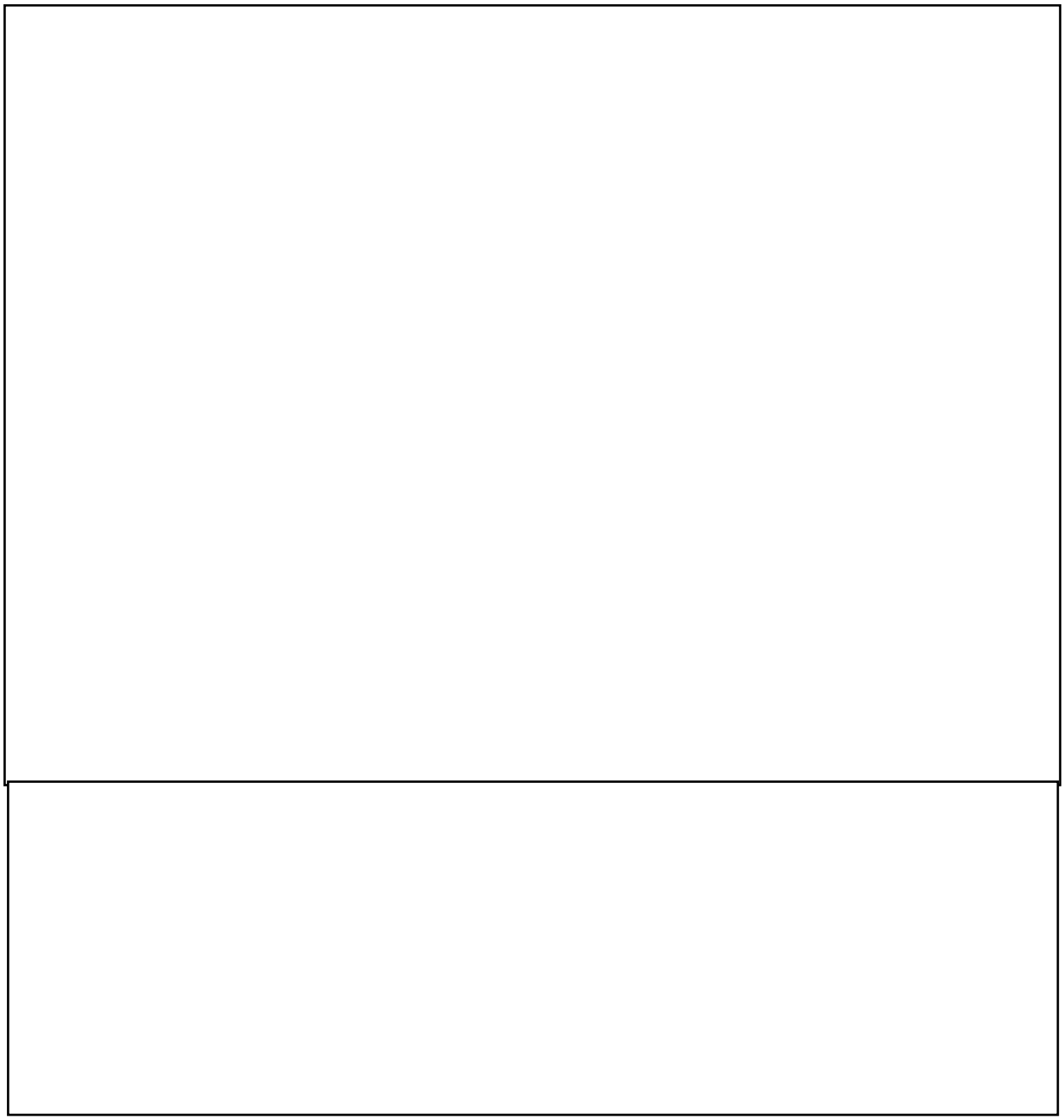

8.3 PUBLIC-KEY ALGORITHMS

Historically, distributing the keys has always been the weakest link in most cryptosystems. No matter how strong a cryptosystem was, if an intruder could steal the key, the system was worthless. Cryptologists always took for granted that the encryption key and decryption key were the same (or easily derived from one another). But the key had to be distributed to all users of the system. Thus, it seemed as if there was an inherent problem. Keys had to be protected from theft, but they also had to be distributed, so they could not be locked in a bank vault.

In 1976, two researchers at Stanford University, Diffie and Hellman (1976), proposed a radically new kind of cryptosystem, one in which the encryption and decryption keys were so different that the decryption key could not feasibly be

derived from the encryption key. In their proposal, the (keyed) encryption algorithm, E, and the (keyed) decryption algorithm, D, had to meet three requirements. These requirements can be stated simply as follows:

1. $D(E(P)) = P$.
2. It is exceedingly difficult to deduce D from E.
3. E cannot be broken by a chosen plaintext attack.

The first requirement says that if we apply D to an encrypted message, $E(P)$, we get the original plaintext message, P, back. Without this property, the legitimate receiver could not decrypt the ciphertext. The second requirement speaks for itself. The third requirement is needed because, as we shall see in a moment, intruders may experiment with the algorithm to their hearts' content. Under these conditions, there is no reason that the encryption key cannot be made public.

The method works like this. A person, say, Alice, who wants to receive secret messages, first devises two algorithms meeting the above requirements. The encryption algorithm and Alice's key are then made public, hence the name **public-key cryptography**. Alice might put her public key on her home page on the Web, for example. We will use the notation E_A to mean the encryption algorithm parameterized by Alice's public key. Similarly, the (secret) decryption algorithm parameterized by Alice's private key is D_A. Bob does the same thing, publicizing E_B but keeping D_B secret.

Now let us see if we can solve the problem of establishing a secure channel between Alice and Bob, who have never had any previous contact. Both Alice's encryption key, E_A, and Bob's encryption key, E_B, are assumed to be in publicly readable files. Now Alice takes her first message, P, computes $E_B(P)$, and sends it to Bob. Bob then decrypts it by applying his secret key D_B [i.e., he computes $D_B(E_B(P)) = P$]. No one else can read the encrypted message, $E_B(P)$, because the encryption system is assumed to be strong and because it is too difficult to derive D_B from the publicly known E_B. To send a reply, R, Bob transmits $E_A(R)$. Alice and Bob can now communicate securely.

A note on terminology is perhaps useful here. Public-key cryptography requires each user to have two keys: a public key, used by the entire world for encrypting messages to be sent to that user, and a private key, which the user needs for decrypting messages. We will consistently refer to these keys as the *public* and *private* keys, respectively, and distinguish them from the *secret* keys used for conventional symmetric-key cryptography.

8.3.1 RSA

The only catch is that we need to find algorithms that indeed satisfy all three requirements. Due to the potential advantages of public-key cryptography, many researchers are hard at work, and some algorithms have already been published.

One good method was discovered by a group at M.I.T. (Rivest et al., 1978). It is known by the initials of the three discoverers (Rivest, Shamir, Adleman): **RSA**. It has survived all attempts to break it for more than 30 years and is considered very strong. Much practical security is based on it. For this reason, Rivest, Shamir, and Adleman were given the 2002 ACM Turing Award. Its major disadvantage is that it requires keys of at least 1024 bits for good security (versus 128 bits for symmetric-key algorithms), which makes it quite slow.

The RSA method is based on some principles from number theory. We will now summarize how to use the method; for details, consult the paper.

1. Choose two large primes, p and q (typically 1024 bits).
2. Compute $n = p \times q$ and $z = (p - 1) \times (q - 1)$.
3. Choose a number relatively prime to z and call it d.
4. Find e such that $e \times d = 1 \ mod \ z$.

With these parameters computed in advance, we are ready to begin encryption. Divide the plaintext (regarded as a bit string) into blocks, so that each plaintext message, P, falls in the interval $0 \le P < n$. Do that by grouping the plaintext into blocks of k bits, where k is the largest integer for which $2^k < n$ is true.

To encrypt a message, P, compute $C = P^e \pmod{n}$. To decrypt C, compute $P = C^d \pmod{n}$. It can be proven that for all P in the specified range, the encryption and decryption functions are inverses. To perform the encryption, you need e and n. To perform the decryption, you need d and n. Therefore, the public key consists of the pair (e, n) and the private key consists of (d, n).

The security of the method is based on the difficulty of factoring large numbers. If the cryptanalyst could factor the (publicly known) n, he could then find p and q, and from these z. Equipped with knowledge of z and e, d can be found using Euclid's algorithm. Fortunately, mathematicians have been trying to factor large numbers for at least 300 years, and the accumulated evidence suggests that it is an exceedingly difficult problem.

According to Rivest and colleagues, factoring a 500-digit number would require 10^{25} years using brute force. In both cases, they assumed the best known algorithm and a computer with a 1-μsec instruction time. With a million chips running in parallel, each with an instruction time of 1 nsec, it would still take 10^{16} years. Even if computers continue to get faster by an order of magnitude per decade, it will be many years before factoring a 500-digit number becomes feasible, at which time our descendants can simply choose p and q still larger.

A trivial pedagogical example of how the RSA algorithm works is given in Fig. 8-17. For this example, we have chosen $p = 3$ and $q = 11$, giving $n = 33$ and $z = 20$. A suitable value for d is $d = 7$, since 7 and 20 have no common factors. With these choices, e can be found by solving the equation $7e = 1 \pmod{20}$, which yields $e = 3$. The ciphertext, C, corresponding to a plaintext message, P, is

given by $C = P^3 \pmod{33}$. The ciphertext is decrypted by the receiver by making use of the rule $P = C^7 \pmod{33}$. The figure shows the encryption of the plaintext "SUZANNE" as an example.

Plaintext (P)			Ciphertext (C)		After decryption	
Symbolic	Numeric	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	Symbolic
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E
Sender's computation				Receiver's computation		

Figure 8-17. An example of the RSA algorithm.

Because the primes chosen for this example are so small, P must be less than 33, so each plaintext block can contain only a single character. The result is a monoalphabetic substitution cipher, not very impressive. If instead we had chosen p and $q \approx 2^{512}$, we would have $n \approx 2^{1024}$, so each block could be up to 1024 bits or 128 eight-bit characters, versus 8 characters for DES and 16 characters for AES.

It should be pointed out that using RSA as we have described is similar to using a symmetric algorithm in ECB mode—the same input block gives the same output block. Therefore, some form of chaining is needed for data encryption. However, in practice, most RSA-based systems use public-key cryptography primarily for distributing one-time session keys for use with some symmetric-key algorithm such as AES or triple DES. RSA is too slow for actually encrypting large volumes of data but is widely used for key distribution.

8.3.2 Other Public-Key Algorithms

Although RSA is widely used, it is by no means the only public-key algorithm known. The first public-key algorithm was the knapsack algorithm (Merkle and Hellman, 1978). The idea here is that someone owns a large number of objects, each with a different weight. The owner encodes the message by secretly selecting a subset of the objects and placing them in the knapsack. The total weight of the objects in the knapsack is made public, as is the list of all possible objects and their corresponding weights. The list of objects in the knapsack is kept secret. With certain additional restrictions, the problem of figuring out a possible list of objects with the given weight was thought to be computationally infeasible and formed the basis of the public-key algorithm.

The algorithm's inventor, Ralph Merkle, was quite sure that this algorithm could not be broken, so he offered a $100 reward to anyone who could break it. Adi Shamir (the "S" in RSA) promptly broke it and collected the reward. Undeterred, Merkle strengthened the algorithm and offered a $1000 reward to anyone who could break the new one. Ronald Rivest (the "R" in RSA) promptly broke the new one and collected the reward. Merkle did not dare offer $10,000 for the next version, so "A" (Leonard Adleman) was out of luck. Nevertheless, the knapsack algorithm is not considered secure and is not used in practice any more.

Other public-key schemes are based on the difficulty of computing discrete logarithms. Algorithms that use this principle have been invented by El Gamal (1985) and Schnorr (1991).

A few other schemes exist, such as those based on elliptic curves (Menezes and Vanstone, 1993), but the two major categories are those based on the difficulty of factoring large numbers and computing discrete logarithms modulo a large prime. These problems are thought to be genuinely difficult to solve—mathematicians have been working on them for many years without any great breakthroughs.

8.4 DIGITAL SIGNATURES

The authenticity of many legal, financial, and other documents is determined by the presence or absence of an authorized handwritten signature. And photocopies do not count. For computerized message systems to replace the physical transport of paper-and-ink documents, a method must be found to allow documents to be signed in an unforgeable way.

The problem of devising a replacement for handwritten signatures is a difficult one. Basically, what is needed is a system by which one party can send a signed message to another party in such a way that the following conditions hold:

1. The receiver can verify the claimed identity of the sender.
2. The sender cannot later repudiate the contents of the message.
3. The receiver cannot possibly have concocted the message himself.

The first requirement is needed, for example, in financial systems. When a customer's computer orders a bank's computer to buy a ton of gold, the bank's computer needs to be able to make sure that the computer giving the order really belongs to the customer whose account is to be debited. In other words, the bank has to authenticate the customer (and the customer has to authenticate the bank).

The second requirement is needed to protect the bank against fraud. Suppose that the bank buys the ton of gold, and immediately thereafter the price of gold