{\Longleftarrow} \left(\frac{a}{n}\right) = 1, \quad (2.40)$$

however,

$$a \in \overline{Q}_n \Longleftarrow \left(\frac{a}{n}\right) = -1. \quad (2.41)$$

Let  $J_n = \{a \in (\mathbb{Z}/n\mathbb{Z})^* : \left(\frac{a}{n}\right) = 1\}$ , then  $\tilde{Q}_n = J_n - Q_n$ . Thus,  $\tilde{Q}_n$  is the set of all pseudosquares modulo  $n$ ; it contains those elements of  $J_n$  that do not belong to  $Q_n$ . Readers may wish to compare this result to Fermat's little theorem, namely (assuming  $\gcd(a, n) = 1$ ),

$$n \text{ is prime} \implies a^{n-1} \equiv 1 \pmod{n}, \quad (2.42)$$

but

$$n \text{ is prime} \stackrel{\times}{\Longleftarrow} a^{n-1} \equiv 1 \pmod{n}, \quad (2.43)$$

however,

$$n \text{ is composite} \Longleftarrow a^{n-1} \not\equiv 1 \pmod{n}. \quad (2.44)$$

The QRP can then be further restricted to:

Given a composite  $n$  and an integer  $a \in J_n$ , decide whether or not  $a \in Q_n$ .

For example, when  $n = 21$ , we have  $J_{21} = \{1, 4, 5, 16, 17, 20\}$  and  $Q_{21} = \{1, 4, 16\}$ , thus  $\tilde{Q}_{21} = \{5, 17, 20\}$ . So, the QRP problem for  $n = 21$  is actually to distinguish squares  $\{1, 4, 16\}$  from pseudosquares  $\{5, 17, 20\}$ . The only method we know for distinguishing squares from pseudosquares is to factor  $n$ ; since integer factorization is computationally infeasible, the QRP problem is computationally infeasible. In what follows, we shall present a cryptosystem whose security is based on the infeasibility of the QRP; it was first proposed by Goldwasser and Micali in 1984 [32] in 1984, under the term *probabilistic encryption*.

**Algorithm 2.6 (Quadratic residuosity-based cryptography).** This algorithm uses the randomized method to encrypt messages and is based on the QRP. The algorithm divides into three parts: key generation, message encryption, and decryption.

- [1] Key generation: Both Alice and Bob should do the following to generate their public and secret keys:
  - [a] Select two large distinct primes  $p$  and  $q$ , each with roughly the same size, say, each with  $\beta$  bits.
  - [b] Compute  $n = pq$ .  
Select a  $y \in \mathbb{Z}/n\mathbb{Z}$ , such that  $y \in \overline{Q}_n$  and  $\left(\frac{y}{n}\right) = 1$ . ( $y$  is thus a pseudosquare modulo  $n$ ).
  - [c] Make  $(n, y)$  public, but keep  $(p, q)$  secret.
- [2] Encryption: To send a message to Alice, Bob should do the following:
  - [a] Obtain Alice's public-key  $(n, y)$ .
  - [c] Represent the message  $m$  as a binary string  $m = m_1 m_2 \cdots m_k$  of length  $k$ .
  - [d] For  $i$  from 1 to  $k$  do
    - [d-1] Choose at random an  $x \in (\mathbb{Z}/n\mathbb{Z})^*$  and call it  $x_i$ .
    - [d-2] Compute  $c_i$ :

$$c_i = \begin{cases} x_i^2 \bmod n, & \text{if } m_i = 0, \quad (\text{r.s.}) \\ yx_i^2 \bmod n, & \text{if } m_i = 1, \quad (\text{r.p.s.}), \end{cases} \quad (2.45)$$

where r.s. and r.p.s. represent random square and random pseudosquare, respectively.

Send the  $k$ -tuple  $c = (c_1, c_2, \dots, c_k)$  to Alice. (Note first that each  $c_i$  is an integer with  $1 \leq c_i < n$ . Note also that since  $n$  is a  $2\beta$ -bit integer, it is clear that the cipher-text  $c$  is a much longer string than the original plain-text  $m$ .)

- [3] Decryption: To decrypt Bob's message, Alice should do the following:

[a] For  $i$  from 1 to  $k$  do

[a-1] Evaluate the Legendre symbol:

$$e'_i = \left( \frac{c_i}{p} \right). \quad (2.46)$$

[a-2] Compute  $m_i$ :

$$m_i = \begin{cases} 0, & \text{if } e'_i = 1 \\ 1, & \text{otherwise.} \end{cases} \quad (2.47)$$

That is,  $m_i = 0$  if  $c_i \in Q_n$ , otherwise,  $m_i = 1$ .

Finally, get the decrypted message  $m = m_1 m_2 \cdots m_k$ .

**Remark 2.8.** The above encryption scheme has the following interesting features:

- 1) The encryption is random in the sense that the same bit is transformed into different strings depending on the choice of the random number  $x$ . For this reason, it is called *probabilistic* (or *randomized*) encryption.
- 2) Each bit is encrypted as an integer modulo  $n$  and hence is transformed into a  $2\beta$ -bit string.
- 3) It is semantically secure against any threat from a polynomially bounded attacker, provided that the QRP is hard.

**Example 2.8.** In what follows we shall give an example of how Bob can send the message “HELP ME” to Alice using the above cryptographic method. We use the binary equivalents of letters as defined in Table 2.1. Now, both Alice

**Table 2.1.** The binary equivalents of letters

Letter	Binary code	Letter	Binary code	Letter	Binary code
A	00000	B	00001	C	00010
D	00011	E	00100	F	00101
G	00110	H	00111	I	01000
J	01001	K	01010	L	01011
J	01001	K	01010	L	01011
M	01100	N	01101	O	01110
P	01111	Q	10000	R	10001
S	10010	T	10011	U	10100
V	10101	W	10110	X	10111
Y	11000	Z	11001	□	11010

and Bob proceed as follows:

## [1] Key generation:

- Alice chooses  $(n, y) = (21, 17)$  as a public-key, where  $n = 21 = 3 \cdot 7$  is a composite and  $y = 17 \in \tilde{Q}_{21}$  (since  $17 \in J_{21}$  but  $17 \notin Q_{21}$ ), so that Bob can use the public-key to encrypt his message and send it to Alice.
- Alice keeps the prime factorization  $(3, 7)$  of 21 as a secret; since  $(3, 7)$  will be used as a private decryption key. (Of course, here we just show an example; in practice, the prime factors  $p$  and  $q$  should be at last 100 digits.)

## [2] Encryption:

- Bob converts his plaintext HELP ME to the binary stream  $M = m_1 m_2 \cdots m_{35}$ :

00111 00100 01011 01111 11010 01100 00100.

(To save space, we only consider how to encrypt and decrypt  $m_2 = 0$  and  $m_3 = 1$ ; readers are suggested to encrypt and decrypt the whole binary stream.)

- Bob randomly chooses integers  $x_i \in (\mathbb{Z}/21\mathbb{Z})^*$ . Suppose he chooses  $x_2 = 10$  and  $x_3 = 19$  which are elements of  $(\mathbb{Z}/21\mathbb{Z})^*$ .
- Bob computes the encrypted message  $C = c_1 c_2 \cdots c_k$  from the plaintext  $M = m_1 m_2 \cdots m_k$  using (2.45). To get, for example,  $c_2$  and  $c_3$ , Bob performs:

$$\begin{aligned} c_2 &= x_2^2 \bmod 21 = 10^2 \bmod 21 = 16, & \text{since } m_2 = 0, \\ c_3 &= y \cdot x_3^2 \bmod 21 = 17 \cdot 19^2 \bmod 21 = 5, & \text{since } m_3 = 1. \end{aligned}$$

(Note that each  $c_i$  is an integer reduced to 21, i.e.,  $m_i$  is a bit, but its corresponding  $c_i$  is not a bit but an integer, which is a string of bits, determined by Table 2.1.)

- Bob then sends  $c_2$  and  $c_3$  along with all other  $c_i$ 's to Alice.

[3] Decryption: To decrypt Bob's message, Alice evaluates the Legendre symbols  $\left(\frac{c_i}{p}\right)$  and  $\left(\frac{c_i}{q}\right)$ . Since Alice knows the prime factorization  $(p, q)$  of  $n$ , it should be easy for her to evaluate these Legendre symbols. For example, for  $c_2$  and  $c_3$ , Alice first evaluates the Legendre symbols  $\left(\frac{c_i}{p}\right)$ :

$$\begin{aligned} e'_2 &= \left(\frac{c_2}{p}\right) = \left(\frac{16}{3}\right) = \left(\frac{4^2}{3}\right) = 1, \\ e'_3 &= \left(\frac{c_3}{p}\right) = \left(\frac{5}{3}\right) = \left(\frac{2}{3}\right) = -1. \end{aligned}$$

then she gets

$$\begin{aligned} m_2 &= 0, & \text{since } e'_2 &= 1, \\ m_3 &= 1, & \text{since } e'_3 &= -1. \end{aligned}$$

**Remark 2.9.** The scheme introduced above is a good extension of the public-key idea but encrypts messages bit by bit. It is completely secure with respect to semantic security as well as bit security.<sup>1</sup> However, a major disadvantage of the scheme is the message expansion by a factor of  $\log n$  bit. To improve the efficiency of the scheme, Blum and Goldwasser [8] proposed in 1984 another randomized encryption scheme, in which the ciphertext is only longer than the plaintext by a constant number of bits; this scheme is comparable to the RSA scheme, both in terms of speed and message expansion.

## Problems and Exercises for Sect. 2.2

1. The RSA function  $M \mapsto C \bmod n$  is a trap-door one-way, as it is computationally intractable to invert the function if the prime factorization  $n = pq$  is unknown. Give your own trap-door one-way functions that can be used to construct public-key cryptosystems. Justify your answer.

2. Show that

$$M \equiv M^{ed} \pmod{n},$$

where  $ed \equiv 1 \pmod{\phi(n)}$ .

3. Let the ciphertexts  $C_1 \equiv M_1^e \pmod{n}$  and  $C_2 \equiv M_2^e \pmod{n}$  be as follows, where  $e = 9137$  and  $n$  is the following RSA-129 number:

46604906435060096392391122387112023736039163470082768\_  
24341038329668507346202721798200029792506708833728356\_  
7804532383891140719579,

65064096938511069741528313342475396648978551735813836\_  
77796350373814720928779386178787818974157439185718360\_  
8196124160093438830158.

Find  $M_1$  and  $M_2$ .

---

<sup>1</sup>Bit security is a special case of semantic security. Informally, bit security is concerned with not only that the whole message is not recoverable but also that individual bits of the message are not recoverable. The main drawback of the scheme is that the encrypted message is much longer than its original plaintext.

4. Let

$$\begin{aligned}
 e_1 &= 9007, \\
 e_2 &= 65537, \\
 n &= 114381625757888867669235779976146612010218296721242362\_ \\
 &\quad 562561842935706935245733897830597123563958705058989075\_ \\
 &\quad 147599290026879543541, \\
 C_1 &\equiv M^{e_1} \pmod{n}, \\
 &\equiv 10420225094119623841363838260797412577444908472492959\_ \\
 &\quad 12574337458892652977717171824130246429380783519790899\_ \\
 &\quad 45343407464161377977212, \\
 C_2 &\equiv M^{e_2} \pmod{n} \\
 &\equiv 76452750729188700180719970517544574710944757317909896\_ \\
 &\quad 04134098748828557319028078348030908497802156339649075\_ \\
 &\quad 9750600519496071304348.
 \end{aligned}$$

Find the plaintext  $M$ .

5. (Rivest) Let

$$k = 2^{2^t} \pmod{n}$$

where

$$\begin{aligned}
 n &= 63144660830728888937993571261312923323632988 \\
 &\quad 18330841375588990772701957128924885547308446 \\
 &\quad 05575320651361834662884894808866350036848039 \\
 &\quad 65881713619876605218972678101622805574753938 \\
 &\quad 38308261759713218926668611776954526391570120 \\
 &\quad 69093997368008972127446466642331918780683055 \\
 &\quad 20679512530700820202412462339824107377537051 \\
 &\quad 27344494169501180975241890667963858754856319 \\
 &\quad 80550727370990439711973361466670154390536015 \\
 &\quad 25433739825245793135753176536463319890646514 \\
 &\quad 02133985265800341991903982192844710212464887 \\
 &\quad 45938885358207031808428902320971090703239693 \\
 &\quad 49199627789953233201840645224764639663559373 \\
 &\quad 67009369212758092086293198727008292431243681, \\
 t &= 79685186856218.
 \end{aligned}$$

Find  $k$ . (Note that to find  $k$ , one needs to find  $2^t \pmod{\phi(n)}$  first; however, to find  $\phi(n)$  one needs to factor  $n$  first.)

6. (Knuth) Let

$$\{C_1, C_2\} \equiv \{M_1^3, M_2^3\} \pmod{n}$$



where

$$\begin{aligned}
 C_1 &= 687502836437089289878995350604407990716898140258583443 \\
 &\quad 035535588237479271080090293049630566651268112334056274 \\
 &\quad 332612142823187203731181519639442616568998924368271227 \\
 &\quad 5123771458797372299204125753023665954875641382171 \\
 C_2 &= 713013988616927464542046650358646224728216664013755778 \\
 &\quad 567223219797011593220849557864249703775331317377532696 \\
 &\quad 534879739201868887567829519032681632688812750060251822 \\
 &\quad 3884462866157583604931628056686699683334519294663 \\
 n &= 779030228851015954236247565470557836248576762097398394 \\
 &\quad 108440222213572872511709998585048387648131944340510932 \\
 &\quad 265136815168574119934775586854274094225644500087912723 \\
 &\quad 2585749337061853958340278434058208881085485078737.
 \end{aligned}$$

Find  $\{M_1, M_2\}$ . (Note that there are two known ways to find  $\{M_1, M_2\}$ :

$$M_i \equiv \sqrt[3]{C_i} \pmod{n},$$

$$M_i \equiv C_i^d \pmod{n},$$

where  $i = 1, 2$ . But in either way, one needs to find  $n$  first.

7. The original version of the RSA cryptosystem:

$$C \equiv M^e \pmod{n}, \quad M \equiv C^d \pmod{n},$$

with

$$ed \equiv 1 \pmod{\phi(n)}$$

is a type of deterministic cryptosystem, in which the same ciphertext is obtained for the same plaintext even at a different time. That is,

$$\begin{aligned}
 M_1 &\xrightarrow{\text{Encryption at Time 1}} C_1, \\
 M_1 &\xrightarrow{\text{Encryption at Time 2}} C_1, \\
 &\quad \vdots \\
 M_1 &\xrightarrow{\text{Encryption at Time } t} C_1.
 \end{aligned}$$

A randomized cryptosystem is one in which different ciphertext is obtained at a different time even for the same plaintext

$$M_1 \xrightarrow{\text{Encryption at Time 1}} C_1,$$

$$\begin{array}{ccc}
 M_1 & \xrightarrow{\text{Encryption at Time 2}} & C_2, \\
 & \vdots & \\
 M_1 & \xrightarrow{\text{Encryption at Time } t} & C_t,
 \end{array}$$

with  $C_1 \neq C_2 \neq \dots \neq C_t$ . Describe a method to make RSA a randomized cryptosystem.

8. Describe a man-in-the-middle attack on the original version of the RSA cryptosystem.
9. Show that cracking RSA or any IFP-based cryptography is generally equivalent to solving the IFP problem.
10. Let

$$\begin{aligned}
 n = & 21290246318258757547497882016271517497806703963277216278233 \\
 & 3832153847057041325010289010897698254819258255135092526096 \\
 & 02369983944024335907529
 \end{aligned}$$

$$\begin{aligned}
 C &\equiv M^2 \pmod{n} \\
 &= 51285205060243481188122109876540661122140906807437327290641 \\
 &\quad 6063392024247974145084119668714936527203510642341164827936 \\
 &\quad 3932042884271651389234
 \end{aligned}$$

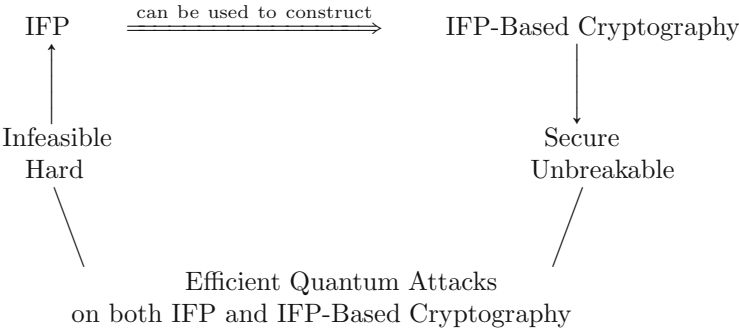
Find the plaintext  $M$ .

## 2.3 Quantum Attacks on IFP and IFP-Based Cryptography

As the security of RSA or any IFP-related cryptography relies on the intractability of the IFP problem, if IFP can be solved in polynomial time, all the IFP-related cryptography can be broken efficiently in polynomial time. In this section, we discuss quantum attacks on IFP and IFP-related cryptography.

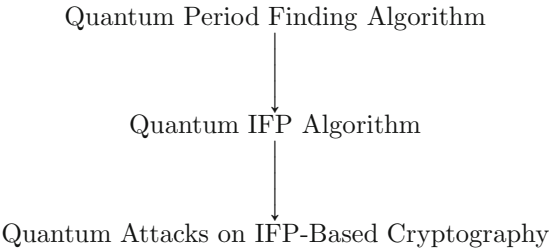
**Relationships Between IFP and IFP-Based Cryptography**

As can be seen, IFP is a conjectured (i.e., unproved) infeasible problem in computational number theory; this would imply that the cryptographic system-based DLP is secure and unbreakable in polynomial time:



Thus, anyone who can solve IFP can break IFP-based cryptography. With this regard, solving IFP is equivalent to breaking IFP-based cryptography. As everybody knows at present, no efficient algorithm is known for solving IFP, therefore, no efficient algorithm for cracking IFP-based cryptography. However, Shor [73] showed that IFP can be solved in  $\mathcal{BQP}$ , where  $\mathcal{BQP}$  is the class of problem that is efficiently solvable in polynomial time on a quantum Turing machine (see Fig. 2.8).

Hence, all IFP-based cryptographic systems can be broken in polynomial time on a quantum computer. Incidentally, the quantum factoring attack is intimately connected to the order finding problem which can be done in polynomial time on a quantum computer. More specifically, using the quantum order finding algorithm, the quantum factoring attack can break all IFP-based cryptographic systems, such as RSA and Rabin, which can be broken completely in polynomial time on a quantum computer :



### Algorithms for Quantum Computation: Discrete Logarithms and Factoring

Peter W. Shor  
AT&T Bell Labs  
Room 2D-149  
600 Mountain Ave.  
Murray Hill, NJ 07974, USA

#### Abstract

*A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a cost in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their computational properties. This paper gives Las Vegas algorithms for finding discrete logarithms and factoring integers on a quantum computer that take a number of steps which is polynomial in the input size, e.g., the number of digits of the integer to be factored. These two problems are generally considered hard on a classical computer and have been used as the basis of several proposed cryptosystems. (We thus give the first examples of quantum cryptanalysis.)*

#### 1 Introduction

Since the discovery of quantum mechanics, people have found the behavior of the laws of probability in quantum mechanics counterintuitive. Because of this behavior, quantum mechanical phenomena behave quite differently than the phenomena of classical physics that we are used to. Feynman seems to have been the first to ask what effect this has on computation [13, 14]. He gave arguments as to why this behavior might make it intrinsically computationally expensive to simulate quantum mechanics on a classical (or von Neumann) computer. He also suggested the possibility of using a computer based on quantum mechanical principles to avoid this problem, thus implicitly asking the converse question: by using quantum mechanics in a computer can you compute more efficiently than on a classical computer. Other early work in the field of quantum mechanics and computing was done by Benioff

[1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum computation relates to classical complexity classes. We will thus first give a brief intuitive discussion of complexity classes for those readers who do not have this background. There are generally two resources which limit the ability of computers to solve large problems: time and space (i.e., memory). The field of analysis of algorithms considers the asymptotic demands that algorithms make for these resources as a function of the problem size. Theoretical computer scientists generally classify algorithms as efficient when the number of steps of the algorithms grows as a polynomial in the size of the input. The class of problems which can be solved by efficient algorithms is known as P. This classification has several nice properties. For one thing, it does a reasonable job of reflecting the performance of algorithms in practice (although an algorithm whose running time is the tenth power of the input size, say, is not truly efficient). For another, this classification is nice theoretically, as different reasonable machine models produce the same class P. We will see this behavior reappear in quantum computation, where different models for quantum machines will vary in running times by no more than polynomial factors.

There are also other computational complexity classes discussed in this paper. One of these is PSPACE, which are those problems which can be solved with an amount of memory polynomial in the input size. Another important complexity class is NP, which intuitively is the class of exponential search problems. These are problems which may require the search of an exponential size space to find



0272-5428/94 \$04.00 © 1994 IEEE

124

**Figure 2.8.** David Deutsch and the first page of his 1985 paper

## Order Finding Problem

We first present some basic concept of the *order* of an element in a multiplicative group.

**Definition 2.4.** Let  $G = \mathbb{Z}_N^*$  be a finite multiplicative group, and  $x \in G$  a randomly chosen integer (element). Then order of  $x$  in  $G$ , or order of an element  $a$  modulo  $N$ , sometimes denoted by  $\text{order}(x, N)$ , is the smallest positive integer  $r$  such that

$$x^r \equiv 1 \pmod{N}.$$

**Example 2.9.** Let  $5 \in \mathbb{Z}_{104}^*$ . Then  $\text{order}(5, 104) = 4$ , since 4 is the smallest positive integer satisfying

$$5^4 \equiv 1 \pmod{104}.$$

**Theorem 2.7.** Let  $G$  be a finite group and suppose that  $x \in G$  has finite order  $r$ . If  $x^k = 1$ , then  $r \mid k$ .

**Example 2.10.** Let  $5 \in \mathbb{Z}_{104}^*$ . As  $5^{24} \equiv 1 \pmod{104}$ , so,  $4 \mid 24$ .

**Definition 2.5.** Let  $G$  be a finite group, then the number of elements in  $G$ , denoted by  $|G|$ , is called the *order* of  $G$ .

**Example 2.11.** Let  $G = \mathbb{Z}_{104}^*$ . Then there are 48 elements in  $G$  that are relatively prime to 104 (two numbers  $a$  and  $b$  are relatively prime if  $\gcd(a, b) = 1$ ), namely,

1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 41, 43  
45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 67, 69, 71, 73, 75, 77, 79, 81  
83, 85, 87, 89, 93, 95, 97, 99, 101, 103

Thus,  $|G| = 48$ . That is, the order of the group  $G$  is 48.

**Theorem 2.8 (Lagrange).** Let  $G$  be a finite group. Then the order of an element  $x \in G$  divides the order of the group  $G$ .

**Example 2.12.** Let  $G = \mathbb{Z}_{104}^*$ . Then the order of  $G$  is 48, whereas the order of the element  $5 \in G$  is 4. Clearly  $4 \mid 48$ .

**Corollary 2.1.** If a finite group  $G$  has order  $r$ , then  $x^r = 1$  for all  $x \in G$ .

**Example 2.13.** Let  $G = \mathbb{Z}_{104}^*$  and  $|G| = 48$ . Then

$$\begin{aligned} 1^{48} &\equiv 1 \pmod{104} \\ 3^{48} &\equiv 1 \pmod{104} \\ 5^{48} &\equiv 1 \pmod{104} \\ 7^{48} &\equiv 1 \pmod{104} \\ &\vdots \\ 101^{48} &\equiv 1 \pmod{104} \\ 103^{48} &\equiv 1 \pmod{104}. \end{aligned}$$

Now, we are in a position to present our two main theorems as follows.

**Theorem 2.9.** Let  $C$  be the RSA ciphertext, and  $\text{order}(C, N)$  the order of  $C \in \mathbb{Z}_N^*$ . Then

$$d \equiv 1/e \pmod{\text{order}(C, N)}.$$

**Corollary 2.2.** Let  $C$  be the RSA ciphertext, and  $\text{order}(C, N)$  the order of  $C \in \mathbb{Z}_N^*$ . Then

$$M \equiv C^{1/e \pmod{\text{order}(C, N)}} \pmod{N}$$

Thus, to recover the RSA  $M$  from  $C$ , it suffices to just find the order of  $C$  modulo  $N$ .

Now, we return to the actual computation of the order of an element  $x$  in  $G = \mathbb{Z}_N^*$ . Finding the order of an element  $x$  in  $G$  is, in theory, not a

problem: Just keep multiplying until we get to “1,” the identity element of the multiplicative group  $G$ . For example, let  $N = 179359$ ,  $x = 3 \in G$ , and  $G = \mathbb{Z}_{179359}^*$ , such that  $\gcd(3, 179359) = 1$ . To find the order  $r = \text{order}(3, 179359)$ , we just keep multiplying until we get to “1”:

$$\begin{array}{rclcl}
 3^1 & \text{mod} & 179359 & = & 3 \\
 3^2 & \text{mod} & 179359 & = & 9 \\
 3^3 & \text{mod} & 179359 & = & 27 \\
 & & & & \vdots \\
 3^{1000} & \text{mod} & 179359 & = & 31981 \\
 3^{1001} & \text{mod} & 179359 & = & 95943 \\
 3^{1002} & \text{mod} & 179359 & = & 108470 \\
 & & & & \vdots \\
 3^{14716} & \text{mod} & 179359 & = & 99644 \\
 3^{14717} & \text{mod} & 179359 & = & 119573 \\
 3^{14718} & \text{mod} & 179359 & = & 1.
 \end{array}$$

Thus, the order  $r$  of 3 in the multiplicative group  $\mathcal{G} = (\mathbb{Z}/179359\mathbb{Z})^*$  is 14718, that is,  $\text{ord}_{179359}(3) = 14718$ .

**Example 2.14.** Let

$$\begin{aligned}
 N &= 5515596313 \\
 e &= 1757316971 \\
 C &= 763222127 \\
 r &= \text{order}(C, N) = 114905160
 \end{aligned}$$

Then

$$\begin{aligned}
 M &\equiv C^{1/e \bmod r} \pmod{N} \\
 &\equiv 763222127^{1/1757316971 \bmod 114905160} \pmod{5515596313} \\
 &\equiv 1612050119
 \end{aligned}$$

Clearly, this result is correct, since

$$\begin{aligned}
 M^e &\equiv 1612050119^{1757316971} \\
 &\equiv 763222127 \\
 &\equiv C \pmod{5515596313}
 \end{aligned}$$

It must also be noted, however, that in practice, the above computation for finding the order of  $x \in (\mathbb{Z}/N\mathbb{Z})^*$  may not work, since for an element  $x$  in a large group  $\mathcal{G}$  with  $N$  having more than 200 digits, the computation of  $r$  may require more than  $10^{150}$  multiplications. Even if these multiplications could be carried out at the rate of 1,000 billion/s on a supercomputer, it would take approximately  $3 \cdot 10^{80}$  years to arrive at the answer. Thus, the order finding

problem is intractable on conventional digital computers. The problem is, however, tractable on quantum computers, provided that a practical quantum computer is available.

It is worthwhile pointing out that although the order is hard to find, the exponentiation is easy to compute. Suppose we want to compute  $x^e \bmod n$  with  $x, e, n \in \mathbb{N}$ . Suppose moreover that the binary form of  $e$  is as follows:

$$e = \beta_k 2^k + \beta_{k-1} 2^{k-1} + \cdots + \beta_1 2^1 + \beta_0 2^0, \quad (2.48)$$

where each  $\beta_i$  ( $i = 0, 1, 2, \dots, k$ ) is either 0 or 1. Then we have

$$\begin{aligned} x^e &= x^{\beta_k 2^k + \beta_{k-1} 2^{k-1} + \cdots + \beta_1 2^1 + \beta_0 2^0} \\ &= \prod_{i=0}^k x^{\beta_i 2^i} \\ &= \prod_{i=0}^k \left( x^{2^i} \right)^{\beta_i}. \end{aligned} \quad (2.49)$$

Furthermore, by the exponentiation law,

$$x^{2^{i+1}} = (x^{2^i})^2, \quad (2.50)$$

and so the final value of the exponentiation can be obtained by *repeated squaring and multiplication* operations. For example, to compute  $a^{100}$ , we first write  $100_{10} = 1100100_2 := e_6 e_5 e_4 e_3 e_2 e_1 e_0$ , and then compute

$$\begin{aligned} a^{100} &= ((((((a)^2 \cdot a)^2)^2 \cdot a)^2)^2 \\ &\Rightarrow a, a^3, a^6, a^{12}, a^{24}, a^{25}, a^{50}, a^{100}. \end{aligned} \quad (2.51)$$

Note that for each  $e_i$ , if  $e_i = 1$ , we perform a *squaring* and a *multiplication* operation (except “ $e_6 = 1$ ,” for which we just write down  $a$ , as indicated in the first bracket); otherwise, we perform only a *squaring* operation. That is,

$e_6$	1	$a$	$a$	initialization
$e_5$	1	$(a)^2 \cdot a$	$a^3$	squaring and multiplication
$e_4$	0	$((a)^2 \cdot a)^2$	$a^6$	squaring
$e_3$	0	$((((a)^2 \cdot a)^2)^2)$	$a^{12}$	squaring
$e_2$	1	$(((((a)^2 \cdot a)^2)^2)^2 \cdot a$	$a^{25}$	squaring and multiplication
$e_1$	0	$((((((a)^2 \cdot a)^2)^2)^2 \cdot a)^2$	$a^{50}$	squaring
$e_0$	0	$((((((((a)^2 \cdot a)^2)^2)^2 \cdot a)^2)^2$	$a^{100}$	squaring
		$\parallel$		
		$a^{100}$		

The following is the algorithm, which runs in  $\mathcal{O}(\log e)$  arithmetic operations and  $\mathcal{O}((\log e)(\log n)^2)$  bit operations.

**Algorithm 2.7 (Fast modular exponentiation  $x^e \bmod n$ ).** This algorithm will compute the modular exponentiation

$$c \equiv x^e \pmod{n},$$

where  $x, e, n \in \mathbb{N}$  with  $n > 1$ . It requires at most  $2 \log e$  and  $2 \log e$  divisions (divisions are only needed for modular operations; they can be removed if only  $c = x^e$  are required to be computed).

[1] [Precomputation] Let

$$e_{\beta-1}e_{\beta-2} \cdots e_1e_0 \quad (2.52)$$

be the binary representation of  $e$  (i.e.,  $e$  has  $\beta$  bits). For example, for  $562 = 1000110010$ , we have  $\beta = 10$  and

$$\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ e_9 & e_8 & e_7 & e_6 & e_5 & e_4 & e_3 & e_2 & e_1 & e_0 \end{array}$$

[2] [Initialization] Set  $c \leftarrow 1$ .

[3] [Modular Exponentiation] Compute  $c = x^e \bmod n$  in the following way:

for  $i$  from  $\beta - 1$  down to 0 do  
 $c \leftarrow c^2 \bmod n$  (squaring)  
 if  $e_i = 1$  then  
 $c \leftarrow c \cdot x \bmod n$  (multiplication)

[4] [Exit] Print  $c$  and terminate the algorithm.

## Quantum Order Computing

It may be the case that, as the famous physicist Feynman mentioned, nobody understands quantum mechanics, some progress has been made in quantum mechanics, particularly in quantum computing and quantum cryptography. In this section, we present a quantum algorithm for computing the order of an element  $x$  in the multiplicative group  $\mathbb{Z}_N^*$ , due to Shor [69]. The main idea of Shor's algorithm is as follows. First of all, we create two quantum registers for our quantum computer: Register-1 and Register-2. Of course, we can create just one single quantum memory register partitioned into two parts. Secondly, we create in Register-1 a superposition of the integers  $a = 0, 1, 2, 3, \dots$  which will be the arguments of  $f(a) = x^a \pmod{N}$ , and load Register-2 with all zeros. Thirdly, we compute in Register-2  $f(a) = x^a \pmod{N}$  for each input  $a$ . (Since the values of  $a$  are kept in Register-1, this can be done reversibly.) Fourthly, we perform the discrete Fourier transform on Register-1. Finally, we observe both registers of the machine and find the order  $r$  that satisfies  $x^r \equiv 1 \pmod{N}$ . The following is a brief description of the quantum algorithm for the order finding problem.



**Algorithm 2.8 (Quantum order finding attack).** Given RSA ciphertext  $C$  and modulus  $N$ , this attack will first find the order,  $r$ , of  $C$  in  $\mathbb{Z}_N^*$ , such that  $C^r \equiv 1 \pmod{N}$ , then recover the plaintext  $M$  from the ciphertext  $C$ . Assume the quantum computer has two quantum registers: Register-1 and Register-2, which hold integers in binary form.

- [1] (Initialization) Find a number  $q$ , a power of 2, say  $2^t$ , with  $N^2 < q < 2N^2$ .
- [2] (Preparation for quantum registers) Put in the first  $t$ -qubit register, Register-1, the uniform superposition of states representing numbers  $a \pmod{q}$ , and load Register-2 with all zeros. This leaves the machine in the state  $|\Psi_1\rangle$ :

$$|\Psi_1\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle |0\rangle.$$

(Note that the joint state of both registers are represented by  $|\text{Register-1}\rangle$  and  $|\text{Register-2}\rangle$ ). What this step does is put each qubit in Register-1 into the superposition

$$\frac{1}{\sqrt{2}} (|0\rangle + |1\rangle).$$

- [3] (Power Creation) Fill in the second  $t$ -qubit register, Register-2, with powers  $C^a \pmod{N}$ . This leaves the machine in state  $|\Psi_2\rangle$ :

$$|\Psi_2\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle |C^a \pmod{N}\rangle.$$

This step can be done reversibly since all the  $a$ 's were kept in Register-1.

- [4] (Perform a quantum FFT) Apply FFT on Register-1. The FFT maps each state  $|a\rangle$  to

$$\frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} \exp(2\pi iac/q) |c\rangle.$$

That is, we apply the unitary matrix with the  $(a, c)$  entry equal to  $\frac{1}{\sqrt{q}} \exp(2\pi iac/q)$ . This leaves the machine in the state  $|\Psi_3\rangle$ :

$$|\Psi_3\rangle = \frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} \exp(2\pi iac/q) |c\rangle |C^a \pmod{N}\rangle.$$

- [5] (Periodicity Detection in  $x^a$ ) Observe both  $|c\rangle$  in Register-1 and  $|C^a \pmod{N}\rangle$  in Register-2 of the machine, measure both arguments of this superposition, obtaining the values of  $|c\rangle$  in the first argument and some  $|x^k \pmod{n}\rangle$  as the answer for the second one ( $0 < k < r$ ).

- [6] (Extract  $r$ ) Extract the required value of  $r$ . Given the pure state  $|\Psi_3\rangle$ , the probabilities of different results for this measurement will be given by the probability distribution:

$$\begin{aligned}
 \text{Prob}(c, C^k \pmod N) &= \left| \frac{1}{q} \sum_{\substack{a=0 \\ C^a \equiv a^k \pmod N}}^{q-1} \exp(2\pi i ac/q) \right|^2 \\
 &= \left| \frac{1}{q} \sum_{B=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i (br+k)c/q) \right|^2 \\
 &= \left| \frac{1}{q} \sum_{B=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i b\{rc\}/q) \right|^2
 \end{aligned}$$

where  $\{rc\}$  is  $rc \pmod N$ . As shown in [69],

$$\begin{aligned}
 \frac{-r}{2} \leq \{rc\} \leq \frac{-r}{2} &\implies \frac{-r}{2} \leq rc - dq \leq \frac{-r}{2}, \text{ for some } d \\
 &\implies \text{Prob}(c, C^k \pmod N) > \frac{1}{3r^2}.
 \end{aligned}$$

then we have

$$\left| \frac{c}{q} - \frac{d}{r} \right| \leq \frac{1}{2q}.$$

Since  $\frac{c}{q}$  were known,  $r$  can be obtained by the continued fraction expansion of  $\frac{c}{q}$ .

- [7] (Code Breaking) Once the order  $r$ ,  $r = \text{order}(C, N)$ , is found, then compute:

$$M \equiv C^{1/e \pmod r} \pmod N.$$

Hence, decodes the RSA code  $C$ .

**Theorem 2.10. (Complexity of Quantum Order Finding Attack).** Quantum order attack can find  $\text{order}(C, N)$  and recover  $M$  from  $C$  in time  $\mathcal{O}((\log N)^{2+\epsilon})$ .

**Remark 2.10.** This quantum attack is for particular RSA ciphertexts  $C$ . In this special case, the factorization of the RSA modulus  $N$  is not needed. In the next section, we shall consider the more general quantum attack by factoring  $N$ .

## Quantum Integer Factorization

Instead of finding the order of  $C$  in  $\mathbb{Z}_N^*$ , one can take this further to a more general case: find the order of an element  $x$  in  $\mathbb{Z}_N^*$ , denoted by  $\text{order}(x, N)$ , where  $N$  is the RSA modulus. Once the order of an element  $x$  in  $\mathbb{Z}_N^*$  is found, and it is even, it will have a good chance to factor  $N$ , of course in polynomial time, by just calculating

$$\left\{ \gcd(x^{r/2} + 1, N), \gcd(x^{r/2} - 1, N) \right\}.$$

For instance, for  $x = 3$ ,  $r = 14718$ , and  $N = 179359$ , we have

$$\left\{ \gcd(3^{14718/2} + 1, 179359), \gcd(3^{14718/2} - 1, 179359) \right\} = (67, 2677),$$

and hence the factorization of  $N$ :

$$N = 179359 = 67 \cdot 2677.$$

The following theorem shows that the probability for  $r$  to work is high.

**Theorem 2.11.** Let the odd integer  $N > 1$  have exactly  $k$  distinct prime factors. For a randomly chosen  $x \in \mathbb{Z}_N^*$  with multiplicative order  $r$ , the probability that  $r$  is even and that

$$x^{r/2} \not\equiv -1 \pmod{N}$$

is least  $1 - 1/2^{k-1}$ . More specifically, if  $N$  has just two prime factors (this is often the case for the RSA modulus  $N$ ), then the probability is at least  $1/2$ .

### Algorithm 2.9 (Quantum algorithm for integer factorization).

Given integers  $x$  and  $N$ , the algorithm will

- find the order of  $x$ , i.e., the smallest positive integer  $r$  such that

$$x^r \equiv 1 \pmod{N},$$

- find the prime factors of  $N$  and compute the decryption exponent  $d$ ,
- decode the RSA message.

Assume the machine has two quantum registers: Register-1 and Register-2, which hold integers in binary form.

- [1] (Initialization) Find a number  $q$ , a power of 2, say  $2^t$ , with  $N^2 < q < 2N^2$ .
- [2] (Preparation for quantum registers) Put in the first  $t$ -qubit register, Register-1, the uniform superposition of states representing numbers  $a \pmod{q}$ , and load Register-2 with all zeros. This leaves the machine in the state  $|\Psi_1\rangle$ :

$$|\Psi_1\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle |0\rangle.$$

(Note that the joint state of both registers are represented by  $|\text{Register-1}\rangle$  and  $|\text{Register-2}\rangle$ ). What this step does is put each qubit in Register-1 into the superposition

$$\frac{1}{\sqrt{2}} (|0\rangle + |1\rangle).$$

[3] (Base Selection) Choose a random  $x \in [2, N-2]$  such that  $\gcd(x, N) = 1$ .

[4] (Power Creation) Fill in the second  $t$ -qubit register, Register-2, with powers  $x^a \pmod{N}$ . This leaves the machine in state  $|\Psi_2\rangle$ :

$$|\Psi_2\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle |x^a \pmod{N}\rangle.$$

This step can be done reversibly since all the  $a$ 's were kept in Register-1.

[5] (Perform a quantum FFT) Apply FFT on Register-1. The FFT maps each state  $|a\rangle$  to

$$\frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} \exp(2\pi iac/q) |c\rangle.$$

That is, we apply the unitary matrix with the  $(a, c)$  entry equal to  $\frac{1}{\sqrt{q}} \exp(2\pi iac/q)$ . This leaves the machine in the state  $|\Psi_3\rangle$ :

$$|\Psi_3\rangle = \frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} \exp(2\pi iac/q) |c\rangle |x^a \pmod{N}\rangle.$$

[6] (Periodicity Detection in  $x^a$ ) Observe both  $|c\rangle$  in Register-1 and  $|x^a \pmod{N}\rangle$  in Register-2 of the machine, measure both arguments of this superposition, obtaining the values of  $|c\rangle$  in the first argument and some  $|x^k \pmod{n}\rangle$  as the answer for the second one ( $0 < k < r$ ).

[7] (Extract  $r$ ) Extract the required value of  $r$ . Given the pure state  $|\Psi_3\rangle$ , the probabilities of different results for this measurement will be given by the probability distribution:

$$\begin{aligned} \text{Prob}(c, x^k \pmod{N}) &= \left| \frac{1}{q} \sum_{\substack{a=0 \\ x^a \equiv x^k \pmod{N}}}^{q-1} \exp(2\pi iac/q) \right|^2 \\ &= \left| \frac{1}{q} \sum_{B=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i(br+k)c/q) \right|^2 \\ &= \left| \frac{1}{q} \sum_{B=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi ib\{rc\}/q) \right|^2 \end{aligned}$$

where  $\{rc\}$  is  $rc \bmod N$ . As showed in [69],

$$\begin{aligned} \frac{-r}{2} \leq \{rc\} \leq \frac{-r}{2} &\implies \frac{-r}{2} \leq rc - dq \leq \frac{-r}{2}, \text{ for some } d \\ &\implies \text{Prob}(c, x^k \bmod N) > \frac{1}{3r^2}. \end{aligned}$$

then we have

$$\left| \frac{c}{q} - \frac{d}{r} \right| \leq \frac{1}{2q}.$$

Since  $\frac{c}{q}$  were known,  $r$  can be obtained by the continued fraction expansion of  $\frac{c}{q}$ .

- [8] (Resolution) If  $r$  is odd, go to Step [3] to start a new base. If  $r$  is even, then try to compute. Once  $r$  is found, the factors of  $N$  can be *possibly*

$$\{\gcd(x^{r/2} - 1, N), \gcd(x^{r/2} + 1, N)\}$$

Hopefully, this will produce two factors of  $N$ .

- [9] (Computing  $d$ ) Once  $N$  is factored and  $p$  and  $q$  are found, then compute

$$d \equiv 1/e \pmod{(p-1)(q-1)}.$$

- [10] (Code Break) As soon as  $d$  is found, and RSA ciphertext encrypted by the public-key  $(e, N)$ , can be decrypted by this  $d$  as follows:

$$M \equiv C^d \pmod{N}.$$

**Theorem 2.12 (Complexity of Quantum Factoring).** Quantum factoring algorithm can factor the RSA modulus  $N$  and break the RSA system in time  $\mathcal{O}((\log N)^{2+\epsilon})$ .

**Remark 2.11.** The attack discussed in Algorithm 2.9 is more general than that in Algorithm 2.8. Algorithm 2.9 also implies that if a practical quantum computer can be built, then the RSA cryptosystem can be completely broken, and a quantum resistant cryptosystem must be developed and used to replace the RSA cryptosystem.

**Example 2.15.** On 19 December 2001, IBM made the first demonstration of the quantum factoring algorithm [77] that correctly identified 3 and 5 as the factors of 15. Although the answer may appear to be trivial, it may have good potential and practical implication. In this example, we show how to factor 15 quantum-mechanically [56]:

- [1] Choose at random  $x = 7$  such that  $\gcd(x, N) = 1$ . We aim to find  $r = \text{order}_{15} 7$  such that  $7^r \equiv 1 \pmod{15}$ .

- [2] Initialize two four-qubit registers to state 0:

$$|\Psi_0\rangle = |0\rangle|0\rangle.$$

- [3] Randomize the first register as follows:

$$|\Psi_0\rangle \rightarrow |\Psi_1\rangle = \frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} |k\rangle|0\rangle.$$

- [4] Unitarily compute the function  $f(a) \equiv 13^a \pmod{15}$  as follows:

$$\begin{aligned} |\Psi_1\rangle \rightarrow |\Psi_2\rangle &= \frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} |k\rangle |13^k \pmod{15}\rangle \\ &= \frac{1}{\sqrt{2^t}} [ |0\rangle|1\rangle + |1\rangle|7\rangle + |2\rangle|4\rangle + |3\rangle|13\rangle + \\ &\quad |4\rangle|1\rangle + |5\rangle|7\rangle + |6\rangle|4\rangle + |7\rangle|13\rangle + \\ &\quad |8\rangle|1\rangle + |9\rangle|7\rangle + |10\rangle|4\rangle + |11\rangle|13\rangle + \\ &\quad + \dots ] \end{aligned}$$

- [5] We now apply the FFT to the second register and measure it (it can be done in the first), obtaining a random result from 1, 7, 4, 13. Suppose we incidently get 4, then the state input to FFT would be

$$\sqrt{\frac{4}{2^t}} [ |2\rangle + |6\rangle + |10\rangle + |14\rangle + \dots ].$$

After applying FFT, some state

$$\sum_{\lambda} \alpha_{\lambda} |\lambda\rangle$$

with the probability distribution for  $q = 2^t = 2048$  (see [56]). The final measurement gives 0, 512, 1024, 2048, each with probability almost exactly 1/4. Suppose  $\lambda = 1536$  was obtained from the measurement. Then we compute the continued fraction expansion

$$\frac{\lambda}{q} = \frac{1536}{2048} = \frac{1}{1 + \frac{1}{3}}, \text{ with convergents } \left[ 0, 1, \frac{3}{4} \right]$$

Thus,  $r = 4 = \text{order}_{15}(7)$ . Therefore,

$$\gcd(x^{r/2} \pm 1, N) = \gcd(7^2 \pm 1, 15) = (5, 3).$$

**Remark 2.12.** Quantum factoring is still in its very earlier stage and will not threaten the security of RSA at least at present, as the current quantum computer can only factor a number with only 2 digits such as 15 which is essentially hopeless.

**Exercises and Problems for Sect. 2.3**

1. Show that if in Shor's factoring algorithm, we have

$$\left| \frac{c}{2^m} - \frac{d}{r} \right| < \frac{1}{2n^2}$$

and

$$\left| \frac{c}{2^m} - \frac{d_1}{r_1} \right| < \frac{1}{2n^2},$$

then

$$\frac{d}{r} = \frac{d_1}{r_1}.$$

2. Show that in case  $r \nmid 2^n$ , Shor's factoring algorithm [70] needs to be repeated only  $\mathcal{O}(\log \log r)$  steps in order to achieve the high probability of success.
3. Let  $0 < s \leq m$ . Fix an integer  $x_0$  with  $0 \leq x_0 < 2^s$ . Show that

$$\sum_{\substack{0 \leq c < 2^m \\ c \equiv c_0 \pmod{2^s}}} e^{2\pi i c x / 2^m} = \begin{cases} 0 & \text{if } x \not\equiv 0 \pmod{2^{m-s}} \\ 2^{m-s} e^{2\pi i x c_0 / 2^m} & \text{if } x \equiv 0 \pmod{2^{m-s}} \end{cases}$$

4. There are currently many pseudo-simulations of Shor's quantum factoring algorithm; for example, the paper by Schneiderman, Stanley, and Aravind [66] gives one of the simulations in Maple, whereas Browne [12] presents an efficient classical simulation of the quantum Fourier transform based on [66]. Construct your own Java (C/C++, Mathematica or Maple) program to simulate Shor's quantum factoring algorithm and discrete logarithm algorithm.
5. Both ECM factoring algorithm and NFS factoring algorithm are very well suited for parallel implementation. Is it possible to utilize the quantum parallelism to implement ECM and NSF algorithms? If so, give a complete description the quantum ECM and NFS algorithms.
6. Pollard [58] and Strassen [75] showed that FFT can be utilized to factor an integer  $n$  in  $\mathcal{O}(n^{1/4+\epsilon})$  steps, deterministically. Is it possible to replace the classical FFT with a quantum FFT in the Pollard–Strassen method, in order to obtain a deterministic quantum polynomial-time factoring algorithm (i.e., to obtain a  $\mathcal{QP}$  factoring algorithm rather than the  $\mathcal{BQP}$  algorithm as proposed by Shor)? If so, give a full description of the  $\mathcal{QP}$  factoring algorithm.
7. At the very heart of the Pollard  $\rho$ -method for IFP lives the phenomenon of periodicity. Develop a quantum period-finding algorithm, if possible, for the  $\rho$  factoring algorithm.

## 2.4 Conclusions, Notes, and Further Reading

The theory of prime numbers is one of the oldest subject in number theory and indeed in the whole of mathematics, whereas the IFP is one of the oldest number-theoretic problems in the field. The root of the problem can be traced back to Euclid's *Elements* [25], although it was first clearly stated in Gauss' *Disquisitiones* [29]. With the advent of modern public-key cryptography, it has an important application in the construction of unbreakable public-key cryptographic schemes and protocols, such as RSA [28, 64], Rabin [62], and zero-knowledge proofs [33]. IFP is currently a very hot and applicable research topic, and there are many good references in the field; for a general reading, the following references are highly recommended: [1, 4, 11, 13, 17, 19, 21, 23, 40, 45, 50, 53, 61, 63, 87].

IFP-based cryptography forms an important class of public-key cryptography. In particular, RSA cryptography is the most famous and widely used cryptographic schemes in today's Internet world. More information on IFP-based cryptography can be found in [9, 20, 30, 31, 36, 37, 39, 42, 52, 76, 84], and [86].

Shor's discovery of the quantum factoring algorithm [69, 70, 70–73] in 1994 generated a great deal of research and interest in the field. Quantum computers provided a completely new paradigm for the theory of computation, and it was the first time to show that IFP can be solved efficiently in polynomial time on a quantum computer. Now, there are many good references on quantum computation, particularly on quantum factoring. Readers who wish to know more about quantum computers and quantum computation are suggested to consult the following references: [2, 5–7, 16, 22, 24, 35, 43, 48, 51, 56, 74, 77–83, 85, 88, 89], and [90]. Feynman is perhaps the father of quantum computation whose original idea about quantum computers may be found in [26, 27].

In addition to quantum computation for factoring, there are also some other non-classical computations for factoring such as molecular DNA-based factoring and attacking. For example, Chang et al. proposed some fast parallel molecular DNA algorithms for factoring large integers [14] and for breaking RSA cryptography [15].

## REFERENCES

- [1] L.M. Adleman, Algorithmic number theory – the complexity contribution, in *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science* (IEEE, New York, 1994), pp. 88–113



- [2] L.M. Adleman, J. DeMarrais, M.D.A. Huang, Quantum computability. *SIAM J. Comput.* **26**(5), 1524–1540 (1997)
- [3] M. Agrawal, N. Kayal, N. Saxena, Primes is in P. *Ann. Math.* **160**(2), 781–793 (2004)
- [4] D. Atkins, M. Graff, A.K. Lenstra, P.C. Leyland, The magic words are Squeamish Ossifrage, in *Advances in Cryptology – ASIACRYPT’94*. Lecture Notes in Computer Science, vol. 917 (Springer, Berlin, 1995), pp. 261–277
- [5] C.H. Bennett, D.P. DiVincenzo, Quantum information and computation. *Nature* **404**, 247–255 (2000)
- [6] C.H. Bennett, E. Bernstein et al., Strengths and weakness of quantum computing. *SIAM J. Comput.* **26**(5), 1510–1523 (1997)
- [7] E. Bernstein, U. Vazirani, Quantum complexity theory. *SIAM J. Comput.* **26**(5), 1411–1473 (1997)
- [8] M. Blum, S. Goldwasser, An efficient probabilistic public-key encryption scheme that hides all partial information, in *Advances in Cryptography, CRYPTO ’84*. Proceedings, Lecture Notes in Computer Science, vol. 196 (Springer, Berlin, 1985), pp. 289–302
- [9] D. Boneh, Twenty years of attacks on the RSA cryptosystem. *Not. AMS* **46**(2), 203–213 (1999)
- [10] R.P. Brent, An improved Monte Carlo factorization algorithm. *BIT* **20**, 176–184 (1980)
- [11] D.M. Bressoud, *Factorization and Primality Testing* (Springer, New York, 1989)
- [12] D.E. Browne, Efficient classical simulation of the quantum Fourier transform. *New J. Phys.* **9**, 146, 1–7 (2007)
- [13] J.P. Buhler, P. Stevenhagen (eds.), *Algorithmic Number Theory* (Cambridge University Press, Cambridge, 2008)
- [14] W.L. Chang, M. Guo, M.S.H. Ho, Fast parallel molecular algorithms for DNA-based computation: factoring integers. *IEEE Trans. Nanobioscience* **4**(2), 149–163 (2005)
- [15] W.L. Chang, K.W. Lin et al., Molecular solutions of the RSA public-key cryptosystem on a DNA-based computer. *J. Supercomput.* **56**(2), 129–163 (2011)
- [16] I.L. Chuang, R. Laflamme, P. Shor, W.H. Zurek, Quantum computers, factoring, and decoherence. *Science* **270**, 1633–1635 (1995)
- [17] H. Cohen, in *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics, vol. 138 (Springer, Berlin, 1993)
- [18] D. Coppersmith, Small solutions to polynomial equations, and low exponent RSA vulnerability. *J. Cryptol.* **10**, 233–260 (1997)
- [19] T.H. Cormen, C.E. Ceiserson, R.L. Rivest, *Introduction to Algorithms*, 3rd edn. (MIT, Cambridge, 2009)
- [20] J.S. Coron, A. May, Deterministic polynomial-time equivalence of computing the RSA secret key and factoring. *J. Cryptol.* **20**(1), 39–50 (2007)
- [21] R. Crandall, C. Pomerance, *Prime Numbers – A Computational Perspective*, 2nd edn. (Springer, Berlin, 2005)
- [22] D. Deutsch, Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. Lond. Ser. A* **400**, 96–117 (1985)

- [23] J.D. Dixon, Factorization and primality tests. *Am. Math. Mon.* **91**(6), 333–352 (1984)
- [24] A. Ekert, R. Jozsa, Quantum computation and Shor’s factoring algorithm. *SIAM J. Comput.* **26**(5), 1510–1523 (1997)
- [25] Euclid, in *The Thirteen Books of Euclid’s Elements*, 2nd edn. Translated by T.L. Heath. Great Books of the Western World, vol. 11 (William Benton Publishers, New York, 1952)
- [26] R.P. Feynman, Simulating physics with computers. *Int. J. Theor. Phys.* **21**, 467–488 (1982)
- [27] R.P. Feynman, in *Feynman Lectures on Computation*, ed. by A.J.G. Hey, R.W. Allen (Addison-Wesley, Reading, 1996)
- [28] M. Gardner, Mathematical games – a new kind of Cipher that would take millions of years to break. *Sci. Am.* **237**(2), 120–124 (1977)
- [29] C.F. Gauss, *Disquisitiones Arithmeticae*, G. Fleischer, Leipzig, 1801. English translation by A.A. Clarke (Yale University Press, Yale, 1966) Revised English translation by W.C. Waterhouse (Springer, Berlin, 1975)
- [30] O. Goldreich, *Foundations of Cryptography: Basic Tools* (Cambridge University Press, Cambridge, 2001)
- [31] O. Goldreich, *Foundations of Cryptography: Basic Applications* (Cambridge University Press, Cambridge, 2004)
- [32] S. Goldwasser, S. Micali, Probabilistic encryption. *J. Comput. Syst. Sci.* **28**, 270–299 (1984)
- [33] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)
- [34] J. Grobchadl, The Chinese remainder theorem and its application in a high-speed RSA Crypto chip, in *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC’00)* (IEEE, New York, 2000), pp. 384–393
- [35] J. Grustka, *Quantum Computing* (McGraw-Hill, New York, 1999)
- [36] M.J. Hinek, *Cryptanalysis of RSA and Its Variants* (Chapman & Hall/CRC Press, London/West Palm Beach, 2009)
- [37] J. Hoffstein, J. Pipher, J.H. Silverman, *An Introduction to Mathematical Cryptography* (Springer, Berlin, 2008)
- [38] K. Ireland, M. Rosen, in *A Classical Introduction to Modern Number Theory*, 2nd edn. Graduate Texts in Mathematics, vol. 84 (Springer, Berlin, 1990)
- [39] S. Katzenbeisser, *Recent Advances in RSA Cryptography* (Kluwer, Dordrecht, 2001)
- [40] T. Kleinjung et al., Factorization of a 768-bit RSA modulus, in *CRYPTO 2010*, ed. by T. Rabin. Lecture Notes in Computer Science, vol. 6223 (Springer, New York, 2010), pp. 333–350
- [41] D.E. Knuth, *The Art of Computer Programming III – Sorting and Searching*, 2nd edn. (Addison-Wesley, Reading, 1998)
- [42] A.G. Konheim, *Computer Security and Cryptography* (Wiley, New York, 2007)
- [43] B.P. Lanyou, T.J. Weinhold et al., Experimental demonstration of a compiled version of Shor’s algorithm’ with quantum entanglement. *Phys. Rev. Lett.* **99**, 250504, 4 (2007)
- [44] R.S. Lehman, Factoring large integers. *Math. Comput.* **28**, 126, 637–646 (1974)

- [45] A.K. Lenstra, Integer factoring. *Des. Codes Cryptography* **19**(2/3), 101–128 (2000)
- [46] A.K. Lenstra, H.W. Lenstra Jr. (eds.), in *The Development of the Number Field Sieve*. Lecture Notes in Mathematics, vol. 1554 (Springer, Berlin, 1993)
- [47] H.W. Lenstra Jr., Factoring integers with elliptic curves. *Ann. Math.* **126**, 649–673 (1987)
- [48] S.J. Lomonaco Jr., Shor’s quantum factoring algorithm. *AMS Proc. Symp. Appl. Math.* **58**, 19 (2002)
- [49] J.F. McKee, Turning Euler’s factoring methods into a factoring algorithm. *Bull. Lond. Math. Soc.* **28**, 351–355 (1996)
- [50] J.F. McKee, R. Pinch, Old and new deterministic factoring algorithms, in *Algorithmic Number Theory*. Lecture Notes in Computer Science, vol. 1122 (Springer, Berlin, 1996), pp. 217–224
- [51] N.D. Mermin, *Quantum Computer Science* (Cambridge University Press, Cambridge, 2007)
- [52] R.A. Mollin, *RSA and Public-Key Cryptography* (Chapman & Hall/CRC Press, London/West Palm Beach, 2003)
- [53] P.L. Montgomery, Speeding Pollard’s and elliptic curve methods of factorization. *Math. Comput.* **48**, 243–264 (1987)
- [54] P.L. Montgomery, A survey of modern integer factorization algorithms. *CWI Q.* **7**(4), 337–394 (1994)
- [55] M.A. Morrison, J. Brillhart, A method of factoring and the factorization of  $F_7$ . *Math. Comput.* **29**, 183–205 (1975)
- [56] M.A. Nielson, I.L. Chuang, *Quantum Computation and Quantum Information*, 10th Anniversary edn. (Cambridge University Press, Cambridge, 2010)
- [57] S.C. Pohlig, M. Hellman, An improved algorithm for computing logarithms over  $GF(p)$  and its cryptographic significance. *IEEE Trans. Inf. Theor.* **24**, 106–110 (1978)
- [58] J.M. Pollard, Theorems on factorization and primality testing. *Proc. Camb. Phil. Soc.* **76**, 521–528 (1974)
- [59] J.M. Pollard, A Monte Carlo method for factorization. *BIT* **15**, 331–332 (1975)
- [60] C. Pomerance, The quadratic Sieve factoring algorithm, in *Proceedings of Eurocrypt 84*. Lecture Notes in Computer Science, vol. 209 (Springer, Berlin, 1985), pp. 169–182
- [61] C. Pomerance, A tale of two sieves. *Not. AMS* **43**(12), 1473–1485 (1996)
- [62] M. Rabin, Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science (1979)
- [63] H. Riesel, *Prime Numbers and Computer Methods for Factorization* (Birkhäuser, Boston, 1990)
- [64] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public key cryptosystems. *Comm. ACM* **21**(2), 120–126 (1978)
- [65] R.L. Rivest, B. Kaliski, RSA Problem, in *Encyclopedia of Cryptography and Security*, ed. by H.C.A. van Tilborg (Springer, Berlin, 2005)
- [66] J.F. Schneiderman, M.E. Stanley, P.K. Aravind, A pseudo-simulation of Shor’s quantum factoring algorithm, 20 pages (2002) [arXiv:quant-ph/0206101v1]

- [67] D. Shanks, class number, a theory of factorization, and genera, in *Proceedings of Symposium of Pure Mathematics*, vol. XX, State Univ. New York, Stony Brook, 1969 (American Mathematical Society, Providence, 1971), pp. 415–440
- [68] D. Shanks, Analysis and improvement of the continued fraction method of factorization, Abstract 720-10-43. Am. Math. Soc. Not. **22**, A-68 (1975)
- [69] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in *Proceedings of 35th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, Silver Spring, 1994), pp. 124–134
- [70] P. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
- [71] P. Shor, Quantum computing. Documenta Math. Extra Volume ICM **I**, 467–486 (1998)
- [72] P. Shor, Introduction to quantum algorithms. AMS Proc. Symp. Appl. Math. **58**, 17 (2002)
- [73] P. Shor, Why haven't more quantum algorithms been found? J. ACM **50**(1), 87–90 (2003)
- [74] D.R. Simon, On the power of quantum computation. SIAM J. Comput. **26**(5), 1471–1483 (1997)
- [75] V. Strassen, Einige Resultate über Berechnungskomplexität. Jahresber. Dtsch. Math. Ver. **78**, 1–84 (1976/1997)
- [76] W. Trappe, L. Washington, *Introduction to Cryptography with Coding Theory*, 2nd edn. (Prentice-Hall, Englewood Cliffs, 2006)
- [77] L.M.K. Vandersypen, M. Steffen, G. Breyta, C.S. Tannoni, M.H. Sherwood, I.L. Chuang, Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature **414**, 883–887 (2001)
- [78] R. Van Meter, K.M. Itoh, Fast quantum modular exponentiation. Phys. Rev. A **71**, 052320 (2005)
- [79] R. Van Meter, W.J. Munro, K. Nemoto, Architecture of a quantum multicomputer implementing Shor's algorithm, in *Theory of Quantum Computation, Communication and Cryptography*, ed. by Y. Kawano, M. Mosca. Lecture Note in Computer Science, vol. 5106 (Springer, Berlin, 2008), pp. 105–114
- [80] U.V. Vazirani, On the power of quantum computation. Phil. Trans. R. Soc. Lond. **A356**, 1759–1768 (1998)
- [81] U.V. Vazirani, Fourier transforms and quantum computation, in *Proceedings of Theoretical Aspects of Computer Science* (Springer, Berlin, 2000), pp. 208–220
- [82] U.V. Vazirani, A survey of quantum complexity theory. AMS Proc. Symp. Appl. Math. **58**, 28 (2002)
- [83] J. Watrous, in *Quantum Computational Complexity*. Encyclopedia of Complexity and System Science (Springer, New York, 2009), pp. 7174–7201
- [84] H. Wiener, Cryptanalysis of short RSA secret exponents. IEEE Trans. Inf. Theor. **36**(3), 553–558 (1990)
- [85] C.P. Williams, *Explorations in Quantum Computation*, 2nd edn. (Springer, New York, 2011)
- [86] S.Y. Yan, *Cryptanalytic Attacks on RSA* (Springer, Berlin, 2008)
- [87] S.Y. Yan, in *Primality Testing and Integer Factorization in Public-Key Cryptography*. Advances in Information Security, vol. 11, 2nd edn. (Springer, New York, 2009)

- [88] N.S. Yanofsky, M.A. Mannucci, *Quantum Computing for Computer Scientists* (Cambridge University Press, Cambridge, 2008)
- [89] A.C. Yao, Quantum circuit complexity, in *Proceedings of Foundations of Computer Science* (IEEE, New York, 1993), pp. 352–361
- [90] C. Zalka, Fast versions of Shor’s quantum factoring algorithm. LANA e-print quant-ph 9806084, p. 37 (1998)



<http://www.springer.com/978-1-4419-7721-2>

Quantum Attacks on Public-Key Cryptosystems

Yan, S.Y.

2013, VIII, 207 p., Hardcover

ISBN: 978-1-4419-7721-2