messages being chosen at random for checking. 3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources. However, if a message authentication tag were attached to the program, it could be checked whenever assurance is required of the integrity of the program. 3.1 / Approaches to Message Authentication 81 Thus, there is a place for both authentication and encryption in meeting security requirements. Message Authentication Code One authentication technique involves the use of a secret key to generate a small block of data, known as a message authentication code (MAC), that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key KAB. When A has a message to send to B, it calculates the message authentication code as a function of the message and the key: MACM = F(KAB, M). The message plus code are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new message authentication code. The received code is compared to the calculated code (Figure 3.1). If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then the following statements apply: 1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the code, then the receiver's calculation of the code will differ from the received code. Because the attacker is assumed not to know the secret key, the attacker cannot alter the code to correspond to the alterations in the message. Figure 3.1 Message Authentication Using a Message Authentication Code MAC algorithm MAC algorithm MAC K K Compare Message Transmit 82 chapter 3 / Public-Key Cryptography and Message Authentication 2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper code. 3. If the message includes a sequence number (such as is used with HDLC and TCP), then the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number. A number of algorithms could be used to generate the code. The NIST specification, FIPS PUB 113, recommends the use of DES. DES is used to generate an encrypted version of the message, and the last number of bits of ciphertext are used as the code. A 16- or 32-bit code is typical. The process just described is similar to encryption. One difference is that the authentication algorithm need not be reversible, as it must for decryption. Because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption. One-Way Hash Function An alternative to the message authentication code is the one-way hash function. As with the message authentication code, a hash function accepts a variable-size message M as input and produces a fixed-size message digest H(M) as output. Unlike the MAC, a hash function does not take a secret key as input. To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic. Figure 3.2 illustrates three ways in which the message can be authenticated. The message digest can be encrypted using conventional encryption (part a); if it is assumed that only the sender and receiver share the encryption key, then authenticity is assured. The message digest can be encrypted using public-key encryption (part b); this is explained in Section 3.5. The public-key approach has two advantages: (1) It provides a digital signature as well as message authentication. (2) It does not require the distribution of keys to communicating parties. These two approaches also have an advantage over approaches that encrypt the entire message in that less computation is required. Nevertheless, there has been interest in developing a technique that avoids encryption altogether. Several reasons for this interest are pointed out in [TSUD92]: ■ Encryption software is quite slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system. ■ Encryption hardware costs are nonnegligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability. ■ Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead. ■ An encryption algorithm may be protected by a patent. Figure 3.2c shows a technique that uses a hash function but no encryption for message authentication. This technique assumes that two communicating parties, say A and B, share a common secret value SAB. When A has a message to send to B, it calculates the hash function over the concatenation of the secret value 3.1 / Approaches to Message Authentication 83 and the message: MDM = H(SAB }M).2 It then sends [M}MDM] to B. Because B possesses SAB, it can recompute H(SAB }M) and verify MDM. Because the secret value itself is not sent, it is not possible for an attacker to modify an intercepted message. As long as the secret value remains secret, it is also not possible for an attacker to generate a false message. 2 } denotes concatenation. Figure 3.2 Message Authentication Using a One-Way Hash Function Source A Destination B Message Message Compare Message H H E (a) Using conventional encryption K D K Message Message Message Compare Message H H H E S S (b) Using public-key encryption (c) Using secret value PRa PUa Compare H D Message Message 84 chapter 3 / Public-Key Cryptography and Message Authentication A variation on the third technique, called HMAC, is the one adopted for IP security (described in Chapter 9); it also has been specified for SNMPv3 (Chapter 13). 3.2 Secure Hash Functions The one-way hash function, or secure hash function, is important not only in message authentication but in digital signatures. In this section, we begin with a discussion of requirements for a secure hash function. Then we look at the most important hash function, SHA. Hash Function Requirements The purpose of a hash function is to produce a “fingerprint” of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the following properties: 1. H can be applied to a block of data of any size. 2. H produces a fixed-length output. 3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical. 4. For any given code h, it is computationally infeasible to find x such that H(x) = h. A hash function with this property is referred to as one-way or preimage resistant. 3 5. For any given block x, it is computationally infeasible to find y ≠ x with H(y) = H(x). A hash function with this property is referred to as second pre-image resistant. This is sometimes referred to as weak collision resistant. 6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). A hash function with this property is referred to as collision resistant. This is sometimes referred to as strong collision resistant. The first three properties are requirements for the practical application of a hash function to message authentication. The fourth property, preimage resistant, is the “one-way” property: It is easy to generate a code given a message, but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value (Figure 3.2c). The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message M and the hash code C = H(SAB }M). The attacker then inverts the hash function to obtain SAB }M = H-1 (C). Because the attacker now has both M and SAB }M, it is a trivial matter to recover SAB. The second preimage resistant property guarantees that it is impossible to find an alternative message with the same hash value as a given message. This prevents 3 For f(x) = y, x is said to be a preimage of y. Unless f is one-to-one, there may be multiple preimage values for a given y. 3.2 / Secure Hash Functions 85 forgery when an encrypted hash code is used (Figures 3.2a and b). If this property were not true, an attacker would be capable of the following sequence: First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code. A hash function that satisfies the first five properties in the preceding list is referred to as a weak hash function. If the sixth property is also satisfied, then it is referred to as a strong hash function. The sixth property, collision resistant, protects against a sophisticated class of attack known as the birthday attack. Details of this attack are beyond the scope of this book. The attack reduces the strength of an m-bit hash function from 2m to 2m/2. See [STAL13] for details. In addition to providing authentication, a message digest also provides data integrity. It performs the same function as a frame check sequence: If any bits in the message are accidentally altered in transit, the message digest will be in error. Security of Hash Functions As with symmetric encryption, there are two approaches to attacking a secure hash function: cryptanalysis and brute-force attack. As with symmetric encryption algorithms, cryptanalysis of a hash function involves exploiting logical weaknesses in the algorithm. The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. For a hash code of length n, the level of effort required is proportional to the following: Preimage resistant 2n Second preimage resistant 2n Collision resistant 2n/2 If collision resistance is required (and this is desirable for a general-purpose secure hash code), then the value 2n/2 determines the strength of the hash code against brute-force attacks. Van Oorschot and Wiener [VANO94] presented a design for a $10 million collision search machine for MD5, which has a 128-bit hash length, that could find a collision in 24 days. Thus, a 128-bit code may be viewed as inadequate. The next step up, if a hash code is treated as a sequence of 32 bits, is a 160-bit hash length. With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision. With today’s technology, the time would be much shorter, so that 160 bits now appears suspect. Simple Hash Functions All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n-bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function. One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as Ci = bi1 ⊕ bi2 ⊕ g ⊕ bim 86 chapter 3 / Public-Key Cryptography and Message Authentication where Ci = ith bit of the hash code, 1 … i … n m = number of n-bit blocks in the input bij = ith bit in jth block ⊕ = XOR operation Figure 3.3 illustrates this operation; it produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2-n . With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2-128, the hash function on this type of data has an effectiveness of 2-112. A simple way to improve matters is to perform a 1-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as 1. Initially set the n-bit hash value to zero. 2. Process each successive n-bit block of data: a. Rotate the current hash value to the left by one bit. b. XOR the block into the hash value. This has the effect of “randomizing” the input more completely and overcoming any regularities that appear in the input. Although the second procedure provides a good measure of data integrity, it is virtually useless for data security when an encrypted hash code is used with a plaintext message, as in Figures 3.2a and b. Given a message, it is an easy matter to produce a new message that yields that hash code: Simply prepare the desired alternate message and then append an n-bit block that forces the combined new message plus block to yield the desired hash code. Although a simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted, you may still feel that such a simple function could be useful when the message as well as the hash code are encrypted. But one must be careful. A technique originally proposed by the National Bureau of Standards used Figure 3.3 Simple Hash Function Using Bitwise XOR bit 1 Block 1 Block 2 Block m Hash code b11 b21 bn1 bn2 bnm b22 b2m b12 b1m C1 C2 Cn bit 2 bit n 3.2 / Secure Hash Functions 87 the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message using the cipher block chaining (CBC) mode. We can define the scheme as follows: Given a message consisting of a sequence of 64-bit blocks X1, X2, c , XN, define the hash code C as the block-by-block XOR or all blocks and append the hash code as the final block: C = XN+1 = X1 ⊕ X2 ⊕ g ⊕ XN Next, encrypt the entire message plus hash code using CBC mode to produce the encrypted message Y1, Y2, c , YN+1. [JUEN85] points out several ways in which the ciphertext of this message can be manipulated in such a way that it is not detectable by the hash code. For example, by the definition of CBC (Figure 2.9), we have X1 = IV ⊕ D(K, Y1) Xi = Yi-1 ⊕ D(K, Yi) XN+1 = YN ⊕ D(K, YN+1) But XN+1 is the hash code: XN+1 = X1 ⊕ X2 ⊕ g ⊕ XN = [IV ⊕ D(K, Y1)] ⊕ [Y1 ⊕ D(K, Y2)] ⊕ g ⊕ [YN-1 ⊕ D(K, YN)] Because the terms in the preceding equation can be XORed in any order, it follows that the hash code would not change if the ciphertext blocks were permuted. The SHA Secure Hash Function In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA). Indeed, because virtually every other widely used hash function had been found to have substantial cryptanalytic weaknesses, SHA was more or less the last remaining standardized hash algorithm by 2005. SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993. When weaknesses were discovered in SHA (now known as SHA-0), a revised version was issued as FIPS 180-1 in 1995 and is referred to as SHA-1. The actual standards document is entitled “Secure Hash Standard.” SHA is based on the hash function MD4, and its design closely models MD4. SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA with hash value lengths of 256, 384, and 512 bits known as SHA-256, SHA-384, and SHA-512, respectively. Collectively, these hash algorithms are known as SHA-2. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. A revised document was issued as FIP PUB 180-3 in 2008, which added a 224-bit version (Table 3.1). SHA-1 and SHA-2 are also specified in RFC 6234, which essentially duplicates the material in FIPS 180-3 but adds a C code implementation. In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on SHA-2 by 2010. Shortly thereafter, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 269 operations, far fewer than the 280 operations previously 88 chapter 3 / Public-Key Cryptography and Message Authentication thought needed to find a collision with an SHA-1 hash [WANG05]. This result should hasten the transition to SHA-2. In this section, we provide a description of SHA-512. The other versions are quite similar. The algorithm takes as input a message with a maximum length of less than 2128 bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks. Figure 3.4 depicts the overall processing of a message to produce a digest. The processing consists of the following steps. Step 1 Append padding bits: The message is padded so that its length is congruent to 896 modulo 1024 [length K 896 (mod 1024)]. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024. The padding consists of a single 1 bit followed by the necessary number of 0 bits. Step 2 Append length: A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding). The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure 3.4, the expanded message is represented as the sequence of 1024-bit blocks M1, M2, c , MN, so that the total length of the expanded message is N * 1024 bits. Step 3 Initialize hash buffer: A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These registers are initialized to the following 64-bit integers (hexadecimal values): a = 6A09E667F3BCC908 e = 510E527FADE682D1 b = BB67AE8584CAA73B f = 9B05688C2B3E6C1F c = 3C6EF372FE94F82B g = 1F83D9ABFB41BD6B d = A54FF53A5F1D36F1 h = 5BE0CD19137E2179 These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers. SHA-1 SHA-224 SHA-256 SHA-384 SHA-512 Message Digest Size 160 224 256 384 512 Message Size 6264 6264 6264 62128 62128 Block Size 512 512 512 1024 1024 Word Size 32 32 32 64 64 Number of Steps 80 64 64 80 80 Note: All sizes are measured in bits. Table 3.1 Comparison of SHA Parameters 3.2 / Secure Hash Functions 89 Step 4 Process message in 1024-bit (128-word) blocks: The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 3.4. The logic is illustrated in Figure 3.5. Each round takes as input the 512-bit buffer value abcdefgh and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, Hi-1. Each round t makes use of a 64-bit value Wt derived from the current 1024-bit block being processed (Mi). Each round also makes use of an additive constant Kt , where 0 ... t ... 79 indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. The constants provide a “randomized” set of 64-bit patterns, which should eliminate any regularities in the input data. The output of the 80th round is added to the input to the first round (Hi-1) to produce Hi . The addition is done independently for each of the eight words in the buffer with each of the corresponding words in Hi-1, using addition modulo 264. Step 5 Output: After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest. Figure 3.4 Message Digest Generation Using SHA-512 N × 1024 bits M1 M2 H2 MN Message hash code 1024 bits 1024 bits 1024 bits L bits L 128 bits 512 bits 512 bits 512 bits 1000000..0 + = word-by-word addition mod 264 H1 F + IV = H0 F + F + HN 90 chapter 3 / Public-Key Cryptography and Message Authentication The SHA-512 algorithm has the property that every bit of the hash code is a function of every bit of the input. The complex repetition of the basic function F produces results that are well mixed; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code. Unless there is some hidden weakness in SHA-512, which has not so far been published, the difficulty of coming up with two messages having the same message digest is on the order of 2256 operations, while the difficulty of finding a message with a given digest is on the order of 2512 operations. SHA-3 SHA-2, particularly the 512-bit version, would appear to provide unassailable security. However, SHA-2 shares the same structure and mathematical operations as its predecessors, and this is a cause for concern. Because it would take years to find a suitable replacement for SHA-2, should it become vulnerable, NIST announced in 2007 a competition to produce the next-generation NIST hash function, which is to be called SHA-3. Following are the basic requirements that must be satisfied by any candidate for SHA-3: Figure 3.5 SHA-512 Processing of a Single 1024-Bit Block 64 Mi Wt Hi Hi–1 W0 W79 Kt K0 K79 a b c Round 0 d e f g h a b c Round t d e f g h Message schedule a b c Round 79 d e f g h + + + + + + + + 3.3 / Message Authentication Codes 91 1. It must be possible to replace SHA-2 with SHA-3 in any application by a simple drop-in substitution. Therefore, SHA-3 must support hash value lengths of 224, 256, 384, and 512 bits. 2. SHA-3 must preserve the online nature of SHA-2. That is, the algorithm must process comparatively small blocks (512 or 1024 bits) at a time instead of requiring that the entire message be buffered in memory before processing it. In 2012, NIST selected a winning submission and formally published SHA-3. A detailed presentation of SHA-3 is provided in Chapter 15. 3.3 Message Authentication Codes HMAC In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash code, such as SHA-1. The motivations for this interest are as follows: ■ Cryptographic hash functions generally execute faster in software than conventional encryption algorithms such as DES. ■ Library code for cryptographic hash functions is widely available. A hash function such as SHA-1 was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The approach that has received the most support is HMAC [BELL96a, BELL96b]. HMAC has been issued as RFC 2104, has been chosen as the mandatoryto-implement MAC for IP Security, and is used in other Internet protocols, such as Transport Layer Security (TLS) and Secure Electronic Transaction (SET). HMAC Design Objectives RFC 2104 lists the following design objectives for HMAC. ■ To use, without modifications, available hash functions. In particular, hash functions that perform well in software, and for which code is freely and widely available ■ To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required ■ To preserve the original performance of the hash function without incurring a significant degradation ■ To use and handle keys in a simple way ■ To have a well-understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the embedded hash function The first two objectives are important to the acceptability of HMAC. HMAC treats the hash function as a “black box.” This has two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC. 92 chapter 3 / Public-Key Cryptography and Message Authentication In this way, the bulk of the HMAC code is prepackaged and ready to use without modification. Second, if it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module. This could be done if a faster hash function were desired. More important, if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one. The last design objective in the preceding list is, in fact, the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths. We return to this point later in this section, but first we examine the structure of HMAC. Hmac Algorithm Figure 3.6 illustrates the overall operation of HMAC. The following terms are defined: H = embedded hash function (e.g., SHA-1) M = message input to HMAC (including the padding specified in the embedded hash function) Figure 3.6 HMAC Structure K+ Si So Y0 Y1 YL–1 b bits b bits b bits b bits ipad K+ opad IV Hash n bits n bits pad to b bits IV Hash n bits n bits HMAC(K, M) H(Si || M) 3.3 / Message Authentication Codes 93 Yi = ith block of M, 0 ... i ... (L - 1) L = number of blocks in M b = number of bits in a block n = length of hash code produced by embedded hash function K = secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length is 7 n K+ = K padded with zeros on the left so that the result is b bits in length ipad = 00110110 (36 in hexadecimal) repeated b/8 times opad = 01011100 (5C in hexadecimal) repeated b/8 times Then HMAC can be expressed as HMAC(K, M) = H[(K+ ⊕ opad) } H[(K+ ⊕ ipad) }M]] In words, HMAC is defined as follows: 1. Append zeros to the left end of K to create a b-bit string K+ (e.g., if K is of length 160 bits and b = 512, then K will be appended with 44 zero bytes). 2. XOR (bitwise exclusive-OR) K+ with ipad to produce the b-bit block Si . 3. Append M to Si . 4. Apply H to the stream generated in step 3. 5. XOR K+ with opad to produce the b-bit block So. 6. Append the hash result from step 4 to So. 7. Apply H to the stream generated in step 6 and output the result.

Note that the XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. In effect, by passing Si and So through the hash algorithm, we have pseudorandomly generated two keys from K. HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the basic hash function (for Si , So, and the block produced from the inner hash). MACs Based on Block Ciphers In this section, we look at several MACs based on the use of a block cipher. Cipher-Based Message Authentication Code (CMAC) The Cipher-based Message Authentication Code mode of operation is for use with AES and triple DES. It is specified in SP 800-38B. First, let us consider the operation of CMAC when the message is an integer multiple n of the cipher block length b. For AES, b = 128, and for triple DES, b = 64. The message is divided into n blocks (M1, M2, c , Mn). The algorithm makes use of a k-bit encryption key K and an n-bit key, K1. For AES, the key size k is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows (Figure 3.7). 94 chapter 3 / Public-Key Cryptography and Message Authentication $C1 = E(K, M1)$ $C2 = E(K, [M2 \oplus C1])$ $C3 = E(K, [M3 \oplus C2])$ # # # $Cn = E(K, [MN \oplus Cn\text{-}1 \oplus K1])$ T = MSBTlen(Cn) where T = message authentication code, also referred to as the tag Tlen = bit length of T MSBs(X) = the s leftmost bits of the bit string X If the message is not an integer multiple of the cipher block length, then the final block is padded to the right (least significant bits) with a 1 and as many Figure 3.7 Cipher-Based Message Authentication Code (CMAC) K K Encrypt K T Encrypt Encrypt MSB(Tlen) M1 K1 K2 M2 Mn (a) Message length is integer multiple of block size K K Encrypt K T Encrypt Encrypt MSB(Tlen) 10...0 (b) Message length is not integer multiple of block size b k M1 M2 Mn 3.3 / Message Authentication Codes 95 0s as necessary so that the final block is also of length b. The CMAC operation then proceeds as before, except that a different n-bit key K2 is used instead of K1. To generate the two n-bit keys, the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting ciphertext by a left shift of one bit and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey. Counter with Cipher Block Chaining-Message Authentication Code The Counter with Cipher Block Chaining-Message Authentication Code (CCM) mode of operation, defined in SP 800-38C, is referred to as an authenticated encryption mode. “Authenticated encryption” is a term used to describe encryption systems that simultaneously protect confidentiality and authenticity (integrity) of communications. Many applications and protocols require both forms of security, but until recently the two services have been designed separately. The key algorithmic ingredients of CCM are the AES encryption algorithm (Section 2.2), the CTR mode of operation (Section 2.5), and the CMAC authentication algorithm. A single key K is used for both encryption and MAC algorithms. The input to the CCM encryption process consists of three elements. 1. Data that will be both authenticated and encrypted. This is the plaintext message P of data block. 2. Associated data A that will be authenticated but not encrypted. An example is a protocol header that must be transmitted in the clear for proper protocol operation but which needs to be authenticated. 3. A nonce N that is assigned to the payload and the associated data. This is a unique value that is different for every instance during the lifetime of a protocol association and is intended to prevent replay attacks and certain other types of attacks. Figure 3.8 illustrates the operation of CCM. For authentication, the input includes the nonce, the associated data, and the plaintext. This input is formatted as a sequence of blocks B0 through Br. The first block contains the nonce plus some formatting bits that indicate the lengths of the N, A, and P elements. This is followed by zero or more blocks that contain A, followed by zero or more blocks that contain P. The resulting sequence of blocks serves as input to the CMAC algorithm, which produces a MAC value with length Tlen, which is less than or equal to the block length

(Figure 3.8a). For encryption, a sequence of counters is generated that must be independent of the nonce. The authentication tag is encrypted in CTR mode using the single counter Ctr0. The Tlen most significant bits of the output are XORed with the tag to produce an encrypted tag. The remaining counters are used for the CTR mode encryption of the plaintext (Figure 2.11). The encrypted plaintext is concatenated with the encrypted tag to form the ciphertext output (Figure 3.8b). 96 chapter 3 / Public-Key Cryptography and Message Authentication 3.4 Public-Key Cryptography Principles Of equal importance to conventional encryption is public-key encryption, which finds use in message authentication and key distribution. This section looks first at the basic concept of public-key encryption and takes a preliminary look at key distribution issues. Section 3.5 examines the two most important public-key algorithms: RSA and Diffie–Hellman. Section 3.6 introduces digital signatures. Figure 3.8 Counter with Cipher Block Chaining-Message Authentication Code (a) Authentication (b) Encryption B0 Ctr0 B1 B2 Br Tag Tag Nonce Plaintext Plaintext Ciphertext Ass. Data K CMAC MSB(Tlen) K CTR Ctr1, Ctr2, ..., Ctrm K Encrypt 3.4 / Public-Key Cryptography Principles 97 Public-Key Encryption Structure Public-key encryption, first publicly proposed by Diffie and Hellman in 1976 [DIFF76], is the first truly revolutionary advance in encryption in literally thousands of years. Public-key algorithms are based on mathematical functions rather than on simple operations on bit patterns, such as are used in symmetric encryption algorithms. More important, public-key cryptography is asymmetric, involving the use of two separate keys—in contrast to the symmetric conventional encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication. Before proceeding, we should first mention several common misconceptions concerning public-key encryption. One is that public-key encryption is more secure from cryptanalysis than conventional encryption. In fact, the security of any encryption scheme depends on (1) the length of the key and (2) the computational work involved in breaking a cipher. There is nothing in principle about either conventional or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis. A second misconception is that public-key encryption is a general-purpose technique that has made conventional encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that conventional encryption will be abandoned. Finally, there is a feeling that key distribution is trivial when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for conventional encryption. In fact, some form of protocol is needed, often involving a central agent, and the procedures involved are no simpler or any more efficient than those required for conventional encryption. A public-key encryption scheme has six ingredients (Figure 3.9a). ■ Plaintext: This is the readable message or data that is fed into the algorithm as input. ■ Encryption algorithm: The encryption algorithm performs various transformations on the plaintext. ■ Public and private key: This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input. ■ Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts. ■ Decryption algorithm: This algorithm accepts the ciphertext and the matching key and produces the original plaintext. As the names suggest, the public key of the pair is made public for others to use, while the private key is known only to its owner. A general-purpose public-key cryptographic algorithm relies on one key for encryption and a different but related key for decryption. 98 chapter 3 / Public-Key Cryptography and Message Authentication The essential steps are the following: 1. Each user generates a pair of keys to be used for the encryption and decryption of messages. 2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 3.9a Figure 3.9 Public-Key Cryptography Plaintext input Bobs's public key ring Transmitted ciphertext Y = E[PUa, X] X = D[PRa, Y] Plaintext Encryption algorithm output (e.g., RSA) Decryption algorithm Joy Mike Mike Bob Ted Alice X Plaintext input Transmitted ciphertext Y = E[PRb, X] X = D[PUb, Y] Plaintext Encryption algorithm output (e.g., RSA) X Alice's public key Bob's public key PUa Bob's private PUb key PRb PRa Alice's private key Decryption algorithm Alice's public key ring Joy Ted Bob (a) Encryption with public key Alice Bob (b) Encryption with private key Alice 3.4 / Public-Key Cryptography Principles 99 suggests, each user maintains a collection of public keys obtained from others. 3. If Bob wishes to send a private message to Alice, Bob encrypts the message using Alice's public key. 4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key. With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user protects his or her private key, incoming communication is secure. At any time, a user can change the private key and publish the companion public key to replace the old public key. The key used in conventional encryption is typically referred to as a secret key. The two keys used for public-key encryption are referred to as the public key and the private key. Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with conventional encryption. Applications for Public-Key Cryptosystems Before proceeding, we need to clarify one aspect of public-key cryptosystems that is otherwise likely to lead to confusion. Public-key systems are characterized by the use of a cryptographic type of algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key, the receiver's public key, or both to perform some type of cryptographic function. In broad terms, we can classify the use of public-key cryptosystems into three categories: ■ Encryption/decryption: The sender encrypts a message with the recipient's public key. ■ Digital signature: The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message. ■ Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties. Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 3.2 indicates the applications supported by the algorithms discussed in this chapter: RSA and Diffie–Hellman. This table also includes the Digital Signature Standard (DSS) and elliptic-curve cryptography, also mentioned later in this chapter. One general observation can be made at this point. Public-key algorithms require considerably more computation than symmetric algorithms for comparable security and a comparable plaintext length. Accordingly, public-key algorithms are used only for short messages or data blocks, such as to encrypt a secret key or PIN. 100 chapter 3 / Public-Key Cryptography and Message Authentication Requirements for Public-Key Cryptography The cryptosystem illustrated in Figure 3.9 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfill [DIFF76]: 1. It is computationally easy for a party B to generate a pair (public key PUb, private key PRb). 2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M, to generate the corresponding ciphertext: C = E(PUb, M) 3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message: M = D(PRb, C) = D[PRb, E(PUb, M)] 4. It is computationally infeasible for an opponent, knowing the public key, PUb, to determine the private key, PRb. 5. It is computationally infeasible for an opponent, knowing the public key, PUb, and a ciphertext, C, to recover the original message, M. We can add a sixth requirement that, although useful, is not necessary for all public-key applications. 1. Either of the two related keys can be used for encryption, with the other used for decryption. M = D[PUb, E(PRb, M)] = D[PRb, E(PUb, M)] 3.5 Public-Key Cryptography Algorithms Two widely used public-key algorithms are RSA and Diffie–Hellman. We look at both of these in this section and then briefly introduce two other algorithms.4 4 This section uses some elementary concepts from number theory. For a review, see Appendix A. Algorithm Encryption/Decryption Digital Signature Key Exchange RSA Yes Yes Yes Diffie–Hellman No No Yes DSS No Yes No Elliptic curve Yes Yes Yes Table 3.2 Applications for Public-Key Cryptosystems 3.5 / Public-Key Cryptography Algorithms 101 The RSA Public-Key Encryption Algorithm One of the first public-key schemes was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978 [RIVE78]. The RSA scheme has until recently reigned supreme as the most widely accepted and implemented approach to public-key encryption. Currently, both RSA and elliptic-curve cryptography are widely used. RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and n - 1 for some n. Basic RSA Encryption and Decryption Encryption and decryption are of the following form period for some plaintext block M and ciphertext block C: C = Me mod n M = Cd mod n = (Me) d mod n = Med mod n Both sender and receiver must know the values of n and e, and only the receiver knows the value of d. This is a public-key encryption algorithm with a public key of KU = {e, n} and a private key of KR = {d, n}. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met. 1. It is possible to find values of e, d, n such that Med mod n = M for all M 6 n. 2. It is relatively easy to calculate Me and Cd for all values of M 6 n. 3. It is infeasible to determine d given e and n. The first two requirements are easily met. The third requirement can be met for large values of e and n. Figure 3.10 summarizes the RSA algorithm. Begin by selecting two prime numbers p and q and calculating their product n, which is the modulus for encryption and decryption. Next, we need the quantity f(n), referred to as the Euler totient of n, which is the number of positive integers less than n and relatively prime to n. Then select an integer e that is relatively prime to f(n) [i.e., the greatest common divisor of e and f(n) is 1]. Finally, calculate d as the multiplicative inverse of e, modulo f(n). It can be shown that d and e have the desired properties. Suppose that user A has published its public key and that user B wishes to send the message M to A. Then B calculates C = Me (mod n) and transmits C. On receipt of this ciphertext, user A decrypts by calculating M = Cd (mod n). An example, from [SING99], is shown in Figure 3.11. For this example, the keys were generated as follows: 1. Select two prime numbers, p = 17 and q = 11. 2. Calculate n = pq = 17 * 11 = 187. 3. Calculate f(n) = (p - 1)(q - 1) = 16 * 10 = 160. 4. Select e such that e is relatively prime to f(n) = 160 and less than f(n); we choose e = 7. 5. Determine d such that de mod 160 = 1 and d 6 160. The correct value is d = 23, because 23 * 7 = 161 = (1 * 160) + 1. The resulting keys are public key PU = {7, 187} and private key PR = {23, 187}. The example shows the use of these keys for a plaintext input of M = 88. 102 chapter 3 / Public-Key Cryptography and Message Authentication For encryption, we need to calculate C = 887 mod 187. Exploiting the properties of modular arithmetic, we can do this as follows: 887 mod 187 = [(884 mod 187) * (882 mod 187) * (881 mod 187)] mod 187 881 mod 187 = 88 882 mod 187 = 7744 mod 187 = 77 884 mod 187 = 59,969,536 mod 187 = 132 887 mod 187 = (88 * 77 * 132) mod 187 = 894,432 mod 187 = 11 Figure 3.10 The RSA Algorithm Select p, q Select integer e Calculate n = p × q Calculate d Calculate f(n) = (p – 1)(q – 1) p and q both prime, p ≠ q Key Generation Encryption gcd (f(n), e) = 1; 1 < e < f(n) Public key de mod f(n) = 1 KU = {e, n} Plaintext: M < n Ciphertext: C = Me (mod n) Private key KR = {d, n} Decryption Ciphertext: C Plaintext: M = Cd (mod n) Figure 3.11 Example of RSA Algorithm Encryption plaintext 88 plaintext 88 ciphertext 11 88 mod 187 = 11 PU = 7, 187 Decryption 7 11 mod 187 = 88 PR = 23, 187 23 3.5 / Public-Key Cryptography Algorithms 103 For decryption, we calculate M = 1123 mod 187: 1123 mod 187 = [(111 mod 187) * (112 mod 187) * (114 mod 187) * (118 mod 187) * (118 mod 187)] mod 187 111 mod 187 = 11 112 mod 187 = 121 114 mod 187 = 14,641 mod 187 = 55 118 mod 187 = 214,358,881 mod 187 = 33 1123 mod 187 = (11 * 121 * 55 * 33 * 33) mod 187 = 79,720,245 mod 187 = 88 Security Considerations The security of RSA depends on it being used in such a way as to counter potential attacks. Four possible attack approaches are as follows: ■ Mathematical attacks: There are several approaches, all equivalent in effort to factoring the product of two primes. The defense against mathematical attacks is to use a large key size. Thus, the larger the number of bits in d, the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run. SP 800-131A (Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, November 2015) recommends the use of a 2048-bit key size. A recent report from the European Union Agency for Network and Information Security (Algorithms, key size and parameters report—2014, November 2014) recommends a 3072-bit key length. Either of these lengths should provide adequate security for a considerable time into the future. ■ Timing attacks: These depend on the running time of the decryption algorithm. Various approaches to mask the time required so as to thwart attempts to deduce key size have been suggested, such as introducing a random delay. ■ Chosen ciphertext attacks: This type of attack exploits properties of the RSA algorithm by selecting blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis. These attacks can be thwarted by suitable padding of the plaintext. To counter sophisticated chosen ciphertext attacks, RSA Security Inc., a leading RSA vendor and former holder of the RSA patent, recommends modifying the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP). A full discussion of the threats and OAEP are beyond our scope; see [POIN02] for an introduction and [BELL94a] for a thorough analysis. Here, we simply summarize the OAEP procedure. Figure 3.12 depicts OAEP encryption. As a first step, the message M to be encrypted is padded. A set of optional parameters, P, is passed through a hash function, H. The output is then padded with zeros to get the desired length in the overall data block (DB). Next, a random seed is generated and passed through another hash function, called the mask generating function (MGF). The resulting hash value is bitby-bit XORed with DB to produce a maskedDB. The maskedDB is in turn passed 104 chapter 3 / Public-Key Cryptography and Message Authentication through the MGF to form a hash that is XORed with the seed to produce the masked seed. The concatenation of the maskedseed and the maskedDB forms the encoded message EM. Note that the EM includes the padded message masked by the seed, and the seed masked by the maskedDB. The EM is then encrypted using RSA. Diffie–Hellman Key Exchange The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [DIFF76] and is generally referred to as the Diffie–Hellman key exchange. A number of commercial products employ this key exchange technique. The purpose of the algorithm is to enable two users to securely exchange a secret key that then can be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of the keys. The Diffie–Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. First, we define a primitive root of a prime number p as one whose Figure 3.12 Encryption Using Optimal Asymmetric Encryption Padding (OAEP) seed maskedseed DB maskedDB M EM padding H(P) MGF MGF P P = encoding parameters M = message to be encoded H = hash function DB = data block MGF = mask generating function EM = encoded message 3.5 / Public-Key Cryptography Algorithms 105 powers generate all the integers from 1 to p - 1. That is, if a is a primitive root of

the prime number p, then the numbers a mod p, a2 mod p, c , ap-1 mod p are distinct and consist of the integers from 1 through p - 1 in some permutation. For any integer b less than p and a primitive root a of prime number p, one can find a unique exponent i such that b = ai mod p 0 ... i ... (p - 1) The exponent i is referred to as the discrete logarithm, or index, of b for the base a, mod p. We denote this value as dloga, p(b).5 The Algorithm With this background, we can define the Diffie–Hellman key exchange, which is summarized in Figure 3.13. For this scheme, there are two publicly known numbers: a prime number q and an integer a that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer XA 6 q and computes YA = aXA mod q. Similarly, user B independently selects a random integer XB 6 q and computes YB = aXB mod q. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as K = (YB) XA mod q and user B computes the key as K = (YA) XB mod q. These two calculations produce identical results: K = (YB) XA mod q = (aXB mod q) XA mod q = (aXB) XA mod q = aXBXA mod q = (aXA) XB mod q = (aXA mod q) XB mod q = (YA) XB mod q The result is that the two sides have exchanged a secret value. Furthermore, because XA and XB are private, an adversary only has the following ingredients to work with: q, a, YA, and YB. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute XB = dloga,q(YB) The adversary can then calculate the key K in the same manner as user B does. The security of the Diffie–Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible. 5 Many texts refer to the discrete logarithm as the index. There is no generally agreed notation for this concept, much less an agreed name. 106 chapter 3 / Public-Key Cryptography and Message Authentication Figure 3.13 The Diffie–Hellman Key Exchange Alice Bob Alice and Bob share a prime q and A, such that A < q and A is a primitive root of q Alice generates a private key XA such that XA < q Alice calculates a public key YA = AXA mod q Alice receives Bob’s public key YB in plaintext Alice calculates shared secret key K = (YB) XA mod q Bob calculates shared secret key K = (YA) XB mod q Bob receives Alice’s public key YA in plaintext Bob calculates a public key YB = AXB mod q Bob generates a private key XB such that XB < q Alice and Bob share a prime q and A, such that A < q and A is a primitive root of q YA YB Here is an example. Key exchange is based on the use of the prime number q = 353 and a primitive root of 353, in this case a = 3. A and B select secret keys XA = 97 and XB = 233, respectively. Each computes its public key: A computes YA = 397 mod 353 = 40. B computes YB = 3233 mod 353 = 248. After they exchange public keys, each can compute the common secret key: A computes K = (YB) XA mod 353 = 24897 mod 353 = 160. B computes K = (YA) XB mod 353 = 40233 mod 353 = 160. We assume an attacker would have available the following information: q = 353; a = 3; YA = 40; YB = 248 In this simple example, it would be possible to determine the secret key 160 by brute force. In particular, an attacker E can determine the common key by discovering a solution to the equation 3a mod 353 = 40 or the equation 3b mod 353 = 248. The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponent value of 97, which provides 397 mod 353 = 40. With larger numbers, the problem becomes impractical. 3.5 / Public-Key Cryptography Algorithms 107 Key Exchange Protocols Figure 3.13 shows a simple protocol that makes use of the Diffie–Hellman calculation. Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. User A can generate a one-time private key XA, calculate YA, and send that to user B. User B responds by generating a private value XB, calculating YB, and sending YB to user A. Both users can now calculate the key. The necessary public values q and a would need to be known ahead of time. Alternatively, user A could pick values for q and include those in the first message. As an example of another use of the Diffie–Hellman algorithm, suppose that a group of users (e.g., all users on a LAN) each generate a long-lasting private value XA and calculate a public value YA. These public values, together with global public values for q and a, are stored in some central directory. At any time, user B can access user A's public value, calculate a secret key, and use that to send an encrypted message to user A. If the central directory is trusted, then this form of communication provides both confidentiality and a degree of authentication. Because only A and B can determine the key, no other user can read the message (confidentiality). Recipient A knows that only user B could have created a message using this key (authentication). However, the technique does not protect against replay attacks. Man-in-the-Middle Attack The protocol depicted in Figure 3.13 is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows (Figure 3.14): 1. Darth prepares for the attack by generating two random private keys XD1 and XD2, and then computing the corresponding public keys YD1 and YD2. 2. Alice transmits YA to Bob. 3. Darth intercepts YA and transmits YD1 to Bob. Darth also calculates K2 = (YA) XD2 mod q. 4. Bob receives YD1 and calculates K1 = (YD1) XB mod q. 5. Bob transmits YB to Alice. 6. Darth intercepts YB and transmits YD2 to Alice. Darth calculates K1 = (YB) XD1 mod q. 7. Alice receives YD2 and calculates K2 = (YD2) XA mod q. At this point, Bob and Alice think that they share a secret key. Instead Bob and Darth share secret key K1, and Alice and Darth share secret key K2. All future communication between Bob and Alice is compromised in the following way: 1. Alice sends an encrypted message M: E(K2, M). 2. Darth intercepts the encrypted message and decrypts it to recover M. 3. Darth sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob. 108 chapter 3 / Public-Key Cryptography and Message Authentication The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates; these topics are explored later in this chapter and in Chapter 4. Other Public-Key Cryptography Algorithms Two other public-key algorithms have found commercial acceptance: DSS and elliptic-curve cryptography. Digital Signature Standard The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS PUB 186, known as the Digital Signature Standard (DSS). The DSS makes use of the SHA-1 and presents a new digital signature technique, the Digital Signature Algorithm Figure 3.14 Man-in-the-Middle Attack Alice Darth Bob Private key XA Public key YA = AXA mod q Private key XB Public key YB = AXB mod q Private keys XD1, XD2 Public keys YD1 = AXD1 mod q YD2 = AXD2 mod q YA Secret key K2 = (YA) XD2 mod q Secret key K1 = (YB) XD1 mod q Secret key K1 = (YD1) XB mod q Secret key K2 = (YD2) XA mod q Alice and Darth share K2 Bob and Darth share K1 YD2 YD1 YB 3.6 / Digital Signatures 109 (DSA). The DSS 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. The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Elliptic-Curve Cryptography The vast majority of the products and standards that use public-key cryptography for encryption and digital signatures use RSA. The bit length for secure RSA use has increased over recent years, and this has put a heavier processing load on applications using RSA. This burden has ramifications, especially for electronic commerce sites that conduct large numbers of secure transactions. Recently, a competing system has begun to challenge RSA: elliptic curve cryptography (ECC). Already, ECC is showing up in standardization efforts, including the IEEE P1363 Standard for Public-Key Cryptography. The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing processing overhead. On the other hand, although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there has been sustained cryptanalytic interest in probing for weaknesses. Thus, the confidence level in ECC is not yet as high as that in RSA. ECC is fundamentally more difficult to explain than either RSA or Diffie– Hellman, and a full mathematical description is beyond the scope of this book. The technique is based on the use of a mathematical construct known as the elliptic curve. 3.6 Digital Signatures NIST FIPS PUB 186-4 [Digital Signature Standard (DSS), July 2013] defines a digital signature as follows: The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity, and signatory non-repudiation. Thus, a digital signature is a data-dependent bit pattern, generated by an agent as a function of a file, message, or other form of data block. Another agent can access the data block and its associated signature and verify that (1) the data block has been signed by the alleged signer and that (2) the data block has not been altered since the signing. Further, the signer cannot repudiate the signature. FIPS 186-4 specifies the use of one of three digital signature algorithms: ■ Digital Signature Algorithm (DSA): The original NIST-approved algorithm, which is based on the difficulty of computing discrete logarithms. ■ RSA Digital Signature Algorithm: Based on the RSA public-key algorithm. ■ Elliptic Curve Digital Signature Algorithm (ECDSA): Based on ellipticcurve cryptography. In this section, we provide a brief overview of the digital signature process, then describe the RSA digital signature algorithm. 110 chapter 3 / Public-Key Cryptography and Message Authentication Digital Signature Generation and Verification Figure 3.15 is a generic model of the process of making and using digital signatures. All of the digital signature schemes in FIPS 186-4 have this structure. Suppose that Bob wants to send a message to Alice. Although it is not important that the message be kept as a 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, serve as input to a digital signature generation algorithm that 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 Figure 3.15 Simplified Depiction of Essential Elements of Digital Signature Process Bob Alice Bob’s signature for M Message M Cryptographic hash function Digital signature generation algorithm Digital signature verication algorithm h Message M Cryptographic hash function h S Message M S Return signature valid or not valid Bob’s private key (a) Bob signs a message (b) Alice veries the signature Bob’s public key 3.6 / Digital Signatures 111 a hash value for the message and (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. It is important to emphasize that the encryption process just described does not provide confidentiality. That is, the message being sent is safe from alteration, but not safe from eavesdropping. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, there is no protection of confidentiality because any observer can decrypt the message by using the sender’s public key. RSA Digital Signature Algorithm The essence of the RSA digital signature algorithm is to encrypt the hash of the message to be signed using RSA. However, as with the use of RSA for encryption of keys or short messages, the RSA digital signature algorithm first modifies the hash value to enhance security. There are several approaches to this, one of which is the RSA Probabilistic Signature Scheme (RSA-PSS). RSA-PSS is the latest of the RSA schemes and the one that RSA Laboratories recommends as the most secure of the RSA digital signature schemes. We provide a brief overview here; for more detail see [STAL16]. Figure 3.16 illustrates the RSS-PSS signature generation process. The steps are as follows: 1. Generate a hash value, or message digest, mHash from the message M to be signed. 2. Pad mHash with a constant value padding1 and pseudorandom value salt to form M' 3. Generate hash value H from M'. 4. Generate a block DB consisting of a constant value padding 2 and salt. 5. Use the mask generating function MGF, which produces a randomized output from input H of the same length as DB. 6. Create the encoded message (EM) block by padding H with the hexadecimal constant BC and the XOR of H and DB. 7. Encrypt EM with RSA using the signer's private key. The objective with this algorithm 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. 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. 112 chapter 3 / Public-Key Cryptography and Message Authentication 3.7 Key Terms, Review Questions, and Problems Hash Hash MGF M Signature padding1 mHash salt maskedDB bc padding2 salt Mœ = DB = EM = H E signer's private key Figure 3.16 RSA-PSS Encoding and Signature Generation Key Terms authenticated encryption Diffie–Hellman key exchange digital signature Digital Signature Standard (DSS) elliptic-curve cryptography (ECC) HMAC key exchange MD5 message authentication message authentication code (MAC) message digest one-way hash function private key public key public-key certificate public-key encryption RSA secret key secure hash function SHA-1 strong collision resistant weak collision resistant Review Questions 3.1 List three approaches to message authentication. 3.2 What is a message authentication code? 3.7 / Key Terms, Review Questions, and Problems 113 3.3 What is a one-way hash function? How is it different from message authentication code? 3.4 List the properties of a strong hash function. 3.5 Compare SHA-1 and SHA-2 with respect to SHA parameters. 3.6 List the design objectives for HMAC. 3.7 Differentiate between public-key cryptosystem and symmetric encryption algorithm. 3.8 List the public-key cryptography algorithm used in digital signatures, encryption/ decryption, and key exchange. 3.9 In what way is the Diffie–Hellman key exchange algorithm insecure against a man-in-the-middle attack? Problems 3.1 Consider a 32-bit hash function defined as the concatenation of two 16-bit functions: XOR and RXOR, which are defined in Section 3.2 as "two simple hash functions." a. Will this checksum detect all errors caused by an odd number of error bits? Explain. b. Will this checksum detect all errors caused by an even number of error bits? If not, characterize the error patterns that will cause the checksum to fail. c. Comment on the effectiveness of this function for use as a hash function for authentication. 3.2 Suppose H(m) is a collision-resistant hash function that maps a message of arbitrary bit length into an n-bit hash value. Is it true that, for all messages x, x' with x [x', we have H(x) ≠ H(x')? Explain your answer. 3.3 State the value of the padding field in SHA-512 if the length of the message is a. 4987 bits b. 4199 bits c. 1227 bits 3.4 State the value of the length field in SHA-512 if the length of the message is a. 3967 bits b. 3968 bits c. 3969 bits 3.5 a. Consider the following hash function. Messages are in the form of a sequence of decimal numbers, M = (a1, a2, c, at). The hash value h is calculated as ¢ a t i=1 ai≤mod n, for some predefined value n. Does this hash function satisfy any of the requirements for a hash function listed in Section 3.2? Explain your answer. b. Repeat part (a) for the hash function h = ¢ a t i=1 (ai) 2 ≤mod n. c. Calculate the hash function of part (b) for M = (237, 632, 913, 423, 349) and n = 757. 3.6 This problem introduces a hash function similar in spirit to SHA that operates on letters instead of binary data. It is called the toy tetragraph hash (tth).6 Given a message consisting of a sequence of letters, tth produces a hash value consisting of four letters. First, tth divides the message into blocks of 16 letters, ignoring spaces, punctuation, and capitalization. If the message length is not divisible by 16, it is padded out with nulls. A four-number running total is maintained that starts out with the value (0, 0, 0, 0); this is input to the compression function for processing the first block. The compression function consists of two rounds. Round 1: Get the next block of text and arrange it as a row-wise 4 block of text and covert it to numbers (A = 0, B = 1, etc.). For example, for the block ABCDEFGHIJKLMNOP, we have 6 I thank William K. Mason of the magazine staff of The Cryptogram for providing this example. 114 chapter 3 / Public-Key Cryptography and Message Authentication A B C D E F G H I J K L M N O P 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Then, add each column mod 26 and add the result to the running total, mod 26. In this example, the running total is (24, 2, 6, 10). Round 2: Using the matrix from round 1, rotate the first row left by 1, second row left by 2, third row left by 3, and reverse the order of the fourth row. In our example: B C D A G H E F L I J K P O N M 1 2 3 0 6 7 4 5 11 8 9 10 15 14 13 12 Now, add each column mod 26 and add the result to the running total. The new running total is (5, 7, 9, 11). This running total is now the input into the first round of the compression function for the next block of text. After the final block is processed, convert the final running total to letters. For example, if the message is ABCDE FGHIJKLMNOP, then the hash is FHJL. a. Draw figures comparable to Figures 3.4 and 3.5 to depict the overall tth logic and the compression function logic. b. Calculate the hash function for the 48-letter message “I leave twenty million dollars to my friendly cousin Bill.” c. To demonstrate the weakness of tth, find a 48-letter block that produces the same hash as that just derived. Hint: Use lots of A’s. 3.7 It is possible to use a hash function to construct a block cipher with a structure similar to DES. Because a hash function is one way and a block cipher must be reversible (to decrypt), how is it possible? 3.8 Now consider the opposite problem: Use an encryption algorithm to construct a oneway hash function. Consider using RSA with a known key. Then process a message consisting of a sequence of blocks as follows: Encrypt the first block, XOR the result with the second block and encrypt again, and so on. Show that this scheme is not secure by solving the following problem. Given a two-block message B1, B2, and its hash, we have RSAH(B1, B2) = RSA(RSA(B1) $\oplus$ B2) Given an arbitrary block C1, choose C2 so that RSAH(C1, C2) = RSAH(B1, B2). Thus, the hash function does not satisfy weak collision resistance. 3.9 One of the most widely used MACs, referred to as the Data Authentication Algorithm, is based on DES. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero (Figure 2.9). The data (e.g., message, record, file, or program) to be authenticated is grouped into contiguous 64-bit blocks: P1, P2, c, PN. If necessary, the final block is padded on the right with 0s to form a full 64-bit block. The MAC consists of either the entire ciphertext block CN or the leftmost M bits of the block with 16 … M … 64. Show that the same result can be produced using the cipher feedback mode. 3.10 In this problem, we will compare the security services that are provided by digital signatures (DS) and message authentication codes (MAC). We assume that Oscar 3.7 / Key Terms, Review Questions, and Problems 115 is able to observe all messages sent from Alice to Bob and vice versa. Oscar has no knowledge of any keys but the public one in case of DS. State whether and how (i) DS and (ii) MAC protect against each attack. The value auth(x) is computed with a DS or a MAC algorithm, respectively. a. (Message integrity) Alice sends a message x = “Transfer $1000 to Mark” in the clear and also sends auth(x) to Bob. Oscar intercepts the message and replaces “Mark” with “Oscar.” Will Bob detect this? b. (Replay) Alice sends a message x = “Transfer $1000 to Oscar” in the clear and also sends auth(x) to Bob. Oscar observes the message and signature and sends them 100 times to Bob. Will Bob detect this? c. (Sender Authentication with cheating third party) Oscar claims that he sent some message x with a valid auth(x) to Bob, but Alice claims the same. Can Bob clear the question in either case? d. (Authentication with Bob cheating) Bob claims that he received a message x with a valid signature auth(x) from Alice (e.g., “Transfer $1000 from Alice to Bob”) but Alice claims she has never sent it. Can Alice clear this question in either case? 3.11 Figure 3.17 shows an alternative means of implementing HMAC. a. Describe the operation of this implementation. b. What potential benefit does this implementation have over that shown in Figure 3.6? Figure 3.17 Efficient Implementation of HMAC b bits b bits b bits Precomputed Computed per message IV Hash n bits b bits n bits Pad to b bits n bits n bits HMAC(K, M) f IV b bits f f Si So Y0 Y1 ipad K+ K+ opad YL–1 H(Si || M) 116 chapter 3 / Public-Key Cryptography and Message Authentication 3.12 In this problem, we demonstrate that for CMAC, a variant that XORs the second key after applying the final encryption doesn’t work. Let us consider this for the case of the message being an integer multiple of the block size. Then the variant can be expressed as VMAC(K, M) = CBC(K, M) $\oplus$ K1. Now suppose an adversary is able to ask for the MACs of three messages: the message 0 = 0n , where n is the cipher block size; the message 1 = 1 n ; and the message 1 } 0. As a result of these three queries, the adversary gets T0 = CBC(K, 0) $\oplus$ K1; T1 = CBC(K, 1) $\oplus$ K1 and T2 = CBC(K, [CBC(K, 1)]) $\oplus$ K1. Show that the adversary can compute the correct MAC for the (unqueried) message 0 } (T0 $\oplus$ T1). 3.13 Prior to the discovery of any specific public-key schemes, such as RSA, an existence proof was developed whose purpose was to demonstrate that public-key encryption is possible in theory. Consider the functions f1(x1) = z1; f2(x2, y2) = z2; f3(x3, y3) = z3, where all values are integers with 1 ... xi , yi , zi ... N. Function f1 can be represented by a vector M1 of length N in which the kth entry is the value of f1(k). Similarly, f2 and f3 can be represented by N * N matrices M2 and M3. The intent is to represent the encryption/decryption process by table lookups for tables with very large values of N. Such tables would be impractically huge but in principle could be constructed. The scheme works as follows: Construct M1 with a random permutation of all integers between 1 and N; that is, each integer appears exactly once in M1. Construct M2 so that each row contains a random permutation of the first N integers. Finally, fill in M3 to satisfy the condition: f3(f2(f1(k), p), k) = p for all k, p with 1 ... k, p ... N In words, 1. M1 takes an input k and produces an output x. 2. M2 takes inputs x and p giving output z. 3. M3 takes inputs z and k and produces p. The three tables, once constructed, are made public. a. It should be clear that it is possible to construct M3 to satisfy the preceding condition. As an example, fill in M3 for the following simple case: 5 5 2 3 4 1 4 4 2 5 1 3 M1 = 2 M2 = 1 3 2 4 5 M3 = 3 3 1 4 2 5 1 2 5 3 4 1 Convention: The ith element of M1 corresponds to k = i. The ith row of M2 corresponds to x = i; the jth column of M2 corresponds to p = j. The ith row of M3 corresponds to z = i; the jth column of M3 corresponds to k = j. b. Describe the use of this set of tables to perform encryption and decryption between two users. c. Argue that this is a secure scheme. 3.14 Perform encryption and decryption using the RSA algorithm (Figure 3.10) for the following: a. p = 3; q = 11, e = 7; M = 2 b. p = 5; q = 11, e = 3; M = 5 c. p = 7; q = 11, e = 17; M = 2 d. p = 11; q = 13, e = 11; M = 3 e. p = 17; q = 11, e = 7; M = 88 Hint: Decryption is not as hard as you think; use some finesse. 3.15 In a public-key system using RSA, you intercept the ciphertext C = 16 sent to a user whose public key is e = 6, n = 40. What is the plaintext M? 3.7 / Key Terms, Review Questions, and Problems 117 3.16 In an RSA system, the public key of a given user is e = 7, n = 137. What is the private key of this user? 3.17 Suppose we have a set of blocks encoded with the RSA algorithm and we don’t have the private key. Assume n = pq, e is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with n. Does this help us in any way? 3.18 Show how RSA can be represented by matrices M1, M2, and M3 of Problem 3.4. 3.19 Consider the following scheme. 1. Pick an odd number, E. 2. Pick two prime numbers, P and Q, where (P - 1)(Q - 1) is relatively prime to E. 3. Multiply P and Q to get N. 5. Calculate D = (P - 1)(Q - 1)(E + 1) + 1 E Is this scheme equivalent to RSA? Show why or why not. 3.20 Suppose Bob uses the RSA cryptosystem with a very large modulus n for which the factorization cannot be found in a reasonable amount of time. Suppose Alice sends a message to Bob by representing each alphabetic character as an integer between 0 and 25 (A S 0, c, Z S 25), and then encrypting each number separately using RSA with large e and large n. Is this method secure? If not, describe the most efficient attack against this encryption method. 3.21 Consider a Diffie–Hellman scheme with a common prime q = 353 and a primitive root a = 3. a. If user A has public key YA = 40, what is A's private key XA? b. If user B has public key YB = 248, what is the shared secret key K? This page intentionally left blank 119 4.1 Remote User Authentication Principles The NIST Model for Electronic User Authentication Means of Authentication 4.2 Symmetric Key Distribution Using Symmetric Encryption 4.3 Kerberos Kerberos Version 4 Kerberos Version 5 4.4 Key Distribution Using Asymmetric Encryption Public-Key Certificates Public-Key Distribution of Secret Keys 4.5 X.509 Certificates Certificates X.509 Version 3 4.6 Public-Key Infrastructure PKIX Management Functions PKIX Management Protocols 4.7 Federated Identity Management Identity Management Identity Federation 4.8 Key Terms, Review Questions, and Problems Chapter Key Distribution and User Authentication Part Two: Network Security Applications 120 chapter 4 / Key Distribution and User Authentication This chapter covers two important related concepts. First is the complex topic of cryptographic key distribution, involving cryptographic, protocol, and management considerations. This chapter gives the reader a feel for the issues involved and provides a broad survey of the various aspects of key management and distribution. This chapter also examines some of the authentication functions that have been developed to support network-based user authentication. The chapter includes a detail discussion of one of the earliest and also one of the most widely used key distribution and user authentication services: Kerberos. Next, the chapter looks at key distribution schemes that rely on asymmetric encryption. This is followed by a discussion of X.509 certificates and public-key infrastructure. Finally, the concept of federated identity management is introduced. 4.1 Remote User Authentication Principles In most computer security contexts, user authentication is the fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability. RFC 4949 (Internet Security Glossary) defines user authentication as the process of verifying an identity claimed by or for a system entity. This process consists of two steps: ■■ Identification step: Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.) ■■ Verification step: Presenting or generating authentication information that corroborates the binding between the entity and the identifier. For example, user Alice Toklas could have the user identifier ABTOKLAS. This information needs to be stored on any server or computer system that Alice wishes to use and could be known to system administrators and other users. A typical item of authentication information associated with this user ID is a password, Learning Objectives After studying this chapter, you should be able to: ◆ Understand the issues involved in the use of symmetric encryption to distribute symmetric keys. ◆ Give a presentation on Kerberos. ◆ Explain the differences between versions 4 and 5 of Kerberos. ◆ Understand the issues involved in the use of asymmetric encryption to distribute symmetric keys. ◆ List and explain the elements in an X.509 certificate. ◆ Present an overview of public-key infrastructure concepts. ◆ Understand the need for a federated identity management system. 4.1 / Remote User Authentication Principles 121 which is kept secret (known only to Alice and to the system). If no one is able to obtain or guess Alice's password, then the combination of Alice's user ID and password enables administrators to set up Alice's access permissions and audit her activity. Because Alice's ID is not secret, system users can send her e-mail, but because her password is secret, no one can pretend to be Alice. In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication. As defined in Chapter 3, message authentication is a procedure that allows communicating parties to verify that the contents of a received message have not been altered and that the source is authentic. This chapter is concerned solely with user authentication. The NIST Model for Electronic User Authentication NIST SP 800-63-2 (Electronic Authentication Guideline, August 2013) defines electronic user authentication as the process of establishing confidence in user identities that are presented electronically to an information system. Systems can use the authenticated identity to determine if the authenticated individual is authorized to perform particular functions, such as database transactions or access to system resources. In many cases, the authentication and transaction or other authorized function take place across an open network such as the Internet. Equally, authentication and subsequent authorization can take place locally, such as across a local area network. SP 800-63-2 defines a general model for user authentication that involves a number of entities and procedures. We discuss this model with reference to Figure 4.1. The initial requirement for performing user authentication is that the user must be registered with the system. The following is a typical sequence for registration. An applicant applies to a registration authority (RA) to become a subscriber of a credential service provider (CSP). In this model, the RA is a trusted entity that establishes and vouches for the identity of an applicant to a CSP. The CSP then engages Figure 4.1 The NIST SP 800-63-2 E-Authentication Architectural Model Registration authority (RA) Registration, credential issuance, and maintenance E-Authentication using token and credential Identity proofing User registration Token, credential Registration/issuance Authenticated session Authenticated protocol Exchange Authenticated assertion Registration Confirmation Token/credential Validation Relying party (RP) Verifier Subscriber/ claimant Credential service provider (RA) 122 chapter 4 / Key Distribution and User Authentication in an exchange with the subscriber. Depending on the details of the overall authentication system, the CSP issues some sort of electronic credential to the subscriber. The credential is a data structure that authoritatively binds an identity and additional attributes to a token possessed by a subscriber, and can be verified when presented to the verifier in an authentication transaction. The token could be an encryption key or an encrypted password that identifies the subscriber. The token may be issued by the CSP, generated directly by the subscriber, or provided by a third party. The token and credential may be used in subsequent authentication events. Once a user is registered as a subscriber, the actual authentication process can take place between the subscriber and one or more systems that perform authentication and, subsequently, authorization. The party to be authenticated is called a claimant and the party verifying that identity is called a verifier. When a claimant successfully demonstrates possession and control of a token to a verifier through an authentication protocol, the verifier can verify that the claimant is the subscriber named in the corresponding credential. The verifier passes on an assertion about the identity of the subscriber to the relying party (RP). That assertion includes identity information about a subscriber, such as the subscriber name, an identifier assigned at registration, or other subscriber attributes that were verified in the registration process. The RP can use the authenticated information provided by the verifier to make access control or authorization decisions. An implemented system for authentication will differ from or be more complex than this simplified model, but the model illustrates the key roles and functions needed for a secure authentication system. Means of Authentication There are four general means of authenticating a user’s identity, which can be used alone or in combination: ■■ Something the individual knows: Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions. ■■ Something the individual possesses: Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a token. ■■ Something the individual is (static biometrics): Examples include recognition by fingerprint, retina, and face. ■■ Something the individual does (dynamic biometrics): Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm. All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Furthermore, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience. For networkbased user authentication, the most important methods involve cryptographic keys and something the individual knows, such as a password. 4.2 / Symmetric Key Distribution Using Symmetric Encryption 123 4.2 Symmetric Key Distribution Using Symmetric Encryption For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the “key distribution technique,” a term that refers to the means of delivering a key to two parties that wish to exchange data, without allowing others to see the key. Key distribution can be achieved in a number of ways. For two parties A and B, there are the following options: 1. A key could be selected by A and physically delivered to B. 2. A third party could select the key and physically deliver it to A and B. 3. If A and B have previously and recently used a key, one party could transmit the new key to the other, using the old key to encrypt the new key. 4. If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B. Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is only going to be exchanging data with its partner on the other end of the link. However, for end-toend encryption over a network, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys supplied dynamically. The problem is especially difficult in a wide-area distributed system. Option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys are revealed. Even if frequent changes are made to the link encryption keys, these should be done manually. To provide keys for end-to-end encryption, option 4 is preferable. For option 4, two kinds of keys are used: ■■ Session key: When two end systems (hosts, terminals, etc.) wish to communicate, they establish a logical connection (e.g., virtual circuit). For the duration of that logical connection, called a session, all user data are encrypted with a onetime session key. At the conclusion of the session the session key is destroyed. ■■ Permanent key: A permanent key is a key used between entities for the purpose of distributing session keys. A necessary element of option 4 is a key distribution center (KDC). The KDC determines which systems are allowed to communicate with each other. When permission is granted for two systems to establish a connection, the key distribution center provides a one-time session key for that connection. In general terms, the operation of a KDC proceeds as follows: 1. When host A wishes to set up a connection to host B, it transmits a connection-request packet to the KDC. The communication