Elements Of Cryptography

The discussion of computer security issues and threats in the previous chapters makes it clear that cryptography provides a solution to many security problems. Without cryptography, the main task of a hacker would be to break into a computer, locate sensitive data, and copy it. Alternatively, the hacker may intercept data sent between computers, analyze it, and help himself to any important or useful "nuggets." Encrypting sensitive data complicates these tasks, because in addition to obtaining the data, the wrongdoer also has to decrypt it. Cryptography is therefore a very useful tool in the hands of security workers, but is not a panacea. Even the strongest cryptographic methods cannot prevent a virus from damaging data or deleting files. Similarly, DoS attacks are possible even in environments where all data is encrypted.

Because of the importance of cryptography, this chapter provides an introduction to the principles and concepts behind the many encryption algorithms used by modern cryptography. More historical and background material, descriptions of algorithms, and examples, can be found in [Salomon 03] and in the many other texts on cryptography, code breaking, and data hiding that are currently available in libraries, bookstores, and the Internet.

Cryptography is the art and science of making data impossible to read. The task of the various encryption methods is to start with plain, readable data (the *plaintext*) and scramble it so it becomes an unreadable *ciphertext*. Each encryption method must also specify how the ciphertext can be decrypted back into the plaintext it came from, and Figure 1 illustrates the relation between plaintext, ciphertext, encryption, and decryption.

Thus, cryptography hides or obscures the meaning of data, but does not hide the data itself. Hiding data is also a useful computer security technique. A small data file (the payload) can be hidden inside a larger file (the cover), such that an examination of the cover will not uncover the data and will not raise any suspicion.

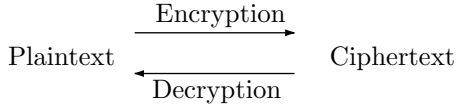

Figure 1: Encryption and Decryption.

1 Principles of Cryptography

First, a simple classification. The field of cryptography is huge and covers many methods and approaches. At the most basic level, these methods can be classified into codes and ciphers.

A code is a short symbol or word that replaces an entire message. Codes are secure but are not general purpose. Before a spy is sent to a foreign country he and his runner may agree on a set of codes. The words *happy* and *sad* used by the spy in otherwise-innocuous sentences may indicate good and bad economies in the foreign country, whereas *deep* and *shallow* may be codes for success and failure of the spy's mission. It is easy to see why the use of codes is limited, but it is also true that codes can be broken. If the same spy sends many messages that use the same codes, then clever codebreakers who intercept the messages may eventually guess the meaning of certain codes, and then test their guesses by applying them to future messages to see if the guesses make sense.

A cipher is a rule that tells how to scramble (encrypt) data in a nonrandom way, so it can later be unscrambled (decrypted). Perhaps the simplest example of a cipher is to replace each letter with the one following it (cyclically) n positions in the alphabet. This is the well-known *Caesar cipher*. Here is an example for $n = 3$ (note how `X` is replaced by `A`).

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
DEFGHIJKLMNOPQRSTUVWXYZABC
```

The top line is the *plain alphabet* and the bottom line is the *cipher alphabet*. The plaintext COME BACK is encrypted by this method to the unreadable ciphertext FRPH EDFN. This simple cipher illustrates several important facts about ciphers as follows:

- Decrypting this cipher requires knowledge of n (3 in our example). Thus, the number n is the *key* of this cipher. Long experience with ciphers has convinced cryptographers that the security provided by a cipher depends on the key, and not on the encryption method. The reason for this is easy to understand. When two persons want to communicate privately, they can develop an encryption method without a key and hope that no one will break it, but when a bank needs encryption for its sensitive operations, it cannot keep its encryption algorithm secret. Many people work for a bank, they come and go and it is inevitable that the details of such a secret algorithm will leak out. It makes more sense for a bank (and for other entities such as government agencies, army units, and spies) to use a well-tested, key-based commercial encryption program and base their security on the key.

- There are 26 letters in the English alphabet, so n can take the values 1 through 25 (the keys 0 and 26 produce ciphertext that's identical to the plaintext). Thus, the *keyspace* of the Caesar cipher is very small, which makes it easy to break this cipher by trying all 25 keys. A practical, useful cipher must have a very large keyspace. Current cryptographic algorithms are executed on computers and are therefore based on binary numbers. Keys typically vary in size from 64 to 256 bits (see exercise 1).

- In the Caesar cipher, a plainletter is always encrypted to the same cipherletter. Such a cipher is called *monoalphabetic* and it is easy to break because the ciphertext reflects the statistical properties of the plaintext. In English, for example, the most common letter is `E`. A plaintext encrypted with a monoalphabetic cipher that replaces `E` with `K`, will produce ciphertext with `K` as its most common letter. Section 3 describes a simple monoalphabetic cipher designed in antiquity by Polybius. A better approach is offered by a *polyalphabetic* encryption method, where the same plainletter is encrypted to different cipherletters. However, modern ciphers are based on binary numbers, not on letters, so they use different approaches.

2 Kerckhoffs's Principle

The entire field of cryptography is based on an important assumption, namely, that some information can be kept and disseminated securely, accessible only to authorized persons. This information is the *key* used by an encryption algorithm.

An important principle in cryptography, due to the Dutch linguist Auguste Kerckhoffs von Nieuwenhoff [Kerckhoffs 83], states that the security of an encrypted message must depend on keeping the key secret. It should not depend on keeping the encryption algorithm secret. This principle is widely accepted and implies that there must be many possible keys to an algorithm; the *keyspace* must be very large. The Caesar algorithm, for example, is very weak because its keyspace is so small. Notice that a large keyspace is a necessary but not a sufficient condition of security. An encryption algorithm may have an immense keyspace but may nevertheless be weak.

Kerckhoffs's Principle
One should assume that the method used to encipher data is known to the opponent, and that security must lie in the choice of key. This does not necessarily imply that the method should be public, just that it is considered public during its creation. —Auguste Kerckhoffs

It is possible to use a brute force approach to break a cipher. A would-be codebreaker can simply search the entire keyspace—every possible key! There are, however, two good reasons why this approach is impractical and has at best a limited value. One reason is the large number of keys, and the other is the problem of recognizing the correct plaintext once it has been

obtained when the right key is tried. Table 2 lists the times it takes to check all the keys, for several key sizes n. The table lists the times for the cases where one mega and one giga keys are checked each second (a mega is 2^{20}, about a million, and a giga is 2^{30}, about a billion). It is clear that doubling the key size more than doubles the total number of keys. In fact, for an n-bit key, the number of keys is 2^n, so it grows exponentially with n. Recognizing the plaintext is not easier. If the plaintext is text, it may be recognized by a computer program by looking up words in a dictionary. Even this can be defeated by artificially inserting many nonsense "words" in the text, but the plaintext may be compressed data, or executable machine code. These types of data are random or close to random and may be virtually impossible to recognize. The plaintext may also be image, video, or audio data, and these types, although nonrandom, may not always be easy to recognize.

n	2^n	Time			
	(approx)	1M tests/s		1G tests/s	
32	4.3×10^9	4096	sec	4	sec
40	1.1×10^{12}	291	hrs	1024	sec
56	72×10^{15}	2179	yrs	777	days
64	1.84×10^{19}	557845	yrs	545	yrs
128	3.4×10^{38}	10^{25}	yrs	10^{22}	yrs

Table 2: The Security Provided by Certain Key Sizes.

◇ **Exercise 1:** Cryptographers often have to refute the following statement: "I can always crack an encrypted message by trying the entire key space. With a really fast computer, I can easily try all the possible 64-bit keys. Besides, I may succeed on the first try." Use real-life examples to illustrate the fallacy of this boast.

3 Polybius's Monoalphabetic Cipher

Before we get to the details of modern, computer-based encryption algorithms, we present two examples of old, letter-based ciphers. This section and the next one describe a monoalphabetic and polyalphabetic ciphers developed in antiquity by the same person.

The second century B.C. Greek author and historian Polybius (or Polybios) had an interest in cryptography and developed the simple monoalphabetic cipher that today bears his name. The cipher is based on a small square of letters, so when applied to English text, the number of letters is artificially reduced from 26 to 25 by considering `I` and `J` identical. The resulting 25 letters are arranged in a 5×5 square (Figure 3a) where each letter is identified by its row and column (integers in the interval $[1,5]$). Encrypting is done by replacing each plainletter with its coordinates in the Polybius square. Thus,

the plaintext `POLYBIUS␣CIPHER` is encrypted into the numeric sequence 35, 34, 31, 54, 12, 24, 45, 43, 13, 24, 35, 23, 15, and 42.

Even though the ciphertext consists of numbers, this cipher is still monoalphabetic and can easily be broken. An experienced cryptanalyst will quickly discover that the ciphertext consists of 2-digit integers where each digit is in the interval $[1, 5]$, and that the integer 15 appears about 12% of the time. The ciphertext may be written as a sequence of digits, such as 3 5 3 4 3 1 5 4 1 2 2 4 4 5 4 3 1 3 2 4 3 5 2 3 1 5 4 2, but this does not significantly strengthen the method. A key may be added to the basic cipher, in accordance with Kerckhoffs's principle (Section 2). The key `polybius␣cipher` becomes, after the removal of spaces and duplicate letters, the string `polybiuscher`. The rest of the alphabet is appended to this string, and the result is the Polybius square of Figure 3b.

◇ **Exercise 2:** Suggest another way to extend the key `polybius␣cipher` to the complete set of 25 letters.

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	i/j	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

(a)

	1	2	3	4	5
1	p	o	l	y	b
2	i/j	u	s	c	h
3	e	r	a	d	f
4	g	k	m	n	q
5	t	v	w	x	z

(b)

Figure 3: The Polybius Monoalphabetic Cipher.

The monoalphabetic Polybius cipher is sometimes called the *nihilistic cipher* or a *knock cipher* because it was used by the Russian Nihilists, the opponents of the Czar, to communicate in prison by knocking the numbers on the walls between cells. They naturally used the old 35-letter Cyrillic alphabet, and so had a 6×6 Polybius square. Each letter was transmitted by tapping its two coordinates (each an integer in the interval $[1, 6]$) on the wall.

Another variant embeds the digits 6–9 in the ciphertext randomly, to act as nulls or placebos, to confuse any would-be codebreakers or listening jailers.

The monoalphabetic Polybius cipher arranges the one-dimensional string of letters in a two-dimensional square. The method can therefore be extended by increasing the number of dimensions. Since $3^3 = 27$, it makes sense to have a three-dimensional box of size $3 \times 3 \times 3$ and to store 27 symbols in it. Each symbol can be encrypted to a triplet of digits, each in the interval $[0, 2]$ (these are ternary digits, or trits). If the original Polybius method, using a square, can be called *bipartite*, its three-dimensional extension may be called *tripartite*. Similarly, since $2^8 = 256$, it is possible to construct an eight-dimensional structure with 256 symbols, where each symbol can be

coded with eight bits. This may be termed *octopartite* encryption and it is the basis of the EBCDIC (extended BCD Interchange code) used in IBM mainframe computers.

Today, the monoalphabetic Polybius cipher is used mostly to convert letter sequences to numeric sequences. Section 4 discusses a polyalphabetic version of this ancient cipher.

4 Polybius's Polyalphabetic Cipher

The simple Polybius monoalphabetic cipher of Section 3 can be extended to a polyalphabetic cipher. A long key is chosen (normally text from a book) and is encrypted by the Polybius square of Figure 3a. The result is a sequence of two-digit numbers.

The plaintext is also encrypted by the same square, resulting in another sequence of 2-digit numbers. The two sequences are combined by adding corresponding numbers, but the addition is done without propagating any carries. Assuming that the plaintext is `POLYBIUS␣CIPHER`, and the key is the text `happy families are all alike...`, the two sequences and their (carryless) sum are

Plaintext	35	34	31	54	12	24	45	43	13	24	35	23	15	42
Key	23	11	35	35	54	21	11	32	24	31	24	15	43	11
Ciphertext	**58**	**45**	**66**	**89**	**66**	**45**	**56**	**75**	**37**	**55**	**59**	**38**	**58**	**53**

The digits of the two numbers added are in the interval $[1, 5]$, so each digit of the sum is in the interval $[2, 10]$, where 10 is written as a single 0.

Decrypting is done by subtracting the key from the ciphertext. In our example, this operation is summarized by the three lines

Ciphertext	**58**	**45**	**66**	**89**	**66**	**45**	**56**	**75**	**37**	**55**	**59**	**38**	**58**	**53**
Key	23	11	35	35	54	21	11	32	24	31	24	15	43	11
Plaintext	35	34	31	54	12	24	45	43	13	24	35	23	15	42

The carryless addition simplifies the subtraction. Each digit of the ciphertext is greater than the corresponding digit of the key, except for cipherdigits that are zero. If a cipherdigit is zero, it should be replaced by the number 10 before subtraction.

◇ **Exercise 3:** In a 6×6 Polybius square, each digit is in the interval $[1, 6]$. When two digits are added, the sum is in the interval $[2, 12]$, where 10, 11, and 12 are considered 0, 1, and 2, respectively. Is it still possible to add two numbers without propagating carries?

Even though this cipher employs numbers, it is similar to other polyalphabetic ciphers because the numbers are related to letters and the letter frequencies are reflected in the numbers. The resulting ciphertext can be broken with methods similar to those used to break other polyalphabetic

ciphers. The polyalphabetic Polybius cipher is similar to the one-time pad of Section 5 and can provide absolute security if the key is as long as the plaintext, is random, and is used only once.

> Polybius had fared better than most of the leaders and intellectuals that Rome had taken from Achaea. While a prisoner, he met the head of one of Rome's great families, Scipio Aemilianus. Scipio found Polybius good company and exchanged books with him. He took Polybius with him on military campaigns, and he introduced Polybius to Rome's high society.
>
> —From [Toynbee 52]

5 The One-Time Pad

The encryption methods discussed so far start with a plaintext that's a string of characters (letters, digits, and punctuation marks) from a certain alphabet and produce ciphertext whose symbols are drawn from the same alphabet. Modern cryptography is based on computers which use binary numbers, so current encryption techniques assume the binary alphabet whose symbols are 0 and 1. We start with the one-time pad (also called the *Vernam cipher*), a simple, secure, but not always practical algorithm for encrypting binary data. The encryption key of the one-time pad is a long, random binary string that's used just once. The key is distributed to all the parties authorized to use the cipher, and it employs the exclusive-OR (XOR) logical operation to encrypt and decrypt binary data.

The rule of encryption is to perform the XOR of the next bit of the plaintext and the next bit of the key. The result is the next bit of the ciphertext. Similarly, decrypting the next bit of ciphertext is done by XORing it with the next bit of the key. The result is the next bit of the plaintext.

This interesting and important cipher was developed by Gilbert S. Vernam in 1917, and United States patent #1310719 was issued to him in 1918.

The vernam cipher is secure because the resulting ciphertext is a random string of bits. It does not contain any patterns from the plaintext and does not provide the codebreaker with any clues to the plaintext. However, the key has to be long (at least as long as the plaintext) and it should be used just once (the reason for that is explained below). Thus, the one-time pad can be used only in applications where long keys can be distributed securely and often.

It is easy to show that if the keystream of a Vernam cipher is a random sequence of bits, then the ciphertext is random even if the plaintext isn't random. We denote the ith bits of the plaintext, the keystream, and the ciphertext by d_i, k_i, and $c_i = d_i \oplus k_i$, respectively. We assume that the keystream is random, i.e., $P(k_i = 0) = 0.5$ and $P(k_i = 1) = 0.5$. The plaintext isn't random, so we assume that $P(d_i = 0) = p$, which implies $P(d_i = 1) = 1 - p$. Table 4 summarizes the four possible cases of d_i and k_i and their probabilities. The values of c_i and their probabilities for those cases are also listed. It is easy to see from the table that the probability of c_i

being 0 is $P(c_i = 0) = p/2 + (1-p)/2 = 1/2$, and similarly $P(c_i = 1) = 1/2$. The ciphertext of the Vernam cipher is therefore random, which makes this simple method unbreakable.

d_i	$P(d_i)$	k_i	$P(k_i)$	c_i	$P(c_i)$
0	p	0	1/2	0	$p/2$
0	p	1	1/2	1	$p/2$
1	$1-p$	0	1/2	1	$(1-p)/2$
1	$1-p$	1	1/2	0	$(1-p)/2$

Table 4: Truth Table of a Vernam Cipher.

◇ **Exercise 4:** In principle, the Vernam cipher can be broken by a brute force approach where every key is tried. This approach is impractical because the key tends to be long, resulting in an immense keyspace. Also, each plaintext produced by such a search will have to be checked. Most such plaintexts would look random and would be immediately ignored, but some may appear meaningful and may have to be carefully examined. However, the chance that a wrong key would produce meaningful plaintext is very small. Advance arguments to support this claim.

In order to achieve good security, the one-time pad should be used just once. The Venona project [NSA-venona 04] run by the United States Army's signal intelligence service during 1943–1980 is a good, practical example of a case where this rule was broken, with significant results. Using the one-time pad more than once breaks one of the cherished rules of cryptography, namely, to avoid repetitions, and may provide enough clues to a would-be codebreaker to reconstruct the random key and use it to decipher messages. Here is how such deciphering can be done.

We assume that two ciphertexts are given and it is known or suspected that they were encrypted with the same random keystream. We select a short, common word, such as `the`. We can assume that the first message contains some occurrences of this word, so we start by assuming that the *entire* first message consists of copies of this word. We then figure out the random keystream needed to encrypt a series of `the` into the first ciphertext, and try to decrypt the second ciphertext with this key (a key that we can consider *the first guess*). Any part of this first-guess key that corresponds to an actual `the` in the first message would decrypt a small part of the second message correctly. Applying the first-guess key to the second ciphertext may therefore result in plaintext with some meaningful words and fragments.

The next step is to guess how to expand those fragments in the second plaintext and use the improved plaintext to produce a second-guess key. This key is then applied to the first ciphertext to produce a first plaintext that has some recognizable words and fragments and is therefore a little better than just a series of `the`. Using our knowledge of the language, we can expand those fragments and use the improved plaintext to construct a third-guess key.

After a few iterations, both plaintexts may have so much recognizable material that the rest can be guessed with more certainty, thereby leading to complete decipherment.

Even though it offers absolute security, the one-time pad is generally impractical, because the one-time pads have to be generated and distributed safely to every member of an organization that may be very large (such as an army division). This method can be used only in a limited number of applications, such as exchanging top-secret messages between a government and its ambassador abroad or between world leaders.

(In principle, the one-time pad can be used, or rather abused, in cases where the sender wants to remain unaccountable. Once an encrypted message A is decrypted to plaintext B and the one-time pad is destroyed, there is no way to redecrypt A and thus to associate it with B. The sender may deny sending B and may claim that decrypting A had to result in something else.)

In practice, stream ciphers are used instead of the Vernam cipher. Stream ciphers employ keystreams that are *pseudorandom* bit strings. Such a bit string is generated by repeatedly applying a recursive relation, so it is deterministic and therefore not truly random. Still, if a sequence of n pseudorandom bits does not repeat itself, it can be used as the keystream for a stream cipher with relative safety.

6 The Key Distribution Problem

The problem of key distribution has already been mentioned. Kerckhoffs' principle (Section 2) states that the security of a cryptographic method depends on the encryption key, not the encryption algorithm, being kept secret. In cases where a large group of users is authorized to send and receive encrypted messages, each person in the group has to have the key and is responsible for keeping it secret. The larger the group, the greater the chance that the key will fall into the wrong hands. Also, encryption keys have to be changed from time to time, and distributing the new key securely to a large group of users (especially under difficult conditions, such as a war) is a delicate and complex task.

For many years it was strongly believed that the key distribution problem has no satisfactory solution, but an ideal, simple solution was found in the 1970s and has since become the foundation upon which much of modern cryptography is based.

The following narrative illustrates the nature of the solution. Suppose that Alice wants to send Bob a secure message. She places the message in a strong box, locks it with a padlock, and mails it to Bob. Bob receives the box safely, but then realizes that he does not have the key to the padlock. This is a simplified version of the key distribution problem, and it has a simple, unexpected solution. Bob simply adds another padlock to the box and mails it back to Alice. Alice removes her padlock and mails the box to Bob, who removes his lock, opens the box, and reads the message.

◇ **Exercise 5:** (Easy.) When restricted to physical boxes and keys, this problem has a simpler solution. Once Bob verifies receipt of the box, Alice can mail him the key under a separate envelope. Explain why this solution cannot be applied to the case where the messages and keys are files sent between computers.

The cryptographic equivalent is similar. We start with a similar, albeit unsatisfactory, approach. Imagine a group of users of a particular encryption algorithm, where each user has a private key that is unknown to anyone else. Also imagine a user, Alice, who wants to send an encrypted message to another user, Bob. Alice encrypts the message with her private key (a key unknown to Bob) and sends it. Bob receives the encrypted message, encrypts it again, with his key, and sends the doubly-encrypted message back to Alice. Alice now decrypts the message with her key, but the message is still encrypted with Bob's key. Alice sends the message again to Bob, who decrypts it with his key and can read it.

The trouble with this simple scenario is that most ciphers must obey the LIFO (last in first out) rule. The last cipher used to encrypt a doubly-encrypted message must be the first one used to decipher it. This is easy to see in the case of a monoalphabetic cipher. Suppose that Alice's key replaces `D` with `P` and `L` with `X` and Bob's key replaces `P` with `L`. After encrypting a message twice, first with Alice's key and then with Bob's key, any `D` in the message becomes an `L`. However, when Alice's key is used to decipher the `L`, it replaces it with `X`. When Bob's key is used to decipher the `X`, it replaces it with something different from the original `D`. The same LIFO rule applies to most polyalphabetic cipher.

7 Diffie–Hellman–Merkle Keys

However, there is a way out, a discovery made in 1976 by Whitfield Diffie, Martin Hellman, and Ralph Merkle. Their revolutionary Diffie–Hellman–Merkle key exchange method makes it possible to securely exchange a cryptographic key (or any piece of data) over an unsecured channel. It involves the concept of a *one-way function*, a function that either does not have an inverse or whose inverse is not unique. Most functions have simple inverses. The inverse of the exponential function $y = e^x$, for example, is the natural logarithm $x = \log_e y$. However, modular arithmetic provides an example of a simple and useful one-way function. The value of the modulo function $f(x) = x \bmod p$ is the remainder of the integer division $x \div p$ and is an integer in the interval $[0, p-1]$. Table 5 illustrates the one-way nature of modular arithmetic by listing values of $3^x \bmod 7$ for 10 values of x. It is easy to see, for example, that the number 3 is the value of $3^x \bmod 7$ for $x = 1$ and $x = 7$. The point is that the same number is the value of this function for infinitely more values of x, effectively making it impossible to reverse this simple function.

x	1	2	3	4	5	6	7	8	9	10
3^x	3	9	27	81	243	729	2187	6561	19683	59049
$3^x \bmod 7$	3	2	6	4	5	1	3	2	6	4

Table 5: Ten Values of $3^x \bmod 7$.

◇ **Exercise 6:** Find some real-world processes that are one-way either in principle or in practice.

Based on this interesting property of modular arithmetic, the three researchers came up with an original and unusual scheme for distributing keys. The process is summarized in Figure 6. The process requires the modular function $L^x \bmod P$, whose two parameters P (a large prime, about 512 bits) and L should satisfy $L < P$. The two parties have to select values for L and P, but these values don't have to be secret.

	Alice	**Bob**
1	Selects a secret number a, say, 4	Selects a secret number b, say, 7
2	Computes $\alpha = 5^a \bmod 13 = 624 \bmod 13 = 1$ and sends α to Bob	Computes $\beta = 5^b \bmod 13 = 78125 \bmod 13 = 8$ and sends β to Alice
3	Computes the key by $\beta^a \bmod 13 = 4096 \bmod 13 = 1$	Computes the key by $\alpha^b \bmod 13 = 1 \bmod 13 = 1$

Notice that knowledge of α, β, and the function is not enough to compute the key. Either a or b is needed, but these are kept secret.

Figure 6: Three Steps to Compute the Same Key.

Careful study of Figure 6 shows that even if the messages exchanged between Alice and Bob are intercepted, and even if the values $L = 5$ and $P = 13$ that they use are known, the key still cannot be derived, since the values of either a or b are also needed, but both are kept secret by the two parties.

This breakthrough has proved that cryptographic keys can be securely exchanged through unsecured channels, and users no longer have to meet personally to agree on keys or to trust couriers to deliver them. However, the Diffie–Hellman–Merkle key exchange method described in Figure 6 is inefficient. In the ideal case, where both users are online at the same time, they can go through the process of Figure 6 (select the secret numbers a and b, compute and exchange the values of α and β, and calculate the key) in just a few minutes. If they cannot be online at the same time (for example,

if they live in very different time zones), then the process of determining the key may take a day or longer.

◇ **Exercise 7:** Show why the steps of Figure 6 produce the same key for Alice and for Bob.

◇ **Exercise 8:** Why should P be large and why should L be less than P?

The following analogy may explain why a one-way function is needed to solve the key distribution problem. Imagine that Bob and Alice want to agree on a certain paint color and keep it secret. Each starts with a container that has one liter of paint of a certain color, say, R. Each adds one liter of paint of a secret color. Bob may add a liter of paint of color G and Alice may add a liter of color B. Neither knows what color was added by the other one. They then exchange the containers (which may be intercepted and examined). When each gets the other's container, each adds another liter of his or her secret paint. A little thinking shows that the two containers end up with paint of the same color. Intercepting and examining the containers on their ways is fruitless, because one cannot unmix paints. Mixing paints is a one-way operation.

8 Public-Key Cryptography

In 1975, a little after the Diffie–Hellman–Merkle key exchange was published, Whitfield Diffie came up with the concept of an *asymmetric key*. Traditionally, ciphers use symmetric keys. The same key is used to encrypt and decrypt a message. Decrypting is the exact reverse of encrypting. Cryptography with an asymmetric key requires two keys, one for encrypting and the other for decrypting. This seems a trivial concept but is in fact revolutionary. In an asymmetric cipher, there is no need to distribute keys or to compute them by exchanging data as in the Diffie–Hellman–Merkle key exchange scheme. Alice could decide on two keys for her secret messages, make the encryption key public, and keep the decryption key secret (this is her private key). Bob could then use Alice's public key to encrypt messages and send them to Alice. Anyone intercepting such a message would not be able to decipher it because this requires the secret decryption key that only Alice knows.

> Whitfield Diffie took cryptography out of the hands of the spooks and made privacy possible in the digital age—by inventing the most revolutionary concept in encryption since the Renaissance.
>
> —*Wired*, November 1994

9 RSA Cryptography

It was clear to Diffie that a cipher based on an asymmetric key would be the ideal solution to the troublesome problem of key distribution and would completely revolutionize cryptography. Unfortunately, he was unable to actually come up with such a cipher. The first, simple, practical, and secure public-key cipher, known today as RSA cryptography, was finally developed in 1977 by Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA was a triumphal achievement, an achievement based on the properties of prime numbers.

A prime number, as most know, is a number with no divisors. More accurately, it is a positive integer N whose only divisors are 1 and itself. (Nonprime integers are called composites.) For generations, prime numbers and their properties (the mathematical discipline of number theory) were of interest to mathematicians only and had no practical applications whatsoever. RSA cryptography found an interesting, original, and very practical application for these numbers. This application relies on the most important property of prime numbers, the property that justifies the name *prime*. Any positive integer can be represented as the product of prime numbers (its prime factors) *in one way only*. In other words, any integer has a unique prime factorization. For example, the number 65,535 can be represented as the product of integers in many ways, but there is only one set of primes, namely 3, 5, 17, and 257, whose product equals 65,535.

The main idea behind RSA is to choose two large primes p and q that together constitute the private key. The public key N is their product $N = p \times q$ (naturally, it is a composite). The important (and surprising) point is that multiplying large integers is a one-way function! It is relatively easy to multiply integers, even very large ones, but it is practically impossible, or at least extremely time consuming, to find the prime factors of a large integer, with hundreds of digits. Today, after millennia of research (primes have been known to the ancients), no efficient method for factoring numbers has been discovered. All existing factoring algorithms are slow and may take years to factor an integer consisting of a few hundred decimal digits. The factoring challenge that used to be offered (with prizes) by RSA laboratories until 2007 [RSA 10] testifies to the accuracy of this statement.

To summarize, we know that the public key N has a unique prime factorization and that its prime factors constitute the private key. However, if N is large enough, we will not be able to factor it, even with the fastest computers, which makes RSA a secure cipher. Having said that, no one has proved that a fast factorization method does not exist. It is not inconceivable that someone will come up with such an algorithm that would render RSA (impregnable for more than two decades) useless and would stimulate researchers to discover a different public-key cipher.

(Recent declassifying of secret British documents suggests that a cipher very similar to RSA had been developed by James Ellis and his colleagues starting in 1969. They worked for the British government communications headquarters, GCHQ, and so had to keep their work secret. See [Singh 99].

James Ellis, a mathematician and computer scientist, joined GCHQ (then at Eastcote, West London) in 1952, having previously worked for the Admiralty.

—From `http://www.gchq.gov.uk/history/heroes.html#ellis`

The following is a comment made by an anonymous reviewer: "Another reason why the GCHQ work was never applied was because no one there believed it to be useful. It took private industry to recognize its worth. In other words, it wasn't just secrecy that prevented its use until it was reinvented in the open literature.")

And now, to the details of RSA. These are deceptively simple, but the use of large numbers requires special arithmetic routines to be implemented and carefully debugged. We assume that Alice has selected two large primes p and q as her private key. She has to compute and publish two more numbers as her public key. They are $N = p \cdot q$ and e. The latter can be any integer, but it should be relatively prime to $(p-1)(q-1)$, a number denoted by ϕ. Notice that N must be unique (if Joe has selected the same N as his public key, then he knows the values of p and q), but e does not have to be. To encrypt a message M (an integer) intended for Alice, Bob gets her public key (N and e), computes $C = M^e \bmod N$, and sends C to Alice through an open communications channel. To decrypt the message, Alice starts by computing the decryption key d from $e \times d = 1 \bmod \phi$, then uses d to compute $M = C^d \bmod N$.

The security of the encrypted message depends on the one-way nature of the modulo function. Since the encrypted message C is $M^e \bmod N$, and since both N and e are public, the message can be decrypted by inverting the modulo function. However, as mentioned earlier, this function is impossible to invert (or, rather has too many inverses) for large values of N. It is important to understand that polyalphabetic ciphers, block ciphers, and stream ciphers can be as secure as RSA, are easier to implement, and are faster to execute, but they are symmetric and therefore suffer from the problem of key distribution.

The use of large numbers requires special routines for the arithmetic operations. Specifically, the operation M^e may be problematic, since M may be a large number. One way to simplify this operation is to break the message M up into small segments. Another option is to break up e into a sum of terms and use each term separately. For example, if $e = 7 = 1 + 2 + 4$, then the computation becomes

$$M^e \bmod N = [(M^1 \bmod N) \times (M^2 \bmod N) \times (M^4 \bmod N)] \bmod N.$$

Here is an example of RSA encryption and decryption using small parameters. We select the two small primes $p = 137$ and $q = 191$ as our private key and compute $N = p \cdot q = 26{,}167$ and $\phi = (p-1)(q-1) = 25{,}840$. We also select $e = 3$ and, using the extended Euclidean algorithm, find a value $d = 17{,}227$ such that $e \cdot d = 1 \bmod \phi$. The public key is the pair $(N, e) = (26167, 3)$, and the decryption key is $d = 17{,}227$. For the plaintext M, we select the 4-character string `abcd` whose ASCII codes are 97,

98, 99, and 100, or in binary 01100001, 01100010, 01100011, and 01100100. These are grouped into the two 16-bit blocks $0110000101100010_2 = 24{,}930$ and $0110001101100100_2 = 25{,}444$. Encrypting the blocks is done by

$$C = M^e \bmod N = 24{,}930^3 \bmod 26{,}167 = 23{,}226,$$
$$C = M^e \bmod N = 25{,}444^3 \bmod 26{,}167 = 23{,}081.$$

Decrypting the two blocks C of ciphertext is done by

$$C^d \bmod N = 23{,}226^{17227} \bmod 26{,}167 = 24{,}930,$$
$$C^d \bmod N = 23{,}081^{17227} \bmod 26{,}167 = 25{,}444.$$

RSA in Two Lines of Perl

Adam Back (adam@cypherspace.org) has created an implementation of RSA in just two lines of Perl. It uses dc, an arbitrary-precision arithmetic package that ships with most UNIX systems. Here's the Perl code:

```
print pack"C*",split/\D+/,‘echo "16iII*o\U@{$/=$z;
                                        [(pop,pop,unpack"H*",<>
)]}\EsMsKsN0[lN*1lK[d2%Sa2/d0<X+d*lMLa^*lN%0]dsX
                                    x++lMlN/dsM0<J]dsJxp"|dc‘
```

The security of RSA depends on the infeasibility of factoring large numbers, and this paragraph shows that this problem is equivalent to keeping the private quantity d secret. It turns out that knowledge of d can lead to an efficient factoring of N in the following way. We know that $e \cdot d = 1 \bmod \phi$, which implies that there is an integer k such that $e \cdot d - 1 = k\phi$. From this it follows (by one of the many theorems proved by the great Leonhard Euler) that $a^{ed-1} = 1 \bmod \phi$ for all integers a in the interval $[0, n-1]$. If we now use the notation $ed - 1 = 2^s t$, where t is odd, then it can be shown that there is an integer $b \in [1, s]$ such that $a^{2^{b-1}t} \neq \pm 1 \bmod N$ and $a^{2^b t} = 1 \bmod N$ for at least half of the integers a. We therefore conclude that if such a and b are known, then the greatest common divisor of $a^{2^{b-1}t} - 1$ and N is a factor of N. Factoring N may therefore be done by selecting a random integer a in the proper range and looking for an integer $b \in [1, s]$ that satisfies the property above. With the computing power currently available, this can be achieved quickly and easily.

The value of parameter e is also important. Encryption requires the computation of M^e, so small e implies faster encryption. However, small values of e may also lead to a weaker ciphertext, as the following example illustrates. Suppose that Alice wants to encrypt a message M and send it to three recipients whose public keys are N_1, N_2, and N_3. Suppose also that all three recipients use the same small e, say, $e = 3$ as in our example. Alice can then compute $C_i = M^3 \bmod N_i$ for $i = 1, 2, 3$ and send the three ciphertexts C_i to their recipients. If Eve intercepts the ciphertexts, however, she may

have an easy way to decrypt them. She may use Gauss's algorithm to solve the three modular equations $x = C_i \bmod N_i$ for $i = 1, 2, 3$. The solution x will be in the interval $[0, N_1N_2N_3)$. We know that $M^3 < N_1N_2N_3$, so from the Chinese remainder theorem we conclude that x must equal M^3. Hence, Eve can decrypt the message by computing $\sqrt[3]{x}$. This vulnerability can be avoided by using large values for e or, in cases where the same message is sent to several recipients, by appending a random string to each message in order to make the plaintexts different.

The Chinese remainder theorem states that if the integers n_1 through n_k are pairwise relatively prime, then the system of equations

$$x = a_1 \bmod n_1, \quad x = a_2 \bmod n_2, \ldots, \quad x = a_k \bmod n_k,$$

has a unique solution modulo the quantity $n = n_1n_2 \ldots n_k$.

Gauss's theorem states that the solution to the system of equations in the Chinese remainder theorem can be expressed as $x = \sum_{i=1}^{k} a_iN_iM_i \bmod n$, where $N_i = n/n_i$ and $M_i = N_i^{-1} \bmod n_i$.

Another potential vulnerability of RSA is its *multiplicative property*. We denote by $\mathcal{Z}_N$ the set of integers modulo N, i.e., $\{0, 1, \ldots, N-1\}$. We similarly denote by $\mathcal{Z}_N^*$ the set of integers $\{a \in \mathcal{Z}_n | \gcd(a, N) = 1\}$. In the special case where N is prime, $\mathcal{Z}_N^* = \{1, 2, \ldots, N-1\}$. It can be shown that $\mathcal{Z}_N$ is closed under multiplication and constitutes a multiplicative group.

Assume that $C_i = M_i^e \bmod N$ for $i = 1, 2$ (i.e., two messages use the same N). This implies that $(M_1M_2)^e = C_1C_1 \bmod N$. The RSA multiplicative property can now be stated as follows. The ciphertext C that corresponds to the plaintext $M = M_1M_2 \bmod N$ is $C = C_1C_2 \bmod N$. An attack by Eve on messages decrypted by Alice is possible if Alice has a certain amount of trust in Eve and is willing to encrypt certain messages for her. Suppose that Eve wants to decrypt a message $C = M^e \bmod N$ that she has intercepted on its way to Alice. Eve can select a random integer $x \in \mathcal{Z}_N^*$, compute $C' = C \cdot X^e \bmod N$, and ask Alice to encrypt C'. If Alice complies, she will compute $M' = (C')^d \bmod N$ and send M' to Eve, who can then use the relation

$$M' = (C')^d \bmod N = C^d(X^e)^d = M \cdot X \bmod N$$

to compute $M = M'X^{-1} \bmod N$.

The *cycling attack* poses another threat to RSA security. Given a plaintext M in the interval $[0, N-1]$, it is encrypted to $C = M^e \bmod N$. The ciphertext C is therefore also an integer in the same interval. Thus, M and C are *permutations* of each other. Because of this, there must be a positive integer k such that $C^{e^k} \bmod N = C$ and $C^{e^{k-1}} \bmod N = M$. The cycling attack uses the public key (N, e) and an intercepted ciphertext C to compute the sequence $C^e \bmod N$, $C^{e^2} \bmod n$, $C^{e^3} \bmod N$, and so on, until one of those numbers, $C^{e^k} \bmod n$, equals C. This reveals the value of k and makes it possible to decrypt C by computing $M = C^{e^{k-1}} \bmod N$. Special software has to be used in these calculations, since the quantities C^{e^k} grow very quickly.

Regardless of the key used, some plaintexts are encrypted by RSA to themselves. Thus, $M^e \bmod N$ may sometimes equal M. Such messages are referred to as *unconcealed* and are rare. Nevertheless, RSA encryption software should compare every ciphertext C to its plaintext M and alert the user when an unconcealed message is detected.

The various problems mentioned here are eliminated by the Public-Key Cryptography Standards (PKCS) developed by RSA Security. The interested reader should consult especially PKCS #1 [PKCS 04].

RSA stands for "resists serious attack."

10 SSL: Secure Socket Layer

The following is a dramatization. Alice is hunched over her computer, browsing the Internet. Her wedding to Bob is in a week, and she is still looking for a wedding dress. She has just found a beautiful cream-colored layered chiffon dress that is exactly her size (36–24–36) and is within her price range. It is sold online by `ChiffonDresses.com`. Alice takes out her credit card, ready to send her number and order the dress, but her hand suddenly freezes in midair. She has just remembered that important transactions on the Internet require special security. She checks the bottom-left corner of her screen and yes, there is a small lock, similar to the one shown here, that assures her that the transaction she is about to perform is secure (the URL also changes to `https` instead of `http`). She can order her dress with confidence, being reasonably certain that no one can intercept and steal her credit card number.

This scenario is common. Most of us perform sensitive transactions over the Internet, and we expect them to be private. Online purchasing is one example. Online banking, where a bank account can be reviewed by a customer after a PIN is sent, is another.

This section describes the SSL (secure socket layer) protocol employed by all major Web browsers, as well as by other software, to secure messages sent over the Internet. First, a disclaimer. SSL provides secure communications but cannot guarantee total security. A credit card number or other sensitive information sent over the Internet by the SSL protocol is encrypted and cannot be compromised while in transit. When it arrives at its destination, however, the security provided by SSL ceases and the information may become vulnerable. A dishonest employee may steal it. An insecure data base may be taken over by a hacker and its content copied and misused. The conclusion is simple. Don't trust SSL all the way. Trust it only for communicating your sensitive data. If there is any reason to doubt the integrity of the receiver, don't send the data. The Better Business Bureau [bbbseal 05] is one source that can be employed to evaluate the integrity of a commercial organization.

SSL was developed at Netscape Communications, Inc. in 1994 in response to users' demands for secure Internet communications. It has since

evolved and been strengthened considerably by several organizations. Today, the most-common SSL protocol is TLS (transport layer security), and there are other versions of SSL, such as an open version (openssl) and a version for wireless communications (WTLS).

Two recommended references are [Rescorla 00] and [Thomas 00].

As before, we assume two protagonists, Alice and Bob. Alice plays the part of a consumer trying to purchase an item online. Bob is the seller. The SSL protocol proceeds in the following steps:

1. An authentication protocol is executed by Alice to make sure that Bob is really who he claims to be. Bob's public key is sent to Alice as part of this protocol. The protocol is based on the public-key concept and employs RSA encryption and also a trusted third party.

2. Alice selects a random key for encrypting her sensitive information. This key is encrypted with Bob's public key and is sent to Bob.

3. Alice uses this key to encrypt her sensitive data with a fast, strong encryption algorithm such as DES or AES. Bob uses the same key and algorithm to decrypt the data. Several messages can be exchanged this way between the two parties in complete privacy.

It is obvious that step 1 is the most important part of SSL. It provides secure communications over an insecure channel. This step is complex and slow, which is why it is used only for communicating a short (normally 128-bit) key. The sensitive data itself is encrypted with a fast cipher. This step depends on a basic property of the RSA encryption algorithm. Data encrypted with a public key can be decrypted only with the corresponding private key, but data can also be encrypted with the private key and decrypted only with the corresponding public key. With this in mind, we start with a simple authentication protocol. (We use the notation "`<message> key`" to indicate a message encrypted with a certain key.) If Alice wants to authenticate Bob, she can send him a short message and have Bob encrypt it with his private key and return the result.

Alice → Bob: `Authenticate this.`
Bob → Alice: `<Authenticate this> Bob's private key.`

Alice now decrypts this result with Bob's public key. If the result matches her original message, she has authenticated Bob. This simple protocol has two drawbacks, as follows:

1. Alice must know Bob's public key. If Alice and Bob are members of a group—say, both are scientists and have been communicating by email for a while—then their public keys are known to all the group's members because they are included in each email message. However, if Bob is an organization, such as a new online store, Alice may not have its public key. Even if Bob sends his public key to Alice, she cannot be sure that it really came from Bob's store; it could have been sent by Eve pretending to be Bob and trying to steal Alice's card number.

2. Encrypting a message with your private key and sending it to Alice leads to weak security. Remember that Alice has the original message. If she also has its private-key encryption, she may use both to pretend to be Bob.

Our simple protocol needs improvements. The first one eliminates the need to encrypt Alice's message with Bob's private key. Instead, Bob selects a new message, computes its *message digest*, encrypts the digest with his private key, and sends the (plain) message and the encrypted digest to Alice.

Alice → Bob: `Looking for Bob.`
Bob → Alice: `I'm Bob, <Digest[I'm Bob]> Bob's private key.`

The digest of a message is a function of the message with the following useful properties: (1) it is practically infeasible to compute the original message from its digest and (2) the chance of finding another message that will produce the same digest is extremely small. In practice, a digest is a hash function that hashes text of any length to a small (typically 128-bit) number. The SHA-1 hash function is currently popular as a digest generator. It has replaced the (older and somewhat similar) MD5 function, which is described below.

With this protocol, Bob still has to send a message (`I'm Bob`) and the encrypted version of its digest (this is known as a digital signature), but now he can select the message, which gives him more protection from an unscrupulous Alice. The protocol constitutes authentication because Alice has a plain message and the private-key encryption of its digest. She can decrypt the digest, digest the message, and compare the two digests. There is still the problem of having Bob's public key and being certain that it is Bob's, and no one else's public key. Here is what may happen if Eve pretends to be Bob.

Alice → Bob: `I'm looking for Bob.`
Eve → Alice: `I'm Bob, Eve's-public-key.`
Alice → Bob: `Are you?`
Eve → Alice: `Of course I am, <Digest[Of course I am]> Eve's private key.`

The solution to this dilemma involves a third, trusted party, an escrow, that issues *certificates*. When Bob opens his store, he applies for a certificate from an escrow. The escrow company sends an inspector to check Bob and his facilities and to look at their operations and verify their identification. If all is satisfactory, a certificate is issued, but it has an expiration date and has to be renewed periodically. Admittedly, this solution is not elegant. In principle, we would like a protocol that involves just the two communicating parties, but in practice a third party is needed. A certificate contains the following fields Figure 7:

1. The name of the certificate issuer (the escrow)
2. A digital signature of the certificate issuer
3. The name of the subject, Bob (the entity for which the certificate is issued)
4. The subject's public key
5. The certificate's expiration date

Figure 8 is a detailed listing of the fields of a typical certificate. The issuer part and the subject part have the fields `C` (two-letter international

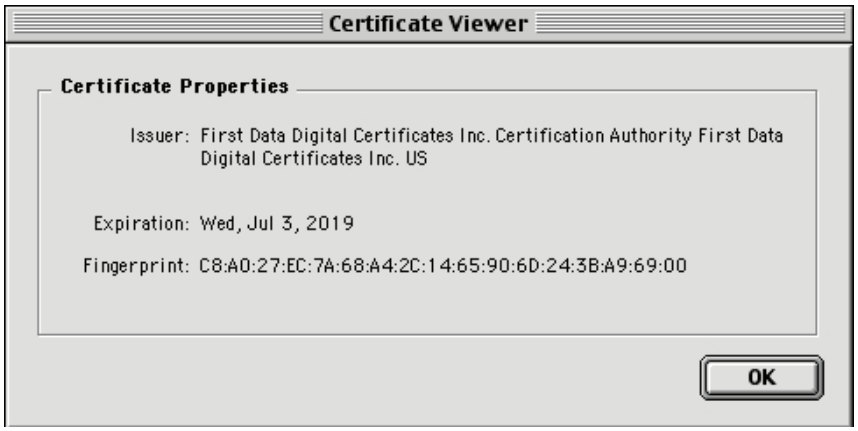

Figure 7: A Certificate.

country code), `ST` (state or province), `L` (locality), `O` (organization name), and `OU` (organizational unit).

Many organizations apply for certificates, so there must be many certificate issuers. Alice doesn't know all of them. She can be expected to know only a few. There is therefore a need for root certificate issuers. Every leading Web browser comes with a list of root certificates preinstalled. A root certificate (also known as a *CA* or *certificate authority*) belongs to a trusted authority that can issue certificates to other, smaller certificate issuers after checking each to make sure it can be trusted. The encryption preferences (or security preferences) menu of the browser can display the list of CA certificates it knows. If the certificate has expired, the browser displays a dialog box similar to Figure 9 that gives the user a choice to continue the sensitive web session or terminate it.

Figure 9: An Expired Certificate.

With certificates, the SSL protocol proceeds as follows:

```
Certificate:
Data:
Version: 1 (0x0)
Serial Number: 0 (0x0)
Signature Algorithm: md5WithRSAEncryption
Issuer:
  C=US,
  ST=NC,
  L=Cary,
  O=My New Outfit, Inc.,
  OU=Sales,
  CN=ntbox.somewhere.com/Email=me@somewhere.com

Validity
  Not Before: Oct 7 04:19:24 1999 GMT
  Not After : Oct 6 04:19:24 2000 GMT

Subject:
  C=US,
  ST=NC,
  L=Cary,
  O=My New Outfit, Inc.,
  OU=Sales,
  CN=ntbox.somewhere.com/Email=me@somewhere.com

Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):

00:c9:dd:68:31:ca:1c:ab:74:7c:21:a8:de:71:22:
25:ec:48:dd:54:34:b5:b8:be:ad:96:cf:56:ad:a2:
7d:9f:81:d5:62:3a:f1:c2:03:4d:8d:73:a3:cb:ac:
f8:f4:d7:95:0d:3f:9e:2c:8f:5f:d3:40:91:09:79:
21:c4:8b:f6:0a:3b:2c:c7:42:3d:2c:c3:5b:17:68:
58:2e:47:42:1e:24:41:1d:59:ba:57:0c:26:63:2e:
46:55:72:e5:1e:61:6c:6e:c2:73:ad:e0:68:ed:70:
a9:43:73:69:b5:c3:9f:64:54:d6:12:11:f3:10:38:
42:e8:54:82:23:f7:20:26:03

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

4f:27:7b:c5:f1:52:33:bc:f8:50:19:b9:98:e6:3b:08:9b:4b:
7b:24:f8:80:10:18:a4:25:6a:39:b1:75:35:05:64:54:ec:5e:
e4:c1:88:fb:7f:72:d1:32:f4:8c:0d:08:28:7e:7e:a5:5f:61:
9c:cc:b4:5c:13:f0:71:a8:d0:56:58:11:e6:b8:35:0a:01:b7:
72:7f:e8:a7:b6:82:aa:52:5d:05:29:d8:48:ba:26:8e:ed:41:
38:86:b8:62:2e:9a:f1:be:99:3c:20:76:57:0f:70:4b:a6:18:
82:aa:90:0c:1f:18:05:c3:98:b8:20:9e:e5:64:02:0d:01:4e:
c4:4e
```

Figure 8: A Detailed Certificate.

Alice → Bob: `I'm looking for Bob.`
Bob → Alice: `I'm Bob, Bob's certificate.`
Alice → Bob: `Are you really?`
Bob → Alice: `Definitely, <Digest[Definitely]> Bob's private key.`

It is the certificate that provides Alice (or, in practice, her Web browser) with Bob's public key. To verify that this is really Bob's certificate, Alice's browser reads the certificate's issuer name (say, `Y`) and signature from the certificate. The digital signature contains the issuer's own certificate, which has been issued by one of the root issuers (call it `X`). The browser has a list of the root certificate issuers, and it communicates with root issuer `X` to verify the certificate of issuer `Y`. This is a slow, tedious process, so it is used as little as possible.

Eve may try to impersonate Bob in this protocol, so we have to keep her in mind. She can execute step 2 in the protocol, because she may have Bob's certificate from a past transaction with him, but she cannot execute step 4 because she doesn't have Bob's private key.

The protocol above is the first and most important step in the complete, three-step SSL protocol. Once it has been executed, Alice is confident that she is dealing with Bob and that she has his public key. In the second step, Alice (or rather her browser) selects a random number to serve as a secret key and sends it to Bob, encrypted with his public key, as a short message. Only Bob can decrypt this message, but again we have to place ourselves in Eve's shoes. What can she do? She cannot decrypt this short message, but she can damage it on its way to Bob. This may be useful to Eve, so the SSL protocol must have a way for Bob to identify damaged messages. One way of verifying a message is to append to it a message authentication code (MAC) that consists of a digest of the message and of the secret key. Eve doesn't know the secret key, so she cannot generate the MAC. Here is the revised protocol

Alice → Bob: `Is this Bob?`
Bob → Alice: `I'm Bob, Bob's certificate.`
Alice → Bob: `Are you really?`
Bob → Alice: `Definitely, <Digest[Definitely]> Bob's private key.`
Alice → Bob: `You have been authenticated.`
Alice → Bob: `Here is our new key <secret-key> Bob's public key.`
Alice computes: `MAC=Digest[My CC # is 12345, secret-key].`
Alice → Bob: `<My CC # is 12345, MAC> secret-key.`

Bob knows to expect two-part messages, where the second part consists of the MAC of the first part. Any corrupted message can easily be identified by Bob. Once Bob has received the new secret key (normally, 40, 56, or 128 bits) from Alice, the two can exchange messages with confidence. The messages are encrypted with this key by a secure, fast algorithm, such as DES or RC4.

As noted earlier, SSL was developed at Netscape Communications in 1994. Just a year later, several hackers discovered a weakness in the Netscape implementation of the first SSL version. It turned out that the secret key

was selected by a pseudorandom number generator (PRNG) whose seed was a combination of the current time (just the seconds and microseconds) and the process id. Netscape programmers believed that such a combination was sufficiently random and would lead to pseudorandom numbers that could be used as secure keys. However, someone intercepting information packets sent by a browser can have a good idea of the time (in seconds) when the packets were generated. Also, someone with access to any account on the operating system where the Netscape browser is running can find the id of any process. The microseconds part of the seed can then be found by trying the million values between 0 and 999,999. It seems that Netscape has since improved the way the seed of the PRNG is computed.

To understand the rest, you ought to have a notion
of two areas in computer science: digital libraries
(databases) and digital magic (cryptography).

—Dmitri Asonov

Answers to Exercises

1: The number of 64-bit keys is 2^{64} = 18,446,744,073,709,551,620 or approximately 1.8×10^{19}. The following examples illustrate the magnitude of this key space.

1. 2^{64} seconds equal 584,942,417,355 years.

2. The unit of electrical current is the Ampere. One Ampere is defined as 6.24×10^{18} electrons per second. Even this huge number is smaller than 2^{64}.

3. Even light, traveling at 299,792,458 m/s, takes 61,531,714,963 seconds (about 1,951 years) to cover 2^{64} meters. This distance is therefore about 1,951 light years.

4. In a fast, 5 GHz computer, the clock ticks five billion times per second. In one year, the clock ticks $5 \cdot 10^9 \cdot (3 \cdot 10^7) = 1.5 \cdot 10^{17}$ times.

5. The mass of the sun is roughly $2 \cdot 10^{31}$ kg and the mass of a single proton is approximately $1.67 \cdot 10^{-27}$ kg. There are therefore approximately 10^{58} protons in the sun. This number is about 2^{193}, so searching a keyspace of 193 bits is equivalent to trying to find a single proton in the sun (ignoring the fact that all protons are identical and that the sun is hot). The proverbial "needle in a haystack" problem pales in comparison.

6. The term *femto*, derived from the Danish *femten*, meaning *fifteen*, stands for 10^{-15}. Thus, a femtometer is 10^{-15} m, and a cubic femtometer is 10^{-45} cubic meters, an incredibly small unit of volume. A light year is 10^{16} meters, so assuming that the universe is a sphere of radius 15 billion light years, its volume is $(4/3)\pi(15 \times 10^9 \times 10^{16})^3 = 1.41372 \times 10^{79}$ cubic meters or about 10^{124} cubic femtometers. This is roughly 2^{411}, so searching a keyspace of 411 bits is like trying to locate a particular cubic femtometer in the entire universe.

These examples illustrate the power of large numbers and should convince any rational person that breaking a code by searching the entire keyspace is a delusion. As for the claim that "there is a chance that the first key tried will be the right one," for a 64-bit keyspace this chance is 2^{-64}. To get a

feeling for how small this number is, consider that light travels 1.6×10^{-11} meters (about the size of 10 atoms laid side by side) in 2^{-64} seconds.

2: Follow each letter in the key `polybiuscher` with its first successor that is still not included in the key. Thus, `p` should be followed by `q` and `o` should be followed by `p`, but because `p` is already included in the key (as are `q`, `r`, and `s`), the `o` is followed by `t`. This process produces first the 22-letter string `pqotlmyzbcikuvswhnefrx` which is then extended in the same way to become the 25-letter string `paqdogtlmyzbcikuvswhnefrx`.

3: Yes, as is easy to see by examining the following examples (notice the two occurrences of 22 in the ciphertext and how they produce different plaintexts):

Plaintext	+	66	05	66	11	61	Ciphertext	−	**22**	**61**	**88**	**22**	**27**
Key		66	66	22	11	66	Key		66	66	22	11	66
Ciphertext		**22**	**61**	**88**	**22**	**27**	Plaintext		66	05	66	11	61

4: The average word size in English is 4–5 letters. We therefore start by examining 4-letter words. There are 26 letters, so the number of combinations of four letters is $26^4 = 456{,}976$. A good English-language dictionary contains about 100,000 words. Assuming that half these words have 4 letters, the percentage of valid 4-letter words is $50000/26^4 \approx 0.11$. The percentage of 5-letter words is obtained similarly as $50000/26^5 \approx 0.004$. Random text may therefore have some short (2–4 letters) words, and very few 5–6 letter words, but longer words would be very rare.

5: When an encrypted message is sent by Alice to Bob, it can be intercepted by Eve and copied. When the key is later sent, Eve may intercept it and use it to decrypt the message.

6: Mixing salt and pepper is a one-way operation in practice (in principle, they can be separated). Heat flow from high to low temperature in a closed system is a one-way process in principle. Giving birth is one-way in principle, while squeezing glue out of a tube is one-way in practice.

7: This is a direct result of the properties of the modulo function. In step 3, Alice computes

$$\beta^a \bmod 13 = (5^b \bmod 13)^a \bmod 13 = 5^{b \cdot a} \bmod 13,$$

and Bob computes the identical expression

$$\alpha^b \bmod 13 = (5^a \bmod 13)^b \bmod 13 = 5^{a \cdot b} \bmod 13.$$

8: The final key is computed, in step 3, as $L^{a \cdot b} \bmod P$ (or, identically, as $L^{b \cdot a} \bmod P$), so it is an integer in the range $[0, P-1]$. Thus, there are only P possible values for the key, which is why P should be large. If we allow values L greater than P, then a user may accidentally select an L that is a multiple of P, which results in a key of 0, thereby providing an eavesdropper with useful information. If P is a prime and if $L < P$, then P is not a prime factor of L^x, so $L^x \bmod P$ cannot be zero.

We learn more by looking for the answer to a question and not finding it than we do from learning the answer itself.
—Lloyd Chudley Alexander (1924—2007)

http://www.springer.com/978-0-85729-005-2

Elements of Computer Security
Salomon, D.
2010, XX, 375 p., Softcover
ISBN: 978-0-85729-005-2