CHAPTER 13

Digital Signatures

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Present an overview of the digital signature process.
- Understand the ElGamal digital signature scheme.
- Understand the Schnorr digital signature scheme.
- Understand the NIST digital signature scheme.
- Compare and contrast the NIST digital signature scheme with the ElGamal and Schnorr digital signature schemes.
- Understand the elliptic curve digital signature scheme.
- Understand the RSA-PSS digital signature scheme.

The most important development from the work on public-key cryptography is the digital signature. The digital signature provides a set of security capabilities that would be difficult to implement in any other way.

Figure 13.1 is a generic model of the process of constructing and using digital signatures. All of the digital signature schemes discussed in this chapter have this structure. Suppose that Bob wants to send a message to Alice. Although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him. For this purpose, Bob uses a secure hash function, such as SHA-512, to generate a hash value for the message. That hash value, together with Bob's private key serves as input to a digital signature generation algorithm, which produces a short block that functions as a **digital signature**. Bob sends the message with the signature attached. When Alice receives the message plus signature, she (1) calculates a hash value for the message; (2) provides the hash value and Bob's public key as inputs to a digital signature verification algorithm. If the algorithm returns the result that the signature is valid, Alice is assured that the message must have been signed by Bob. No one else has Bob's private key and therefore no one else could have created a signature that could be verified for this message with Bob's public key. In addition, it is impossible to alter the message without access to Bob's private key, so the message is authenticated both in terms of source and in terms of data integrity.

We begin this chapter with an overview of digital signatures. We then present the ElGamal and Schnorr digital signature schemes, understanding of which makes it easier to understand the NIST Digital Signature Algorithm (DSA). The chapter then covers the two other important standardized digital signature schemes: the Elliptic Curve Digital Signature Algorithm (ECDSA) and the RSA Probabilistic Signature Scheme (RSA-PSS).

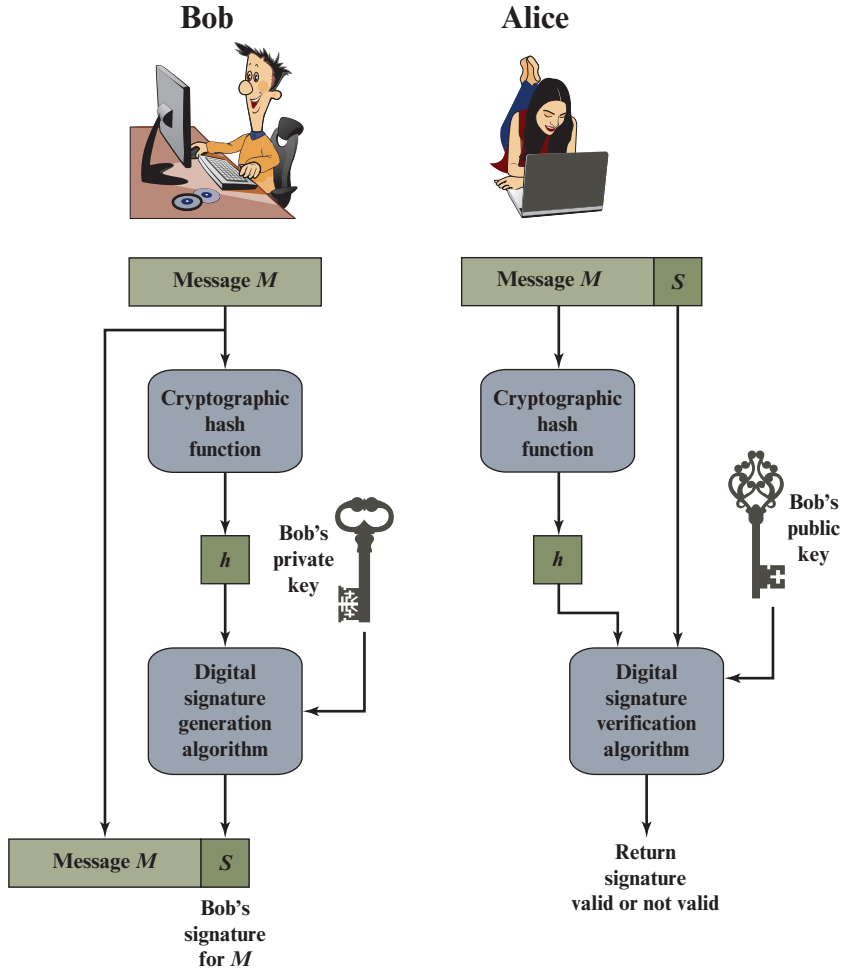


Figure 13.1 Simplified Depiction of Essential Elements of Digital Signature Process

13.1 DIGITAL SIGNATURES

Properties

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. Several forms of dispute between the two parties are possible.

For example, suppose that John sends an authenticated message to Mary, using one of the schemes of Figure 12.1. Consider the following disputes that could arise.

1. Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.
2. John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

Both scenarios are of legitimate concern. Here is an example of the first scenario: An electronic funds transfer takes place, and the receiver increases the amount of funds transferred and claims that the larger amount had arrived from the sender. An example of the second scenario is that an electronic mail message contains instructions to a stockbroker for a transaction that subsequently turns out badly. The sender pretends that the message was never sent.

In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature must have the following properties:

- It must verify the author and the date and time of the signature.
- It must authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

Attacks and Forgeries

[GOLD88] lists the following types of attacks, in order of increasing severity. Here A denotes the user whose signature method is being attacked, and C denotes the attacker.

- **Key-only attack:** C only knows A's public key.
- **Known message attack:** C is given access to a set of messages and their signatures.
- **Generic chosen message attack:** C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages. The attack is generic, because it does not depend on A's public key; the same attack is used against everyone.
- **Directed chosen message attack:** Similar to the generic attack, except that the list of messages to be signed is chosen after C knows A's public key but before any signatures are seen.
- **Adaptive chosen message attack:** C is allowed to use A as an "oracle." This means that C may request from A signatures of messages that depend on previously obtained message-signature pairs.

[GOLD88] then defines success at breaking a signature scheme as an outcome in which C can do any of the following with a non-negligible probability:

- **Total break:** C determines A's private key.
- **Universal forgery:** C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.
- **Selective forgery:** C forges a signature for a particular message chosen by C.
- **Existential forgery:** C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

Digital Signature Requirements

On the basis of the properties and attacks just discussed, we can formulate the following requirements for a digital signature.

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information only known to the sender to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

A secure hash function, embedded in a scheme such as that of Figure 13.1, provides a basis for satisfying these requirements. However, care must be taken in the design of the details of the scheme.

Direct Digital Signature

The term **direct digital signature** refers to a digital signature scheme that involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source.

Confidentiality can be provided by encrypting the entire message plus signature with a shared secret key (symmetric encryption). Note that it is important to perform the signature function first and then an outer confidentiality function. In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

The validity of the scheme just described depends on the security of the sender's private key. If a sender later wishes to deny sending a particular message, the sender can claim that the private key was lost or stolen and that someone else forged his or her signature. Administrative controls relating to the security of private keys

can be employed to thwart or at least weaken this ploy, but the threat is still there, at least to some degree. One example is to require every signed message to include a **timestamp** (date and time) and to require prompt reporting of compromised keys to a central authority.

Another threat is that a private key might actually be stolen from X at time T. The opponent can then send a message signed with X's signature and stamped with a time before or equal to T.

The universally accepted technique for dealing with these threats is the use of a digital certificate and certificate authorities. We defer a discussion of this topic until Chapter 14, and focus in this chapter on digital signature algorithms.

13.2 ELGAMAL DIGITAL SIGNATURE SCHEME

Before examining the NIST Digital Signature Algorithm, it will be helpful to understand the ElGamal and Schnorr signature schemes. Recall from Chapter 10, that the ElGamal encryption scheme is designed to enable encryption by a user's public key with decryption by the user's private key. The ElGamal signature scheme involves the use of the private key for digital signature generation and the public key for digital signature verification [ELGA84, ELGA85].

Before proceeding, we need a result from number theory. Recall from Chapter 2 that for a prime number q, if α is a primitive root of q, then

$$\alpha, \alpha^2, \ \ldots \ , \alpha^{q-1}$$

are distinct (mod q). It can be shown that, if α is a primitive root of q, then

1. For any integer m, $\alpha^m \equiv 1 \pmod{q}$ if and only if $m \equiv 0 \pmod{q-1}$.
2. For any integers, i, j, $\alpha^i \equiv \alpha^j \pmod{q}$ if and only if $i \equiv j \pmod{q-1}$.

As with ElGamal encryption, the global elements of **ElGamal digital signature** are a prime number q and α, which is a primitive root of q. User A generates a private/public key pair as follows.

1. Generate a random integer X_A, such that $1 < X_A < q - 1$.
2. Compute $Y_A = \alpha^{X_A} \bmod q$.
3. A's private key is X_A; A's pubic key is $\{q, \alpha, Y_A\}$.

To sign a message M, user A first computes the hash $m = \mathrm{H}(M)$, such that m is an integer in the range $0 \leq m \leq q - 1$. A then forms a digital signature as follows.

1. Choose a random integer K such that $1 \leq K \leq q - 1$ and $\gcd(K, q - 1) = 1$. That is, K is relatively prime to $q - 1$.
2. Compute $S_1 = \alpha^K \bmod q$. Note that this is the same as the computation of C_1 for ElGamal encryption.
3. Compute $K^{-1} \bmod (q - 1)$. That is, compute the inverse of K modulo $q - 1$.
4. Compute $S_2 = K^{-1}(m - X_A S_1) \bmod (q - 1)$.
5. The signature consists of the pair (S_1, S_2).

Any user B can verify the signature as follows.

1. Compute $V_1 = \alpha^m \bmod q$.
2. Compute $V_2 = (Y_A)^{S_1}(S_1)^{S_2} \bmod q$.

The signature is valid if $V_1 = V_2$. Let us demonstrate that this is so. Assume that the equality is true. Then we have

$\alpha^m \bmod q = (Y_A)^{S_1}(S_1)^{S_2} \bmod q$	assume $V_1 = V_2$
$\alpha^m \bmod q = \alpha^{X_A S_1}\alpha^{KS_2} \bmod q$	substituting for Y_A and S_1
$\alpha^{m-X_A S_1} \bmod q = \alpha^{KS_2} \bmod q$	rearranging terms
$m - X_A S_1 \equiv KS_2 \bmod (q-1)$	property of primitive roots
$m - X_A S_1 \equiv KK^{-1}(m - X_A S_1) \bmod (q-1)$	substituting for S_2

For example, let us start with the prime field GF(19); that is, $q = 19$. It has primitive roots {2, 3, 10, 13, 14, 15}, as shown in Table 2.7. We choose $\alpha = 10$.

Alice generates a key pair as follows:

1. Alice chooses $X_A = 16$.
2. Then $Y_A = \alpha^{X_A} \bmod q = \alpha^{16} \bmod 19 = 4$.
3. Alice's private key is 16; Alice's pubic key is $\{q, \alpha, Y_A\} = \{19, 10, 4\}$.

Suppose Alice wants to sign a message with hash value $m = 14$.

1. Alice chooses $K = 5$, which is relatively prime to $q - 1 = 18$.
2. $S_1 = \alpha^K \bmod q = 10^5 \bmod 19 = 3$ (see Table 2.7).
3. $K^{-1} \bmod (q-1) = 5^{-1} \bmod 18 = 11$.
4. $S_2 = K^{-1}(m - X_A S_1) \bmod (q-1) = 11\,(14 - (16)(3)) \bmod 18 = -374 \bmod 18 = 4$.

Bob can verify the signature as follows.

1. $V_1 = \alpha^m \bmod q = 10^{14} \bmod 19 = 16$.
2. $V_2 = (Y_A)^{S_1}(S_1)^{S_2} \bmod q = (4^3)(3^4) \bmod 19 = 5184 \bmod 19 = 16$.

Thus, the signature is valid because $V_1 = V_2$.

13.3 SCHNORR DIGITAL SIGNATURE SCHEME

As with the ElGamal digital signature scheme, the Schnorr signature scheme is based on discrete logarithms [SCHN89, SCHN91]. The Schnorr scheme minimizes the message-dependent amount of computation required to generate a signature. The main work for signature generation does not depend on the message and can be done during the idle time of the processor. The message-dependent part of the signature generation requires multiplying a $2n$-bit integer with an n-bit integer.

The scheme is based on using a prime modulus p, with $p - 1$ having a prime factor q of appropriate size; that is, $p - 1 \equiv 0 \pmod q$. Typically, we use $p \approx 2^{1024}$ and $q \approx 2^{160}$. Thus, p is a 1024-bit number, and q is a 160-bit number, which is also the length of the SHA-1 hash value.

The first part of this scheme is the generation of a private/public key pair, which consists of the following steps.

1. Choose primes p and q, such that q is a prime factor of $p - 1$.
2. Choose an integer a, such that $a^q \equiv 1 \bmod p$. The values a, p, and q comprise a global public key that can be common to a group of users.
3. Choose a random integer s with $0 < s < q$. This is the user's private key.
4. Calculate $v = a^{-s} \bmod p$. This is the user's public key.

A user with private key s and public key v generates a signature as follows.

1. Choose a random integer r with $0 < r < q$ and compute $x = a^r \bmod p$. This computation is a preprocessing stage independent of the message M to be signed.
2. Concatenate the message with x and hash the result to compute the value e:

$$e = \mathrm{H}(M \| x)$$

3. Compute $y = (r + se) \bmod q$. The signature consists of the pair (e, y).

Any other user can verify the signature as follows.

1. Compute $x' = a^y v^e \bmod p$.
2. Verify that $e = \mathrm{H}(M \| x')$.

To see that the verification works, observe that

$$x' \equiv a^y v^e \equiv a^y a^{-se} \equiv a^{y-se} \equiv a^r \equiv x \pmod p$$

Hence, $\mathrm{H}(M \| x') = \mathrm{H}(M \| x)$.

13.4 NIST DIGITAL SIGNATURE ALGORITHM

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the **Digital Signature Algorithm (DSA)**. The DSA makes use of the Secure Hash Algorithm (SHA) described in Chapter 12. The DSA was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2, subsequently updated to FIPS 186-3 in 2009, and FIPS 186-4 in 2013. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss DSA.

The DSA Approach

The DSA uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

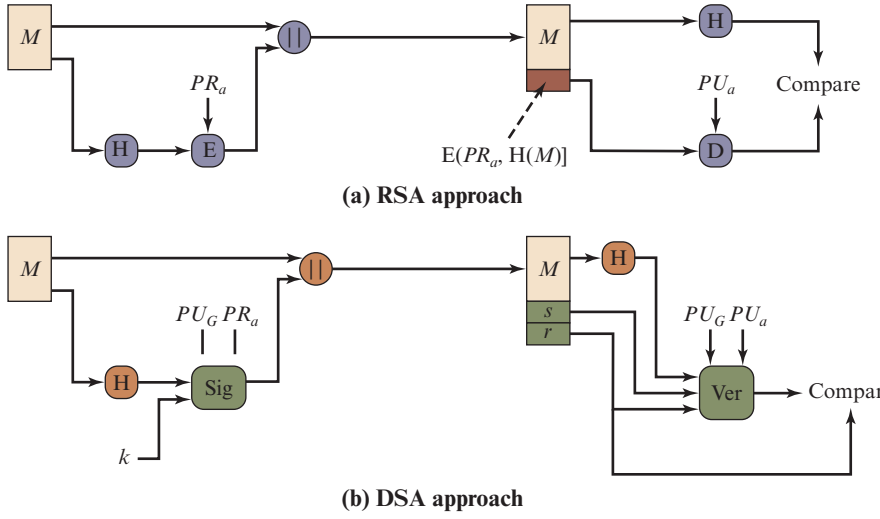


Figure 13.2 Two Approaches to Digital Signatures

Figure 13.2 contrasts the DSA approach for generating digital signatures to that used with RSA. In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

The DSA approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PR_a) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G).[1] The result is a signature consisting of two components, labeled s and r.

At the receiving end, the hash code of the incoming message is generated. The hash code and the signature are inputs to a verification function. The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

We turn now to the details of the algorithm.

[1]It is also possible to allow these additional parameters to vary with each user so that they are a part of a user's public key. In practice, it is more likely that a global public key will be used that is separate from each user's public key.

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64; i.e., bit length L between 512 and 1024 bits in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{N-1} < q < 2^N$ i.e., bit length of N bits

g $= h(p - 1)/q$ is an exponent mod p, where h is any integer with $1 < h < (p - 1)$ such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1}(H(M) + xr)] \bmod q$

Signature $= (r, s)$

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [H(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1}y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$

M = message to be signed

H(M) = hash of M using SHA-1

M', r', s' = received versions of M, r, s

Figure 13.3 The Digital Signature Algorithm (DSA)

The Digital Signature Algorithm

DSA is based on the difficulty of computing discrete logarithms (see Chapter 2) and is based on schemes originally presented by ElGamal [ELGA85] and Schnorr [SCHN91].

Figure 13.3 summarizes the algorithm. There are three parameters that are public and can be common to a group of users. An N-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides $(p - 1)$. Finally, g is chosen to be of the form $h^{(p-1)/q} \bmod p$, where h is an integer between 1 and $(p - 1)$ with the restriction that g must be greater than 1.[2] Thus, the global public-key components of DSA are the same as in the Schnorr signature scheme.

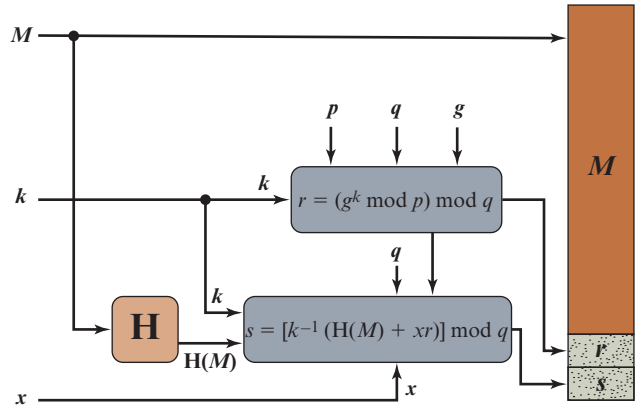
With these parameters in hand, each user selects a private key and generates a public key. The private key x must be a number from 1 to $(q - 1)$ and should be chosen randomly or pseudorandomly. The public key is calculated from the private key as $y = g^x \bmod p$. The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p (see Chapter 2).

[2] In number-theoretic terms, g is of order q mod p; see Chapter 2.

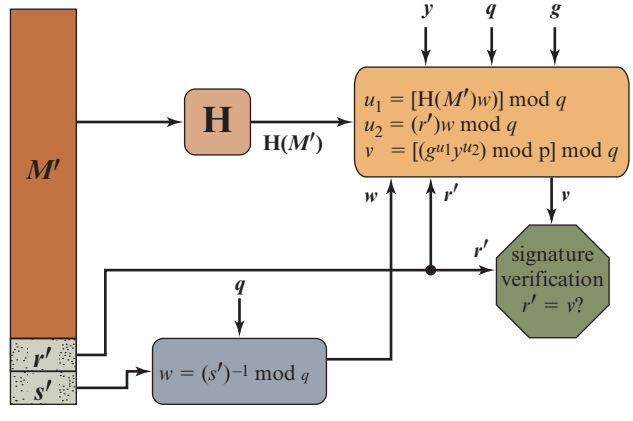
The signature of a message M consists of the pair of numbers r and s, which are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message $H(M)$, and an additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.

Let $M, r', \text{and } s'$ be the received versions of $M, r, \text{and } s$, respectively. Verification is performed using the formulas shown in Figure 13.3. The receiver generates a quantity v that is a function of the public key components, the sender's public key, the hash code of the incoming message, and the received versions of r and s. If this quantity matches the r component of the signature, then the signature is validated.

Figure 13.4 depicts the functions of signing and verifying.



(a) Signing



(b) Verifying

Figure 13.4 DSA Signing and Verifying

The structure of the algorithm, as revealed in Figure 13.4, is quite interesting. Note that the test at the end is on the value r, which does not depend on the message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of k (mod q) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious from Figure 13.3 or Figure 13.4 that such a scheme would work. A proof is provided in FIPS 186-4.

Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover k from r or to recover x from s.

Another point worth noting is that the only computationally demanding task in signature generation is the exponential calculation $g^k \bmod p$. Because this value does not depend on the message to be signed, it can be computed ahead of time. Indeed, a user could precalculate a number of values of r to be used to sign documents as needed. The only other somewhat demanding task is the determination of a multiplicative inverse, k^{-1}. Again, a number of these values can be precalculated.

13.5 ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM

As was mentioned, the 2009 version of FIPS 186 includes a new digital signature technique based on elliptic curve cryptography, known as the **Elliptic Curve Digital Signature Algorithm (ECDSA)**. ECDSA is enjoying increasing acceptance due to the efficiency advantage of elliptic curve cryptography, which yields security comparable to that of other schemes with a smaller key bit length.

First we give a brief overview of the process involved in ECDSA. In essence, four elements are involved.

1. All those participating in the digital signature scheme use the same global domain parameters, which define an elliptic curve and a point of origin on the curve.
2. A signer must first generate a public, private key pair. For the private key, the signer selects a random or pseudorandom number. Using that random number and the point of origin, the signer computes another point on the elliptic curve. This is the signer's public key.
3. A hash value is generated for the message to be signed. Using the private key, the domain parameters, and the hash value, a signature is generated. The signature consists of two integers, r and s.
4. To verify the signature, the verifier uses as input the signer's public key, the domain parameters, and the integer s. The output is a value v that is compared to r. The signature is verified if $v = r$.

Let us examine each of these four elements in turn.

Global Domain Parameters

Recall from Chapter 10 that two families of elliptic curves are used in cryptographic applications: prime curves over Z_p and binary curves over GF(2^m). For ECDSA, prime curves are used. The global domain parameters for ECDSA are the following:

q	a prime number
a, b	integers that specify the elliptic curve equation defined over Z_q with the equation $y^2 = x^3 + ax + b$
G	a base point represented by $G = (x_g, y_g)$ on the elliptic curve equation
n	order of point G; that is, n is the smallest positive integer such that $nG = O$. This is also the number of points on the curve.

Key Generation

Each signer must generate a pair of keys, one private and one public. The signer, let us call him Bob, generates the two keys using the following steps:

1. Select a random integer $d, d \in [1, n - 1]$
2. Compute $Q = dG$. This is a point in $E_q(a, b)$
3. Bob's public key is Q and private key is d.

Digital Signature Generation and Authentication

With the public domain parameters and a private key in hand, Bob generates a digital signature of 320 bits for message m using the following steps:

1. Select a random or pseudorandom integer $k, k \in [1, n - 1]$
2. Compute point $P = (x, y) = kG$ and $r = x \bmod n$. If $r = 0$ then go to step 1
3. Compute $t = k^{-1} \bmod n$
4. Compute $e = H(m)$, where H is one of the SHA-2 or SHA-3 hash functions
5. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = O$ then go to step 1
6. The signature of message m is the pair (r, s).

Alice knows the public domain parameters and Bob's public key. Alice is presented with Bob's message and digital signature and verifies the signature using the following steps:

1. Verify that r and s are integers in the range 1 through $n - 1$
2. Using SHA, compute the 160-bit hash value $e = H(m)$
3. Compute $w = s^{-1} \bmod n$
4. Compute $u_1 = ew$ and $u_2 = rw$
5. Compute the point $X = (x_1, y_1) = u_1G + u_2Q$
6. If $X = O$, reject the signature else compute $v = x_1 \bmod n$
7. Accept Bob's signature if and only if $v = r$

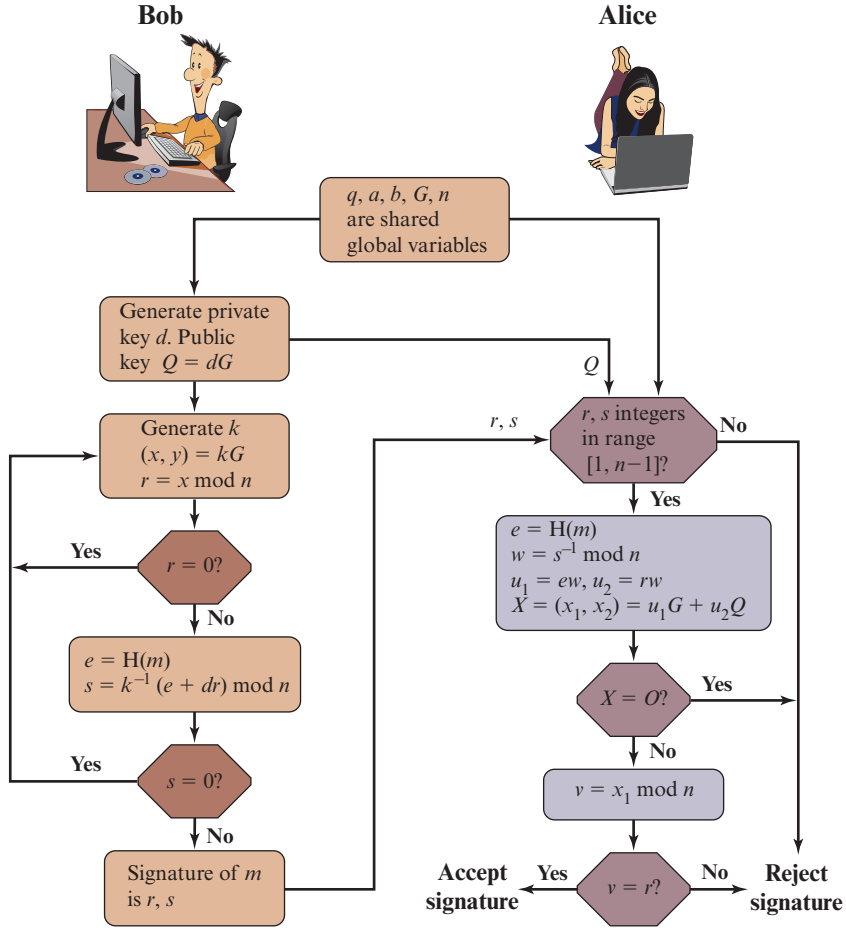


Figure 13.5 ECDSA Signing and Verifying

Figure 13.5 illustrates the signature authentication process. We can verify that this process is valid as follows. If the message received by Alice is in fact signed by Bob, then

$$s = k^{-1}(e + dr) \bmod n$$

Then

$$\begin{aligned} k &= s^{-1}(e + dr) \bmod n \\ k &= (s^{-1}e + s^{-1}dr) \bmod n \\ k &= (we + wdr) \bmod n \\ k &= (u_1 + u_2 d) \bmod n \end{aligned}$$

Now consider that

$$u_1 G + u_2 Q = u_1 G + u_2 dG = (u_1 + u_2 d)G = kG$$

In step 6 of the verification process, we have $v = x_1 \bmod n$, where point $X = (x_1, y_1) = u_1G + u_2Q$. Thus we see that $v = r$ since $r = x \bmod n$ and x is the x coordinate of the point kG and we have already seen that $u_1G + u_2Q = kG$.

13.6 RSA-PSS DIGITAL SIGNATURE ALGORITHM

In addition to the NIST Digital Signature Algorithm and ECDSA, the 2009 version of FIPS 186 also includes several techniques based on RSA, all of which were developed by RSA Laboratories and are in wide use. A worked-out example, using RSA, is available at this book's Web site.

In this section, we discuss the RSA Probabilistic Signature Scheme (RSA-PSS), which is the latest of the RSA schemes and the one that RSA Laboratories recommends as the most secure of the RSA schemes.

Because the RSA-based schemes are widely deployed in many applications, including financial applications, there has been great interest in demonstrating that such schemes are secure. The three main RSA signature schemes differ mainly in the padding format the signature generation operation employs to embed the hash value into a message representative, and in how the signature verification operation determines that the hash value and the message representative are consistent. For all of the schemes developed prior to PSS, it has not been possible to develop a mathematical proof that the signature scheme is as secure as the underlying RSA encryption/decryption primitive [KALI01]. The PSS approach was first proposed by Bellare and Rogaway [BELL96c, BELL98]. This approach, unlike the other RSA-based schemes, introduces a randomization process that enables the security of the method to be shown to be closely related to the security of the RSA algorithm itself. This makes RSA-PSS more desirable as the choice for RSA-based digital signature applications.

Mask Generation Function

Before explaining the RSA-PSS operation, we need to describe the mask generation function (MGF) used as a building block. MGF(X, *maskLen*) is a pseudorandom function that has as input parameters a bit string X of any length and the desired length L in octets of the output. MGFs are typically based on a secure cryptographic hash function such as SHA-1. An MGF based on a hash function is intended to be a cryptographically secure way of generating a message digest, or hash, of variable length based on an underlying cryptographic hash function that produces a fixed-length output.

The MGF function used in the current specification for RSA-PSS is MGF1, with the following parameters:

Options	Hash	hash function with output *hLen* octets
Input	X	octet string to be masked
	maskLen	length in octets of the mask
Output	*mask*	an octet string of length *maskLen*

MGF1 is defined as follows:

1. **Initialize variables.**

   ```
   T = empty string
   k = ⌈maskLen/hLen⌉ - 1
   ```

2. **Calculate intermediate values.**

   ```
   for counter = 0 to k
   Represent counter as a 32-bit string C
   T = T || Hash(X || C)
   ```

3. **Output results.**

   ```
   mask = the leading maskLen octets of T
   ```

In essence, MGF1 does the following. If the length of the desired output is equal to the length of the hash value ($maskLen = hLen$), then the output is the hash of the input value X concatenated with a 32-bit counter value of 0. If $maskLen$ is greater than $hLen$, the MGF1 keeps iterating by hashing X concatenated with the counter and appending that to the current string T. So that the output is

$$\text{Hash}\,(X\|0)\|\text{Hash}(X\|1)\|\ \ldots\ \|\text{Hash}(X\|k)$$

This is repeated until the length of T is greater than or equal to $maskLen$, at which point the output is the first $maskLen$ octets of T.

The Signing Operation

MESSAGE ENCODING The first stage in generating an RSA-PSS signature of a message M is to generate from M a fixed-length message digest, called an encoded message (EM). Figure 13.6 illustrates this process. We define the following parameters and functions:

Options	Hash	hash function with output $hLen$ octets. The current preferred alternative is SHA-1, which produces a 20-octet hash value.
	MGF	mask generation function. The current specification calls for MGF1.
	$sLen$	length in octets of a pseudorandom number referred to as the salt. Typically $sLen = hLen$, which for the current version is 20 octets.
Input	M	message to be encoded for signing.
	$emBits$	This value is one less than the length in bits of the RSA modulus n.
Output	EM	encoded message. This is the message digest that will be encrypted to form the digital signature.
Parameters	$emLen$	length of EM in octets $= \lceil emBits/8 \rceil$.
	padding_1	hexadecimal string 00 00 00 00 00 00 00 00; that is, a string of 64 zero bits.

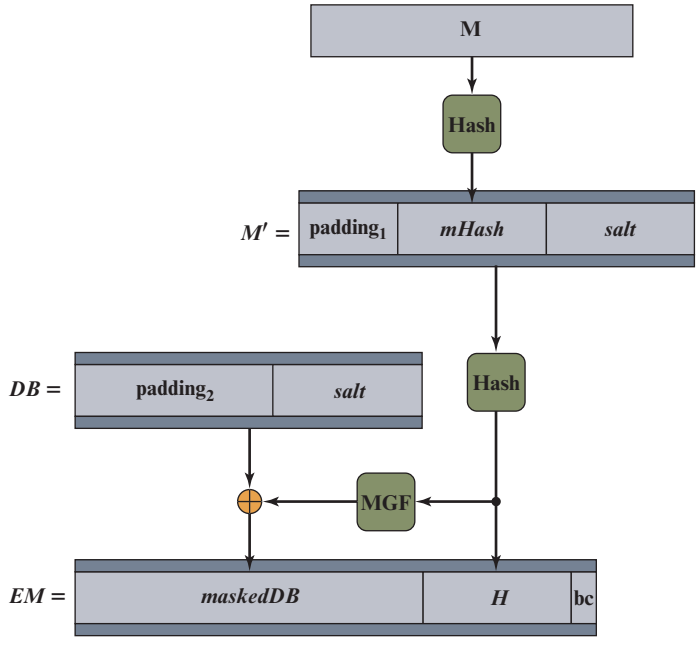


Figure 13.6 RSA-PSS Encoding

padding_2	hexadecimal string of 00 octets with a length ($emLen - sLen - hLen - 2$) octets, followed by the hexadecimal octet with value 01.
salt	a pseudorandom number.
bc	the hexadecimal value BC.

The encoding process consists of the following steps.

1. Generate the hash value of M: $mHash = \text{Hash}(M)$
2. Generate a pseudorandom octet string *salt* and form block $M' = \text{padding}_1 \| mHash \| salt$
3. Generate the hash value of M': $H = \text{Hash}(M')$
4. Form data block $DB = \text{padding}_2 \| salt$
5. Calculate the MGF value of H: $dbMask = \text{MGF}(H, emLen - hLen - 1)$
6. Calculate $maskedDB = DB \oplus dbMsk$
7. Set the leftmost $8emLen - emBits$ bits of the leftmost octet in $maskedDB$ to 0
8. $EM = maskedDB \| H \| 0\text{xbc}$

We make several comments about the complex nature of this message digest algorithm. All of the RSA-based standardized digital signature schemes involve appending one or more constants (e.g., padding_1 and padding_2) in the process of forming the message digest. The objective is to make it more difficult for an adversary to find another message that maps to the same message digest

as a given message or to find two messages that map to the same message digest. RSA-PSS also incorporates a pseudorandom number, namely the salt. Because the salt changes with every use, signing the same message twice using the same private key will yield two different signatures. This is an added measure of security.

Forming the Signature We now show how the signature is formed by a signer with private key $\{d, n\}$ and public key $\{e, n\}$ (see Figure 9.5). Treat the octet string EM as an unsigned, nonnegative binary integer m. The signature s is formed by encrypting m as follows:

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

Let k be the length in octets of the RSA modulus n. For example if the key size for RSA is 2048 bits, then $k = 2048/8 = 256$. Then convert the signature value s into the octet string S of length k octets.

Signature Verification

Decryption For signature verification, treat the signature S as an unsigned, nonnegative binary integer s. The message digest m is recovered by decrypting s as follows:

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

Then, convert the message representative m to an encoded message EM of length $emLen = \lceil (modBits - 1)/8 \rceil$ octets, where $modBits$ is the length in bits of the RSA modulus n.

EM Verification EM verification can be described as follows:

Options	Hash	hash function with output $hLen$ octets.
	MGF	mask generation function.
	$sLen$	length in octets of the salt.
Input	M	message to be verified.
	EM	the octet string representing the decrypted signature, with length $emLen = \lceil emBits/8 \rceil$.
	$emBits$	This value is one less than the length in bits of the RSA modulus n.
Parameters	padding_1	hexadecimal string 00 00 00 00 00 00 00 00; that is, a string of 64 zero bits.
	padding_2	hexadecimal string of 00 octets with a length $(emLen - sLen - hLen - 2)$ octets, followed by the hexadecimal octet with value 01.

1. Generate the hash value of M: $mHash = \text{Hash}(M)$
2. If $emLen < hLen + sLen + 2$, output "inconsistent" and stop
3. If the rightmost octet of EM does not have hexadecimal value BC, output "inconsistent" and stop

4. Let *maskedDB* be the leftmost $emLen - hLen - 1$ octets of EM, and let H be the next $hLen$ octets
5. If the leftmost $8emLen - emBits$ bits of the leftmost octet in *maskedDB* are not all equal to zero, output "inconsistent" and stop
6. Calculate $dbMask = \text{MGF}(H, emLen - hLen - 1)$
7. Calculate $DB = maskedDB \oplus dbMsk$
8. Set the leftmost $8emLen - emBits$ bits of the leftmost octet in DB to zero
9. If the leftmost $(emLen - hLen - sLen - 1)$ octets of DB are not equal to padding_2, output "inconsistent" and stop
10. Let *salt* be the last $sLen$ octets of DB
11. Form block $M' = \text{padding}_1 \| mHash \| salt$
12. Generate the hash value of M': $\text{H}' = \text{Hash}(M')$
13. If $H = H'$, output "consistent." Otherwise, output "inconsistent"

Figure 13.7 illustrates the process. The shaded boxes labeled H and H' correspond, respectively, to the value contained in the decrypted signature and the value generated from the message M associated with the signature. The remaining three shaded areas contain values generated from the decrypted signature and compared to known constants. We can now see more clearly the different roles played by the constants and the pseudorandom value *salt*, all of which are embedded in the

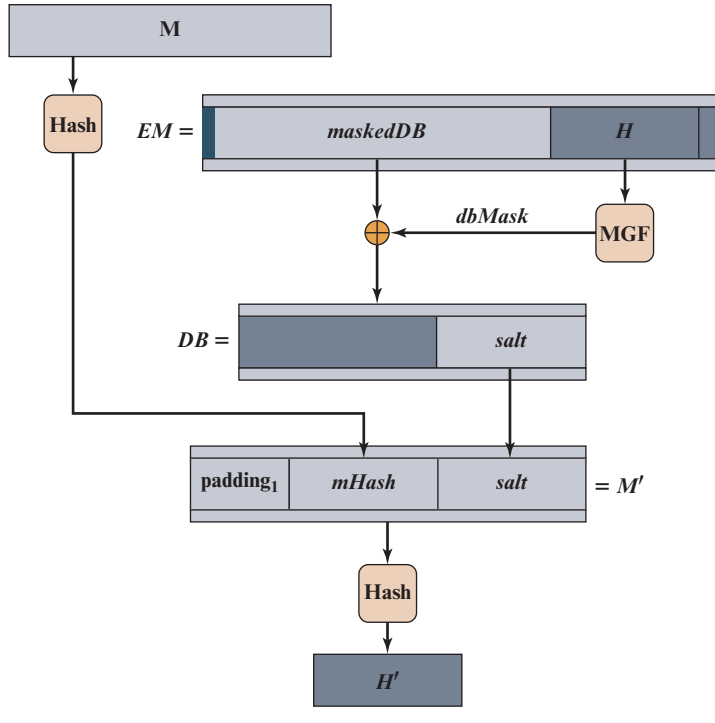


Figure 13.7 RSA-PSS EM Verification

EM generated by the signer. The constants are known to the verifier, so that the computed constants can be compared to the known constants as an additional check that the signature is valid (in addition to comparing H and H'). The salt results in a different signature every time a given message is signed with the same private key. The verifier does not know the value of the salt and does not attempt a comparison. Thus, the salt plays a similar role to the pseudorandom variable k in the NIST DSA and in ECDSA. In both of those schemes, k is a pseudorandom number generated by the signer, resulting in different signatures from multiple signings of the same message with the same private key. A verifier does not and need not know the value of k.

13.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

digital signature	ElGamal digital signature	timestamp
Digital Signature Algorithm (DSA)	Elliptic Curve Digital Signature Algorithm (ECDSA)	
direct digital signature		

Review Questions

13.1 List two disputes that can arise in the context of message authentication.

13.2 What are the properties a digital signature should have?

13.3 What requirements should a digital signature scheme satisfy?

13.4 What is the difference between direct and arbitrated digital signature?

13.5 In what order should the signature function and the confidentiality function be applied to a message, and why?

13.6 What are some threats associated with a direct digital signature scheme?

Problems

13.1 Dr. Watson patiently waited until Sherlock Holmes finished. "Some interesting problem to solve, Holmes?" he asked when Holmes finally logged out.

"Oh, not exactly. I merely checked my email and then made a couple of network experiments instead of my usual chemical ones. I have only one client now and I have already solved his problem. If I remember correctly, you once mentioned cryptology among your other hobbies, so it may interest you."

"Well, I am only an amateur cryptologist, Holmes. But of course I am interested in the problem. What is it about?"

"My client is Mr. Hosgrave, director of a small but progressive bank. The bank is fully computerized and of course uses network communications extensively. The bank already uses RSA to protect its data and to digitally sign documents that are communicated. Now the bank wants to introduce some changes in its procedures; in particular, it needs to digitally sign some documents by *two* signatories.

1. The first signatory prepares the document, forms its signature, and passes the document to the second signatory.

2. The second signatory as a first step must verify that the document was really signed by the first signatory. She then incorporates her signature into the document's signature so that the recipient, as well as any member of the public, may verify that the document was indeed signed by both signatories. In addition, only the second signatory has to be able to verify the document's signature after the first step; that is, the recipient (or any member of the public) should be able to verify only the complete document with signatures of both signatories, but not the document in its intermediate form where only one signatory has signed it. Moreover, the bank would like to make use of its existing modules that support RSA-style digital signatures."

"Hm, I understand how RSA can be used to digitally sign documents by *one* signatory, Holmes. I guess you have solved the problem of Mr. Hosgrave by appropriate generalization of RSA digital signatures."

"Exactly, Watson," nodded Sherlock Holmes. "Originally, the RSA digital signature was formed by encrypting the document by the signatory's private decryption key 'd', and the signature could be verified by anyone through its decryption using publicly known encryption key 'e'. One can verify that the signature S was formed by the person who knows d, which is supposed to be the only signatory. Now the problem of Mr. Hosgrave can be solved in the same way by slight generalization of the process, that is …"

Finish the explanation.

13.2 DSA specifies that if the signature generation process results in a value of $s = 0$, a new value of k should be generated and the signature should be recalculated. Why?

13.3 What happens if a k value used in creating a DSA signature is compromised?

13.4 The DSA document includes a recommended algorithm for testing a number for primality.

1. **[Choose w]** Let w be a random odd integer. Then $(w - 1)$ is even and can be expressed in the form $2^a m$ with m odd. That is, 2^a is the largest power of 2 that divides $(w - 1)$.
2. **[Generate b]** Let b be a random integer in the range $1 < b < w$.
3. **[Exponentiate]** Set $j = 0$ and $z = b^m \bmod w$.
4. **[Done?]** If $j = 0$ and $z = 1$, or if $z = w - 1$, then w passes the test and may be prime; go to step 8.
5. **[Terminate?]** If $j > 0$ and $z = 1$, then w is not prime; terminate algorithm for this w.
6. **[Increase j]** Set $j = j + 1$. If $j < a$, set $z = z^2 \bmod w$ and go to step 4.
7. **[Terminate]** w is not prime; terminate algorithm for this w.
8. **[Test again?]** If enough random values of b have been tested, then accept w as prime and terminate algorithm; otherwise, go to step 2.
   a. Explain how the algorithm works.
   b. Show that it is equivalent to the Miller–Rabin test described in Chapter 2.

13.5 With DSA, because the value of k is generated for each signature, even if the same message is signed twice on different occasions, the signatures will differ. This is not true of RSA signatures. What is the practical implication of this difference?

13.6 Consider the problem of creating domain parameters for DSA. Suppose we have already found primes p and q such that $q | (p - 1)$. Now we need to find $g \in Z_p$ with g of order q mod p. Consider the following two algorithms:

Algorithm 1	Algorithm 2
repeat	**repeat**
select $g \in Z_p$	**select** $h \in Z_p$
$h \leftarrow g^q \bmod p$	$g \leftarrow h^{(p-1)/q} \bmod p$
until $(h = 1$ and $g \neq 1)$	**until** $(g \neq 1)$
return g	**return** g

a. What happens in Algorithm 1 if $ord(g) = q$ is chosen?
b. What happens in Algorithm 2 if $ord(g) = q$ is chosen?
c. Suppose $p = 64891$ and $q = 421$. How many loop iterations do you expect Algorithm 1 to make before it finds a generator?
d. If p is 512 bits and q is 128 bits, would you recommend using Algorithm 1 to find g? Explain.
e. Suppose $p = 64891$ and $q = 421$. What is the probability that Algorithm 2 computes a generator in its very first loop iteration? (If it is helpful, you may use the fact that $\sum_{(d|n)} \psi(d) = n$ when answering this question.)

13.7 It is tempting to try to develop a variation on Diffie–Hellman that could be used as a digital signature. Here is one that is simpler than DSA and that does not require a secret random number in addition to the private key.

Public elements:	q	prime number
	α	$\alpha < q$ and α is primitive root of q
Private key:	X	$X < q$
Public key:	$Y = \alpha^X \bmod q \bmod q$	

To sign a message M, compute $h = H(M)$, which is the hash code of the message. We require that $\gcd(h, q - 1) = 1$. If not, append the hash to the message and calculate a new hash. Continue this process until a hash code is produced that is relatively prime to $(q - 1)$. Then, calculate Z to satisfy $Z \equiv X \times h \bmod (q - 1)$. The signature of the message is $\sigma = \alpha^Z$. To verify the signature, a user computes t such that $t \times h = 1 \bmod (q - 1)$ and verifies $Y = \sigma^t \bmod q$.
a. Show that this scheme works. That is, show that the verification process produces an equality if the signature is valid.
b. Show that the scheme is unacceptable by describing a simple technique for forging a user's signature on an arbitrary message.

13.8 Assume a technique for a digital signature scheme using a cryptographic one-way hash function (H) as follows. To sign an n-bit message, the sender randomly generates in advance $2n$ 64-bit cryptographic keys: $k_1, k_2, \ldots, k_n\, k'_1, k'_2, \ldots, k'_n$ which are kept private. The sender generates the following two sets of validation parameters, which are made public.

$$v_1, v_2, \ldots, v_n \text{ and } v'_1, v'_2, \ldots, v'_n$$

where

$$v_i = H(k_i \| 0), v'_i = H(k'_i \| 1)$$

The user sends the appropriate k_i or k'_i according to whether M_i is 0 or 1, respectively. For example, if the first 3 bits of the message are 011, then the first three keys of the signature are k_1, k'_2, and k'_3.
a. How does the receiver validate the message?
b. Is the technique secure?
c. How many times can the same set of secret keys be safely used for different messages?
d. What, if any, practical problems does this scheme present?