Indeed, some vendors have created special instructions to support AES encryption and decryption [832, 833, 1242].

Like the DES, the AES has spurred studies in cryptanalysis. One effect of these studies is a deeper understanding of how block ciphers work, how to cryptanalyze them, and how to design them to resist attacks. Other effects of these studies remain to be seen.

10.3 Public Key Cryptography

In 1976, Diffie and Hellman [564] proposed a new type of cryptography that distinguished between encipherment and decipherment keys. One of the keys would be publicly known; the other would be kept private by its owner. Symmetric cryptography requires the sender and recipient to share a common key. Public key cryptography does not. If the encipherment key is public, to send a secret message simply encipher the message with the recipient's public key. Then send it. The recipient can decipher it using his private key. Chapter 11, "Key Management," discusses how to make public keys available to others.

Interestingly, James Ellis, a cryptographer working for the British government's Communications-Electronics Security Group, developed the concept of public key cryptography (which he called "non-secret encryption") in a January 1970 report. Two of his colleagues found practical implementations. This work remained classified until 1997 [629].

Because one key is public, and its complementary key must remain secret, a public key cryptosystem must meet the following three conditions:

- It must be computationally easy to encipher or decipher a message given the appropriate key.
- It must be computationally infeasible to derive the private key from the public key.
- It must be computationally infeasible to determine the private key from a chosen plaintext attack.

The first system to meet these requirements generates a shared session key (see Section 11.2.3.1).

Public key systems are based on hard problems. The first type uses NP-complete problems that have special cases that are easy to solve. The system transforms that simpler problem into the more general problem. The information to do this is called "trapdoor information." If an adversary finds that information, the problem can be transformed back into the simpler one, and the adversary can break the system.

EXAMPLE: An early public key cipher was based on the knapsack problem. Given a set of numbers $A = \{a_1, \ldots, a_n\}$ and an integer C, find a subset of

A whose integers add exactly to C. This problem is NP-complete. However, if the a_i are chosen so that each $a_i > a_{i-1} + \cdots + a_1$, then the knapsack is called *superincreasing* and can easily be solved. Merkle and Hellman [1324] developed trapdoor information allowing them to construct a trapdoor knapsack from a superincreasing one.

In 1982, Shamir developed a polynomial-time method for determining trapdoor information [1723], thereby breaking the knapsack cipher. In 1984, Brickell extended this by showing how to break a cipher consisting of iterated knapsacks [293].

A second type is based on hard mathematical problems such as finding the factors of a very large number. The RSA cryptosystem (see Section 10.3.2) provides confidentiality, authentication, and integrity using a problem related to factoring.

An important comment about the examples in this section is necessary.

In the examples that follow, we will use small numbers for pedagogical purposes. In practice, the numbers would be much larger, and often the encipherment schemes will use additional techniques to prevent the success of attacks such as precomputation (see Section 12.1.1) and changing the order of the ciphertext blocks (see Section 12.1.2).

10.3.1 El Gamal

The El Gamal cryptosystem [627] provides message secrecy. It is based on the discrete logarithm problem.

Definition 10–2. Let n, g, and b be integers with $0 \leq a < n$ and $0 \leq b < n$. The *discrete logarithm problem* is to find an integer k such that $0 \leq k < n$ and $a = g^k \bmod n$.

Choose a prime number p with $p-1$ having at least one large factor. Choose some g such that $1 < g < p$; g is called a *generator*, because repeatedly adding g to itself, and reducing $\bmod p$, will generate all integers between 0 and $p-1$ inclusive. Next, select an integer k_{priv} such that $1 < k_{priv} < p-1$, and take $y = g^{k_{priv}} \bmod p$. Then k_{priv} will be the private key and the triplet $K_{pub} = (p, g, y)$ will be the public key.

EXAMPLE: Alice chooses $p = 262643$, a prime number; $p - 1 = 262642 = 2 \times 131321$ has at least one large factor, so her choice is suitable. She chooses $g = 9563$ and the public key $k_{priv} = 3632$. Then

$$y = g^{k_{priv}} \bmod p = 9563^{3632} \bmod 262643 = 27459$$

so the public key is $K_{pub} = (p, g, y) = (262643, 9563, 27459)$.

$$(144749)88247^{-3632} \bmod 262643 = 41812$$
$$(5198)152432^{-3632} \bmod 262643 = 1111$$

Translating this into characters, this is PUP PIE SAR ESM ALL, or PUPPIESARESMALL, which was indeed what Bob sent.

The El Gamal cryptosystem provides strength comparable to other cryptosystems but uses a shorter key. It also introduces randomness into the cipher, so the same letter enciphered twice produces two different ciphertexts. This prevents attacks that depend upon repetition. However, care must be taken; if a random integer k is used twice, an attacker who obtains the plaintext for one message can easily decipher the other (see Exercise 10). Also, notice that c_2 is a linear function of m, so an attacker can forge messages that are multiples of previously enciphered messages. As an example, if (c_1, c_2) is the ciphertext of message m, (c_1, nc_2) is the ciphertext corresponding to nm. Protocols using El Gamal must prevent an attacker from being able to forge this type of message.

Network security protocols often use El Gamal due to its shorter key length. See Section 12.5.3 for an example. It can also be used for authentication (see Section 10.5.2.2).

10.3.2 RSA

The RSA cryptosystem was first described publicly in 1978 [1598]. Unknown at the time was the work of Clifford Cocks in 1973, where he developed a similar cryptosystem. This work was classified, and only became public in the late 1990s [629].

RSA is an exponentiation cipher. Choose two large prime numbers p and q, and let $n = pq$. The totient $\phi(n)$ of n is the number of numbers less than n with no factors in common with n. It can be shown that $\phi(n) = (p-1)(q-1)$ (see Exercise 12).

EXAMPLE: Let $n = 10$. The numbers that are less than 10 and are relatively prime to (have no factors in common with) n are 1, 3, 7, and 9. Hence, $\phi(10) = 4$. Similarly, if $n = 21$, the numbers that are relatively prime to n are 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, and 20. So $\phi(21) = 12$.

Choose an integer $e<n$ that is relatively prime to $\phi(n)$. Find a second integer d such that $ed \bmod \phi(n) = 1$. The public key is (e, n), and the private key is d.

Let m be a message. Then

$$c = m^e \bmod n$$

and

$$m = c^d \bmod n$$

Exercise 13 shows why this works.

When implementing this cipher, two issues are the computation of the modular exponentiation and finding two large primes. Exercise 19 shows how to compute the modular exponentiation quickly. Large prime numbers are found by generating large random numbers and then testing them for primality [291, 1607, 1826, 1955].

EXAMPLE: Let $p = 181$ and $q = 1451$. Then $n = 262631$ and $\phi(n) = 261000$. Alice chooses $e = 154993$, so her private key is $d = 95857$. As in the El Gamal example, Bob wants to send Alice the message "PUPPIESARESMALL," so he encodes it the same way, giving the plaintext 152015 150804 180017 041812 001111. Using Alice's public key, the ciphertext is

$$\begin{aligned}
152015^{154993} \bmod 262631 &= 220160\\
150804^{154993} \bmod 262631 &= 135824\\
180017^{154993} \bmod 262631 &= 252355\\
041812^{154993} \bmod 262631 &= 245799\\
001111^{154993} \bmod 262631 &= 070707
\end{aligned}$$

or 220160 135824 252355 245799 070707.

In addition to confidentiality, RSA can provide data and origin authentication; this is used in digital signatures (see Section 10.5.2.1). If Alice enciphers her message using her private key, anyone can read it, but if anyone alters it, the (altered) ciphertext cannot be deciphered correctly.

EXAMPLE: Suppose Alice wishes to send Bob the same message in such a way that Bob will be sure that Alice sent it. She enciphers the message with her private key and sends it to Bob. As indicated above, the plaintext is represented as 152015 150804 180017 041812 001111. Using Alice's private key, the ciphertext is

$$\begin{aligned}
152015^{95857} \bmod 262631 &= 072798\\
150804^{95857} \bmod 262631 &= 259757\\
180017^{95857} \bmod 262631 &= 256449\\
041812^{95857} \bmod 262631 &= 089234\\
001111^{95857} \bmod 262631 &= 037974
\end{aligned}$$

or 072798 259757 256449 089234 037974. In addition to origin authenticity, Bob can be sure that no letters were altered.

Providing both confidentiality and authentication requires enciphering with the sender's private key and the recipient's public key.

EXAMPLE: Suppose Alice wishes to send Bob the message "PUPPIESARE-SMALL" in confidence and authenticated. Again, assume that Alice's private key is 95857. Take Bob's public key to be 45593 (making his private key 235457). The plaintext is represented as 152015 150804 180017 041812 001111. The encipherment is

$$\begin{aligned}
(152015^{95857} \bmod 262631)^{45593} \bmod 262631 &= 249123 \\
(150804^{95857} \bmod 262631)^{45593} \bmod 262631 &= 166008 \\
(180017^{95857} \bmod 262631)^{45593} \bmod 262631 &= 146608 \\
(041812^{95857} \bmod 262631)^{45593} \bmod 262631 &= 092311 \\
(001111^{95857} \bmod 262631)^{45593} \bmod 262631 &= 096768
\end{aligned}$$

or 249123 166008 146608 092311 096768.

The recipient uses the recipient's private key to decipher the message and the sender's public key to authenticate it. Bob receives the ciphertext above, 249123 166008 146608 092311 096768. The decipherment is

$$\begin{aligned}
(249123^{235457} \bmod 262631)^{154993} \bmod 262631 &= 152012 \\
(166008^{235457} \bmod 262631)^{154993} \bmod 262631 &= 150804 \\
(146608^{235457} \bmod 262631)^{154993} \bmod 262631 &= 180017 \\
(092311^{235457} \bmod 262631)^{154993} \bmod 262631 &= 041812 \\
(096768^{235457} \bmod 262631)^{154993} \bmod 262631 &= 001111
\end{aligned}$$

or 152015 150804 180017 041812 001111. This corresponds to the message Alice sent.

The use of a public key system provides a technical type of nonrepudiation of origin. The message is deciphered using Alice's public key. Because the public key is the inverse of the private key, only the private key could have enciphered the message. Because Alice is the only one who knows this private key, only she could have enciphered the message. The underlying assumption is that Alice's private key has not been compromised, and that the public key bearing her name really does belong to her.

In practice, no one would use blocks of the size presented here. The issue is that, even if n is very large, if one character per block is enciphered, RSA can be broken using the techniques used to break symmetric substitution ciphers (see Sections 10.2.2 and 12.1.3). Furthermore, although no individual block can be altered without detection (because the attacker presumably does not have access to the private key), an attacker can rearrange blocks and change the meaning of the message.

EXAMPLE: A general sends a message to headquarters asking if the attack is on. Headquarters replies with the message "ON" enciphered using an RSA cipher with a 2,048-bit modulus, but each letter is enciphered separately. An attacker intercepts the message and swaps the order of the blocks. When the general deciphers the message, it will read "NO," the opposite of the original plaintext.

Moreover, if the attacker knows that headquarters will send one of two messages (here, "NO" or "ON"), the attacker can use a technique called "forward search" or "precomputation" to break the cipher (see Section 12.1.1). For this reason, plaintext is usually padded with random data to make up a block. This can eliminate the problem of forward searching, because the set of possible plaintexts becomes too large to precompute feasibly.

A different general sends the same request as in the example above. Again, headquarters replies with the message "ON" enciphered using an RSA cipher with a 2,048-bit modulus. Each letter is enciphered separately, but the first 10 bits of each block contain the number of the block, the next 8 bits contain the character, and the remaining 2,030 bits contain random data. If the attacker rearranges the blocks, the general will detect that block 2 arrived before block 1 (as a result of the number in the first 10 bits) and rearrange them. The attacker also cannot precompute the blocks to determine which contains "O," because she would have to compute 2^{2030} blocks, which is computationally infeasible.

10.3.3 Elliptic Curve Ciphers

Miller [1351] and Koblitz [1082] proposed a public key scheme based on *elliptic curves*. This scheme can be applied to any scheme that depends on the discrete logarithm problem. Here, we show a version of El Gamal using elliptic cryptography.

Definition 10–3. An *elliptic curve* is an equation of the form $y^2 = x^3 + ax + b$.

Figure 10–5 shows the plot of the curve $y^2 = x^3 + 4x + 10$. Consider two points on the curve, P_1 and P_2. If $P_1 \neq P_2$, draw a line through them. If $P_1 = P_2$, then draw a tangent to the curve at P_1. Suppose that line intersects the curve at a third point, $P_3 = (x_3, y_3)$. Take $P_4 = (x_3, -y_3)$. We define P_4 to be the sum of P_1 and P_2. Otherwise, the line is vertical, so take $P_1 = (x, y)$ and treat ∞ as another point of intersection with the curve. The third point of intersection is $P_2 = (x, -y)$, so given the above definition of addition, we have $P_1 + \infty = (x, y) = P_1$. Hence the point at ∞ is the identity in addition. It is also its own inverse.

More precisely, let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$. Define

$$m = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{when } P_1 \neq P_2 \\[2ex] \dfrac{3x_1^2 + a}{2y_1} & \text{otherwise} \end{cases}$$

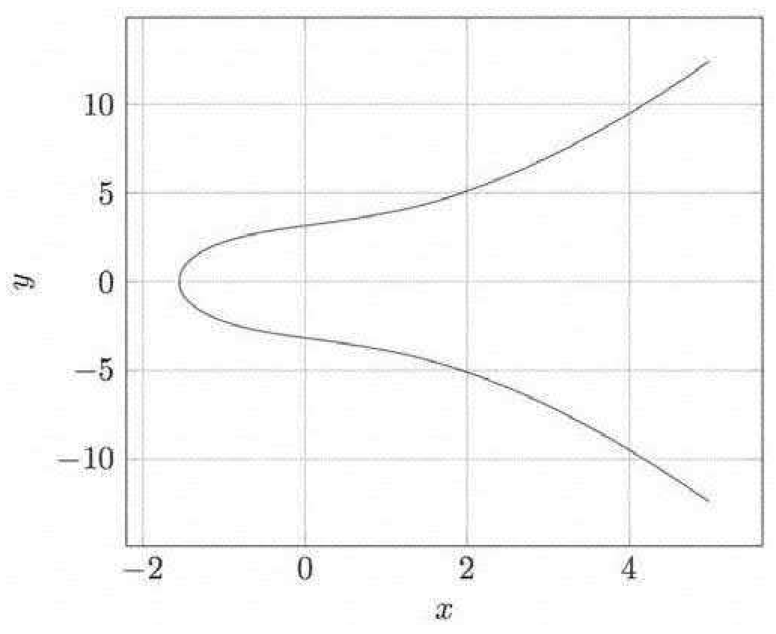

Figure 10–5 Plot of the elliptic curve $y^2 = x^3 + 4x + 10$.

Then $P_3 = P_1 + P_2$, where

$$x_3 = m^2 - x_1 - x_2$$
$$y_3 = m(x_1 - x_3) - y_1$$

Also, if $P_4 = -P_3$, then $x_4 = x_3$ and $y_4 = -y_3$.

This can be turned into a cryptosystem using modular arithmetic, where the modulus used is a prime number p. Thus, the curve of interest is of the form

$$y^2 = x^3 + ax + b \bmod p$$

with p a prime number and $4a^3 + 27b^2 \neq 0$.[3] Suppose we add a point P to itself n times. Call the result Q, so $Q = nP$. If n is large, it is generally very hard to compute from Q and P. This is the basis for the security of the cryptosystem.

Thus, an elliptic curve cryptosystem has four parameters: (a, b, p, P). The private key is a randomly chosen integer $k<p$; in practice, one chooses this number to be less than the number of (integer) points on the curve. The corresponding

[3]More generally, elliptic curves can be over any finite field. When the size of the finite field is a power of 2, the equation has the form $y^2 + xy = x^3 + ax^2 + b$; the rules for addition are also slightly different.

public key is $K = kP$. In the following examples, we shall use the shorthand $(x, y) \bmod p$ to mean $(x \bmod p, y \bmod p)$. Also, $a^{-1} \bmod p$ is the value x that satisfies the equation $ax \bmod p = 1$ (see Section B.3).

To use the elliptic curve version of El Gamal, choose a point P on the curve, and a private key k_{priv}. Then compute $Q = k_{priv}P$. Using the elliptic curve above, this means that the public key is (P, Q, a, p). To encipher a message m, it is first expressed as a point on the elliptic curve. The sender then selects a random number k and computes

$$c_1 = kP$$
$$c_2 = m + kQ$$

and sends those to the recipient. To decipher the message, the recipient computes

$$m = c_2 - k_{priv}c_1$$

EXAMPLE: Alice and Bob now decide to use the elliptic curve version of El Gamal to encipher their messages. They use the same elliptic curve and point as in the previous example. Bob chooses a random number $k_{Bob} = 1847$ as his private key. He then computes his public key $K_{Bob} = k_{Bob}P = 1847(1002, 493) \bmod 2503 = (460, 2083)$.

Alice wants to send Bob the message $m = (18, 1394)$. To encipher it, she chooses a random number $k = 717$, computes

$$c_1 = kP = 717(1002, 493) \bmod 2503 = (2134, 419)$$
$$c_2 = m + kK_{Bob} = (18, 1394) + 717(460, 2083) \bmod 2503 = (221, 1253)$$

and sends c_1 and c_2 to Bob.

To decipher the message, Bob computes

$$k_{Bob}c_1 = 1847(2134, 419) \bmod 2503 = (652, 1943)$$

and uses this to compute

$$m = c_2 - c_1 = (221, 1253) - (652, 1943) \bmod 2503 = (18, 1394)$$

thereby recovering the plaintext message.

The generation of elliptic curves suitable for cryptography is a complex question. In particular, it requires a careful selection of parameters. For example, when $b = 0$ and $p \bmod 4 = 3$, or when $a = 0$ and $p \bmod 3 = 2$, the discrete log problem underlying elliptic curve cryptography becomes significantly easier to solve. Thus, choosing these parameters weakens the cryptosystem. Ways to generate elliptic curves are being studied [440]. Several parameter sets have been recommended for use. The U.S. NIST recommends curves P-192, P-224, P-256, P-384, or P-521 for elliptic curves using a prime modulus, and degree 163, 233, 283, 409, or 571 binary fields [2148]. Certicom recommends these as well, except that the degree 233 binary field is replaced by a degree 239 binary field [2231]. These curves are widely used. Some questions have been raised about the strength of these curves [181]. Other proposed curves include those of the Brainpool standard [1203], Curve1174 [182], Curve25519 [180], and several others [271].

The advantage to using elliptic curves over other forms of public key cryptography is that the keys can be shorter, and hence the computation time is shorter. As an example, elliptic curve cryptography with a key length of 256 to 383 bits provides a level of security comparable to RSA with a modulus of 3,072 bits [126]. Koblitz, Koblitz, and Menezes [1081] review how elliptic curve cryptography became widely accepted, with a discussion of it and RSA.

10.4 Cryptographic Checksums

Suppose Alice wants to send Bob a message of n bits. She wants Bob to be able to verify that the message he receives is the same one that was sent. So she applies a mathematical function, called a *checksum function*, to generate a smaller set of k bits from the original n bits. This smaller set is called the *checksum* or *message digest*. Alice then sends Bob both the message and the associated checksum. When Bob gets the message, he recomputes the checksum and compares it with the one Alice sent. If they match, he assumes that the message has not been changed; if they do not match, then either the message or the checksum has changed, and so they cannot be trusted to be what Alice sent him.

Of course, an adversary can change the message and alter the checksum to correspond to the message. For the moment, assume this will not happen; we will relax this assumption in Section 10.5.

EXAMPLE: The parity bit in the ASCII representation is often used as a single-bit checksum. If *odd parity* is used, the sum of the 1 bits in the ASCII representation of the character, and the parity bit, is odd. Assume that Alice sends Bob the letter "A." In ASCII, the representation of "A" using odd parity is p1000001 in binary, where p represents the parity bit. Because two bits are set, the parity bit is 1 for odd parity.

When Bob gets the message 11000001, he counts the 1 bits in the message. Because this number is odd, Bob believes that the message has arrived unchanged.

to solve. Thus, choosing these parameters weakens the cryptosystem. Ways to generate elliptic curves are being studied [440]. Several parameter sets have been recommended for use. The U.S. NIST recommends curves P-192, P-224, P-256, P-384, or P-521 for elliptic curves using a prime modulus, and degree 163, 233, 283, 409, or 571 binary fields [2148]. Certicom recommends these as well, except that the degree 233 binary field is replaced by a degree 239 binary field [2231]. These curves are widely used. Some questions have been raised about the strength of these curves [181]. Other proposed curves include those of the Brainpool standard [1203], Curve1174 [182], Curve25519 [180], and several others [271].

The advantage to using elliptic curves over other forms of public key cryptography is that the keys can be shorter, and hence the computation time is shorter. As an example, elliptic curve cryptography with a key length of 256 to 383 bits provides a level of security comparable to RSA with a modulus of 3,072 bits [126]. Koblitz, Koblitz, and Menezes [1081] review how elliptic curve cryptography became widely accepted, with a discussion of it and RSA.

10.4 Cryptographic Checksums

Suppose Alice wants to send Bob a message of n bits. She wants Bob to be able to verify that the message he receives is the same one that was sent. So she applies a mathematical function, called a *checksum function*, to generate a smaller set of k bits from the original n bits. This smaller set is called the *checksum* or *message digest*. Alice then sends Bob both the message and the associated checksum. When Bob gets the message, he recomputes the checksum and compares it with the one Alice sent. If they match, he assumes that the message has not been changed; if they do not match, then either the message or the checksum has changed, and so they cannot be trusted to be what Alice sent him.

Of course, an adversary can change the message and alter the checksum to correspond to the message. For the moment, assume this will not happen; we will relax this assumption in Section 10.5.

EXAMPLE: The parity bit in the ASCII representation is often used as a single-bit checksum. If *odd parity* is used, the sum of the 1 bits in the ASCII representation of the character, and the parity bit, is odd. Assume that Alice sends Bob the letter "A." In ASCII, the representation of "A" using odd parity is p1000001 in binary, where p represents the parity bit. Because two bits are set, the parity bit is 1 for odd parity.

When Bob gets the message 11000001, he counts the 1 bits in the message. Because this number is odd, Bob believes that the message has arrived unchanged.

To minimize the probability that a change to either the message or the checksum will be detected, the checksum function must satisfy specific properties.

Definition 10–4. A *cryptographic checksum function* (also called a *strong hash function* or a *strong one-way function*) $h : A \to B$ is a function that has the following properties:

1. For any $x \in A$, $h(x)$ is easy to compute.
2. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$.
3. It is computationally infeasible to find $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$. (Such a pair is called a *collision*.)

The third requirement is often stated as:

$3'$. Given any $x \in A$, it is computationally infeasible to find another $x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$.

However, properties 3 and $3'$ are subtly different. It is considerably harder to find an x' meeting the conditions in property $3'$ than it is to find a pair x and x' meeting the conditions in property 3. To explain why, we need to examine some basics of cryptographic checksum functions.

Given that the checksum contains fewer bits than the message, several messages must produce the same checksum. Ideally, the hashes of all possible messages will be evenly distributed over the set of possible checksums. Furthermore, the checksum that any given message produces can be determined only by computing the checksum. Such a checksum function acts as a random function.

The size of the output of the cryptographic checksum is an important consideration owing to a mathematical principle called the *pigeonhole principle*.

Definition 10–5. The *pigeonhole principle* states that if there are n containers for $n + 1$ objects, at least one container will hold two objects.

To understand its application here, consider a cryptographic checksum function that computes hashes of three bits and a set of files each of which contains five bits. This yields $2^3 = 8$ possible hashes for $2^5 = 32$ files. Hence, at least four different files correspond to the same hash.

Now assume that a cryptographic checksum function computes hashes of 128 bits. The probability of finding a message corresponding to a given hash is 2^{-128}, but the probability of finding two messages with the same hash (that is, with the value of neither message being constrained) is 2^{-64} (see Exercise 24).

Definition 10–6. A *keyed* cryptographic checksum function requires a cryptographic key as part of the computation. A *keyless* cryptographic checksum does not.

Many keyless hash functions have been developed. The best known are MD4 [1593] and MD5 [1594] (128-bit checksums), RIPEMD-160 [575] (160-bit checksum), HAVAL [2096] (128-, 160-, 192-, 224-, and 256-bit checksums), and the Secure Hash Algorithm family of hash functions [2214] (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512). Dobbertin devised a method for generating collisions in MD4 [574] and MD5 [573]. Wang and Yu used a differential attack to find collisions in MD4 and MD5 very quickly [1972]; their attack also works on HAVAL-128, RIPEMD, and the original version of SHA (now called SHA-0) [2213]. Various techniques for finding collisions have found collisions in several SHA hash functions, all on versions with a reduced number of steps (such as 58-step SHA-1 [1971], 70-step SHA-1 [516], 24-step SHA-2, SHA-256, and SHA-512 [1649], and 38-step SHA-256 [1317]).

In 2012, the U.S. NIST selected the Keccak hash function as SHA-3 [2229]. The design of Keccak prevents many of the collision-finding attacks that succeeded in previous hash functions [491]. In 2013, however, a different kind of attack succeeded with 3-round Keccak-384 and Keccak-512, and 5-round Keccak-256 [571]. Only the future will tell whether SHA-3 continues to be as robust as is believed.

10.4.1 HMAC

HMAC is a generic term for an algorithm that uses a keyless hash function and a cryptographic key to produce a keyed hash function [1102]. This mechanism enables Alice to validate that data Bob sent to her is unchanged in transit. Without the key, anyone could change the data and recompute the message authentication code, and Alice would be none the wiser.

The need for HMAC arose because keyed hash functions are derived from cryptographic algorithms. Many countries restrict the import and export of software that implements such algorithms. They do not restrict software implementing keyless hash functions, because such functions cannot be used to conceal information. Hence, HMAC builds on a keyless hash function using a cryptographic key to create a keyed hash function.

Let h be a keyless hash function that hashes data in blocks of b bytes to produce a hash l bytes long. Let k be a cryptographic key. We assume that the length of k is no greater than b; if it is, use h to hash it to produce a new key of length b. Let k' be the key k padded with bytes containing 0 to make b bytes. Let *ipad* be a sequence of bytes containing the bits 00110110 and repeated b times; let *opad* be a similar sequence with the bits 01011100. The HMAC-h function with key k for message m is

$$\text{HMAC-}h(k, m) = h(k' \oplus opad \,||\, h(k' \oplus ipad || m))$$

where $\oplus$ is exclusive or and $||$ is concatenation.

Bellare, Canetti, and Krawczyk [155] analyze the security of HMAC and conclude that the strength of HMAC depends on the strength of the hash function h. Emphasizing this, attacks on HMAC-MD4, HMAC-MD5, HMAC-SHA-0, and HMAC-SHA-1 have been developed, some of which recover partial [451, 1054] or full keys [710,1965]. Bellare [154] extends the analysis by explaining under what conditions HMAC is secure.

Various HMAC functions are used in Internet security protocols (see Chapter 12).

10.5 Digital Signatures

As electronic commerce grows, so does the need for a provably high degree of authentication and integrity. Think of Alice's signature on a contract with Bob. Bob not only has to know that Alice is the other signer and is signing it; he also must be able to prove to a disinterested third party (called a *judge*) that Alice signed it and that the contract he presents has not been altered since Alice signed it. Such a construct plays a large role in managing cryptographic keys as well. This construct is called a *digital signature*.

Definition 10–7. A *digital signature* is a construct that authenticates both the origin and contents of a message in a manner that is provable to a disinterested third party.

The "proof" requirement introduces a subtlety. Let m be a message. Suppose Alice and Bob share a secret key k. Alice sends Bob the message and its encipherment using k. Is this a digital signature?

First, Alice has authenticated the contents of the message, because Bob deciphers the enciphered message and can check that the message matches the deciphered one. Because only Bob and Alice know k, and Bob knows that he did not send the message, he concludes that it has come from Alice. He has authenticated the message origin and integrity. However, based on the mathematics alone, Bob cannot prove that he did not create the message, because he knows the key used to create it. Hence, this is not a digital signature.

Public key cryptography solves this problem. Let d_{Alice} and e_{Alice} be Alice's private and public keys, respectively. Alice sends Bob the message and its encipherment using d_{Alice}. As before, Bob can authenticate the origin and contents of the message, but in this situation a judge can determine that Alice signed the *opad* be a similar sequence with the bits 01011100. The HMAC-*h* function with key k for message m is

$$\text{HMAC-}h(k, m) = h(k' \oplus opad \mid\mid h(k' \oplus ipad \mid\mid m))$$

where $\oplus$ is exclusive or and $\mid\mid$ is concatenation.

Bellare, Canetti, and Krawczyk [155] analyze the security of HMAC and conclude that the strength of HMAC depends on the strength of the hash function h. Emphasizing this, attacks on HMAC-MD4, HMAC-MD5, HMAC-SHA-0, and HMAC-SHA-1 have been developed, some of which recover partial [451, 1054] or full keys [710,1965]. Bellare [154] extends the analysis by explaining under what conditions HMAC is secure.

Various HMAC functions are used in Internet security protocols (see Chapter 12).

10.5 Digital Signatures

As electronic commerce grows, so does the need for a provably high degree of authentication and integrity. Think of Alice's signature on a contract with Bob. Bob not only has to know that Alice is the other signer and is signing it; he also must be able to prove to a disinterested third party (called a *judge*) that Alice signed it and that the contract he presents has not been altered since Alice signed it. Such a construct plays a large role in managing cryptographic keys as well. This construct is called a *digital signature*.

Definition 10–7. A *digital signature* is a construct that authenticates both the origin and contents of a message in a manner that is provable to a disinterested third party.

The "proof" requirement introduces a subtlety. Let m be a message. Suppose Alice and Bob share a secret key k. Alice sends Bob the message and its encipherment using k. Is this a digital signature?

First, Alice has authenticated the contents of the message, because Bob deciphers the enciphered message and can check that the message matches the deciphered one. Because only Bob and Alice know k, and Bob knows that he did not send the message, he concludes that it has come from Alice. He has authenticated the message origin and integrity. However, based on the mathematics alone, Bob cannot prove that he did not create the message, because he knows the key used to create it. Hence, this is not a digital signature.

Public key cryptography solves this problem. Let d_{Alice} and e_{Alice} be Alice's private and public keys, respectively. Alice sends Bob the message and its encipherment using d_{Alice}. As before, Bob can authenticate the origin and contents of the message, but in this situation a judge can determine that Alice signed the message, because only Alice knows the private key with which the message was signed. The judge merely obtains Alice's public key e_{Alice} and uses that to decipher the enciphered message. If the result is the original message, Alice signed it. This is in fact a digital signature.

A digital signature provides the service of nonrepudiation. If Alice claims she never sent the message, the judge points out that the originator signed the message with her private key, which only she knew. Alice at that point may claim that her private key was stolen, or that her identity was incorrectly bound in the certificate (see Chapter 15, "Representing Identity"). The notion of "nonrepudiation" provided here is strictly technical. In fact, Alice's key might have been stolen, and she might not have realized this before seeing the digital signature. Such a claim would require ancillary evidence, and a court or other legal agency would need to handle it. For the purposes of this section, we consider the service of nonrepudiation to be the inability to deny that one's cryptographic key was used to produce the digital signature.

10.5.1 Symmetric Key Signatures

All secret key digital signature schemes rely on a trusted third party. The judge must trust the third party. Merkle's scheme is typical [1322].

Let Cathy be the trusted third party. Alice shares a cryptographic key k_{Alice} with Cathy. Likewise, Bob shares k_{Bob} with Cathy. When Alice wants to send Bob a contract m, she enciphers the message using k_{Alice} and sends it to Bob. Bob sends it to Cathy, who deciphers the message using k_{Alice}, enciphers it with k_{Bob}, and returns this to Bob. He can now decipher it. To verify that Alice sent the message, the judge has Cathy decipher the enciphered message Alice sent and the enciphered message Bob received from Cathy using Alice's and Bob's keys. If they match, the sending is verified; if not, one of them is a forgery.

10.5.2 Public Key Signatures

In our earlier example, we had Alice encipher the message with her private key to produce a digital signature. We now examine two specific systems.

10.5.2.1 RSA Digital Signatures

Section 10.3.2 discussed the RSA system. We observe that using it to authenticate a message produces a digital signature. However, we also observe that the strength of the system relies on the protocol describing how RSA is used as well as on the RSA cryptosystem itself.

First, suppose that Alice wants to trick Bob into signing a message m. She computes two other messages m_1 and m_2 such that $m_1 m_2 \bmod n_{Bob} = m$. She has Bob sign m_1 and m_2. Alice then multiplies the two signatures together $\bmod n_{Bob}$, giving Bob's signature on m (see Exercise 13). The defense is to not sign random documents and, when signing, never sign the document itself; sign a cryptographic hash of the document [1684].

EXAMPLE: Let $n_{Alice} = 262631$, $e_{Alice} = 154993$, $d_{Alice} = 95857$, $n_{Bob} = 288329$, $e_{Bob} = 22579$, and $d_{Bob} = 138091$. Alice and Bob have many possible contracts, each represented by three letters, from which they are to select and sign one.

Alice first asks Bob to sign the sequence 225536, so she can validate his signature. Bob computes

$$225536^{138091} \bmod 288329 = 271316$$

Alice then asks Bob to sign contract "AYE" (002404):

$$002404^{138091} \bmod 288329 = 182665$$

Alice now computes

$$(002404)(225536) \bmod 288329 = 130024$$

She then claims that Bob agreed to contract "NAY" (130024). She presents the signature

$$(271316)(182665) \bmod 288329 = 218646$$

Judge Janice is called, and she computes

$$218646^{22579} \bmod 288329 = 130024$$

Naturally, Janice concludes that Bob is lying, because his public key deciphers the signature. So Alice has successfully tricked Bob.

Enciphering a message and then signing it creates a second problem [60]. Suppose Alice is sending Bob her signature on a confidential contract m. She enciphers it first, then signs it:

$$c = \left(m^{e_{Bob}} \bmod n_{Bob}\right)^{d_{Alice}} \bmod n_{Alice}$$

She then sends the result to Bob. However, Bob wants to claim that Alice sent him the contract M. Bob computes a number r such that $M^r \bmod n_{Bob} = m$. He then republishes his public key as (re_{Bob}, n_{Bob}). Note that the modulus does not change. Now, he claims that Alice sent him M. The judge verifies this using his current public key. The simplest way to fix this is to require all users to use the same exponent but vary the moduli.

EXAMPLE: Smarting from Alice's trick, Bob seeks revenge. He and Alice agree to sign the contract "LUR" (112017). Alice first enciphers it, then signs it:

$$(112017^{22579} \bmod 288329)^{95857} \bmod 262631 = 42390$$

She sends it to Bob. Bob, however, wants the contract to be "EWM" (042212). He computes an r such that $042212^r \bmod 288329 = 112017$; one such r is $r = 9175$. He then computes a new public key $re_{Bob} \bmod \phi(n_{Bob}) = (9175)(22579) \bmod 287184 = 102661$. He replaces his current public key with (102661, 288329), and resets his private key to 161245. He now claims that Alice sent him contract "EWM," signed by her.

Judge Janice is called. She takes the message 42390 and deciphers it:

$$(42390^{154993} \bmod 262631)^{161245} \bmod 288329 = 042212$$

She concludes that Bob is correct.

This attack will not work if one signs first and then enciphers. The reason is that Bob cannot access the information needed to construct a new public key, because he would need to alter Alice's public key (see Exercise 27).

However, signing first and then enciphering enables the recipient to decipher the signed message, re-encrypt it using a third party's public key, and then forward it to the third party. The third party then cannot tell if the original sender sent it directly to him. This is the *surreptitious forwarding attack*. Several solutions to providing enciphered, authenticated messages have been proposed [510, 1101]. One simple solution is to embed the signer's and recipient's names in the signed message. Another is to sign the message, encrypt it, and sign the result; a variant is to encrypt the message, then sign that, and then encrypt the result.

10.5.2.2 El Gamal Digital Signature

This scheme is based on the El Gamal cryptosystem presented in Section 10.3.1. Recall that the generator g is chosen so that $1 < g < p$, where p is a prime number with $p - 1$ having a large factor; the private key d is chosen so that $1 < d < p - 1$; and the public key is (p, g, y), where $y = g^d \bmod p$.

Suppose Alice wants to send Bob a signed contract m. She chooses a number k that is less than, and relatively prime to, $p - 1$ and has not been used before. She computes $a = g^k \bmod p$ and then uses the Extended Euclidean Algorithm (see Appendix B) to find b such that

$$m = (da + kb) \bmod p - 1$$

The pair (a, b) is the signature.