CHAPTER C

# Some number theory background

Many important constructions of cryptographic primitives are based on problems from number theory which seem to be computationally intractable. The most well-known of these problems is that of factoring composite integers. In order to work with these problems, we need to develop some basic materiel on number theory and number theoretic algorithms. Accordingly, we provide here a mini-course on this subject. The materiel here will be used later when we discuss candidate example one-way and trapdoor functions.

There are many sources for information on number theory in the literatures. For example try Angluin's notes [7] and Chapter 33 of Cormen, Leiserson and Rivest [63].

## C.1 Groups: Basics

A *group* is a set $G$ together with some operation, which we denote $*$. It takes pairs of elements to another element, namely $a * b$ is the result of $*$ applied to $a, b$. A group has the following properties:

**(1)** If $a, b \in G$ so is $a * b$

**(2)** The operation is associative: $(a * b) * c = a * (b * c)$

**(3)** There is an *identity element* $I$ such that $I * a = a * I = a$ for all $a \in G$

**(4)** Every $a \in G$ has an inverse, denoted $a^{-1}$, such that $a * a^{-1} = a^{-1} * a = I$.

We will encounter this kind of structure a lot. First recall $\mathsf{Z}, \mathsf{N}$ and $\mathbf{R}$. Now, for example:

- Integers under addition: $I = 0$; $a^{-1} = -a$.
- Real numbers under multiplication: $I = 1$; $a^{-1} = 1/a$.
- What about $\mathsf{N}$ under addition? Not a group!
- What about $\mathsf{Z}$ under multiplication? Not a group!

**Notation:** $a^m$ is $a$ multiplied by itself $m$ times. Etc. Namely, notation is what you expect. What is $a^{-m}$? It is $(a^{-1})^m$. Note that it "works" like it should.

These groups are all infinite. We are usually interested in finite ones. In such a case:

**Def:** We call $|G|$ the order of $G$.

**Fact C.1** Let $m = |G|$. Then $a^m = I$ for any $a \in G$. ▌

We will use this later.

$a \equiv b \pmod{n}$ means that if we divide $a$ by $n$ then the remainder is $b$. (In C, this is $a\%n = b$).

An important set is the set of integers modulo an integer $n$. This is $Z_n = \{0, \ldots, n-1\}$. We will see it is a group under addition modulo $n$. Another related important set is $Z_n^* = \{\, m \,:\, 1 \le m \le n \text{ and } \gcd(m,n) = 1 \,\}$, the set of integers less than $n$ which are relatively prime to $n$. We will see this is a group under multiplication modulo $n$. We let $\phi(n) = |Z_n^*|$. This is the Euler totient function.

A subset $S \subseteq G$ is called a sub-group if it is a group in its own right, under the operation making $G$ a group. In particular if $x, y \in S$ so is $xy$ and $x^{-1}$, and $1 \in S$.

**Fact C.2** Suppose $S$ is a subgroup of $G$. Then $|S|$ divides $|G|$. ▌

## C.2 Arithmatic of numbers: +, *, GCD

Complexity of algorithms operating of a number $a$ is measured in terms of the size (length) of $a$, which is $|a| \approx \lg(a)$. How long do basic operations take? In terms of the number of bits $k$ in the number:

- Addition is linear time. Ie. two $k$-bit numbers can be added in $O(k)$ time.
- Multiplication of $a$ and $b$ takes $O(|a| \cdot |b|)$ bit operations. Namely it is an $O(k^2)$ algorithm.
- Division of $a$ by $b$ (integer division: we get back the quotient and remainder) takes time $O((1 + |q|)|b|)$ where $q$ is the quotient obtained. Thus this too is a quadratic time algorithm.

Euclid's algorithm can be used to compute GCDs in polynomial time. The way it works is to repeatedly use the identity $\gcd(a, b) = \gcd(b, a \bmod b)$. For examples, see page 10 of [7].

What is the running time? Each division stage takes quadratic time and we have $k$ stages, which would say it is a $O(k^3)$ algorithm. But see Problem 33-2, page 850, of [63]. This yields the following:

**Theorem C.3** Euclid's algorithm can be implemented to use only $O(|a| \cdot |b|)$ bit operations to compute $\gcd(a, b)$. That is, for $k$-bit numbers we get a $O(k^2)$ algorithm. ▌

**Fact C.4** $\gcd(a, b) = 1$ if and only if there exist integers $u, v$ such that $1 = au + bv$. ▌

The Extended Euclid Algorithm is given $a, b$ and it returns not only $d = \gcd(a, b)$ but integers $u, v$ such that $d = au + bv$. It is very similar to Euclid's algorithm. We can keep track of some extra information at each step. See page 11 of [7].

## C.3 Modular operations and groups

### C.3.1 Simple operations

Now we go to modular operations, which are the main ones we are interested in

- Addition is now the following: Given $a, b, n$ with $a, b \in Z_n$ compute $a + b \bmod n$. This is still linear time. Ie. two $k$-bit numbers can be added in $O(k)$ time. Why? You can't go much over $N$. If you do, just subtract $n$. That too is linear time.
- Taking $a \bmod n$ means divide $a$ by $n$ and take remainder. Thus, it takes quadratic time.
- Multiplication of $a$ and $b$ modulo $n$: First multiply them, which takes $O(|a| \cdot |b|)$ bit operations. Then divide by $n$ and take the remainder. We saw latter too was quadratic. So the whole thing is quadratic.

$Z_n$ is a group under addition modulo $N$. This means you can add two elements and get back an element of the set, and also subtraction is possible. Under addition, things work like you expect.

We now move to $Z_n^*$. We are interested in the multiplication operation here. We want to see that it is a group, in the sense that you can multiply and divide. We already saw how to multiply.

**Theorem C.5** There is a $O(k^2)$ algorithm which given $a, n$ with $a \in Z_n^*$ outputs $b \in Z_n^*$ satisfying $ab \equiv 1 \pmod{n}$, where $k = |n|$. ▌

See page 12 of [7]. The algorithms uses the extended Euclid. We know that $1 = \gcd(a, n)$. Hence it can find integers $u, v$ such that $1 = au + nv$. Take this modulo $n$ and we get $au \equiv 1 \pmod{n}$. So can set $b = u \bmod n$. Why is this an element of $Z_n^*$? Claim that $\gcd(u, n) = 1$. Why? By Fact C.4, which says that $1 = au + nv$ means $\gcd(u, n) = 1$.

The $b$ found in the theorem can be shown to be unique. Hence:

**Notation:** The $b$ found in the theorem is denoted $a^{-1}$.

### C.3.2 The main groups: $Z_n$ and $Z_n^*$

**Theorem C.6** For any positive integer $n$, $Z_n^*$ forms a group under multiplication modulo $n$. ▌

This means that $a, b \in Z_n^*$ implies $ab \bmod n$ is in $Z_n^*$, something one can verify without too much difficulty. It also means we can multiply and divide. We have an identity (namely 1) and a cancellation law.

**Notation:** We typically stop writing mod$n$ everywere real quick. It should just be understood.

The way to think about $Z_n^*$ is like the real numbers. You can manipulate things like you are used to. The following is a corollary of Fact C.1.

**Theorem C.7** For any $a \in Z_n^*$ it is the case that $a^{\phi(n)} = 1$. ▌

**Corollary C.8** (Fermat's little theorem) If $p$ is prime then $a^{p-1} \equiv 1 \pmod{p}$ for any $a \in \{1, \ldots, p-1\}$. ▌

Why? Because $\phi(p) = p - 1$.

### C.3.3 Exponentiation

This is the most basic operation for public key cryptography. The operation is just that given $a, n, m$ where $a \in Z_n$ and $m$ is an integer, computes $a^m \bmod n$.

**Example C.9** Compute $2^{21} \bmod 22$. Naive way: use 21 multiplications. What's the problem with this? It is an *exponential* time algorithm. Because we want time poly$(k)$ where $k = |n|$. So we do it by repeated squaring:

$$\begin{aligned} 2^1 &\equiv 2 \\ 2^2 &\equiv 4 \\ 2^4 &\equiv 16 \\ 2^8 &\equiv 14 \\ 2^{16} &\equiv 20 \end{aligned}$$

Now $2^{21} = 2^{16+4+1} = 2^{16} * 2^4 * 2^1 = 20 * 16 * 2 \equiv 10 \bmod 21$. ▌

This is the repeated squaring algorithm for exponentiation. It takes cubic time.

**Theorem C.10** There is an algorithm which given $a, n, m$ with $a, m \in Z_n$ outputs $a^m \bmod n$ in time $O(k^3)$ where $k = |n|$. More precisely, the algorithm uses at most $2k$ modular multiplications of $k$-bit numbers. ∎

Algorithm looks at binary expansion of $m$. In example above we have $21 = 10101$. What we did is collect all the powers of two corresponding to the ones and multiply them.

| 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| $a^{16}$ | $a^8$ | $a^4$ | $a^2$ | $a^1$ |
| 1 | 0 | 1 | 0 | 1 |

**Exponentiate**$(a, n, m)$

Let $b_{k-1}...b_1b_0$ be the binary representation of $m$
Let $x_0 = a$
Let $y = 1$
For $i = 0, \ldots, k-1$ do
  If $b_i = 1$ let $y = y * x_i \bmod n$
  Let $x_{i+1} = x_i^2 \bmod n$
Output $y$

## C.4 Chinese remainders

Let $m = m_1 m_2$. Suppose $y \in Z_m$. Consider the numbers

$$\begin{aligned} a_1 &= y \bmod m_1 \quad \in Z_{m_1} \\ a_2 &= y \bmod m_2 \quad \in Z_{m_2} \end{aligned}$$

The chinese remainder theorem considers the question of recombining $a_1, a_2$ back to get $y$. It says there is a *unique* way to do this under some conditions, and under these conditions says how.

**Example C.11** $m = 6 = 3 * 2$

$$\begin{aligned} 0 &\to (0,0) \\ 1 &\to (1,1) \\ 2 &\to (2,0) \\ 3 &\to (0,1) \\ 4 &\to (1,0) \\ 5 &\to (2,1) \end{aligned}$$

∎

**Example C.12** $m = 4 = 2 * 2$

$$\begin{aligned} 0 &\to (0,0) \\ 1 &\to (1,1) \\ 2 &\to (0,0) \\ 3 &\to (1,1) \end{aligned}$$

∎

The difference here is that in the first example, the association is unique, in the second it is not. It turns out uniqueness happens when the $m_1, m_2$ are relatively prime. Here is a simple form of the theorem.

**Theorem C.13** [Chinese Remainder Theorem] Let $m_1, m_2, \ldots, m_k$ be pairwise relatively prime integers. That is, $\gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq k$. Let $a_i \in \mathbf{Z}_{m_i}$ for $1 \leq i \leq k$ and set $m = m_1 m_2 \cdots m_k$. Then there exists a unique $y \in \mathbf{Z}_m$ such that $y \equiv a_i \bmod m_i$ for $i = 1 \ldots k$. Furthermore there is an $O(k^2)$ time algorithm to compute $y$ given $a_1, a_2, m_1, m_2$, where $k = \max(|m_1|, |m_2|)$.

**Proof:** For each $i$, let $n_i = \left(\frac{m}{m_i}\right) \in \mathbf{Z}$. By hypothesis, $\gcd(m_i, n_i) = 1$ and hence $\exists b_i \in \mathbf{Z}_{m_i}$ such that $n_i b_i \equiv 1 \bmod m_i$. Let $c_i = b_i n_i$. Then $c_i = \begin{cases} 1 \bmod m_i \\ 0 \bmod m_j \text{ for } j \neq i \end{cases}$.

Set $y = \sum_{i=1}^{k} c_i a_i \bmod m$. Then $y \equiv a_i \bmod m_i$ for each $i$.

Further, if $y' \equiv a_i \bmod m_i$ for each $i$ then $y' \equiv y \bmod m_i$ for each $i$ and since the $m_i$'s are pairwise relatively prime it follows that $y \equiv y' \bmod m$, proving uniqueness. ▌ ▌

**Remark C.14** The integers $c_i$ appearing in the above proof will be referred to as the Chinese Remainder Theorem coefficients. Note that the proof yields a polynomial time algorithm for finding $y$ because the elements $b_i \in \mathbf{Z}_{m_i}$ can be determined by using the Euclidean algorithm and the only other operations involved are division, multiplication, and addition. ▌

A more general form of the Chinese Remainder Theorem is the following result.

**Theorem C.15** Let $a_i \in \mathbf{Z}_{m_i}$ for $1 \leq i \leq k$. A necessary and sufficient condition that the system of congruences $x \equiv a_i \bmod m_i$ for $1 \leq i \leq k$ be solvable is that $\gcd(m_i, m_j) | (a_i - a_j)$ for $1 \leq i < j \leq k$. If a solution exists then it is unique modulo $\operatorname{lcm}(m_1, m_2, \ldots, m_k)$. ▌

**Solution Of The Quadratic Congruence $a \equiv x^2 \bmod n$ When $a \in \mathbf{Z}_n$.**

First observe that for $p$ an odd prime and $a \in \mathbf{Z}_p^{*2}$ so that $a \equiv x^2 \bmod p$ for some $x \in \mathbf{Z}_p^*$ there are exactly two solutions to $a \equiv x^2 \bmod p$ because $x$ and $-x$ are two distinct solutions modulo $p$ and if $y^2 \equiv a \equiv x^2 \bmod p$ then $p | [(x-y)(x+y)] \Longrightarrow p|(x-y)$ or $p|(x+y)$ so that $y \equiv \pm x \bmod p$. (Note that $x \not\equiv -x \bmod p$ for otherwise, $2x \equiv 0 \bmod p \Rightarrow p|x$ as $p$ is odd.) Thus, for $a \in \mathbf{Z}_p^*$, $a \equiv x^2 \bmod p$ has either 0 or 2 solutions.

Next consider the congruence $a \equiv x^2 \bmod p_1 p_2$ where $p_1$ and $p_2$ are distinct odd primes. This has a solution if and only if both $a \equiv x^2 \bmod p_1$ and $a \equiv x^2 \bmod p_2$ have solutions. Note that for each pair $(x_1, x_2)$ such that $a \equiv x_1^2 \bmod p_1$ and $a \equiv x_2^2 \bmod p_2$ we can combine $x_1$ and $x_2$ by using the Chinese Remainder Theorem to produce a solution $y$ to $a \equiv x^2 \bmod p_1 p_2$ such that $y \equiv x_1 \bmod p_1$ and $y \equiv x_2 \bmod p_2$. Hence, the congruence $a \equiv x^2 \bmod p_1 p_2$ has either 0 or 4 solutions.

More generally, if $p_1, p_2, \ldots, p_k$ are distinct odd primes then the congruence $a \equiv x^2 \bmod p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ has either 0 or $2^k$ solutions. Again, these solutions can be found by applying the Chinese Remainder Theorem to solutions of $a \equiv x^2 \bmod p_i^{\alpha_i}$. Furthermore, for a prime $p$ a solution to the congruence $a \equiv x^2 \bmod p^k$ can be found by first finding a solution $x_0$ to $a \equiv x^2 \bmod p$ by using algorithm A of Lemma 2.39 and viewing it as an approximation of the desired square root. Then the approximation is improved by the iteration $x_j \equiv \frac{1}{2}(x_{j-1} + \frac{a}{x_{j-1}}) \bmod p^{2^j}$ for $j \geq 1$.

**Claim C.16** For each integer $j \geq 0$, $a \equiv x_j^2 \bmod p^{2^j}$. **Proof:** The claim is certainly true for $j = 0$. Suppose that for $j > 0$, $a \equiv x_j^2 \bmod p^{2^j}$.

Then $x_j - a x_j^{-1} \equiv 0 \bmod p^{2^j} \Rightarrow (x_j - a x_j^{-1})^2 \equiv 0 \bmod p^{2^{j+1}}$.

Expanding and adding $4a$ to both sides gives $x_j^2 + 2a + a^2 x_j^{-2} \equiv 4a \bmod p^{2^{j+1}}$ and therefore, $\left(\frac{1}{2}(x_j + \frac{a}{x_j})\right)^2 \equiv a \bmod p^{2^{j+1}}$ or $x_{j+1}^2 \equiv a \bmod p^{2^{j+1}}$.

Hence, the claim follows by induction. ▌ ▌

From the claim, it follows that after $\lceil \log k \rceil$ iterations we will obtain a solution to $a \equiv x^2 \bmod p^k$.

## C.5 Primitive elements and $Z_p^*$

### C.5.1 Definitions

Let $G$ be a group. Let $a \in G$. Look at the powers of $a$, namely $a^0, a^1, a^2, \ldots,$. We let

$$\langle a \rangle \quad = \quad \{\, a^i \;:\; i \geq 0 \,\}\,.$$

Let $m = |G|$ be the order of $G$. We know that $a^0$ is the identity, call it 1, and $a^m = 1$ also. So the sequence repeats after $m$ steps, ie. $a^{m+1} = a$, etc. But it could repeat before too. Let's look at an example.

**Example C.17** $Z_9^* = \{1, 2, 4, 5, 7, 8\}$. Size $\phi(9) = 6$. Then:

$$\begin{aligned} \langle 1 \rangle &= \{1\} \\ \langle 2 \rangle &= \{1, 2, 4, 8, 7, 5\} \\ \langle 4 \rangle &= \{1, 4, 7\} \\ \langle 5 \rangle &= \{1, 5, 7, 8, 4, 2\} \end{aligned}$$

What we see is that sometimes we get everything, sometimes we don't. It might wrap around early. ▌

**Fact C.18** $\langle a \rangle$ is a subgroup of $G$, called the subgroup generated by $a$. ▌

Let $t = |\langle a \rangle|$. Then we know that $t$ divides $m$. And we know that in fact $\langle a \rangle = \{a^0, a^1, \ldots, a^{t-1}\}$. That is, these are the distinct elements. All others are repeats.

**Definition C.19** The order of an element $a$ is the least positive integer $t$ such that $a^t = 1$. That is, $\text{order}(a) = |\langle a \rangle|$. ▌

Computation in the indices can be done modulo $t$. That is, $a^i = a^{i \bmod t}$. This is because $a^t = a^0 = 1$.

What's the inverse of $a^i$? Think what it "should" be: $a^{-i}$. Does this make sense? Well, think of it as $(a^{-1})^i$. This is correct. On the other hand, what is it as member of the subgroup? It must be $a^j$ for some $j$. Well $j = t - i$. In particular, inverses are pretty easy to compute if you are given the index.

Similarly, like for real numbers, multiplication in the base corresponds to addition in the exponent. Namely $a^{i+j} = a^i \cdot a^j$. Etc.

**Definition C.20** An element $g \in G$ is said to be a primitive element, or *generator*, of $G$ if the powers of $g$ generate $G$. That is, $\langle g \rangle = G$ is the *whole* group $G$. A group $G$ is called *cyclic* if it has a primitive element. ▌

Note this means that for any $y \in G$ there is a *unique* $i \in \{0, \ldots, m-1\}$ such that $g^i = y$, where $m = |G|$.

**Notation:** This unique $i$ is denoted $\log_g(y)$ and called the *discrete logarithm of $x$ to base $g$*.

Consider the following problem. Given $g, y$, figure out $\log_g(y)$. How could we do it? One way is to go through all $i = 0, \ldots, m-1$ and for each $i$ compute $g^i$ and check whether $g^i = y$. But this process takes exponential time.

It turns out that computing discrete logarithms is hard for many groups. Namely, there is no known polynomial time algorithm. In particular it is true for $Z_p^*$ where $p$ is a prime.

### C.5.2 The group $Z_p^*$

**Fact C.21** [7, Section 9] The group $Z_p^*$ is cyclic. ▌

**Remark C.22** What is the order of $Z_p^*$? It is $\phi(p)$, the number of positive integers below $p$ which are relatively prime to $p$. Since $p$ is prime this is $p-1$. Note the order is *not* prime! In particular, it is even (for $p \geq 3$). ▌

*A one-way function:* Let $p$ be prime and let $g \in Z_p^*$ be a generator. Then the function $f_{p,g}$: $Z_p \to Z_p^*$ defined by

$$x \mapsto g^x$$

is conjectured to be one-way as long as some technical conditions hold on $p$. That is, there is no efficient algorithm to invert it, for large enough values of the parameters. See Chapter 2.

*Homomorphic properties:* A useful property of the function $f_{p,g}$ is that $g^{a+b} = g^a \cdot g^b$.

Now, how can we use this function? Well, first we have to set it up. This requires two things. First that we can find primes; second that we can find generators.

### C.5.3 Finding generators

We begin with the second. We have to look inside $Z_p^*$ and find a generator. How? Even if we have a candidate, how do we test it? The condition is that $\langle g \rangle = G$ which would take $|G|$ steps to check.

In fact, finding a generator given $p$ is in general a hard problem. In fact even checking that $g$ is a generator given $p, g$ is a hard problem. But what we can exploit is that $p = 2q+1$ with $q$ prime. Note that the order of the group $Z_p^*$ is $p-1 = 2q$.

**Fact C.23** Say $p = 2q+1$ is prime where $q$ is prime. Then $g \in Z_p^*$ is a generator of $Z_p^*$ iff $g^q \neq 1$ and $g^2 \neq 1$. ▌

In other words, testing whether $g$ is a generator is easy given $q$. Now, given $p = 2q+1$, how do we find a generator?

**Fact C.24** If $g$ is a generator and $i$ is not divisible by $q$ or 2 then $g^i$ is a generator.

**Proof:** $g^{iq} = g^{q+(i-1)q} = g^q \cdot (g^{2q})^{(i-1)/2} = g^q \cdot 1 = g^q$ which is not 1 because $g$ is not a generator. Similarly let $i = r + jq$ and we have $g^{2i} = g^{2r} \cdot g^{2jq} = g^{2r}$. But $2r < 2q$ since $r < q$ so $g^{2r} \neq 1$. ▌ ▌

So how many generators are there in $Z_p^*$? All things of form $g^i$ with $i$ not divisible by 2 or $q$ and $i = 1, \ldots, 2q$. Namely all $i$ in $Z_{2q}^*$. So there are $\phi(2q) = q-1$ of them.

So how do we find a generator? Pick $g \in Z_p^*$ at random, and check that $g^q \neq 1$ and $g^2 \neq 1$. If it fails, try again, up to some number of times. What's the probability of failure? In one try it is $(q+1)/2q$ so in $l$ tries it is

$$\left(\frac{q+1}{2q}\right)^l$$

which is rougly $2^{-l}$ because $q$ is very large.

## C.6 Quadratic residues

An element $x \in Z_N^*$ is a square, or quadratic residue, if it has a square root, namely there is a $y \in Z_N^*$ such that $y^2 \equiv x \bmod N$. If not, it is a non-square or non-quadratic-residue. Note a number may have lots of square roots.

It is easy to compute square roots modulo a prime. It is also easy modulo a composite whose prime factorization you know, via Chinese remainders. (In both cases, you can compute all the roots.) But it is hard modulo a composite of unknown factorization. In fact computing square roots is equivalent to factoring.

Also, it is hard to decide quadratic residuosity modulo a composite.

**Fact C.25** If $N$ is the product of two primes, every square $w \in Z_N^*$ has exactly four square roots, $x, -x$ and $y, -y$ for some $x, y \in Z_N^*$. If you have two square roots $x, y$ such that $x \neq \pm y$, then you can easily factor $N$. ▮

The first fact is basic number theory. The second is seen like this. Say $x > y$ are the roots. Consider $x^2 - y^2 = (x-y)(x+y) \equiv 0 \bmod N$. Let $a = x - y$ and $b = x + y \bmod N$. So $N$ divides $ab$. So $p$ divides $ab$. Since $p$ is prime, this means either $p$ divides $a$ or $p$ divides $b$. Since $1 \leq a, b < N$ this means either $\gcd(a, N) = p$ or $\gcd(b, N) = p$. We can compute the gcds and check whether we get a divisor of $N$.

## C.7 Jacobi Symbol

We previously defined the Legendre symbol to indicate the quadratic character of $a \in \mathbf{Z}_p^*$ where $p$ is a prime. Specifically, for a prime $p$ and $a \in \mathbf{Z}_p$

$$\mathbf{J}_p(a) = \begin{cases} 1 & \text{if } a \text{ is a square in } \mathbf{Z}_p^* \\ 0 & \text{if } a = 0 \\ -1 & \text{if } a \text{ is not a square in } \mathbf{Z}_p^* \end{cases}$$

For composite integers, this definition is extended, as follows, giving the *Jacobi Symbol*. Let $n = \prod_{i=1}^{k} {p_i}^{\alpha_i}$ be the prime factorization of $n$. For $a \in \mathbf{Z}_n$ define

$$\mathbf{J}_n(a) = \prod_{i=1}^{k} \mathbf{J}_{p_i}(a)^{\alpha_i}.$$

However, the Jacobi Symbol does not generalize the Legendre Symbol in the respect of indicating the quadratic character of $a \in \mathbf{Z}_n^*$ when $n$ is composite. For example, $\mathbf{J}_9(2) = \mathbf{J}_3(2)\mathbf{J}_3(2) = 1$, although the equation $x^2 \equiv 2 \bmod 9$ has no solution.

The Jacobi Symbol also satisfies identities similar to those satisfied by the Legendre Symbol. We list these here. For proofs of these refer to [156].

1. If $a \equiv b \bmod n$ then $\mathbf{J}_n(a) = \mathbf{J}_n(b)$.
2. $\mathbf{J}_n(1) = 1$.
3. $\mathbf{J}_n(-1) = (-1)^{\frac{n-1}{2}}$.
4. $\mathbf{J}_n(ab) = \mathbf{J}_n(a)\mathbf{J}_n(b)$.
5. $\mathbf{J}_n(2) = (-1)^{\frac{n^2-1}{8}}$.
6. If $m$ and $n$ are relatively prime odd integers then $\mathbf{J}_n(m) = (-1)^{\frac{n-1}{2}\frac{m-1}{2}}\mathbf{J}_m(n)$.

Using these identities, the Jacobi Symbol $\mathbf{J}_n(a)$ where $a \in \mathbf{Z}_n$ can be calculated in polynomial time even without knowing the factorization of $n$. Recall that to calculate the Legendre Symbol in polynomial time we can call upon Euler's Theorem; namely, for $a \in \mathbf{Z}_p^*$, where $p$ is prime, we have $\mathbf{J}_p(a) \equiv a^{\frac{p-1}{2}} \bmod p$. However, for a composite integer $n$ it is not necessarily true that $\mathbf{J}_n(a) \equiv a^{\frac{n-1}{2}} \bmod n$ for $a \in \mathbf{Z}_n^*$. In fact, this statement is true for at most half of the elements in $\mathbf{Z}_n^*$. From this result, we can derive a Monte Carlo primality test as we shall see later.

## C.8 RSA

Here we have a composite modulus $N = pq$ product of two distinct primes $p$ and $q$ of roughly equal length. Let $k = |N|$; this is about 1024, say. It is generally believed that such a number is hard to factor.

Recall that $\phi(N) = |Z_N^*|$ is the Euler Phi function. Note that $\phi(N) = (p-1)(q-1)$. (To be relatively prime to $N$, a number must be divisible neither by $p$ nor by $q$. Eliminating multiples of either yields this. Note we use here that $p \neq q$.)

Now let $e$ be such that $\gcd(e, \phi(N)) = 1$. That is, $e \in Z_{\phi(N)}^*$. The RSA function is defined by

$$\begin{array}{rccc} f\colon & Z_N^* & \to & Z_N^* \\ & x & \mapsto & x^e \bmod N\ . \end{array}$$

We know that $Z_{\phi(N)}^*$ is a group. So $e$ has an inverse $d \in Z_{\phi(N)}^*$. Since $d$ is an inverse of $e$ it satisfies

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

Now let $x \in Z_N^*$ be arbitrary and look at the following computation:

$$(x^e)^d \bmod N = x^{ed \bmod \phi(N)} \bmod N = x^1 \bmod N = x\ .$$

In other words, the function $y \mapsto y^d$ is an inverse of $f$. That is, $f^{-1}(y) = y^d \bmod N$.

Can we find $d$? Easy: computing inverses can be done in quadratic time, as we already say, using the extended GCD algorithm! But note a crucial thing. We are working modulo $\phi(N)$. So finding $d$ this way requires that we know $\phi(N)$. But the latter involves knowing $p, q$.

It seems to be the case that given only $N$ and $e$ it is hard to find $d$. Certainly we agreed it is hard to find $p, q$; but even more, it seems hard to find $d$. This yields the conjecture that RSA defines a trapdoor one-way permutation. Namely given $N, e$ defining $f$, $x \mapsto f(x)$ is easy; $y \mapsto f^{-1}(y)$ is hard; but $f^{-1}$ is easy given $p, q$ (or $d$). Note that this trapdoorness is a property the discrete logarithm problem did not have.

Computation of $f$ is called encryption, and computation of $f^{-1}$ is called decryption.

Both encryption and decryption are exponentiations, which a priori are cubic in $k$ time operations. However, one often chooses $e$ to be small, so encryption is faster. In hardware, RSA is about 1000 times slower than DES; in software it is about 100 times slower, this with small encryption exponent.

Formally, RSA defines a *family of trapdoor permutations.* The family is indexed by a security parameter $k$ which is the size of the modulus. The RSA generator is an algorithm $G$ which on input $1^k$ picks two distinct, random $(k/2)$-bit primes $p, q$, multiplies them to produce $N = pq$, and also computes $e, d$. It outputs $N, e$ as the description of $f$ and $N, d$ as the description of $f^{-1}$. See Chapter 2.

RSA provides the ability to do public key cryptography.

## C.9 Primality Testing

For many cryptographic purposes, We need to find primes. There is no known polynomial time algorithm to test primality of a given integer $n$. What we use are probabilistic, polynomial time (PPT) algorithms.

We will first show that the problem of deciding whether an integer is prime is in NP. Then we will discuss the Solovay-Strassen and Miller-Rabin probabilistic primality tests which efficiently find proofs of compositeness. Finally, we will give a primality test due to Goldwasser and Kilian which uses elliptic curves and which efficiently finds a proof of primality.

### C.9.1 PRIMES $\in$ NP

We will first present two algorithms for testing primality, both of which are inefficient because they require factoring as a subroutine. However, either algorithm can be used to show that the problem of deciding whether

an integer is prime is in NP. In fact, the second algorithm that is presented further demonstrates that deciding primality is in UP∩coUP. Here UP denotes the class of languages $L$ accepted by a polynomial time nondeterministic Turing machine having a unique accepting path for each $x \in L$.

**Definition C.26** Let PRIMES $= \{p : p \text{ is a prime integer}\}$. ▮

## C.9.2 Pratt's Primality Test

Pratt's primality testing algorithm is based on the following result.

**Proposition C.27** For an integer $n > 1$, the following statements are equivalent.

1. $|\mathbf{Z}_n^*| = n - 1$.
2. The integer $n$ is prime.
3. There is an element $g \in \mathbf{Z}_n^*$ such that $g^{n-1} \equiv 1 \bmod n$ and for every prime divisor $q$ of $n-1$, $g^{\frac{n-1}{q}} \not\equiv 1 \bmod n$.

▮

Pratt's algorithm runs as follows on input a prime $p$ and outputs a proof (or certificate) that $p$ is indeed prime.

1. Find an element $g \in \mathbf{Z}_p^*$ whose order is $p-1$.
2. Determine the prime factorization $\prod_{i=1}^{k} q_i^{\alpha_i}$ of $p-1$.
3. Prove that $p$ is prime by proving that $g$ is a generator of $\mathbf{Z}_p^*$. Specifically, check that $g^{p-1} \equiv 1 \bmod p$ and for each prime $q_i$ check that $g^{\frac{p-1}{q_i}} \not\equiv 1 \bmod p$.
4. Recursively show that $q_i$ is prime for $1 \leq i \leq k$.

Note that if $p$ is a prime, then $\mathbf{Z}_p^*$ has $\varphi(p-1) = \Omega(\frac{p}{\log\log p})$ generators (see [178]). Thus, in order to find a generator $g$ by simply choosing elements of $\mathbf{Z}_p^*$ at random, we expect to have to choose $O(\log\log p)$ candidates for $g$. If we find a generator $g$ of $\mathbf{Z}_p^*$ and if we can factor $p-1$ and recursively prove that the prime factors of $p-1$ are indeed primes then we have obtained a proof of the primality of $p$. Unfortunately, it is not known how to efficiently factor $p-1$ for general $p$. Pratt's primality testing algorithm does demonstrate, however, that PRIMES $\in$ NP because both the generator $g$ in step 1 and the required factorization in step 2 can be guessed. Moreover, the fact that the factorization is correct can be verified in polynomial time and the primality of each $q_i$ can be verified recursively by the algorithm. Note also, as Pratt showed in [167] by a simple inductive argument, that the total number of primes involved is $O(\log p)$. Thus, verifying a Pratt certificate requires $O(\log^2 p)$ modular multiplications with moduli at most $p$.

## C.9.3 Probabilistic Primality Tests

## C.9.4 Solovay-Strassen Primality Test

We can derive a Monte Carlo primality test. This algorithm, which we state next, is due to Solovay and Strassen (see [197]).

The Solovay-Strassen primality test runs as follows on input an odd integer $n$ and an integer $k$, indicating the desired reliability.

1. Test if $n = b^e$ for integers $b, e > 1$; if so, output composite and terminate.
2. Randomly choose $a_1, a_2, \ldots, a_k \in \{1, 2, \ldots, n-1\}$.
3. If $\gcd(a_i, n) \neq 1$ for any $1 \leq i \leq k$ then output composite and terminate.
4. Calculate $\alpha_i = a_i^{\frac{n-1}{2}} \bmod n$ and $\beta_i = \mathbf{J}_n(a_i)$.
5. If for any $1 \leq i \leq k$, $\alpha_i \neq \beta_i \bmod n$ then output composite. If for all $1 \leq i \leq k$, $\alpha_i = \beta_i \bmod n$ then output probably prime.

Since the calculations involved in the Solovay-Strassen primality test are all polynomial time computable (verify that this statement is indeed true for step 1), it is clear that the algorithm runs in time polynomial in $\log n$ and $k$. The following result guarantees that if $n$ is composite then in step 5 of the algorithm, $\Pr[\alpha_i = \beta_i \bmod n] \leq \frac{1}{2}$ and thus, $\Pr[\alpha_i = \beta_i \bmod n \text{ for } 1 \leq i \leq k] \leq \left(\frac{1}{2}\right)^k$.

**Proposition C.28** Let $n$ be an odd composite integer which is not a perfect square and let $G = \{a \in \mathbf{Z}_n^* \text{ such that } \mathbf{J}_n(a) \equiv a^{\frac{n-1}{2}} \bmod n\}$. Then $|G| \leq \frac{1}{2}|\mathbf{Z}_n^*|$. ▌

**Proof:** Since $G$ is a subgroup of $\mathbf{Z}_n^*$ it suffices to show that $G \neq \mathbf{Z}_n^*$.

Since $n$ is composite and not a perfect square, it has a nontrivial factorization $n = rp^\alpha$ where $p$ is prime, $\alpha$ is odd, and $\gcd(r, p) = 1$.

Suppose that $a^{\frac{n-1}{2}} \equiv \mathbf{J}_n(a) \bmod n$ for all $a \in \mathbf{Z}_n^*$. Then

$$a^{\frac{n-1}{2}} \equiv \pm 1 \bmod n \text{ for all } a \in \mathbf{Z}_n^*. \tag{C.1}$$

We first show that in fact $a^{\frac{n-1}{2}} \equiv 1 \bmod n$ for all such $a$. If not, then there is an $a \in \mathbf{Z}_n^*$ with $a^{\frac{n-1}{2}} \equiv -1 \bmod n$. By the Chinese Remainder Theorem there is a unique element $b \in \mathbf{Z}_n$ such that $b \equiv 1 \bmod r$ and $b \equiv a \bmod p^\alpha$. Then $b \in \mathbf{Z}_n^*$ and $b^{\frac{n-1}{2}} \equiv 1 \bmod r$ and $b^{\frac{n-1}{2}} \equiv -1 \bmod p^\alpha$ contradicting equation $(C.1)$. Therefore, $\mathbf{J}_n(a) = 1$ for all $a \in \mathbf{Z}_n^*$.

However, by the Chinese Remainder Theorem, there is a unique element $a \in \mathbf{Z}_{rp}$ such that $a \equiv 1 \bmod r$ and $a \equiv z \bmod p$ where $z$ is one of the $\frac{p-1}{2}$ quadratic nonresidues modulo $p$. Then $a \in \mathbf{Z}_n^*$ and thus, $\mathbf{J}_n(a) = \mathbf{J}_r(1)\mathbf{J}_p(z)^\alpha = -1$ because $\alpha$ is odd. This is a contradiction. ▌

Note that if we reach step 5 of the Solovay-Strassen algorithm then $n$ is not a perfect square and each $a_i \in \mathbf{Z}_n^*$ because the algorithm checks for perfect powers in step 1 and computes $\gcd(a_i, n)$ in step 3. Thus, the hypotheses of Proposition C.28 are satisfied and for each $1 \leq i \leq k$, $\Pr[\alpha_i = \beta_i \bmod n] \leq \frac{1}{2}$.

**Remark** The assertion in Proposition C.28 is in fact true even if $n$ is a perfect square. The proof of the more general statement is very similar to the proof of Proposition C.28. For details refer to [7].

Finally, it follows from Proposition C.28 that the Solovay-Strassen algorithm runs correctly with high probability. Specifically,

$$\Pr[\text{Solovay-Strassen outputs probably prime} \mid n \text{ is composite}] \leq (\frac{1}{2})^k$$

and

$$\Pr[\text{Solovay-Strassen outputs probably prime} \mid n \text{ is prime}] = 1.$$

### C.9.5 Miller-Rabin Primality Test

Fermat's Little Theorem states that for a prime $p$ and $a \in \mathbf{Z}_p^*$, $a^{p-1} \equiv 1 \bmod p$. This suggests that perhaps a possible way to test if $n$ is prime might be to check, for instance, if $2^{n-1} \equiv 1 \bmod n$. Unfortunately, there are composite integers $n$ (called base-2 pseudoprimes) for which $2^{n-1} \equiv 1 \bmod n$. For example, $2^{340} \equiv 1 \bmod 341$ and yet $341 = 11 \cdot 31$. In fact, replacing 2 in the above exponentiation by a random $a \in \mathbf{Z}_n^*$ would not help for some values of $n$ because there are composite integers $n$ (called Carmichael numbers) for which $a^{n-1} \equiv 1 \bmod n$ for all $a \in \mathbf{Z}_n^*$. 561, 1105, and 1729 are the first three Carmichael numbers.

The Miller-Rabin primality test overcomes the problems of the simple suggestions just mentioned by choosing several random $a \in \mathbf{Z}_n^*$ for which $a^{n-1} \bmod n$ will be calculated by repeated squaring. While computing each modular exponentiation, it checks whether some power of $a$ is a nontrivial square root of 1 modulo $n$ (that is, a root of 1 not congruent to $\pm 1$ modulo $n$). If so, the algorithm has determined $n$ to be composite. The quality of this test relies on Proposition C.30 which Rabin proved in [171]. For a simpler proof which only yields $|\{b : \mathrm{W}_n(b) \text{ holds } \}| \geq \frac{1}{2}(n-1)$ (but is nevertheless sufficient for the purposes of the Miller-Rabin algorithm) see Chapter 33, pages 842-843 of [63].

**Definition C.29** Let $n$ be an odd positive integer. Denote the following condition on an integer $b$ by $\mathrm{W}_n(b)$:

1. $1 \leq b < n$ and
2. (i) $b^{n-1} \not\equiv 1 \bmod n$ or
   (ii) there is an integer $i$ such that $2^i \mid (n-1)$ and $b^{(n-1)/2^i} \not\equiv \pm 1 \bmod n$ but $\left(b^{(n-1)/2^i}\right)^2 \equiv 1 \bmod n$.

An integer $b$ for which $\mathrm{W}_n(b)$ holds will be called a witness to the compositeness of $n$. ▌

**Remark** Rabin originally defined the condition $\mathrm{W}_n(b)$ to hold if $1 \leq b < n$ and either $b^{n-1} \not\equiv 1 \bmod n$ or for some integer $i$ such that $2^i \mid (n-1)$, $1 < \gcd(b^{(n-1)/2^i} - 1, n) < n$. In [171] Rabin proves that the two definitions for $\mathrm{W}_n(b)$ are equivalent. This condition was in fact first considered by Miller (see [149]), who used it to give a nonprobabilistic test for primality assuming the correctness of the extended Riemann hypothesis. Rabin's results, however, do not require any unproven assumptions.

**Proposition C.30** If $n$ is an odd composite integer then $|\{b : \mathrm{W}_n(b) \text{ holds}\}| \geq \frac{3}{4}(n-1)$. ▌

The Miller-Rabin algorithm runs as follows on input an odd integer $n$ and an integer $k$, indicating the desired reliability.

1. Randomly choose $b_1, b_2, \ldots, b_k \in \{1, 2, \ldots, n-1\}$.
2. Let $n - 1 = 2^l m$ where $m$ is odd.
3. For $1 \leq i \leq k$ compute $b_i^m \bmod n$ by repeated squaring.
4. Compute $b_i^{2^j m} \bmod n$ for $j = 1, 2, \ldots, l$. If for some $j$, $b_i^{2^{j-1} m} \not\equiv \pm 1 \bmod n$ but $b_i^{2^j m} \equiv 1 \bmod n$ then $\mathrm{W}_n(b_i)$ holds.
5. If $b_i^{n-1} \not\equiv 1 \bmod n$ then $\mathrm{W}_n(b_i)$ holds.
6. If for any $1 \leq i \leq k$, $\mathrm{W}_n(b_i)$ holds then output composite. If for all $1 \leq i \leq k$, $\mathrm{W}_n(b_i)$ does not hold then output probably prime.

Proposition C.30 shows that the Miller-Rabin algorithm runs correctly with high probability. Specifically,

$$\Pr[\text{Miller-Rabin outputs probably prime} \mid n \text{ is composite}] \leq (\frac{1}{4})^k$$

and

$$\Pr[\text{Miller-Rabin outputs probably prime} \mid n \text{ is prime}] = 1.$$

Furthermore, Miller-Rabin runs in time polynomial in $\log n$ and $k$ as all the computations involved can be performed in polynomial time.

### C.9.6 Polynomial Time Proofs Of Primality

Each of the two algorithms discussed in the previous section suffers from the deficiency that whenever the algorithm indicates that the input $n$ is prime, then it is prime with high probability, but no certainty is provided. (In other words, the algorithms are Monte Carlo algorithms for testing primality.) However, when either algorithm outputs that the input $n$ is composite then it has determined that $n$ is indeed composite. Thus, the Solovay-Strassen and Miller-Rabin algorithms can be viewed as compositeness provers. In this section we will discuss a primality test which yields in expected polynomial time a short (verifiable in deterministic polynomial time) proof that a prime input is indeed prime. Therefore, for a general integral input we can run such a primality prover in parallel with a compositeness prover and one of the two will eventually terminate either yielding a proof that the input is prime or a proof that the input is composite. This will provide us with a Las Vegas algorithm for determining whether an integer is prime or composite.

### C.9.7 An Algorithm Which Works For Some Primes

Suppose that we could find a prime divisor $q$ of $p-1$ such that $q > \sqrt{p}$. Then the following algorithm can be used to prove the primality of $p$.

1. Determine a prime divisor $q$ of $p-1$ for which $q > \sqrt{p}$.
2. Randomly choose $a \in \mathbf{Z}_p^* - \{1\}$.
3. If $1 < \gcd(a-1, p) < p$ then output that $p$ is composite.
4. Check that $a^q \equiv 1 \bmod p$.
5. Recursively prove that $q$ is prime.

The correctness of this algorithm follows from the next result.

**Claim C.31** If $q > \sqrt{p}$ is a prime and for some $a \in \mathbf{Z}_p^*$ $\gcd(a-1, p) = 1$ and $a^q \equiv 1 \bmod p$ then $p$ is a prime. ▌

**Proof:** Suppose $p$ is not prime. Then there is a prime $d \leq \sqrt{p}$ such that $d \mid p$ and therefore, by the hypothesis, $a \not\equiv 1 \bmod d$ and $a^q \equiv 1 \bmod d$. Thus, in $\mathbf{Z}_d^*$, $\mathrm{ord}(a) \mid q$. But $q$ is prime and $a$ does not have order 1. Hence, $q = \mathrm{ord}(a) \leq |\mathbf{Z}_d^*| = d - 1 < \sqrt{p}$ and this contradicts the assumption that $q > \sqrt{p}$. ▌

Note that if $p$ is prime then in step 4, the condition $a^q \equiv 1 \bmod p$ will be verified with probability at least $\frac{q-1}{p-2} > \frac{1}{\sqrt{p}}$ (since $q > \sqrt{p}$). However, in order for the algorithm to succeed, there must exist a prime divisor $q$ of $p-1$ such that $q > \sqrt{p}$, and this must occur at every level of the recursion. Namely, there must be a sequence of primes $q = q_0, q_1, \ldots, q_k$, where $q_k$ is small enough to identify as a known prime, such that $q_i \mid (q_{i-1} - 1)$ and $q_i > \sqrt{q_{i-1}}$ for $i = 1, \ldots, k$ and this is very unlikely.

This obstacle can be overcome by working with elliptic curves modulo primes $p$ instead of $\mathbf{Z}_p^*$. This will allow us to randomly generate for any prime modulus, elliptic groups of varying orders. In the following sections, we will exploit this additional freedom in a manner similar to Lenstra's elliptic curve factoring algorithm.

### C.9.8 Goldwasser-Kilian Primality Test

The Goldwasser-Kilian primality test is based on properties of elliptic curves. The idea of the algorithm is similar to the primality test presented in Section C.9.7, except that we work with elliptic curves $E_{a,b}(\mathbf{Z}_p)$ instead of $\mathbf{Z}_p^*$. By varying $a$ and $b$ we will be able to find an elliptic curve which exhibits certain desired properties.

The Goldwasser-Kilian algorithm runs as follows on input a prime integer $p \neq 2, 3$ of length $l$ and outputs a proof that $p$ is prime.

1. Randomly choose $a, b \in \mathbf{Z}_p$, rejecting choices for which $\gcd(4a^3 + 27b^2, p) \neq 1$.
2. Compute $|E_{a,b}(\mathbf{Z}_p)|$ using the polynomial time algorithm due to Schoof (see [185]).
3. Use a probabilistic pseudo-primality test (such as Solovay-Strassen or Miller-Rabin) to determine if $|E_{a,b}(\mathbf{Z}_p)|$ is of the form $cq$ where $1 < c \leq O(\log^2 p)$ and $q$ is a probable prime. If $|E_{a,b}(\mathbf{Z}_p)|$ is not of this form then repeat from step 1.
4. Select a point $M = (x, y)$ on $E_{a,b}(\mathbf{Z}_p)$ by choosing $x \in \mathbf{Z}_p$ at random and taking $y$ to be the square root of $x^3 + ax + b$, if one exists. If $x^3 + ax + b$ is a quadratic nonresidue modulo $p$ then repeat the selection process.
5. Compute $q \cdot M$.
   (i) If $q \cdot M = O$ output $(a, b, q, M)$. Then, if $q > 2^{l^{(\frac{1}{\log log l})}}$, [5]), recursively prove that $q$ is prime (specifically, repeat from step 1 with $p$ replaced by $q$). Otherwise, use the deterministic test due to Adleman, Pomerance, and Rumely (see [5]) to show that $q$ is prime and terminate.
   (ii) If $q \cdot M \neq O$ then repeat from step 4.

**Remark** The test mentioned in step 5i is currently the best deterministic algorithm for deciding whether an input is prime or composite. It terminates within $(\log n)^{O(\log \log \log n)}$ steps on input $n$.

### C.9.9 Correctness Of The Goldwasser-Kilian Algorithm

Note first that as we saw in Section C.9.3, the probability of making a mistake at step 3 can be made exponentially small. The correctness of the Goldwasser-Kilian algorithm follows from Theorem C.32. This result is analogous to Claim C.31.

**Theorem C.32** Let $n > 1$ be an integer with $\gcd(n, 6) = 1$. Let $E_{a,b}(\mathbf{Z}_n)$ be an elliptic curve modulo $n$ and let $M \neq O$ be a point on $E_{a,b}(\mathbf{Z}_n)$. If there is a prime integer $q$ such that $q > (n^{1/4} + 1)^2$ and $q \cdot M = O$ then $n$ is prime. ▌

**Proof:** Suppose that $n$ were composite. Then there is a prime divisor $p$ of $n$ such that $p < \sqrt{n}$.

Let $\text{ord}_E(M)$ denote the order of the point $M$ on the elliptic curve $E$. If $q \cdot M = O$ then $q \cdot M_p = O_p$. Thus, $\text{ord}_{E_p}(M_p) \mid q$.

$$\begin{aligned} \text{However, } \text{ord}_{E_p}(M_p) \leq |E_{a,b}(\mathbf{Z}_p)| &\leq p + 1 + 2\sqrt{p} \quad \text{(by Hasse's Inequality)} \\ &< n^{1/2} + 1 + 2n^{1/4} \\ &< q \end{aligned}$$

and since $q$ is prime, we have that $\text{ord}_{E_p}(M_p) = 1$. Therefore, $M_p = O_p$ which implies that $M = O$, a contradiction. ▌

**Theorem C.33** By using the sequence of quadruples output by the Goldwasser-Kilian algorithm, we can verify in time $O(\log^4 p)$ that $p$ is indeed prime. ▌

**Proof:** Let $p_0 = p$. The sequence of quadruples output by the algorithm will be of the form $(a_1, b_1, p_1, M_1), (a_2, b_2, p_2, M_2), \ldots, (a$ where $\gcd(4a_i^3 + 27b_i^2, p_{i-1}) \neq 1$, $M_i \neq O$ is a point on $E_{a_i,b_i}(\mathbf{Z}_{p_{i-1}})$, $p_i > p_{i-1}^{1/2} + 1 + 2p_{i-1}^{1/4}$, and $p_i \cdot M_i = O$ for $1 \leq i \leq k$. These facts can all be verified in $O(\log^3 p)$ time for each value of $i$. By Theorem C.32 it follows that $p_i$ prime $\Rightarrow$ $p_{i-1}$ prime for $1 \leq i \leq k$. Further, note that in step 3 of the algorithm, $c \geq 2$ and hence, $p_i \leq \frac{p_{i-1}+2\sqrt{p_{i-1}}}{2}$. Therefore, the size of $k$ will be $O(\log p)$ giving a total of $O(\log^4 p)$ steps. Finally, $p_k$ can be verified to be prime in $O(\log p)$ time due to its small size. ▌

### C.9.10 Expected Running Time Of Goldwasser-Kilian

The algorithm due to Schoof computes $|E_{a,b}(\mathbf{Z}_p)|$ in $O(\log^9 p)$ time. Then to check that $|E_{a,b}(\mathbf{Z}_p)| = cq$ where $1 < c \leq O(\log^2 p)$ and $q$ is prime requires a total of $O(\log^6 p)$ steps if we use Solovay-Strassen or Miller-Rabin with enough iterations to make the probability of making a mistake exponentially small (the algorithm may have to be run for each possible value of $c$ and each run of the algorithm requires $O(\log^4 p)$ steps).

Next, selecting the point $M = (x, y)$ requires choosing an expected number of at most $\frac{2p}{|E_{a,b}(\mathbf{Z}_p)|-1} \approx 2$ values for $x$ before finding one for which $x^3 + ax + b$ is a quadratic residue modulo $p$. Note that the computation of square roots modulo a prime $p$ (to find $y$) can be done in $O(\log^4 p)$ expected time. Since $|E_{a,b}(\mathbf{Z}_p)| = cq$ where $q$ is prime, $E_{a,b}(\mathbf{Z}_p)$ is isomorphic to $\mathbf{Z}_{c_1 q} \times \mathbf{Z}_{c_2}$ where $c = c_1 c_2$ and $c_2 | c_1$. Therefore, $E_{a,b}(\mathbf{Z}_p)$ has at least $q - 1$ points of order $q$ and hence with probability at least $\frac{q-1}{cq} \approx \frac{1}{c}$, the point $M$ selected in step 4 will have order $q$. Thus, the expected number of points that must be examined before finding a point $M$ of order $q$ will be $c = O(\log^2 p)$. Further, the computation of $q \cdot M$ requires $O(\log p)$ additions, using repeated doubling and so can be done in $O(\log^3 p)$ time. Therefore, dealing with steps 4 and 5 requires $O(\log^5 p)$ expected time.

As remarked previously, the recursion depth is $O(\log p)$. Therefore, the only remaining consideration is to determine how often an elliptic curve $E_{a,b}(\mathbf{Z}_p)$ has to be selected before $|E_{a,b}(\mathbf{Z}_p)| = cq$ where $c = O(\log^2 p)$ and $q$ is prime. By the result of Lenstra concerning the distribution of $|E_{a,b}(\mathbf{Z}_p)|$ in $(p+1-\sqrt{p}, p+1+\sqrt{p})$ (see [131]) this is $O(\frac{\sqrt{p}\log p}{|S|-2})$ where $S$ is the set of integers in $(p+1-\sqrt{p}, p+1+\sqrt{p})$ of the desired form $cq$. Note that $|S| \geq \pi(\frac{p+1+\sqrt{p}}{2}) - \pi(\frac{p+1-\sqrt{p}}{2})$ because $S$ contains those integers in $(p+1-\sqrt{p}, p+1+\sqrt{p})$ which are twice a prime. Therefore, if one assumes that the asymptotic distribution of primes holds in small intervals, then the expected number of elliptic curves that must be considered is $O(\log^2 p)$. However, there is only evidence to assume the following conjecture concerning the number of primes in small intervals.

**Conjecture C.34** There is a positive constant $s$ such that for all $x \in \mathbf{R}_{\geq 2}$, the number of primes between $x$ and $x + \sqrt{2x}$ is $\Omega\left(\frac{\sqrt{x}}{\log^s x}\right)$. ▌

Under this assumption, the Goldwasser-Kilian algorithm proves the primality of $p$ in $O((\log p)^{11+s})$ expected time.

### C.9.11 Expected Running Time On Nearly All Primes

Although the analysis presented in Section C.9.10 relies on the unproven result stated in Conjecture C.34, a theorem due to Heath-Brown concerning the density of primes in small intervals can be used to show that the fraction of primes of length $l$ for which the Goldwasser-Kilian algorithm runs in expected time polynomial in $l$ is at least $1 - O(2^{-l^{\frac{1}{\log\log l}}})$. The Heath-Brown result is the following.

**Theorem C.35** Let $\#_p[a, b]$ denote the number of primes $x$ satisfying $a \leq x \leq b$.
Let $i(a, b) = \begin{cases} 1 \text{ if } \#_p[a, b] \leq \frac{b-a}{2\lfloor \log a \rfloor} \\ 0 \text{ otherwise} \end{cases}$. Then there exists a positive constant $\alpha$ such that
$\sum_{x \leq a \leq 2x} i(a, a + \sqrt{a}) \leq x^{\frac{5}{6}} \log^{\alpha} x$. ▌

Using this theorem, Goldwasser and Kilian were able to prove in [100] that their algorithm terminates in expected time $O(l^{12})$ on at least a $1 - O(2^{l^{\frac{1}{\log\log l}}})$ fraction of those primes of length $l$. In [4] Adleman and Huang showed, by a more careful analysis of the Goldwasser-Kilian algorithm, that in fact the fraction of primes of length $l$ for which Goldwasser-Kilian may not terminate in expected polynomial time is strictly exponentially vanishing. Further, they proposed a new algorithm for proving primality based on hyperelliptic curves which they showed will terminate in exponential polynomial time on all prime inputs. Thus, the goal of obtaining a Las Vegas algorithm has been finally achieved.

## C.10 Factoring Algorithms

In this lecture we discuss some general properties of elliptic curves and present Lenstra's elliptic curve factoring algorithm which uses elliptic curves over $\mathbf{Z}_n$ to factor integers.

### Pollard's $p-1$ Method

We begin by introducing a predecessor of the elliptic curve factoring algorithm which uses ideas analogous to those used in the elliptic curve factoring algorithm. This algorithm, known as Pollard's $p-1$ method, appears in [166]. Let $n$ be the composite number that we wish to split. Pollard's algorithm uses the idea that if we can find integers $e$ and $a$ such that $a^e \equiv 1 \bmod p$ and $a^e \not\equiv 1 \bmod q$ for some prime factors $p$ and $q$ of $n$ then, since $p \mid (a^e - 1)$ and $q \not\mid (a^e - 1)$, $\gcd(a^e - 1, n)$ will be a nontrivial factor of $n$ divisible by $p$ but not by $q$.

The algorithm proceeds as follows on input $n$.

1. Choose an integer $e$ that is a multiple of all integers less than some bound $B$. For example, $e$ might be the least common multiple of all integers $\leq B$. To simplify this, we might even let $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_1, p_2, \ldots, p_{\pi(B)}$ are the primes $\leq B$ and $\alpha_i$ is chosen minimally so that $p_i^{\alpha_i} \geq \sqrt{n} > \min_{p|n}\{p-1\}$.
2. Choose a random integer $a$ between 2 and $n-2$.
3. Compute $a^e \bmod n$ by repeated squaring.
4. Compute $d = \gcd(a^e - 1, n)$ by the Euclidean algorithm. If $1 < d < n$ output the nontrivial factor $d$. Otherwise, repeat from step 2 with a new choice for $a$.

To explain when this algorithm works, assume that the integer $e$ is divisible by every integer $\leq B$ and that $p$ is a prime divisor of $n$ such that $p-1$ is the product of prime powers $\leq B$. Then $e = m(p-1)$ for some integer $m$ and hence $a^e = \left(a^{(p-1)}\right)^m \equiv 1^m = 1 \bmod p$. Therefore, $p \mid \gcd(a^e - 1, n)$ and the only way that we could fail to obtain a nontrivial factor of $n$ in step 4 is if $a^e \equiv 1 \bmod n$. In other words, we could only fail here if for every prime factor $q$ of $n$ the order of $a \bmod q$ divides $e$ and this is unlikely.

Unfortunately, it is not true that for general $n$ there is a prime divisor $p$ of $n$ for which $p-1$ is divisible by no prime power larger than $B$ for a bound $B$ of small size. If $p-1$ has a large prime power divisor for each prime divisor $p$ of $n$, then Pollard's $p-1$ method will work only for a large choice of the bound $B$ and so will be inefficient because the algorithm runs in essentially $O(B)$ time. For example, if $n$ is the product of two different primes $p$ and $q$ where $|p| \approx |q|$ are primes and $p-1$ and $q-1$ are $O(\sqrt{n})$-smooth then the method will likely require a bound $B$ of size $O(\sqrt{n})$.

Reiterating, the problem is that given input $n = \prod {p_i}^{\alpha_i}$ where the $p_i$'s are the distinct prime factors of $n$, we are restricted by the possibility that none of the integers $p_i - 1$ are sufficiently smooth. However, we can ameliorate this restriction by working with the group of points defined over elliptic curves. For each prime $p$ we will obtain a large collection of groups whose orders essentially vary "uniformly" over the interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$. By varying the groups involved we can hope to always find one whose order is smooth. We will then show how to take advantage of such a collection of groups to obtain a factorization of $n$.

## C.11 Elliptic Curves

**Definition C.36** An elliptic curve over a field $F$ is the set of points $(x, y)$ with $x$, $y \in F$ satisfying the Weierstrass equation $y^2 = x^3 + ax + b$ where $a, b \in F$ and $4a^3 + 27b^2 \neq 0$ together with a special point $O$ called the point at infinity. We shall denote this set of points by $E_{a,b}(F)$. ▌

**Remark** The condition $4a^3 + 27b^2 \neq 0$ ensures that the curve is nonsingular. That is, when the field $F$ is $\mathbf{R}$, the tangent at every point on the curve is uniquely defined.

Let $P$, $Q$ be two points on an elliptic curve $E_{a,b}(F)$. We can define the negative of $P$ and the sum $P + Q$ on the elliptic curve $E_{a,b}(F)$ according to the following rules.

1. If $P$ is the point at infinity $O$ then we define $-P$ to be $O$.
   Otherwise, if $P = (x, y)$ then $-P = (x, -y)$.
2. $O + P = P + O = P$.
3. Let $P, Q \neq O$ and suppose that $P = (x_1, y_1)$, $Q = (x_2, y_2)$.
   (i) If $P = -Q$ (that is, $x_1 = x_2$ and $y_1 = -y_2$) then we define $P + Q = O$
   (ii) Otherwise, let

$$\alpha = \frac{y_1 - y_2}{x_1 - x_2} \text{ if } P \neq Q \tag{C.2}$$

   or

$$\alpha = \frac{3x_1^2 + a}{y_1 + y_2} \text{ if } P = Q \tag{C.3}$$

   That is, if the field $F = \mathbf{R}$, $\alpha$ is the slope of the line defined by $P$ and $Q$, if $P \neq Q$, or the slope of the tangent at $P$, if $P = Q$.
   Then $P + Q = R$ where $R = (x_3, y_3)$ with $x_3 = \alpha^2 - (x_1 + x_2)$ and $y_3 = \alpha(x_1 - x_3) - y_1$.

It can be shown that this definition of addition on the elliptic curve is associative and always defined, and thus it imposes an additive abelian group structure on the set $E_{a,b}(F)$ with $O$ serving as the additive identity of the group. We will be interested in $F = \mathbf{Z}_p$ where $p \neq 2, 3$ is a prime. In this case, addition in $E_{a,b}(\mathbf{Z}_p)$ can be computed in time polynomial in $|p|$ as equations $(C.2)$ and $(C.3)$ involve only additions, subtractions, and divisions modulo $p$. Note that to compute $z^{-1} \bmod p$ where $z \in \mathbf{Z}_p^*$ we can use the extended Euclidean algorithm to compute an integer $t$ such that $tz \equiv 1 \bmod p$ and then set $z^{-1} = t \bmod p$.

To illustrate negation and addition, consider the elliptic curve $y^2 = x^3 - x$ over $\mathbf{R}$ as shown in Figure C.1.

Figure C.1: Addition on the elliptic curve $y^2 = x^3 - x$.

The graph is symmetric about the $x$-axis so that the point $P$ is on the curve if and only if $-P$ is on the curve. Also, if the line $l$ through two points $P, Q \neq O$ on the curve $E(\mathbf{R})$ is not vertical then there is exactly one more point where this line intersects the curve. To see this, let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and let $y = \alpha x + \beta$ be the equation of the line through $P$ and $Q$ where $\alpha = \frac{y_1 - y_2}{x_1 - x_2}$ if $P \neq Q$ or $\alpha = \frac{3x_1^2 + a}{y_1 + y_2}$ if $P = Q$ and $\beta = y_1 - \alpha x_1$. Note that in the case where $P = Q$, we take $l$ to be the tangent at $P$ in accordance with the rules of addition on the curve $E(\mathbf{R})$. A point $(x, \alpha x + \beta)$ on the line $l$ lies on the elliptic curve if and only if $(\alpha x + \beta)^2 = x^3 + ax + b$. Thus, there is one intersection point for each root of the cubic equation $x^3 - (\alpha x + \beta)^2 + ax + b = 0$. The numbers $x_1$ and $x_2$ are roots of this equation because $(x_1, \alpha x_1 + \beta)$ and $(x_2, \alpha x_2 + \beta)$ are, respectively, the points $P$ and $Q$ on the curve. Hence, the equation must have a third root $x_3$ where $x_1 + x_2 + x_3 = \alpha^2$. This leads to the expression for $x_3$ mentioned in the rules of addition for the curve $E(\mathbf{R})$. Thus, geometrically, the operation of addition on $E(\mathbf{R})$ corresponds to drawing the line through $P$ and $Q$, letting the third intercept of the line with the curve be $-R = (x, y)$ and taking $R = (x, -y)$ to be the sum $P + Q$.

### C.11.1 Elliptic Curves Over $Z_n$

Lenstra's elliptic curve factoring algorithm works with elliptic curves $E_{a,b}(\mathbf{Z}_n)$ defined over the ring $\mathbf{Z}_n$ where $n$ is an odd, composite integer. The nonsingularity condition $4a^3 + 27b^2 \neq 0$ is replaced by $\gcd(4a^3 + 27b^2, n) = 1$.

The negation and addition rules are given as was done for elliptic curves over fields. However, the addition of two points involves a division (refer to equations $(C.2)$ and $(C.3)$ in the rules for arithmetic on elliptic curves given at the beginning of this section) which is not always defined over the ring $\mathbf{Z}_n$. For addition to be defined the denominators in these equations must be prime to $n$. Consequently, $E_{a,b}(\mathbf{Z}_n)$ is not necessarily a group. Nevertheless, we may define a method for computing multiples $e \cdot P$ of a point $P \in E_{a,b}(\mathbf{Z}_n)$ as follows.

1. Let $a_0 + a_1 2 + \cdots + a_{m-1}2^{m-1}$ be the binary expansion of $e$. Let $j = 0$, $S = 0$.
2. If $a_j = 1$ then $S \leftarrow S + 2^j P$. If this sum is not defined (namely, the division in equation $(C.2)$ or equation $(C.3)$ has failed) then output undefined and terminate.
3. $j \leftarrow j + 1$. If $j = m$ then output $S$ as the defined value for $e \cdot P$ and terminate.
4. Calculate $2^j P := 2^{j-1}P + 2^{j-1}P$. If this sum is not defined then output that $e \cdot P$ cannot be calculated and terminate. Otherwise, repeat from step 2.

The elliptic curve algorithm will use[1] this method which will be referred to as repeated doubling. Note that if the repeated doubling method is unable to calculate a given multiple $e \cdot P$ and outputs undefined then we have encountered points $Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$ on $E_{a,b}(\mathbf{Z}_n)$ such that $Q_1 + Q_2$ is not defined modulo $n$ (the division in equation $(C.2)$ or equation $(C.3)$ has failed) and hence, either $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of $n$.

Next, we state some facts concerning the relationship between elliptic curves defined over $\mathbf{Z}_n$ and elliptic curves defined over $\mathbf{Z}_p$ when $p$ is a prime divisor of $n$. Let $a, b \in \mathbf{Z}_n$ be such that $\gcd(4a^3 + 27b^2, n) = 1$. Let $p$ be a prime divisor of $n$ and let $a_p = a \bmod p$, $b_p = b \bmod p$.

**Fact C.37** $E_{a_p,b_p}(\mathbf{Z}_p)$ is an additive abelian group. ▌

Further, given $P = (x, y) \in E_{a,b}(\mathbf{Z}_n)$, define $P_p = (x \bmod p, y \bmod p)$. $P_p$ is a point on the elliptic curve $E_{a_p,b_p}(\mathbf{Z}_p)$.

**Fact C.38** Let $P$ and $Q$ be two points on $E_{a,b}(\mathbf{Z}_n)$ and let $p$ be a prime divisor of $n$. If $P + Q$ is defined modulo $n$ then $P_p + Q_p$ is defined on $E_{a,b}(\mathbf{Z}_p)$ and $P_p + Q_p = (P + Q)_p$. Moreover, if $P \neq -Q$ then the sum $P + Q$ is undefined modulo $n$ if and only if there is some prime divisor $q$ of $n$ such that the points $P_q$ and $Q_q$ add up to the point at infinity $O$ on $E_{a_q,b_q}(\mathbf{Z}_q)$ (equivalently, $P_q = -Q_q$ on $E_{a_q,b_q}(\mathbf{Z}_q)$). ▌

## C.11.2 Factoring Using Elliptic Curves

The main idea in Lenstra's elliptic curve factoring algorithm is to find points $P$ and $Q$ in $E_{a,b}(\mathbf{Z}_n)$ such that $P + Q$ is not defined in $E_{a,b}(\mathbf{Z}_n)$. We will assume throughout that $n$ has no factors of 2 or 3 because these can be divided out before the algorithm commences.

The algorithm runs as follows on input $n$.

1. Generate an elliptic curve $E_{a,b}(\mathbf{Z}_n)$ and a point $P = (x, y)$ on $E_{a,b}(\mathbf{Z}_n)$ by randomly selecting $x$, $y$ and $a$ in $\mathbf{Z}_n$ and setting $b = y^2 - x^3 - ax \bmod n$.
2. Compute $\gcd(4a^3 + 27b^2, n)$. If $1 < \gcd(4a^3 + 27b^2, n) < n$ then we have found a divisor of $n$ and we stop. If $\gcd(4a^3 + 27b^2, n) = 1$ then $4a^3 + 27b^2 \not\equiv 0 \bmod p$ for every prime divisor $p$ of $n$ and hence $E_{a,b}$ is an elliptic curve over $\mathbf{Z}_p$ for each prime divisor $p$ of $n$ and we may proceed. But if $\gcd(4a^3 + 27b^2, n) = n$ then we must generate another elliptic curve $E_{a,b}$.

[1] This statement is incorrect and will soon be corrected in these notes. For the method used in the algorithm, see section 2.3 of [134]

3. Set $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_1, p_2, \ldots, p_{\pi(B)}$ are the primes $\leq B$ and $\alpha_i$ is chosen maximally so that $p_i^{\alpha_i} \leq C$. $B$ and $C$ are bounds that will be determined later so as to optimize the running time and ensure that the algorithm will most likely succeed.

4. Compute $e \cdot P$ in $E_{a,b}(\mathbf{Z}_n)$ by repeated doubling. Every time before adding two intermediate points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ check if $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of $n$. If so, output the factor and stop. Otherwise, repeat from step 1.

An elliptic curve $E_{a,b}(\mathbf{Z}_n)$ will lead to a nontrivial factorization of $n$ if for some prime factors $p$ and $q$ of $n$, $e \cdot P_p = O$ on $E_{a_p,b_p}(\mathbf{Z}_p)$ but $P_q$ does not have order dividing $e$ on $E_{a_q,b_q}(\mathbf{Z}_q)$. Notice the analogy here between Lenstra's elliptic curve algorithm and Pollard's $p-1$ algorithm. In Pollard's algorithm we seek prime divisors $p$ and $q$ of $n$ such that $e$ is a multiple of the order of $a \in \mathbf{Z}_p^*$ but not a multiple of the order of $a \in \mathbf{Z}_q^*$. Similarly, in Lenstra's algorithm we seek prime divisors $p$ and $q$ of $n$ such that $e$ is a multiple of the order of $P_p \in E_{a_p,b_p}(\mathbf{Z}_p)$ but not a multiple of the order of $P_q \in E_{a_q,b_q}(\mathbf{Z}_q)$. However, there is a key difference in versatility between the two algorithms. In Pollard's algorithm, the groups $\mathbf{Z}_p^*$ where $p$ ranges over the prime divisors of $n$ are fixed so that if none of these groups have order dividing $e$ then the method fails. In Lenstra's elliptic curve algorithm the groups $E_{a_p,b_p}(\mathbf{Z}_p)$ can be varied by varying $a$ and $b$. Hence, if for every prime divisor $p$ of $n$, $|E_{a_p,b_p}(\mathbf{Z}_p)| \not| e$, then we may still proceed by simply working over another elliptic curve; that is, choosing new values for $a$ and $b$.

### C.11.3 Correctness of Lenstra's Algorithm

Suppose that there are prime divisors $p$ and $q$ of $n$ such that $e$ is a multiple of $|E_{a_p,b_p}(\mathbf{Z}_p)|$ but in $E_{a_q,b_q}(\mathbf{Z}_q)$, $P_q$ does not have order dividing $e$. Then $e \cdot P_p = O$ in $E_{a_p,b_p}(\mathbf{Z}_p)$ but $e \cdot P_q \neq O$ in $E_{a_q,b_q}(\mathbf{Z}_q)$ and thus there exists an intermediate addition of two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ in the calculation of $e \cdot P$ such that

$$x_1 \equiv x_2 \bmod p \text{ but } x_1 \not\equiv x_2 \bmod q \text{ if } P_1 \neq P_2$$

or

$$y_1 \equiv -y_2 \bmod p \text{ but } y_1 \not\equiv -y_2 \bmod q \text{ if } P_1 = P_2.$$

Hence, either $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of $n$. The points $P_1$, and $P_2$ will be encountered when $(P_1)_p + (P_2)_p = O$ in $E_{a_p,b_p}(\mathbf{Z}_p)$ but $(P_1)_q + (P_2)_q \neq O$ in $E_{a_q,b_q}(\mathbf{Z}_q)$.

### C.11.4 Running Time Analysis

The time needed to perform a single addition on an elliptic curve can be taken to be $M(n) = O(\log^2 n)$ if one uses the Euclidean algorithm. Consequently, since the computation of $e \cdot P$ uses repeated doubling, the time required to process a given elliptic curve is $O((\log e)(M(n))$. Recall that $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_i^{\alpha_i} \leq C$. Then $e \leq C^{\pi(B)}$ and therefore, $\log e \leq \pi(B) \log C$. Now, let $p$ be the smallest prime divisor of $n$ and consider the choice $B = L^{\beta}(p) = \exp[\beta(\log p)^{1/2}(\log\log p)^{1/2}]$ where $\beta$ will be optimized. Then

$$\log B = \beta(\log p)^{1/2}(\log\log p)^{1/2} = e^{[\log\beta + \frac{1}{2}(\log\log p) + \frac{1}{2}(\log\log\log p)]}$$

and thus

$$\pi(B) \approx \frac{B}{\log B} \approx O\left(e^{[\beta(\log p)^{1/2}(\log\log p)^{1/2} - \frac{1}{2}(\log\log p)]}\right) \approx O(L^{\beta}(p)).$$

Hence, the time required for each iteration of the algorithm is $O(L^{\beta}(p)M(n)(\log C))$. In choosing $C$, note that we would like $e$ to be a multiple of $|E_{a_p,b_p}(\mathbf{Z}_p)|$ for some prime divisor $p$ of $n$ and thus it is sufficient to take $C = |E_{a_p,b_p}(\mathbf{Z}_p)|$ where $p$ is some prime divisor of $n$, provided that $|E_{a_p,b_p}(\mathbf{Z}_p)|$ is $B$-smooth. The value of $p$ is unknown, but if $p$ is the smallest prime divisor of $n$ then $p < \sqrt{n}$. We also know by Hasse's Inequality (see, for example, page 131 of [196]) that $p+1-2\sqrt{p} < |E_{a_p,b_p}(\mathbf{Z}_p)| < p+1+2\sqrt{p}$ and thus $|E_{a_p,b_p}(\mathbf{Z}_p)| < \sqrt{n}+1+2\sqrt[4]{n}$. Hence, it is safe to take $C = \sqrt{n} + 1 + 2\sqrt[4]{n}$.

The only remaining consideration is to determine the expected number of elliptic curves that must be examined before we obtain a factorization of $n$ and this part of the analysis relies on the following result due to Lenstra which appears as Proposition 2.7 in [134].

**Proposition C.39** Let $S = \{s \in \mathbf{Z} : |s - (p+1)| < \sqrt{p} \text{ and } s \text{ is } L(p)^{\beta}\text{-smooth}\}$. Let $n$ be a composite integer that has at least two distincet prime divisors exceeding 3. Then

$$\Pr[\text{Lenstra's algorithm factors } n] \geq \Omega\left(\frac{|S|-2}{2\sqrt{p}}\right)\left(\frac{1}{\log p}\right)$$

(where the probability is taken over $x$, $y$, and $a$ in $\mathbf{Z}_n$). ▌

In other words, the proposition asserts that the probability that a random triple $(x, y, a)$ leads to a factorization of $n$ is essentially the probability that a random integer in the interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$ is $L(p)^{\beta}$-smooth; the latter probability being $\frac{|S|}{2\lfloor\sqrt{p}\rfloor+1}$.

Recall that we dealt earlier with the smoothness of integers less than some bound and saw that a theorem due to Canfield, Erdös and Pomerance (see [52]) implies that

$$\Pr[m \leq x \text{ is } L(x)^{\alpha}\text{-smooth}] = L^{-\frac{1}{2\alpha}}(x).$$

However, as we have just seen, we require here the unproven conjecture that the same result is valid if $m$ is a random integer in the small interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$; specifically, that

$$\Pr[m \in (p+1-\sqrt{p}, p+1+\sqrt{p}) \text{ is } L(p)^{\beta}\text{-smooth}] = L^{-\frac{1}{2\beta}}(p).$$

Consequently, the lower bound on the probability of success in Proposition C.39 can be made explicit. Hence, we have

$$\Pr[\text{Lenstra's algorithm factors } n] \geq \Omega\left(L^{-\frac{1}{2\beta}}(p)\right)\frac{1}{\log p}.$$

Therefore, we expect to have to try $L^{\frac{1}{2\beta}}(p)(\log p)$ elliptic curves before encountering one of $L^{\beta}(p)$-smooth order. Thus, the total running time required is expected to be $O(L^{\frac{1}{2\beta}}(p)(\log p)L^{\beta}(p)M(n)(\log C)) = O(L^{\beta+\frac{1}{2\beta}}(p)(\log^4 n))$. This achieves a minimum of $O(L^{\sqrt{2}}(p)(\log^4 n))$ when $\beta = \frac{1}{\sqrt{2}}$.

**Remark** In step 3 of Lenstra's algorithm a minor practical problem arises with the choice of $B = L^{\beta}(p)$ because the smallest prime divisor $p$ of $n$ is not known before the algorithm begins. This problem can be resolved by taking $B = L^{\beta}(v)$ and performing the algorithm for a gradually increasing sequence of values for $v$ while factorization continues to be unsuccessful and declaring failure if $v$ eventually exceeds $\sqrt{n}$ because the smallest prime divisor of $n$ is less than $\sqrt{n}$.

CHAPTER D

# About PGP

PGP is a free software package that performs cryptographic tasks in association with email systems. In this short appendix we will review some of its features. For a complete description of its functioning readers are referred to Chapter 9 in [198].

## D.1 Authentication

PGP performs authentication of messages using a hash-and-sign paradigm. That is given a message $M$, the process is as following:

- The message is timestamped, i.e. date and time are appended to it;
- it is then hashed using MD5 (see [175]);
- the resulting 128-bit digest is signed with the sender private key using RSA [176];
- The signature is prepended to the message.

## D.2 Privacy

PGP uses a hybrid system to ensure privacy. That is each message is encrypted using a fast symmetric encryption scheme under a one-time key. Such key is encrypted with the receiver public-key and sent together with the encrypted message.

In detail, assume A wants to send an encrypted message to B.

- A compresses the message using the ZIP compression package; let $M$ be the resulting compressed message.
- A generates a 128-bit random key $k$;
- The message $M$ is encrypted under $k$ using the symmetric encryption scheme IDEA (see [129] or Chapter 7 of [198]); let $C$ be the corresponding ciphertext;
- $k$ is encrypted under B's public key using RSA; let $c$ be the corresponding ciphertext.
- The pair $(c, C)$ is sent to B.

If both authentication and privacy are required, the message is first signed, then compressed and then encrypted.

## D.3 Key Size

PGP allows for three key sizes for RSA

- *Casual* 384 bits
- *Commercial* 512 bits
- *Military* 1024 bits

## D.4 E-mail compatibility

Since e-mail systems allow only the transmission of ASCII characters, PGP needs to recovert eventual encrypted parts of the message (a signature or the whole ciphertext) back to ASCII.

In order to do that PGP applies the radix-64 conversion to bring back a binary stream into the ASCII character set. This conversion expands the message by 33%. However because of the original ZIP compression, the resulting ciphertext is still one-third smaller than the original message.

In case the resulting ciphertext is still longer than the limit on some e-mail systems, PGP breaks into pieces and send the messages separately.

## D.5 One-time IDEA keys generation

Notice that PGP does not have session keys, indeed each message is encrypted under a key $k$ generated ad hoc for that message.

The generation of such key is done using a pseudo-random number generator that uses IDEA as a building block. The seed is derived from the keystrokes of the user. That is, form the actual keys being typed and the time intervals between them.

## D.6 Public-Key Management

Suppose you think that $PK$ is the public key of user B, while instead it is C who knows the corresponding secret key $SK$.

This can create two major problems:

1. C can read encrypted messages that A thinks she is sending to B
2. C can have A accept messages as coming from B.

The problem of establishing trust in the connection between a public-key and its owner is at the heart of public-key systems, no just of PGP.

There are various ways of solving this problem:

- *Physical exchange* B could give the key to A in person, stored in a floppy disk.
- *Verification* A could call B on the phone and verify the key with him
- *Certification Authorithy* There could be a trusted center $AUTH$ that signs public keys for the users, establishing the connection between the key and the ID of the user (such a signature is usually referred to as a *certificate*.)

Only the last one seems reasonable and it appears to be the way people are actually implementing public key systems in real life.

PGP does not use any of the above systems, but it rather uses a *decentralized trust* system. Users reciprocally certify each other's keys and one trusts a key to the extent that he/she trusts the user who certify it for it. Details can be found in [198]

CHAPTER E

# Problems

This chapter contains some problems for you to look at.

## E.1 Secret Key Encryption

### E.1.1 DES

Let $\bar{m}$ be the bitwise complement of the string $m$. Let $\mathrm{DES}_K(m)$ denote the encryption of $m$ under DES using key $K$. It is not hard to see that if

$$c = \mathrm{DES}_K(m)$$

then

$$\bar{c} = \mathrm{DES}_{\bar{K}}(\bar{m})$$

We know that a brute–force attack on DES requires searching a space of $2^{56}$ keys. This means that we have to perform that many DES encryptions in order to find the key, in the worst case.

1. Under *known plaintext attack* (i.e., you are given a single pair $(m, c)$ where $c = \mathrm{DES}_K(m)$) do the equations above change the number of DES encryption you perform in a brute–force attack to recover $K$?
2. What is the answer to the above question in the case of *chosen plaintext attack* (i.e., when you are allowed to choose many $m$'s for which you get the pair $(m, c)$ with $c = \mathrm{DES}_K(m)$)?

### E.1.2 Error Correction in DES ciphertexts

Suppose that $n$ plaintext blocks $x_1,\ldots,x_n$ are encrypted using DES producing ciphertexts $y_1, \ldots, y_n$. Suppose that one ciphertext block, say $y_i$, is transmitted incorrectly (i.e. some 1's are changed into 0's and viceversa.) How many plaintext blocks will be decrypted incorrectly if the ECB mode was used for encryption? What if CBC is used?

### E.1.3 Brute force search in CBC mode

A brute-force key search for a known-plaintext attack for DES in the ECB mode is straightforward: given the 64-bit plaintext and the 64 bit ciphertext, try all of the possible $2^{56}$ keys until one is found that generates the known ciphertext from the known plaintext. The situation is more complex for the CBC mode, which includes the use of a 64-bit IV. This seems to introduce an additional 64 bits of uncerntainty.

1. Suggest strategies for known-plaintext attack on the CBC mode that are of the same order of magnitude of effort as the ECB attack.

2. Now consider a ciphertext only attack. For ECB mode the strategy is to try to decrypt the given ciphertext with all possible $2^{56}$ keys and test each result to see if it appears to be a syntactically correct plaintext. Will this strategy work for the CBC mode? If so, explain. If not, describe an attack strategy for the CBC mode and estimate its level of effort.

### E.1.4 E-mail

Electronic mail systems differ in the way in which multiple recipients are handled. In some systems the originating mail handler makes all the necessary copies, and these are sent out independently. An alterantive approach is to determine the route for each destination first. Then a single message is sent out on a common portion of the route and copies are made when the routes diverge (this system is known as *mail-bagging*.)

1. Leaving aside security considerations, discuss the relative advantages and disadvantages of the two methods.

2. Discuss the security requirements and implications of the two methods

## E.2 Passwords

The framework of (a simplified version of) the Unix password scheme is this. We fix some function $h\colon \{0,1\}^k \to \{0,1\}^L$. The user chooses a $k$-bit password, and the system stores the value $y = h(K)$ in the password file. When the user logs in he must supply $K$. The system then computes $h(K)$ and declares you authentic if this value equals $y$.

We assume the attacker has access to the password file and hence to $y$. The intuition is that it is computationally infeasible to recover $K$ from $y$. Thus $h$ must be chosen to make this true.

The specific choice of $h$ made by Unix is $h(K) = \mathrm{DES}_K(0)$ where "0" represents the 64 bit string of all zeros. Thus $k = 56$ and $L = 64$.

In this problem you will analyze the generic scheme and the particular DES based instantiation. The goal is to see how, given a scheme like this, to use the models we have developed in class, in particular to think of DES as a pseudorandom function family.

To model the scheme, let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a pseudorandom function family, having some given insecurity function $\mathbf{Adv}^{\mathrm{prf}}_F(\cdot,\cdot)$, and with $L > k$. We let $T_F$ denote the time to compute $F$. (Namely the time, given $K, x$, to compute $F_K(x)$.) See below for the definition of a one-way function, which we will refer to now.

**(a)** Define $h\colon \{0,1\}^k \to \{0,1\}^L$ by $h(K) = F_K(0)$, where "0" represents the $l$-bit string of all zeros. Prove that $h$ is a one-way function with

$$\mathbf{Adv}^{\mathrm{owf}}_{(}h,t) \;\le\; 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_F(t',1)\ ,$$

where $t' = t + O(l + L + k + T_F)$.

*Hints:* Assume you are given an inverter $I$ for $h$, and construct a distinguisher $D$ such that

$$\mathbf{Adv}^{\mathrm{prf}}_F((\,)D) \;\ge\; \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{owf}}_{h,I}\ .$$

Use this to derive the claimed result.

**(b)** Can you think of possible threats or weaknesses that might arise in a real world usage of such a scheme, but are not covered by our model? Can you think of how to protect against them? Do you think this is a good password scheme "in practice"?

We now provide the definition of security for a one-way function to be used above.

Let $h\colon \{0,1\}^k \to \{0,1\}^L$ be a function. It is one-way, if, intuitively speaking, it is hard, given $y$, to compute a point $x'$ such that $h(x') = y$, when $y$ was chosen by drawing $x$ at random from $\{0,1\}^k$ and setting $y = h(x)$.

In formalizing this, we say an *inverter* for $h$ is an algorithm $I$ that given a point $y \in \{0,1\}^L$ tries to compute this $x'$. We let

$$\mathbf{Adv}^{\mathrm{owf}}_{h,I} \;=\; \mathbf{Pr}\left[\, h(x') = y \;:\; x \xleftarrow{\$} \{0,1\}^k \;;\; y \leftarrow h(x) \;;\; x' \leftarrow I(y) \,\right]$$

be the probability that the inverter is successful, taken over a random choice of $x$ and any coins the inverter might toss. We let

$$\mathbf{Adv}^{\mathrm{owf}}_{h}(t') \;=\; \max_{I}\{\mathbf{Adv}^{\mathrm{owf}}_{h,I}\}\,,$$

where the maximum is over all inverters $I$ that run in time at most $t'$.

## E.3 Number Theory

### E.3.1 Number Theory Facts

Prove the following facts:

1. If $k$ is the number of distinct prime factors of $n$ then the equation $x^2 = 1 \bmod n$ has $2^k$ distinct solutions in $Z_n^*$. *Hint: use Chinese Remainder Theorem*
2. If $p$ is prime and $x \in Z_p^*$ then $(\frac{x}{p}) = x^{\frac{p-1}{2}}$
3. $g$ is a generator of $Z_p^*$ for a prime $p$, iff $g^{p-1} = 1 \bmod p$ and $g^q \neq 1 \bmod p$ for all $q$ prime divisors of $p-1$

### E.3.2 Relationship between problems

Let $n$ be the product of two primes $n = pq$. Describe reducibilities between the following problems (e.g. if we can factor we can invert RSA.) Don't prove anything formally, just state the result.

- computing $\phi(n)$
- factoring $n$
- computing $QR_n(a)$ for some $a \in Z_n^*$
- computing square roots modulo $n$
- computing $k$-th roots modulo $n$, where $gcd(k, \phi(n)) = 1$

### E.3.3 Probabilistic Primality Test

Let $SQRT(p, a)$ denote an expected polynomial time algorithm that on input $p, a$ outputs $x$ such that $x^2 = a \bmod p$ if $a$ is a quadratic residue modulo $p$. Consider the following probabilistic primality test, which takes as an input an odd integer $p > 1$ and outputs "composite" or "prime".

1. Test if there exist $b, c > 1$ such that $p = b^c$. If so output "composite"
2. Choose $i \in Z_p^*$ at random and set $y = i^2$
3. Compute $x = SQRT(p, y)$
4. If $x = i \bmod p$ or $x = -i \bmod p$ output "prime", otherwise output "composite"

(A) Does the above primality test always terminate in expected polynomial time? Prove your answer.

(B) What is the probability that the above algorithm makes an error if $p$ is prime?

(C) What is the probability that the above algorithm makes an error if $p$ is composite?

## E.4 Public Key Encryption

### E.4.1 Simple RSA question

Suppose that we have a set of block encoded with the RSA algorithm and we don't have the private key. Assume $n = pq, e$ is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with $n$. Does this help us in any way?

### E.4.2 Another simple RSA question

In the RSA public-key encyption scheme each user has a public key $n, e$ and a private key $d$. Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new pair $e', d'$. Is this a good idea?

### E.4.3 Protocol Failure involving RSA

Remember that an RSA public–key is a pair $(n, e)$ where $n$ is the product of two primes.

$$RSA_{(n,e)}(m) = m^e \bmod n$$

Assume that three users in a network Alice, Bob and Carl use RSA public–keys $(n_A, 3)$, $(n_B, 3)$ and $(n_C, 3)$ respectively. Suppose David wants to send the *same* message $m$ to the three of them. So David computes

$$y_A = m^3 \bmod n_A \ , \ y_B = m^3 \bmod n_B \ , \ y_C = m^3 \bmod n_C$$

and sends the ciphertext to the relative user.

Show how an eavesdropper Eve can now compute the message $m$ even without knowing any of the secret keys of Alice, Bob and Carl.

### E.4.4 RSA for paranoids

The best factoring algorithm known to date (the *number field sieve*) runs in

$$e^{O(\log^{1/3} n \log\log^{2/3} n)}$$

That is, the running time does not depend on the size of the smallest factor, but rather in the size of the whole composite number.

The above observation seem to suggest that in order to preserve the security of RSA, it may not be necessary to increase the size of both prime factors, but only of one of them.

Shamir suggested the follwong version of RSA that he called *unbalanced RSA* (also known as RSA for paranoids). Choose the RSA modulus $n$ to be 5,000 bits long, the product of a 500-bits prime $p$ and a 4,500-bit prime $q$. Since usually RSA is usually used just to exchange DES keys we can assume that the messages being encrypted are smaller than $p$.

**(A)** How would you choose the public exponent $e$? Is 3 a good choice?

Once the public exponent $e$ is chosen, one computes $d = e^{-1} \bmod \phi(n)$ and keep it secret. The problem with such a big modulus $n$, is that decrypting a ciphertext $c = m^e \bmod n$ may take a long time (since one has to

compute $c^d \bmod n$.) But since we know that $m < p$ we can just use the Chinese Remainder Theorem and compute $m_1 = c^d \bmod p = m$. Shamir claimed that this variant of RSA achieves better security against the advances of factoring, without losing in efficiency.

**(B)** Show how with a single chosen message attack (i.e. obtaining the decryption of a message of your choice) you can completely break the unbalanced RSA scheme, by factoring $n$.

### E.4.5 Hardness of Diffie-Hellman

Recall the Diffie-Hellman key exchange protocol. $p$ is a prime and $g$ a generator of $Z_p^*$. Alice's secret key is a random $a < p$ and her public key is $g^a \bmod p$. Similarly Bob's secret key is a random $b < p$ and his public key is $g^b \bmod p$. Their common key is $g^{ab}$.

In this problem we will prove that if the Diffie-Hellman key exchange protocol is secure for a small fraction of the values $(a, b)$, then it is secure for almost all values $(a, b)$.

Assume that there is a *ppt* algorithm $\mathcal{A}$ that

$$Prob[\mathcal{A}(g^a, g^b) = g^{ab}] > \frac{1}{2} + \epsilon$$

(where the probability is taken over the choices of $(a, b)$ and the internal coin tosses of $\mathcal{A}$)

Your task is to prove that for any $\delta < 1$ there exists a *ppt* algorithm $\mathcal{B}$ such that for all $(a, b)$

$$Prob[\mathcal{B}(g^a, g^b) = g^{ab}] > 1 - \delta$$

(where the probability is now taken only over the coin tosses of $\mathcal{B}$)

### E.4.6 Bit commitment

Consider the following "real life" situation. Alice and Bob are playing "Guess the bit I am thinking". Alice thinks a bit $b = 0, 1$ and Bob tries to guess it. Bob declares his guess and Alice tells him if the guess is right or not.

However Bob is losing all the time so he suspects that Alice is cheating. She hears Bob's guess and she declares she was thinking the opposite bit. So Bob requires Alice to write down the bit in a piece of paper, seal it in an envelope and place the envelope on the table. At this point Alice is committed to the bit. However Bob has no information about what the bit is.

Our goal is to achieve this bit commitment without envelopes. Consider the following method. Alice and Bob together choose a prime $p$ and a generator $g$ of $Z_p^*$. When Alice wants to commit to a bit $b$ she choose a random $x \in Z_p^*$ such that $lsb(x) = b$ and she publishes $y = g^x \bmod p$. Is this a good bit commitment? Do you have a better suggestion?

### E.4.7 Perfect Forward Secrecy

Suppose two parties, Alice and Bob, want to communicate privately. They both hold public keys in the traditional Diffie-Hellman model.

An eavesdropper Eve stores all the encrypted messages between them and one day she manages to break into Alice and Bob's computer and find their secret keys, correspondent to their public keys.

Show how using only public–key cryptography we can achieve *perfect forward secrecy*, i.e., Eve will not be able to gain any knowledge about the messages Alice and Bob exchanged before the disclosure of the secret keys.

### E.4.8 Plaintext-awareness and non-malleability

We say that an encryption scheme is *plaintext–aware* if it is impossible to produce a valid ciphertext without knowing the corresponding plaintext.

Usually plaintext-aware encryption schemes are implemented by adding some redundancy to the plaintext. Decryption of a ciphertext results either in a valid message or in a flag indicating non–validity (if the redundancy is not of the correct form.) Correct decryption convinces the receiver that the sender *knows* the plaintext that was encrypted.

The concept of plaintext–awareness is related to the concept of malleability. We say that an encryption scheme $E$ is *non–malleable* if it given a ciphertext $c = E(m)$ it is impossible to produce a valid ciphertext $c'$ of a related message $m'$.

Compare the two definitions and tell us if one implies the other.

### E.4.9 Probabilistic Encryption

Assume that you have a message $m$ that you want to encrypt in a probabilistic way. For each of the following methods, tell us if you think it is a good or a bad method.

1. Fix $p$ a large prime and let $g$ be a generator. For each bit $b_i$ in $m$, choose at random $x_i \in Z_{p-1}$ such that $lsb(x_i) = b_i$ ($lsb(x)$ = least significant bit of $x$.) The ciphertext is the concatenation of the $y_i = g^{x_i} \bmod p$. What about if you use $x$ such that $msb(x_i) = b_i$?

2. Choose an RSA public key $n, e$ such that $|n| > 2|m|$. Pad $m$ with random bits to get it to the same length of $n$. Let $m'$ be the padded plaintext. Encrypt $c = m^e \bmod n$.

3. Choose an RSA public key $n, e$. Assume that $|m|$ is smaller than $\log \log n$ (you can always break the message in blocks of that size.) Pad $m$ with random bits to get it to the same length of $n$. Let $m'$ be the padded plaintext. Encrypt $c = m^e \bmod n$.

4. Choose two large primes $p, q = 3 \bmod 4$. Let $n = pq$. For each bit $b_i$ in $m$, choose at random $x_i \in Z_n^*$ and set $y_i = x_i^2 \bmod n$ if $b_i = 0$ or $y_i = -x_i^2 \bmod n$ if $b_i = 1$. The ciphertext is the concatenation of the $y_i$'s.

## E.5 Secret Key Systems

### E.5.1 Simultaneous encryption and authentication

Let $(\mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme (cf. Chapter 6and MAC a message authentication code (cf. Chapter 9). Suppose Alice and Bob share two keys $K_1$ and $K_2$ for privacy and authentication respectively. They want to exchange messages $M$ in a private and authenticated way. Consider sending each of the following as a means to this end:

1. $M, \mathrm{MAC}_{K_2}(\mathcal{E}_{K_1}(M))$

2. $\mathcal{E}_{K_1}(M, \mathrm{MAC}_{K_2}(M))$

3. $\mathcal{E}_{K_1}(M), \mathrm{MAC}_{K_2}(M)$

4. $\mathcal{E}_{K_1}(M), \mathcal{E}_{K_1}(\mathrm{MAC}_{K_2}(M))$

5. $\mathcal{E}_{K_1}(M), \mathrm{MAC}_{K_2}(\mathcal{E}_{K_1}(M))$

6. $\mathcal{E}_{K_1}(M, A)$ where $A$ encodes the identity of Alice. Bob decrypts the ciphertext and checks that the second half of the plaintext is $A$

For each say if it secure or not and briefly justify your answer.

## E.6 Hash Functions

### E.6.1 Birthday Paradox

Let $H$ be a hash function that outputs $m$-bit values. Assume that $H$ behaves as a *random oracle*, i.e. for each string $s$, $H(s)$ is uniformly and independently distributed between 0 and $2^m - 1$.

Consider the following brute–force search for a collision: try all possible $s_1, s_2, \ldots$ until a collision is found. (That is, keep hashing until some string yields the same hash value as a previously hashed string.)

Prove that the expected number of hashing performed is approximately $2^{\frac{m}{2}}$.

### E.6.2 Hash functions from DES

In this problem we will consider two proposals to construct hash functions from symmetric block encryption schemes as DES.

Let $E$ denote a symmetric block encryption scheme. Let $E_k(M)$ denote the encryption of the 1–block message $M$ under key $k$. Let $M = M_0 \circ M_1 \circ M_2 \circ \ldots \circ M_n$ denote a message of $n+1$ blocks.

The first proposed hash function $h_1$ works as follows: let $H_0 = M_0$ and then define

$$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1} \quad \text{for} \quad i = 1, \ldots, n.$$

The value of the hash function is defined as

$$h_1(M) = H_n$$

The second proposed hash function $h_2$ is similar. Again $H_0 = M_0$ and then

$$H_i = E_{H_{i-1}}(M_i) \oplus M_i \quad \text{for} \quad i = 1, \ldots, n.$$

The value of the hash function is defined as

$$h_2(M) = H_n$$

For both proposals, show how to find collisions if the encryption scheme $E$ is chosen to be DES.

### E.6.3 Hash functions from RSA

Consider the followng hash function $H$. Fix an RSA key $n, e$ and denote with $RSA_{n,e}(m) = m^e \bmod n$. Let the message to be hashed be $m = m_1 \ldots m_k$. Denote with $h_1 = m_1$ and for $i > 1$,

$$h_i = RSA_{n,e}(h_{i-1}) \oplus m_i$$

Then $H(m) = h_n$. Show how to find a collision.

## E.7 Pseudo-randomness

### E.7.1 Extending PRGs

Suppose you are given a PRG $G$ which stretches a $k$ bit seed into a $2k$ bit pseudorandom sequence. We would like to construct a PRG $G'$ which stretches a $k$ bit seed into a $3k$ bit pseudorandom sequence.

Let $G_1(s)$ denote the first $k$ bits of the string $G(s)$ and let $G_2(s)$ the last $k$ bits (that is $G(s) = G_1(s).G_2(s)$ where $a.b$ denotes the concatenation of strings $a$ and $b$.)

Consider the two constructions

1. $G'(s) = G_1(s).G(G_1(s))$
2. $G''(s) = G_1(s).G(G_2(s))$

For each construction say whether it works or not and justify your answer. That is, if the answer is no provide a simple statistical test that distinguishes the output of, say, $G'$ from a random $3k$ string. If the answer is yes prove it.

### E.7.2 From PRG to PRF

Let us recall the construction of PRFs from PRGs we saw in class. Let $G$ be a length-doubling PRG, from seed of length $k$ to sequences of length $2k$.

Let $G_0(x)$ denote the first $k$ bits of $G(x)$ and $G_1(x)$ the last $k$ bits. In other words $G_0(x) \circ G_1(x) = G(x)$ and $|G_0(x)| = |G_1(x)|$.

For any bit string $z$ recursively define $G_{0 \circ z}(x) \circ G_{1 \circ z}(x) = G(G_z(x))$ with $|G_{0 \circ z}(x)| = |G_{1 \circ z}|$.

The PRF family we constructed in class was defined as $\mathcal{F} = \{f_i\}$. $f_i(x) = G_x(i)$. Suppose instead that we defined $f_i(x) = G_i(x)$. Would that be a PRF family?

## E.8 Digital Signatures

### E.8.1 Table of Forgery

For both RSA and ElGamal say if the scheme is

1. universally forgeable
2. selectively forgeable
3. existentially forgeable

and if it is under which kind of attack.

### E.8.2 ElGamal

Suppose Bob is using the ElGamal signature scheme. Bob signs two messages $m_1$ and $m_2$ with signatures $(r, s_1)$ and $(r, s_2)$ (the same value of $r$ occurs in both signatures.) Suppose also that $gcd(s_1 - s_2, p - 1) = 1$.

1. Show how $k$ can be computed efficiently given this information
2. Show how the signature scheme can subsequently be broken

### E.8.3 Suggested signature scheme

Consider the following discrete log based signature scheme. Let $p$ be a large prime and $g$ a generator. The private key is $x < p$. The public key is $y = g^x \bmod p$.

To sign a message $M$, calculate the hash $h = H(M)$. If $gcd(h, p - 1)$ is different than 1 then append $h$ to $M$ and hash again. Repeat this until $gcd(h, p - 1) = 1$. Then solve for $Z$ in

$$Zh = X \bmod (p - 1)$$

The signature of the message is $s = g^Z \bmod p$. To verify the signature, a user checks that $s^h = Y \bmod p$.

1. Show that valid signatures are always accepted
2. Is the scheme secure?

### E.8.4 Ong-Schnorr-Shamir

Ong, Schnorr and Shamir suggested the following signature scheme.

Let $n$ be a large integer (it is not necessary to know the factorization of $n$.) Then choose $k \in Z_n^*$. Let

$$h = -k^{-2} \bmod n = -(k^{-1})^2 \bmod n$$

The public key is $(n, h)$, the secret key is $k$.

To sign a message $M$, generate a random number $r$, such that $r$ and $n$ are relatively prime. Then calculate

$$S_1 = \frac{M/r + r}{2} \bmod n$$

$$S_2 = \frac{k}{2}(M/r - r)$$

The pair $(S_1, S_2)$ is the signature.

To verify the signature, check that

$$M = S_1^2 + hS_2^2 \bmod$$

1. Prove that reconstructing the private key, from the public key is equivalent to factor $n$.
2. Is that enough to say that the scheme is secure?

## E.9 Protocols

### E.9.1 Unconditionally Secure Secret Sharing

Consider a generic Secret Sharing scheme. A dealer $D$ wants to share a secret $s$ between $n$ trustees so that no $t$ of them have *any* information about $s$, but $t + 1$ can reconstruct the secret. Let $s_i$ be the share of trustee $T_i$. Let $v$ denote the number of possible values that $s$ might have, and let $w$ denote the number of different possible share values that a given trustee might receive, as $s$ is varied. (Let's assume that $w$ is the same for each trustee.)

Argue that $w \geq v$ for any Secret Sharing Scheme. (It then follows that the number of bits needed to represent a share can not be smaller than the number of bits needed to represent the secret itself.)

*Hint: Use the fact that t players have* NO *information about the secret—no matter what t values they have received, any value of s is possible.*

### E.9.2 Secret Sharing with cheaters

Dishonest trustees can prevent the reconstruction of the secret by contributing *bad* shares $\hat{s_i} \neq s_i$. Using the cryptographic tools you have seen so far in the class show how to prevent this denial of service attack.

### E.9.3 Zero–Knowledge proof for discrete logarithms

Let $p$ be a prime and $g$ a generator modulo $p$. Given $y = g^x$ Alice claims she knows the discrete logarithm $x$ of $y$. She wants to convince Bob of this fact but she does not want to reveal $x$ to him. How can she do that? (Give a zero-knowledge protocol for this problem.)

### E.9.4 Oblivious Transfer

An oblivious transfer protocol is a communication protocol between Alice and Bob. Alice runs it on input a value $s$. At the end of the protocol either Bob learns $s$ or he has no information about it. Alice has no idea which event occurred.

An 1-2 oblivious transfer protocol is a communication protocol between Alice and Bob. Alice runs it on inout two values $s_0$ and $s_1$. Bob runs it on input a bit $b$. At the end of the protocol, Bob learns $s_b$ but has no information about $s_{1-b}$. Alice has no information about $b$.

Show that given an oblivious transfer protocol as a black box, one can design a 1-2 oblivious transfer protocol.

### E.9.5 Electronic Cash

Real-life cash has two main properties:

- It is *anonymous*: meaning when you use cash to buy something your identity is not revealed, compare with credit cards where your identity and spending habits are disclosed
- It is *transferable*: that is the vendor who receives cash from you can in turn use it to buy something else. He would not have this possibility if you had payed with a non-transferable check.

The electronic cash proposals we saw in class are all "non–transferable". that is the user gets a coin from the bank, spends it, and the vendor must return the coin to the bank in order to get credit. As such they really behave as anonymous non-transferable checks. In this problem we are going to modify such proposals in order to achieve transferability.

The proposal we saw in class can be abstracted as follows: we have three agents: the Bank, the Userand the Vendor.

The Bankhas a pair of keys $(S, P)$. A signature with $S$ is a *coin* worth a fixed amount (say \$1.). It is possible to make blind signatures, meaning the Usergets a signature $S(m)$ on a message $m$, but the Bankgets no information about $m$.

*Withdrawal protocol*

1. The Userchooses a message $m$
2. The Bankblindly signs $m$ and withdraws \$1 from User's account.
3. The Userrecovers $S(m)$. The coin is the pair $(m, S(m))$.

*Payment Protocol*

1. The Usergives the coin $(m, S(m))$ to the Vendor.
2. The Vendorverifies the Banksignature and sends a random challenge $c$ to the User.
3. The Userreplies with an answer $r$
4. the Vendorverifies that the answer is correct.

The challenge–response protocol is needed in order to detect double–spending. Indeed the system is constructed in such a way that if the Useranswers two different challenges on the same coin (meaning he's trying to spend the coin twice) his identity will be revealed to the Bankwhen the two coins return to the bank. This is why the whole history of the payment protocol must be presented to the Bankwhen the Vendordeposits the coin.

*Deposit protocol*

1. The Vendorsends $m, S(m), c, r$ to the Bank

2. The Bankverifies it and add \$1 to the Vendor's account.

3. The Banksearches its database to see if the coin was deposited already and if it was reconstruct the identity of the double–spender User.

In order to make the whole scheme transferrable we give the bank a different pair of keys $(\mathcal{S}, \mathcal{P})$. It is still possible to make blind signatures with $\mathcal{S}$. However messages signed with $\mathcal{S}$ have no value. We will call them *pseudo-coins*. When people open an account with the Bank, they get a lot of these anonymous pseudo–coins by running the withdrawal protocol with $\mathcal{S}$ as the signature key.

Suppose now the Vendorreceived a payed coin $m, S(m), c, r$ and instead of depositing it wants to use it to buy something from OtherVendor. What she could do is the following:

*Transfer protocol*

1. The Vendorsends $m, S(m), c, r$ and a pseudo–coin $m', \mathcal{S}(m')$ to OtherVendor

2. OtherVendorverifies all signatures and the pair $(c, r)$. Then sends a random challenge $c'$ for the pseudo–coin.

3. Vendorreplies with $r'$

4. OtherVendorchecks the answer.

Notice however that Vendorcan still double–spend the coin $m, S(m), c, r$ if she uses two different pseudo–coins to transfer it to two differnt people. Indeed since she will never answer two different challenges on the same pseudo–coin, her identity will never be revealed. The problem is that there is no link between the real coin and the pseudo-coin used during the transfer protocol. If we could force Vendorto use only one pseudo–coin for each real coin she wants to transfer then the problem would be solved.

Show how to achieve the above goal. You will need to modify both the payment and the transfer protocol.

*Hint: If* Vendor*wants to transfer the true coin she is receiving during the payment protocol, she must be forced* then *to create a link between the true coin and the pseudo–coin she will use for the transfer later. Notice that* Vendor*chooses c at random, maybe c can be chosen in some different way?*

### E.9.6 Atomicity of withdrawal protocol

Recall the protocol that allows a Userto withdraw a coin of \$1 from the Bank. Let $(n, 3)$ be the RSA public key of the Bank.

1. The Userprepares 100 messages $m_1, \ldots, m_{100}$ which are all \$1 coins. The User*blinds* them, that is she chooses at random $r_1, \ldots, r_{100}$ and computes $w_i = r_i^3 m_i$. The Usersends $w_1, \ldots, w_{100}$ to the Bank.

2. The Bankchooses at random 99 of the blindings and asks the Userto open them. That the Bankchooses $i_1, \ldots, i_{99}$ and sends it to the User.

3. The Useropens the required blindings by revealing $r_{i_1}, \ldots, r_{i_{99}}$.

4. The Bankchecks that the blindings are constructed correctly and then finally signs the unopened blinding. W.l.o.g. assume this to be the first one. So the Banksigns $w_1$ by sending to the User$w_1^{\frac{1}{3}} = r_1 m_1^{\frac{1}{3}}$

5. The Userdivides this signature by $r_1$ and gets a signature on $m_1$ which is a valid coin.

Notice that the Userhas a probability of 1/100 to succesfully cheat.

Suppose now that the protocol is not atomic. That is the communication line may go down at the end of each step between the Bankand the User. What protocol should be followed for each step if the line goes down at the end of that step in order to prevent abuse or fraud by either party?

### E.9.7 Blinding with ElGamal/DSS

In class we saw a way to blind messages for signatures using RSA. In this problem we ask you to construct blind signatures for a variation of the ElGamal signature scheme.

The ElGamal-like signature we will consider is as follows. Let $p$ be a large prime, $q$ a large prime dividing $p-1$, $g$ an element of order $q$ in $Z_p^*$, $x$ the secret key of the Bankand $y = g^x$ the corresponding public key. Let $H$ be a collision-free hash function.

When the Bankwants to sign a message $m$ she computes

$$a = g^k \bmod p$$

for a random $k$ and

$$c = H(m, a)$$

and finally

$$b = kc + xa \bmod q$$

The signature of the message $m$ is $sig(m) = (a, b)$. Given the triple $(m, a, b)$ the verification is performed by computing $c = H(m, a)$ and checking that

$$g^b = a^c y^a$$

So the withdrawal protocol could be as following:

1. The Usertells the bank she wants a $1 coin.
2. The Bankreplies with 100 values $a_i = g^{k_i}$ for random $k_i$.
3. The Usersends back $c_i = H(m_i, a_i)$ where $m_i$ are all $1 coins.
4. The Bankasks the user to open 99 of those.
5. The Userreveals 99 of the $m_i$'s.
6. The Bankreplies with $b_i = k_i c_i + x a_i \bmod (p-1)$ for the unopened index $i$

However this is not anonymous since the Bankcan recognize the Userwhen the coin comes back. In order to make the protocol really anonymous, the Userhas to change the value of "challenge" $c_i$ computed at step 3. This modification will allow him to compute a different signature on $m_i$ on her own which will not be recognizable to the Bankwhen the coin comes back. During the protocol the Bankwill check as usual that this modification has been performed correctly by asking the Userto open 99 random blindings.