# Chapter 3

# Cryptography

## 3.1 Introduction

A central problem in secure communication is the following: *how can two parties, a sender and a receiver, communicate so that no eavesdropper can deduce the meaning of the communication*?

Suppose Alice has a message to send to Bob. Henceforth, we take a message to be a string over a specified alphabet; the string to be transmitted is usually called *plaintext*. The plaintext need not be a meaningful sentence. For instance, it could be a password or a credit card number. Any message could be intercepted by an eavesdropper, named Eve; so, it is not advisable to send the message in its plaintext form. Alice will encrypt the plaintext to create a *ciphertext*. Alice and Bob agree on a protocol, so that only Bob knows how to decrypt, i.e., convert the ciphertext to plaintext.

The goal of encryption and decryption is to make it hard (or impossible) for Eve to decrypt the ciphertext while making it easy for Alice to encrypt and Bob to decrypt. This means that Bob has some additional information, called a *key*, which Eve does not possess. When Alice and Bob share knowledge of the key, they are using a symmetric key system. Modern public key cryptography is asymmetric; encrypting uses a public key that is known to every one while decrypting requires a private key known only to the receiver.

The communication medium is really not important. Alice could write her message on a piece of paper (or on a clay tablet) and mail it physically; or she could send the message by email. Alice and Bob could engage in a telephone conversation in which only Alice speaks. Any communication medium is vulnerable, so security is achieved by choosing the encryption (and decryption) algorithms carefully.

## 3.2 Early Encryption Schemes

Secure communication has been important for at least 2,500 years, for military and romantic matters. In the very early days, the messenger simply hid the message, sometimes using invisible ink. As late as in the second world war, the Germans would often shrink a page of plaintext to a very small dot, less than 1mm in diameter, using photographic techniques, and then hide the dot within a full stop in a regular letter.

The aim of cryptography is not to hide the message but its meaning. Some of the earliest efforts simply scrambled the letters of the plaintext; this is called a transposition cypher. Thus, *attack at dawn* may be scrambled to *kntadatacwat* with the spaces removed. Since there are $n!$ permutations of $n$ symbols it may seem impossible for even a computer to do a brute-force search to decode such messages. Bob, the intended recipient, can decrypt only if he is given the scrambling sequence. Transposition cyphers are, actually, quite easy to break; so they are rarely used except by schoolchildren.

### 3.2.1 Substitution Cyphers

A substitution cypher replaces a symbol (or a group of symbols) by another symbol (or a group of symbols). In the simplest case, each symbol is paired with another, and each occurrence of a symbol is replaced by its partner. Given the substitution code in Table 3.1, *attack at dawn* becomes *daadtw da kdcn*.

| a | c | d | k | n | t | w |
|---|---|---|---|---|---|---|
| d | t | k | w | n | a | c |

Table 3.1: Substitution code for a subset of the Roman alphabet

Julius Caesar used a very simple form of substitution in communicating with his generals. He replaced the $i$th symbol of the alphabet by symbol $(i+3) \bmod n$, where $n$, the size of the alphabet, is 26. In general, of course, we can use any permutation of the alphabet, not merely a shift, as Caesar did. Caesar shift cypher is very easy to break; simply try all possible shifts. A general permutation is harder to crack, but not much harder as we will see. In all cases, the receiver must know the permutation in order to decrypt the message.

Cryptanalysis is the name given to unscrambling an intercepted message. For a substitution cypher, the eavesdropper can attempt a cryptanalysis based on the frequencies of letters in long plaintexts. Table 3.2 gives the frequencies (probability of occurrence) of letters in a piece of English text; clearly, different texts exhibit different frequencies, but the numbers given in the table are typical.

As can be seen in Table 3.2, $e$ is the most common letter, followed by $t$ and $a$. The cryptanalysis strategy is to replace the most common letter in the ciphertext by $e$, to see if it makes any sense. If not, then we try the remaining letters in sequence. For the plaintext *attack at dawn*, which has been converted to *daadtw da kdcn*, we first identify the most common letter in the ciphertext,

| Letter | Frequency | Letter | Frequency | Letter | Frequency | Letter | Frequency |
|---|---|---|---|---|---|---|---|
| a | 0.08167 | b | 0.01492 | c | 0.02782 | d | 0.04253 |
| e | 0.12702 | f | 0.02228 | g | 0.02015 | h | 0.06094 |
| i | 0.06966 | j | 0.00153 | k | 0.00772 | l | 0.04025 |
| m | 0.02406 | n | 0.06749 | o | 0.07507 | p | 0.01929 |
| q | 0.00095 | r | 0.05987 | s | 0.06327 | t | 0.09056 |
| u | 0.02758 | v | 0.00978 | w | 0.02360 | x | 0.00150 |
| y | 0.01974 | z | 0.00074 | | | | |

Table 3.2: Frequencies of letters in English text

*d*. Replacing each occurrence of *d* by *e*, the most common symbol, gives us the following string (I have used uppercase letters to show the guesses):

*EaaEtw Ea kEcn*

Since the second word is a two letter word beginning with *E*, which is uncommon except for proper names, we decide to abandon the guess that *d* is *E*. We try replacing *d* by *t*, the next most common symbol, to get

*TaaTtw Ta kTcn*

Now, it is natural to assume that *a* is really *o*, from the word *Ta*. This gives:

*TOOTtw TO kTcn*

It is unlikely that we can make any progress with the first word. So, we start fresh, with *d* set to the next most likely letter, *a*.

A variation of this scheme is to consider the frequencies of pairs of letters in the ciphertext, in the hope of eliminating certain possibilities. In Table 3.3, we take the two most common letters in the ciphertext, *d* and *a*, and compute the number of times they are adjacent to certain other letters; check that *d* and *a* are adjacent to each other 3 times and *d* and *k* are adjacent just once. We see that the adjacency of *d* and *a* is quite common. We may reasonably guess that one of *d* and *a* is a vowel. Because of the presence of the word *da*, both are not vowels.

| | a | c | d | k | n | t | w |
|---|---|---|---|---|---|---|---|
| d | 3 | 1 | | 1 | | 1 | |
| a | 1 | | 3 | | | | |

Table 3.3: Frequencies of pairs of letters

Cryptanalysis based on frequencies takes a lot of guesswork and backtracking. Computers are well suited to this task. It is not too difficult to write a program that does the cryptanalysis on such cyphers. One difficulty is that if the ciphertext is short, the frequencies may not correspond to the letter frequencies we expect. Consider a short text like *quick quiz*, which you may send to a friend regarding the ease of a pop quiz in this class. It will be difficult to decipher this one through frequency analysis, though the pair *qu*, which occurs twice, may help.

**Exercise 21**

Would it improve security to encrypt the plaintext two times using substitution cypher instead of just once? □

### 3.2.2 Electronic Transmission

For electronic transmissions, the encryption and decryption can be considerably more elaborate than the transposition or substitution codes. The first step in any such transmission is to convert the message to be sent to a bit string, by replacing each symbol of the alphabet by its ascii representation, for instance. This string is now the plaintext. Next, the plaintext is broken up into fixed size blocks, typically around 64 bits in length, which are then encrypted and sent.

A simple encryption scheme is as follows. Alice and Bob agree on a key $k$, which is a bit string of the same length as the block. Encrypt $x$ by $y$, where $y = x \oplus k$, i.e., $y$ is the exclusive or of $x$ and $k$. Bob, on receiving $y$, computes $y \oplus k$ which is $(x \oplus k) \oplus k$, i.e., $x$, the original message. Eve can only see $y$, which appears as pure gibberish. The transmission can only be decrypted by someone in possession of key $k$.

There are many variations of this simple scheme. It is better to have a long key, much longer than the block length, so that successive blocks are encrypted using different strings. When the bits from $k$ run out, wrap around and start reusing the bits of $k$ from the beginning. Using a longer key reduces the possibility of the code being broken.

This communication scheme is simple to program; in fact, encryption and decryption have the same program. Each operation is fast, requiring time proportional to a block length for encryption (and decryption). Yet, the scheme has significant drawbacks. If Eve can decode a single block, she can decode all blocks (assuming that the key length is the same as the block length), because given $x$ and $y$ where $y = x \oplus k$, $k$ is simply $x \oplus y$. Also, Alice and Bob will have to agree on a key before the transmission takes place, so the keys have to be transmitted first in a secure manner, a problem known as *key exchange*. For these and other reasons, this form of encryption is rarely used in high security applications.

A major problem in devising a secure communication protocol is that Alice may send several messages to Bob, and as the number of transmissions increase, so is the probability of breaking the code. Therefore, it is advisable to use a different key for each transmission. This idea can be implemented as follows. Alice and Bob possess a common sequence of distinct keys, called a *pad*. Each key in the pad is used for exactly one message transmission. Both parties discard a key after it has been used; so, the pad is a *one-time pad*.

It can be shown that one-time pads are unbreakable. However, the major difficulty is in sharing the pad. How can Alice and Bob agree on a pad to begin with? In ancient times it was conceivable that they could agree on a common book —say, the King James version of the Bible— and use successive strings from the book as keys. However, the need to develop different pads for each pair of communicants, and distribute the pads efficiently (i.e., electronically) and securely, makes this scheme impractical.

Many military victories, defeats and political intrigues over the entire course of human history are directly attributable to security/breakability of codes. Lively descriptions appear in a delightful book by Singh [47].

## 3.3 Public Key Cryptography

The coding schemes given in the last section were symmetric, in the sense that given the encryption mechanism it is easy to see how to decrypt a message. Thus Alice and Bob, the sender and the receiver, both share the same secret, the key. A consequence of this observation is that the key has to be transmitted before any data can be transmitted.

A novel idea is for Bob to publicly announce a function $f$ that is to be used to encrypt any plaintext to be sent to him, i.e., Alice should encrypt $x$ to $f(x)$ and then send the latter to Bob. Function $f$ is the public key of Bob. Upon receiving the message, Bob applies the inverse function $f^{-1}$, a private key, thus obtaining $f^{-1}(f(x))$, i.e., $x$. Eve knows $f$; so, theoretically, she can also compute $x$. However, $f$ is chosen so that it is *computationally intractable* to deduce $f^{-1}$ given only $f$, that is, the computation will take an extraordinarily long time before Eve can deduce $f^{-1}$. Bob, who designed $f$, also designed $f^{-1}$ simultaneously. So, he can decrypt the message.

Let us examine some of the implications of using public key cryptography. First, there is no need to exchange any key, because there are no shared secrets. There is a publicly open database in which every one posts his own public key. Any one may join or drop out of this community at any time. Alice sends a message to Bob by first reading his key, $f$, from the database, applying $f$ to the plaintext $x$ and then sending $f(x)$ to him. Eve can see who sent the message (Alice), who will receive the message (Bob), the public key of the recipient ($f$) and the ciphertext ($f(x)$). Yet, she is powerless, because it will take her eons to decode this message. Note that Alice also cannot decrypt any message sent to Bob by another party.

Function $f$ is called *one-way* because it is easy to apply —$f(x)$ can be computed easily from $x$— though hard to invert, that is, to compute $x$ from $f(x)$ without using additional information. Let me repeat that it is theoretically possible to compute $x$ given $f$ and $f(x)$; simply try all possible messages $y$ of appropriate length as candidates for $x$, compute $f(y)$, and then compare it against $f(x)$. This is not practical for any but the very shortest $x$ because of the number of possible candidates. If the function $f$ is well-chosen, Eve has no other way of decrypting the message. If the message is 64 bits long and she can check $10^{10}$ messages a second, it will still take around 58 years to check all possible messages. She could, of course, be lucky, and get a break early in the search; but, the probability of being lucky is quite low.

The notion of computational intractability was not available to the early cryptanalysts; it is a product of modern computer science. We now know that there are certain problems which, though decidable, are computationally intractable in that they take huge amounts of time to solve. Normally, this is an undesirable situation. We have, however, turned this disadvantage to an advantage by putting the burden of solving an intractable problem on Eve, the eavesdropper.

The idea of public key cryptography using one-way functions is due to Diffie and Hellman [14]. Rivest, Shamir and Adelman [44] were the first to propose a specific one-way function that has remained unbroken (or, so it is believed). In the next section, I develop the theory behind this one-way function.

## 3.3.1 Mathematical Preliminaries

### 3.3.1.1 Modular Arithmetic

Henceforth, all variables are positive integers unless stated otherwise. We write "$x \bmod n$" for the remainder of $x$ divided by $n$. Two integers $x$ and $y$ that have the same remainder after division by $n$ are said to be *congruent* mod $n$; in that case, we write

$$x \overset{\bmod n}{\equiv} y$$

That is,

$$(x \overset{\bmod n}{\equiv} y) \equiv (x \bmod n = y \bmod n)$$

So, $x \overset{\bmod n}{\equiv} y$ means $x - y$ is divisible by $n$.

Note that congruence (mod $n$) is an equivalence relation over integers. Below, we list a few properties of the congruence relation. Variables $u$, $v$, $p$, $x$ and $y$ are positive integers.

- (P1)

$$\frac{\begin{array}{rcl} u & \overset{\bmod p}{\equiv} & v, \\ x & \overset{\bmod p}{\equiv} & y \end{array}}{\begin{array}{rcl} u+x & \overset{\bmod p}{\equiv} & v+y, \\ u-x & \overset{\bmod p}{\equiv} & v-y, \\ u \times x & \overset{\bmod p}{\equiv} & v \times y \end{array}}$$

- (P2; Corollary) For all $n$, $n \geq 0$,

$$\frac{x \overset{\bmod p}{\equiv} y}{x^n \overset{\bmod p}{\equiv} y^n}$$

- (Modular Simplification Rule) Let $e$ be any expression over integers that has only addition, subtraction, multiplication and exponention as its operators. Let $e'$ be obtained from $e$ by replacing any subexpression $t$ of $e$ by ($t$ mod $p$). Then, $e \overset{\text{mod } p}{\equiv} e'$, i.e., $e \bmod p = e' \bmod p$.

  Note that an exponent is not a subexpression; so, it can't be replaced by its mod.

**Examples**

$(20 + 5) \bmod 3 = ((20 \bmod 3) + 5) \bmod 3$
$((x \times y) + g) \bmod p = (((x \bmod p) \times y) + (g \bmod p)) \bmod p$
$x^n \bmod p = (x \bmod p)^n \bmod p$
$x^{2n} \bmod p = (x^2)^n \bmod p = (x^2 \bmod p)^n \bmod p$
$x^n \bmod p = x^{n \bmod p} \bmod p$, is wrong. □

**Relatively Prime** Positive integers $x$ and $y$ are relatively prime iff $\gcd(x, y) = 1$. Since $\gcd(0, x) = x$ for positive $x$, it follows that 0 and $x$ are relatively prime iff $x = 1$. Note that $\gcd(0, 0)$ is undefined.

- (P3) For $p$ and $q$ relatively prime,
  $(u \overset{\text{mod } p}{\equiv} v \wedge u \overset{\text{mod } q}{\equiv} v) \equiv (u \overset{\text{mod } p\times q}{\equiv} v)$

**Exercise 22**

Disprove each of the following conclusions.

$$\frac{u \overset{\text{mod } p}{\equiv} v, \quad x \overset{\text{mod } p}{\equiv} y}{\max(u, x) \overset{\text{mod } p}{\equiv} \max(v, y), \quad u^x \overset{\text{mod } p}{\equiv} v^y}$$

**Solution** Use $p = 3$, $u, v = 2, 2$ and $x, y = 4, 1$. □

The following rule allows us to manipulate exponents, which we can't do using only the modular simplification rule (see the previous exercise).

- (P4; due to Fermat) $b^{p-1} \bmod p = 1$, where $p$ is prime, and $b$ and $p$ are relatively prime.

**Exercise 23**

With $b$ and $p$ as in (P4) show that for any nonnegative integer $m$

$b^m \overset{\text{mod } p}{\equiv} b^{m \bmod (p-1)}$

**Solution** Write $b^m$ as $b^{p-1} \times b^{p-1} \ldots \times b^{m \bmod (p-1)}$. Use (P4) to reduce each $b^{p-1} \bmod p$ to 1. □

#### 3.3.1.2 Extended Euclid Algorithm

We prove the following result: given nonnegative integers $x$ and $y$, where both integers are not zero, there exist integers $a$ and $b$ such that

$$a \times x + b \times y = \gcd(x, y).$$

This result is known as Bézout's lemma. For example, let $x, y = 12, 32$. Then $gcd(x, y) = 4$. And, $a, b = 3, -1$ satisfy the equation.

Note that $a$ and $b$ need not be positive, nor are they unique. In fact, verify that for any solution $(a, b)$, the pair given below is also a solution, where $k$ is any integer.

$$(a + k \times y/ \gcd(x, y), b - k \times x/ \gcd(x, y))$$

We can prove Bézout's lemma easily by applying induction on $x + y$. Later, we show how to extend Euclid's algorithm to compute $(a, b)$, which also constitutes a constructive proof of the lemma.

- $x = 0$ and $y > 0$: Then $\gcd(x, y) = y$. We have to display $a$ and $b$ such that

$$a \times x + b \times y = y$$

Setting $a, b = 0, 1$ satisfies the equation. Similarly, if $y = 0$ and $x > 0$, set $a, b = 1, 0$.

- $x > 0$ and $y > 0$: Without loss in generality, assume that $x \geq y$. Since $x - y + y < x + y$, applying induction, there exist $a'$ and $b'$ such that

$$a' \times (x - y) + b' \times y = \gcd(x - y, y)$$

Note that $\gcd(x, y) = \gcd(x - y, y)$. Therefore,

$$a' \times (x - y) + b' \times y = \gcd(x, y)$$

$$a' \times x + (b' - a') \times y = \gcd(x, y)$$

Set $a, b = a', (b' - a')$ to prove the result.

Next, consider the classical Euclid algorithm for computing gcd. We will modify this algorithm to compute $a$ and $b$ as well.

```
$u, v := x, y$
$\{u \geq 0,\ v \geq 0,\ u \neq 0 \ \vee\ v \neq 0,\ \gcd(x, y) = \gcd(u, v)\}$
**while** $v \neq 0$ **do**
  $u, v := v, u \bmod v$
**od**
$\{\gcd(x, y) = \gcd(u, v),\ v = 0\}$
$\{\gcd(x, y) = u\}$
```

One way of computing $u \bmod v$ is to explicitly compute the quotient $q$, $q = \lfloor u/v \rfloor$, and subtract $v \times q$ from $u$. Thus, $u, v := v, u \bmod v$ is replaced by

```
$q := \lfloor u/v \rfloor$;
$u, v := v, u - v \times q$
```

To compute $a$ and $b$ as required, we augment this program by introducing variables $a, b$ and another pair of variables $c, d$, which satisfy the invariant

$$(a \times x + b \times y = u) \ \wedge \ (c \times x + d \times y = v)$$

An outline of the program is shown below.

```
$u, v := x, y$; $a, b := 1, 0$; $c, d := 0, 1$;
 **while** $v \neq 0$ **do**
   $q := \lfloor u/v \rfloor$;
   $\alpha$ :  $\{(a \times x + b \times y = u) \ \wedge \ (c \times x + d \times y = v)\}$
   $u, v := v, u - v \times q$;
   $a, b, c, d := a', b', c', d'$
   $\beta$ :  $\{(a \times x + b \times y = u) \ \wedge \ (c \times x + d \times y = v)\}$
 **od**
```

The remaining task is to calculate $a', b', c', d'$ so that the given annotations are correct, i.e., the invariant $(a \times x + b \times y = u) \ \wedge \ (c \times x + d \times y = v)$ holds at program point $\beta$. Using backward substitution, we need to show that the following proposition holds at program point $\alpha$.

$$(a' \times x + b' \times y = v) \ \wedge \ (c' \times x + d' \times y = u - v \times q)$$

We are given that the proposition $(a \times x + b \times y = u) \ \wedge \ (c \times x + d \times y = v)$ holds at $\alpha$. Therefore, we may set

$$a', b' = c, d$$

Now, we compute $c'$ and $d'$.

$$\begin{array}{ll} & c' \times x + d' \times y \\ = & \{\text{from the invariant}\} \\ & u - v \times q \\ = & \{a \times x + b \times y = u \text{ and } c \times x + d \times y = v\} \\ & (a \times x + b \times y) - (c \times x + d \times y) \times q \\ = & \{\text{algebra}\} \\ & (a - c \times q) \times x + (b - d \times q) \times y \end{array}$$

So, we may set

$$c', d' = a - c \times q, b - d \times q$$

The complete algorithm is:

| $a$ | $b$ | $u$ | $c$ | $d$ | $v$ | $q$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 17 | 0 | 1 | 2668 | |
| | | | | | | 0 |
| 0 | 1 | 2668 | 1 | 0 | 17 | |
| | | | | | | 156 |
| 1 | 0 | 17 | −156 | 1 | 16 | |
| | | | | | | 1 |
| −156 | 1 | 16 | 157 | −1 | 1 | |
| | | | | | | 16 |
| 157 | −1 | 1 | -2668 | 17 | 0 | |

Table 3.4: Computation with the extended Euclid algorithm

```
$u, v := x, y$; $a, b := 1, 0$; $c, d := 0, 1$;
 while $v \neq 0$ do
   $q := \lfloor u/v \rfloor$;
   $\alpha :\ \{(a \times x + b \times y = u)\ \wedge\ (c \times x + d \times y = v)\}$
   $u, v := v, u - v \times q$;
   $a, b, c, d := c, d, a - c \times q, b - d \times q$
   $\beta :\ \{(a \times x + b \times y = u)\ \wedge\ (c \times x + d \times y = v)\}$
 od
```

At the termination of the algorithm,

$$\begin{array}{ll} & a \times x + b \times y \\ = & \{\text{from the invariant}\} \\ & u \\ = & \{u = \gcd(x, y), \text{ from the annotation of the first program in page 58}\} \\ & \gcd(x, y) \end{array}$$

**Example** Table 3.4 shows the steps of the algorithm for $x, y = 17, 2668$. □

**Exercise 24**
Show that the algorithm terminates, and that $\alpha : \{(a \times x + b \times y = u)\ \wedge\ (c \times x + d \times y = v)\}$ is a loop invariant. Use annotations shown in the program. □

## 3.3.2 The RSA Scheme

A principal, Bob, joins the cryptosystem as follows.

- Choose two large primes $p$ and $q$, $p \neq q$. (There are efficient probabilistic schemes for computing $p$ and $q$).
- Let $n = p \times q$. And, define $\phi(n) = (p - 1) \times (q - 1)$. Any message below $n$ in value can be encrypted.

- Choose integers $d$ and $e$ ($d$ is for decryption and $e$ for encryption) such that

  1. $1 \leq d < n$, $1 \leq e < n$,
  2. both $d$ and $e$ are relatively prime to $\phi(n)$, and
  3. $d \times e \overset{\bmod \phi(n)}{\equiv} 1$.

  It is customary to choose $e$, the encryption key, to be a small value (like 35) which is below $n$ and relatively prime to $\phi(n)$. Once $e$ is chosen, $d$ is uniquely determined, as follows.

  Computation of $d$ is based on the Extended Euclid algorithm of page 58. Set $x := e$ and $y := \phi(n)$ in the formula $a \times x + b \times y = \gcd(x, y)$ to get:

$$\begin{array}{ll} & a \times e + b \times \phi(n) = \gcd(e, \phi(n)) \\ \Rightarrow & \{e \text{ and } \phi(n) \text{ are relatively prime, from the choice of } e\} \\ & a \times e + b \times \phi(n) = 1 \\ \Rightarrow & \{\text{definition of mod}\} \\ & a \times e \overset{\bmod \phi(n)}{\equiv} 1 \end{array}$$

  Now, let $d$ be $a$. If $d$ is positive but higher than $n$, subtract $\phi(n)$ from it enough times to satisfy $1 \leq d < n$. If $d$ is negative (it can't be zero) add $\phi(n)$ to it enough times to make it positive. In either case, $d = a + k \times \phi(n)$, for some $k$. Given that $a \times e \overset{\bmod \phi(n)}{\equiv} 1$, we have $(a + k \times \phi(n)) \times e \overset{\bmod \phi(n)}{\equiv} 1$, for any $k$; therefore, $d \times e \overset{\bmod \phi(n)}{\equiv} 1$.

  We next show that $d$ is relatively prime to $\phi(n)$. Let $r$ be a common divisor of $d$ and $\phi(n)$. We show that $|r| = 1$. From $d \times e \overset{\bmod \phi(n)}{\equiv} 1$, we get $d \times e = k \times \phi(n) + 1$, for some $k$. Let $r|d$ denote that $r$ divides $d$.

$$\begin{array}{ll} & r|d \\ \Rightarrow & \{\text{arithmetic}\} \\ & r|(d \times e) \\ \Rightarrow & \{d \times e = k \times \phi(n) + 1\} \\ & r|(k \times \phi(n) + 1) \\ \Rightarrow & \{\text{since } r \text{ is a divisor of } \phi(n),\ r|(k \times \phi(n)\} \\ & r|1 \\ \Rightarrow & \{\text{arithmetic}\} \\ & |r| = 1 \end{array}$$

  Therefore, $d$ is relatively prime to $\phi(n)$.

- At this stage, Bob has the following variables:

1. $p$ and $q$, which are primes,
2. $n$, which is $p \times q$, and $\phi(n)$, which is $(p-1) \times (q-1)$,
3. $d$ and $e$ which satisfy
   (a) $1 \leq d < n$, $1 \leq e < n$,
   (b) both $d$ and $e$ are relatively prime to $\phi(n)$, and
   (c) $d \times e \overset{\text{mod } \phi(n)}{\equiv} 1$.

- Publicize $(e, n)$ as the public key. Save $(d, n)$ as the private key.

**Note** Observe that the specifications of $d$ and $e$ are symmetric.

**Example** Let $p = 47$ and $q = 59$. Then, $n = p \times q = 2773$, and $\phi(2773) = (47-1) \times (59-1) = 2668$. Let $e = 17$. Now, $d$ is computed as shown in Table 3.4 of page 60. Thus, $d = 157$. Verify that $157 \times 17 \overset{\text{mod } \phi(n)}{\equiv} 1$. □

#### 3.3.2.1 Encryption

To send message $M$ to a principal whose public key is $(e, n)$ and $0 \leq M < n$, send $M'$ where $M' = (M^e \bmod n)$.

**Example; contd.** Let us represent each letter of the alphabet by two digits, with white space = 00 a = 01 b = 02, etc.

Suppose the message to be sent is "bad day". The representation yields: 02010400040125.

Since $n$ is 2773, we can convert any pair of letters to a value below $n$, the largest such pair being $zz$ which is encoded as 2626. Therefore, our block length is 2 letters. We get the following blocks from the encoded message: 0201 0400 0401 2500; we have appended an extra blank at the end of the last block to make all blocks have equal size.

Now for encryption of each block. We use the parameters from the previous example, where $e = 17$. For the first block, we have to compute $0201^{17}$ mod 2773, for the second $0400^{17}$ mod 2773, etc. □

There is an efficient way to raise a number to a given exponent. To compute $M^{17}$, we need not multiply $M$ with itself 16 times. Instead, we see that $M^{17} = M^{16} \times M = (M^8)^2 \times M = ((M^4)^2)^2 \times M = (((M^2)^2)^2)^2 \times M$. The multiplication strategy depends on the binary representation of the exponent. Also, at each stage, we may apply $\bmod n$, so that the result is always less than $n$. Specifically,

$$M^{2t} \bmod n = (M^t \bmod n)^2 \bmod n$$
$$M^{2t+1} \bmod n = ((M^t \bmod n)^2 \times M) \bmod n$$

The following algorithm implements this strategy. Let $e$, the exponent, in binary be: $e_k\ e_{k-1} \ldots e_0$. For $e = 17$, we get 10001. Next, use the following algorithm that looks at the bits of $e$ from the higher to the lower order; the result of encryption is in $C$. The loop invariant is: $C = M^h \bmod n$, where $h$ is the portion of the exponent seen so far, i.e., $e_k\ e_{k-1} \ldots e_i$ (initially, $h = 0$).

```
$C := 1$;
 **for** $i = k..0$ **do**
    **if** $e_i = 0$
       **then** $C := C^2 \bmod n$
       **else** $C := ((C^2 \bmod n) * M) \bmod n$
    **fi**
 **od**
```

We encrypt 0201 to 2710 and 0400 to 0017.

**Exercise 25**

The algorithm to compute $M^e \bmod n$, given earlier, scans the binary representation of $e$ from left to right. It is often easier to scan the representation from right to left, because we can check if $e$ is even or odd easily on a computer. We use:

$$M^{2t} = (M^2)^t$$
$$M^{2t+1} = (M^2)^t \times M$$

Here is an algorithm to compute $M^e$ in $C$. Prove its correctness using the loop invariant $C * m^h = M^e$. Also, modify the algorithm to compute $M^e \bmod n$.

```
$C := 1$; $h, m := e, M$;
 **while** $h \neq 0$ **do**
    **if** $odd(h)$ **then** $C := C * m$ **fi** ;
    $h := h \div 2$;
    $m := m^2$
 **od**
$\{C = M^e\}$
```

### 3.3.2.2 Decryption

On receiving an encrypted message $M'$, $0 \leq M' < n$, Bob, whose private key is $(d, n)$, computes $M''$ as follows.

$$M'' = (M'^d \bmod n)$$

We show below that $M'' = M$.

**Example** We continue with the previous example. The encryption and decryption steps are identical, except for different exponents. We use the encryption algorithm with exponent 157 to decrypt. The encryption of 0201 is 2710 and of 0400 is 0017. Computing $2710^{157} \bmod 2773$ and $0017^{157} \bmod 2773$ yield the original blocks, 0201 and 0400. □

**Lemma 1:** For any $M$, $0 \leq M < n$, $M^{d \times e} \overset{\bmod p}{\equiv} M$.

Proof:

$$
\begin{array}{ll}
 & M^{d\times e} \bmod p \\
= & \{d\times e \overset{\bmod \phi(n)}{\equiv} 1, \text{ and } \phi(n) = (p-1)\times(q-1)\} \\
 & M^{t\times(p-1)+1} \bmod p, \text{ for some } t \\
= & \{\text{rewriting}\} \\
 & ((M^{(p-1)})^t \times M) \bmod p \\
= & \{\text{modular simplification: replace } (M^{(p-1)}) \text{ by } (M^{(p-1)}) \bmod p\} \\
 & ((M^{(p-1)} \bmod p)^t \times M) \bmod p \\
= & \{\text{Consider two cases:} \\
 & \quad \bullet\ M \text{ and } p \text{ are not relatively prime:} \\
 & \quad\ \ \text{Since } p \text{ is prime, } M \text{ is a multiple of } p\text{, i.e.,} \\
 & \quad\ \ (M \bmod p) = 0. \text{ So, } M^{(p-1)} \bmod p = 0. \\
 & \quad\ \ \text{The entire expression is 0, thus equal to } M \bmod p. \\
 & \quad \bullet\ M \text{ and } p \text{ are relatively prime:} \\
 & \quad\ \ \text{Then, } (M^{(p-1)}) \bmod p = 1, \text{ from (P4).} \\
 & \quad\ \ \text{The expression is } (1^t \times M) \bmod p = M \bmod p. \\
 & \} \\
 & M \bmod p
\end{array}
$$

**Lemma 2:** For any $M$, $0 \le M < n$, $(M^{d\times e} \bmod n) = M$.
Proof:

$$
\begin{array}{lll}
M \overset{\bmod p}{\equiv} M^{d\times e} & & \text{, from Lemma 1} \\
M \overset{\bmod q}{\equiv} M^{d\times e} & & \text{, replacing } p \text{ by } q \text{ in Lemma 1} \\
M \overset{\bmod n}{\equiv} M^{d\times e} & & \text{, from above two, using P3 and } n = p\times q \\
(M \bmod n) = (M^{d\times e} \bmod n) & & \text{, from above} \\
M = (M^{d\times e} \bmod n) & & \text{, } M < n\text{; so } M \bmod n = M
\end{array}
$$

We are now ready to prove the main theorem, that encryption followed by decryption yields the original message.

**Theorem:** $M'' = M$.

$$
\begin{array}{ll}
 & M \\
= & \{\text{Lemma 2}\} \\
 & M^{d\times e} \bmod n \\
= & \{\text{arithmetic}\} \\
 & (M^e)^d \bmod n \\
= & \{\text{modular simplification rule}\} \\
 & (M^e \bmod n)^d \bmod n \\
= & \{\text{from the encryption step, } M' = M^e \bmod n\} \\
 & M'^d \bmod n \\
= & \{\text{from the decryption step, } M'' = M'^d \bmod n\} \\
 & M''
\end{array}
$$

#### 3.3.2.3 Breaking RSA is hard, probably

The question of breaking RSA amounts to extracting the plaintext from the ciphertext. Though there is no proof for it, it is strongly believed that in order to break RSA, you will have to compute the private key given only the public key. That is, given $(e, n)$, find $d$ where $d \times e \overset{\bmod \phi(n)}{\equiv} 1$. We show that computing $d$ is as hard as factoring $n$.

It is strongly believed that factoring a large number is intractable[1]. In the naive approach to factoring, we have to test all numbers at least up to $\sqrt{n}$ to find a factor of $n$. If $n$ is a 200 digit number, say, approximately $10^{100}$ computation steps are needed. The best known algorithm for factoring a 200 digit number would take about a million years. We can speed up matters by employing supercomputers and a whole bunch of them to work in parallel. Yet, it is unlikely that factoring would be done fast enough to justify the investment. So, it is strongly believed —though not proven— that RSA is unbreakable.

Next, we show that computing $d$ is easy given $e$ and the factors of $n$. Suppose we have factored $n$ into primes $p$ and $q$. Then, we can compute $\phi(n)$, which is $(p-1) \times (q-1)$. Next, we can compute $d$ as outlined earlier.

The proof in the other direction, —if we have $d$ and $e$ where $d \times e \overset{\bmod \phi(n)}{\equiv} 1$, then we can factor $n$— is more technical. It can be shown that $n$ is easily factored given any multiple of $\phi(n)$, and $(d \times e) - 1$ is a multiple of $\phi(n)$.

An easier result is that $n$ can be factored if $\phi(n)$ is known. Recall that $\phi(n) = (p-1) \times (q-1)$ and $n = p \times q$. Hence, $\phi(n) = n - (p+q) + 1$. From $n$ and $\phi(n)$, we get $p \times q$ and $p+q$. Observe that $p - q = \sqrt{(p-q)^2} = \sqrt{(p+q)^2 - 4 \times p \times q}$. Therefore, $p - q$ can be computed. Then, $p = \frac{(p+q)+(p-q)}{2}$ and $q = \frac{(p+q)-(p-q)}{2}$.

**Exercise 26**
Why is it necessary to choose *distinct* primes for $p$ and $q$?

## 3.4 Digital Signatures and Related Topics

Public key cryptography neatly solves a related problem, affixing a digital signature to a document. Suppose Bob receives a message from someone claiming to be Alice; how can he be sure that Alice sent the message? To satisfy Bob, Alice affixes her signature to the message, as described below.

Alice encrypts the message using her *private key*; this is now a signed message. More formally, let $x$ be the message, $f_a$ and $f_b$ the public keys of Alice and Bob, and $f_a^{-1}$ and $f_b^{-1}$ be their private keys, respectively. Then $f_a^{-1}(x)$ is the message signed by Alice. If the message is intended for Bob's eyes only, she encrypts the signed message with Bob's public key, and sends $f_b(f_a^{-1}(x))$. Bob first decrypts the message using his own private key, then decrypts the signed message using Alice's public key. Alice may also include her name in plaintext,

[1]Peter Shor has developed an algorithm for factoring that runs in $O(\log n)^3$ time. Unfortunately (or, fortunately for cryptopgraphy), the algorithm can run only on a quantum computer.

$f_b(\text{“alice”} \mathbin{+\!\!+} f_a^{-1}(x))$ where $\mathbin{+\!\!+}$ is concatenation, so that Bob will know whose public key he should apply to decrypt the signed message.

We show that such signatures satisfy two desirable properties. First, decrypting any message with the Alice's public key will result in gibberish unless it has been encrypted with her private key. So, if Bob is able to get a meaningful message by decryption, he is convinced that Alice sent the message.

Second, Alice cannot deny sending the message, because no one else has access to her private key. An impartial judge can determine that Alice's signature appears on the document (message) by decrypting it with her public key. Note that the judge does not need access to any private information.

Note that no one can modify this message while keeping Alice's signature affixed to it. Thus, no electronic cutting and pasting of the message/signature is possible.

Observe a very important property of the RSA scheme: any message can be encrypted by the public or the private key and decrypted by its inverse.

Digital signatures are now accepted for electronic documents. A user can sign a check, or a contract, or even a document that has been signed by other parties.

**Another look at one-time pads** We know that one-time pads provide complete security. The only difficulty with them is that both parties to a transmission must have access to the same pad. We can overcome this difficulty using RSA, as follows.

Regard each one-time pad as a random number. Both parties to a transmission have access to a pseudo-random number generator which produces a stream of random numbers. The pseudo-random number generator is public knowledge, but the seed which the two parties use is a shared secret. Since they use the same seed, they will create the same stream of random numbers. Then the encryption can be relatively simple, like taking exclusive or.

This scheme has one drawback, having the seed as a shared secret. RSA does not have this limitation. We can use RSA to establish such a secret: Bob generates a random number as the seed and sends it to Alice, encrypted by Alice's public key. Then, both Bob and Alice know the seed.

**Security of communication with a trusted third party** We have so far assumed that all public keys are stored in a public database and Alice can query the database manager, David, to get Bob's public key (in plaintext). Suppose Eve intercepts the message sent by David to Alice, and replaces the public key of Bob by her own. Then Alice encrypts a message for Bob by Eve's public key. Any message she sends to Bob could be intercepted and decrypted by Eve.

The problem arises because Alice does not know that the message received from David is not authentic. To establish authenticity, David —often called a *trusted third party*— could sign the message. Then, Eve cannot do the substitution as described above.

Trusted third parties play a major role in security protocols, and authenticating such a party is almost always handled by digital signatures.

**Oblivious Transfer** We have studied the oblivious transfer problem in Section 2.4. The problem is as follows (repeated from that section). Alice has two pieces of data $m_0$ and $m_1$. Bob requests one of these data from Alice. The restriction is that Alice should not know which data has been requested (so, she has to send both data in some encoded form) and Bob should be able to extract the data he has requested, but know nothing about the data he has not requested.

We solved this problem in Section 2.4 using a trusted third party, and there was no need for encryption. Here is another solution without using a third party, but one that relies on encryption. Let $E$ and $D$ be the encryption and decryption functions for Alice. We will not need the corresponding functions for Bob, though Alice may want to encrypt her transmissions to avoid eavesdropping.

First, Alice and Bob agree on two random pieces of data, $x_0$ and $x_1$ (Alice could create them and send to Bob). Suppose Bob needs data $m_b$, for $b$ either 0 or 1. Then, Bob creates a random $y$ and sends $p = E(y) \oplus x_b$ to Alice. Alice computes $q_i = D(p \oplus x_i)$, and sends the pair $(r_0, r_1) = (m_0 \oplus q_0, m_1 \oplus q_1)$. We assert that $q_b = y$, because $q_b = D(p \oplus x_b) = D(E(y) \oplus x_b \oplus x_b) = D(E(y)) = y$. Therefore, $r_b = m_b \oplus q_b = m_b \oplus y$, or $m_b = r_b \oplus y$. Bob knows $y$; so, he can compute $m_b$ from the received value $r_b$.

We argue that Alice does not know $b$, the intended data for Bob. This is because Alice receives only $p = E(y) \oplus x_b$, and since $y$ is random, she has no way of telling if $x_0$ or $x_1$ is used in computing $p$. And, Bob does not know $m_0$ if his intention was only to receive $m_1$. This is because he knows only $x$'s and $r$'s, and he can not compute $q_0$ (which is needed to extract $m_0$ from $r_0$).

**Blind Signature** Alice wants Bob to sign a document without revealing its contents to Bob. This is useful if Bob is a notary, so that Bob does not need to know the contents of the document, but merely verify the signature.

Suppose Alice has a document $M$. She would like to have $M^d \bmod n$, where $d$ is Bob's decryption key and $n$ is as described in Section 3.3.2 (page 60). Alice sends $M \times k^e$ to Bob, where $e$ is Bob's encryption key and $k$ is some random number, $1 \leq k < n$. Bob signs the document with his decryption key $d$ and returns $(M \times k^e)^d \bmod n$ to Alice. Now,

$$\begin{array}{ll} & (M \times k^e)^d \bmod n \\ = & \{\text{arithmetic}\} \\ & (M^d \times k^{e \times d}) \bmod n \\ = & \{\text{arithmetic}\} \\ & (M^d \times (k^{e \times d} \bmod n)) \bmod n \\ = & \{\text{From Lemma 2, page 64, } (k^{e \times d} \bmod n = k\} \\ & (M^d \times k) \bmod n \end{array}$$

Alice divides the signed message by $k$, that is, multiplies it by $k^{-1} \bmod n$ to retrieve $M^d \bmod n$.

## 3.5 Block Cipher

The RSA scheme is expensive in computation time, particularly if you want to encrypt large amounts of data, as in video or music encryption. In such cases, we use either *block* or *stream* cipher. In each case, we start with a secret key (this section is mostly about how to form a secret key). For block cipher, data is broken up into fixed length blocks, typically around 128 bits. Each block is individually encrypted and decrypted using the secret key. There are two functions, one for encryption and the other for decryption. Each function takes two arguments, a block and the secret key. The encryption function computes a new block and the decryption function converts it back to the original block.

One of the early block cipher schemes was DES (Data Encryption Standard), adopted as a standard in 1977. It used 64 bit blocks which were deemed insufficiently secure. A newer version, called AES (Advanced Encryption Standard), was adopted in 2001 which operates on 128 bit blocks. Both algorithms operate nearly identically for both encryption and decryption, and have fast hardware implementations. Each algorithm runs for 16 rounds, where each round permutes a block in some way using the secret key.

Stream ciphers operate on a stream of data, such as for video delivery. Encryption has to be extremely fast and online; that is block $b$ is converted to block $b'$ without much processing. Typically, both encryption and decryption have access to a very long random bit string $s$ (typically millions of bits). Encryption forms $b'$ by taking exclusive-or of parts of $s$ with $b$. Each block consumes some part of $s$, so, different blocks are encrypted using different bit strings. Decryption follows the same procedure. The random bit string $s$ is computed by a pseudo-random number generator using a secret key as a seed.

Block ciphers are *symmetric* key ciphers, in contrast with RSA which is asymmetric (the public and private keys are different). Symmetric keys are always hidden except from those who need to know. Before invention of public key cryptography, only symmetric keys were used. The main problem with symmetric keys is that any change in the key has to be communicated to all the parties. At one time, armored trucks were used to deliver the keys. Today, we use a key exchange protocol to create a secret key. One of the first, and still very important, schemes is known as Diffie-Hellman Key Exchange (or Diffie-Hellman-Merkle Key Exchange), which we study next.

**A Simplisic Version of Diffie-Hellman Key Exchange** The scheme described here permits two parties, Alice and Bob, to *form* a secret key for further communication. First, we give a very simple version, which has obvious shortcomings; we add more details later to overcome these problems. One secret key is used for a single session of communication.

First, both Alice and Bob settle upon a known key $g$; this key can be sent in plain text by one of them to the other. Next, Alice creates a secret key $a$ and Bob a secret key $b$. Alice sends $g^a$ to Bob and Bob sends $g^b$ to Alice, both in plain text. Alice uses the received value $g^b$ and her own secret key $a$ to compute $(g^b)^a = g^{ab}$. Similarly, Bob computes $(g^a)^b = g^{ab}$. They use $g^{ab}$ as the secret key.

Eve, the eavesdropper, knows $g$, $g^a$ and $g^b$, because these values are sent in plain text. She can easily compute $a$ and $b$ (by using a slight variation of binary search) and then compute $g^{ab}$. We will modify the protocol to make it very hard for her to compute $a$ or $b$.

The attractive part of this solution is lack of long-term commitment. Alice and Bob can set up a new secret key whenever they want to; therefore, they need use the secret for only one session.

Observe that RSA can also be used to exchange a secret key. Alice sends a public key to Bob which Bob uses to send her a secret. Using RSA, only Alice can decrypt it, and Bob already knows the secret. They can then employ the secret for further communications.

**Precise Description of Diffie-Hellman Key Exchange** Bob and Alice publicly settle on two positive integers $g$ and $p$. Here, $p$ is a prime number; typically $p$ is large (a 300 bit number will do fine for most applications). And, $g$ is smaller than $p$ and a primitive $(p-1)^{th}$ root of 1. This means $g^k \bmod p \neq 1$, for any $k$, $k < p-1$ (and from Fermat's Theorem, P4 in Page 57, $g^{p-1} \bmod p = 1$). For $p = 7$, there are two primitive roots, 3 and 5; for $p = 11$, there are four primitive roots, 2, 6, 7 and 8.

Alice and Bob privately choose integers $a$ and $b$ which are less than $p$; typically $a$ and $b$ are large numbers, so that Eve can not possibly exhaustively test for them. Alice sends $g^a \bmod p$ and Bob sends $g^b \bmod p$. Then Alice computes $(g^b \bmod p)^a \bmod p$. This quantity is $g^{ab} \bmod p$ from, $g^{ab} \bmod p = \{\text{arithmetic}\}\ (g^b)^a \bmod p = \{\text{modular simplification rule}\}\ (g^b \bmod p)^a \bmod p$. Similarly, Bob computes $(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$. They both have $g^{ab} \bmod p$, and they settle on that as the secret key.

As an example, let $g = 3$ and $p = 17$. Verify that $g$ is a $16^{th}$ primitive root of 1. Let Alice choose $a = 5$ and Bob choose $b = 9$. Then, Alice and Bob send to each other $3^5 \bmod 17 = 5$ and $3^9 \bmod 17 = 14$, respectively. Next, Alice computes $14^5 \bmod 17$ and Bob computes $5^9 \bmod 17$, both of which are 12. Each of them now knows 12 as the secret key.

**Choosing $p$ and $g$** We have to choose a large prime number $p$ and a positive integer $g$ which is a $(p-1)^{th}$ root of 1. From the description of the protocol, we don't really need that $g$ be a $(p-1)^{th}$ root of 1; all we need is that $g^k \bmod p \neq 1$, for many values of $k$, $k < p-1$. With this relaxed requirement, we choose $p$ to be a prime of the form $2 \times q + 1$, where $q$ is a prime. Such primes are known as *safe* or *Sophie-Germain* primes. The first few safe primes are: 5, 7, 11, 23, 47, 59, 83, 107, 167, 179, 227, 263.

Given $p = 2 \times q + 1$, from Fermat's Theorem (P4 in Page 57), $g^{2 \times q} \bmod p = 1$ for any $g$. That is, either $g^2 \bmod p = 1$ or $g^q \bmod p = 1$; further, $g^k \bmod p \neq 1$, for all other values of $k$, $1 \leq k \leq q$. We look for a $g$ such that $g^2 \bmod p \neq 1$; then $g^k \bmod p \neq 1$, for all $k$, $1 \leq k < q$. We find such a $g$ by a dumb search, simply looking at $g = 2, 3, 4, \cdots$ until we find one such that $g^2 \bmod p \neq 1$. Usually, this is a very short computation.

**Discrete Logarithm Problem** The task of Eve is to compute $a$ or $b$ given $g$, $p$, $g^a \bmod p$ and $g^b \bmod p$. This is known as the *Discrete Logarithm Problem*, for which there is no known efficient algorithm. And, most computer scientists believe that this is a hard problem algorithmically.

Observe that if Alice could compute Bob's secret $b$ efficiently, then so can Eve. To see this, note that Alice knows $a$, $g$, $p$ and $g^b \bmod p$, from which she can compute $b$. Clearly, $a$ is irrelevant in this computation. Therefore, Eve can similarly compute $b$ from $e$, $g$, $p$ and $g^b \bmod p$, where $e$ is any arbitrary positive integer less than $p$.

**Woman in the Middle attack** The given protocol fails if Eve intercepts every communication. She pretends to be Alice to Bob and Bob to Alice. She substitutes her own keys, $e$ and $e'$, in place of $a$ and $b$. Thus, she establishes a secret $s$ with Bob and another secret $s'$ with Alice, thus decoding all communication between them.

There is no easy way to handle this problem. Each communication has to be authenticated, i.e., Alice must be sure that any message purported to be from Bob is actually from Bob (and similarly for Bob), at least for the duration of the Diffie-Hellman exchange protocol. This can be accomplished by Alice and Bob signing every message they send (using their private keys).

**Acknowledgement** I am thankful to Steve Li for simplifying one of the proofs.