Chapter 13
HASHES
NON-KEYED MESSAGE DIGESTS

In Chapter 12, Alice signed (that is, private key encrypted) her newsletter to assure her clients that the newsletter they received was Alice's genuine, unaltered newsletter. But signing anything except a small message takes too much time.

Message digests make public key cryptography more efficient.

Public key encryption and decryption methods, such as RSA and DSA, are as much as 1,000 times slower than secret key encryption, such as DES or the new AES standard, Rijndael. This means that an encryption that takes one minute in DES takes many hours in RSA, obviously much too long for fast-paced e-commerce. So cryptographers invented message digests to make private key signing more efficient.

This chapter shows how Alice uses signed message digests (instead of a signed message) and how Bob uses Alice's signed message digest to verify the origin (authenticity) and the genuineness (integrity) of Alice's newsletter.

Definition: message digest

A message digest is used as a proxy for a message; it is a shorter, redundant represention of the message.[1] *Redundant*, in computer lingo, refers to the repetition of a message to identify whether the message was modified during transmission.

Communication redundancies verify that the sent message was correctly received. For example, in communications between a control tower and a pilot, shown in Figure 13-1, communication redundancies are used to confirm that the data sent was correctly received. The pilot's reply is a compact, redundant version of the control tower's message.

Message digests act as proxies for larger data.

Message digests add similar redundant assurances to digital data communications. And because a message digest is usually much smaller than the underlying message, it's faster to sign (private key encrypt) and verify (public key decrypt) a message digest than a lengthy message.

1. MACs, discussed in Chapter 7, are also message digests. But MACs are made with secret keys; this chapter looks at message digests that are made without secret keys. The next chapter examines both types of message digests in more depth.

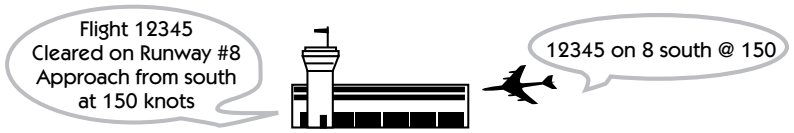

Figure 13-1 Communication redundancies ensure that the message received accurately reflects the message sent.

Coming in For a Landing: Don't Shoot!

Identification Friend or Foe (IFF) devices were developed during World War II so US aircraft wouldn't be shot down by friendly fire. Cryptography was incorporated into the devices to stop the enemy from replicating them. After the war, the IFF device was redesigned to fit into an aircraft's nose. A team led by Horst Feistel, later of DES fame, tested the improved system and also developed the first practical block ciphers. IFF without cryptography evolved into the Mark X, an essential part of civilian and military air traffic control today.

Message digest, aka (cryptographic) hash, aka digital fingerprint, aka cryptographic checksum

A message digest is also analogous to a fingerprint. You can authenticate a person's identity by verifying facial characteristics, name, height, weight, age, knowledge of the mother's maiden name, and so on. Similarly, a fingerprint is also a small piece of data that authenticates identity. Because fingerprints and message digests are used as unique proxies for a much larger whole, message digests are also known as *digital fingerprints* (or *message fingerprints*). They are also referred to as *cryptographic hashes*[2] or *cryptographic checksums*. We use these terms interchangeably as well as using the more abbreviated terms *digest* and *hash*.

Message digest methods supercompress messages.

Message digest methods supercompress messages so that encryption and decryption operate on less data and, therefore, take less time. Secure Hash Algorithm (SHA-1), the message digest algorithm currently recommended by government and private cryptographers, will compress all of Microsoft Office to about the same amount of disk space occupied by 20 *x*s: *xxxxxxxxxxxxxxxxxxxx*.

A message digest cannot be uncompressed.

Although message digests are similar to popular file compression programs such as PKZip, WinZip, gzip, or StuffIt, a major difference is that popular compression programs are made to compress and restore files. Message digest programs can't and don't restore their compressed messages; they *only* compress messages. Just as a person cannot be reconstructed from a fingerprint (not yet anyway), the original file cannot be reconstructed from the message digest (not yet anyway).

2. Cryptographic hashes are not the same as the hashes used in computer programming. Cryptographic hashes, although similar, add important security features.

Detecting Unintentional Modifications

Before we look at message digests, it's instructive to look at other schemes that also act as redundant proxies. Schemes such as parity checking and checksums were invented before their cryptographic offspring and share many of the same goals.

Typical (not cryptographic) checksum programs search for unintentional message modifications that originate in things such as noisy communications channels. Like message digests, checksums are designed to identify whether a message has been modified. For example, suppose that Alice sends the checksum and message to Bob. Bob independently calculates a checksum on the sent data and compares it to the checksum sent by Alice. If the checksums don't match, Bob knows that data were modified or lost during transmission.

Figure 13-2 shows an example of a simplistic checksum error detection code. Alice makes a checksum stating the number of 0's and 1's in her message. Both the message and the checksum are sent to Bob. The checksum program in Bob's computer also computes a checksum (counts the number of 0's and 1's in the message he received) and compares it against the checksum he received from Alice. In Figure 13-2, the checksum Bob independently makes doesn't equal the checksum Alice sent, so the message might be rejected because the number of 0's and 1's received is incorrect. (The message might be accepted if,

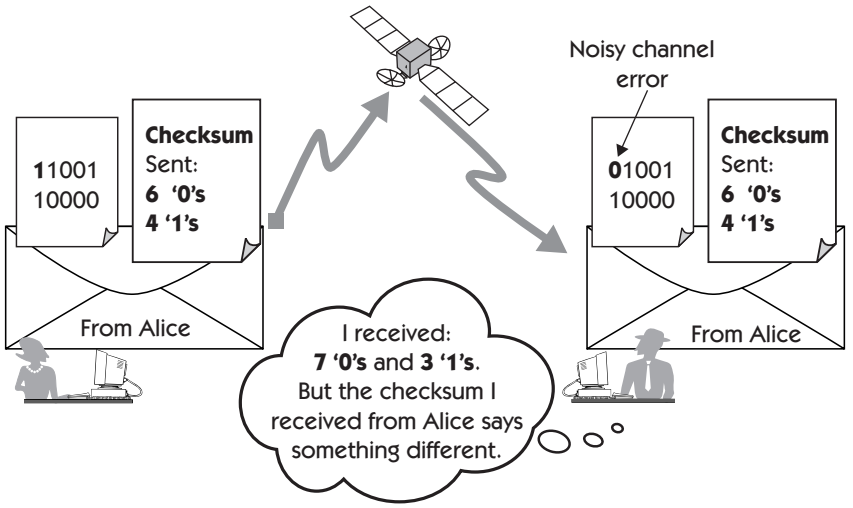

Figure 13-2 Detecting unintentional modifications. Alice makes and sends a checksum of her data (6 0's and 4 1's). Bob independently calculates the number of 0's and 1's and compares it to the checksum sent by Alice. Bob knows that he should have received one fewer 0 bit and one more 1 bit.

for example, it was a graphic containing thousands of bits and the presence of one altered bit was acceptable.)

Message digests are (cryptographic) checksums.

Checksum methods can easily be applied to any message of any size, and the checksum report is usually much smaller than the underlying message. Message digests operate in a similar fashion; you can also apply them to any size message and output a small digest. Message digests are specialized checksums; hence the name *cryptographic checksums*.

Let's review our symbols and add two new ones before we see how Alice and her customers use message digests to prevent BlackHat from perpetrating a fraudulent newsletter. Figure 13-3(a) shows Alice signing, and 13-3(b) shows Bob verifying her newsletter.

Figure 13-3(c) introduces new symbols to illustrate the creation of a message digest. The symbols represent the plaintext message (Alice's newsletter) crumpled into a unique wad of paper. The pictures of Alice's newsletter and its corresponding message digest aren't to scale. In reality, Alice's newsletter could be 10,000,000 bytes or more, whereas a SHA-1 message digest is only 160 bits (20 characters). Figure 13(d) shows a signed (private key encrypted) message digest. Figures 13-3(e) and 13-3(f) contrast with 13-3(b); the message digest (in-

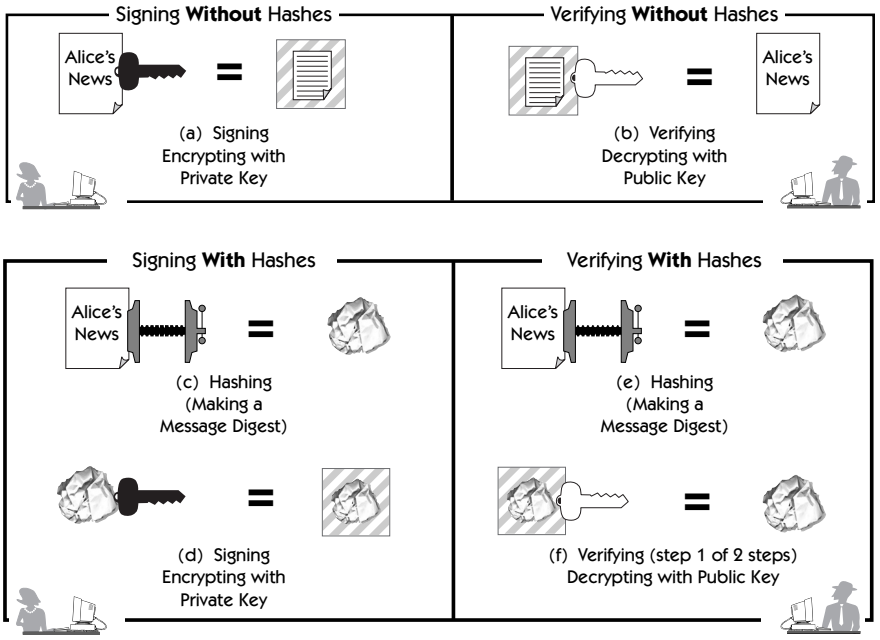

Figure 13-3 (a), (b): Signing and verifying the whole message (no digest). (c), (d): Making and signing a message digest. (e), (f): Making and verifying a message digest.

stead of the message) is used for signing and verification. The rest of this chapter explains how a signed message digest verifies message integrity (ensuring that the message was not altered during transit).

Detecting Intentional Modifications

Because message digest signing and verification is probably one of the most confusing cryptographic processes, let's look at the pieces of that process one step at a time. First, let's see how an unsigned (not private key encrypted) message digest detects intentional modifications, just like the checksum shown in the preceding section. Then we'll show BlackHat successfully attacking an unsigned digest. Finally, we'll show how a signed message digest stops BlackHat.

Figures 13-4 and 13-5 show how Alice and her customers use a message digest as a redundant proxy for her newsletter. In Figure 13-4 Alice makes a message digest and sends both the newsletter and the (unsigned) message digest to Bob (and her other customers). Bob verifies the newsletter's authenticity and integrity with the three steps shown in Figure 13-5.

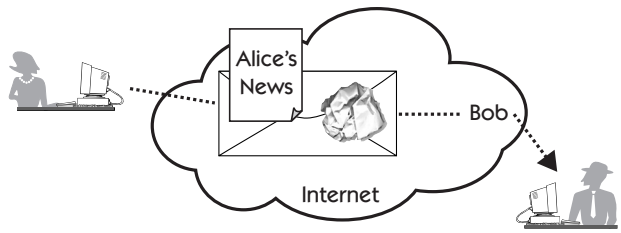

Figure 13-4 Alice sends a message and an unsigned message digest to Bob.

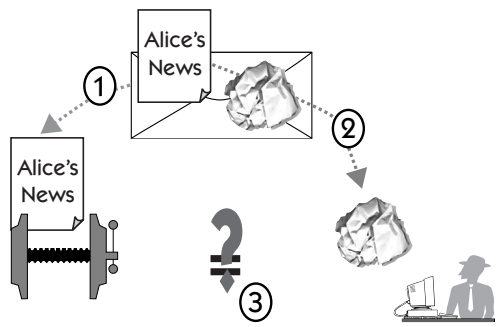

Figure 13-5 Bob independently calculates a digest and compares it to the digest sent by Alice. If they are equivalent, he trusts that the message has not been modified during transit.

1. Bob extracts Alice's newsletter and independently calculates a message digest.
2. Bob extracts the message digest Alice sent to him.
3. If Bob's independently calculated message digest (from step 1) equals the message digest he received from Alice (from step 2), he trusts that the newsletter has not been modified since Alice made the message digest.

BlackHat successfully attacks because Alice hasn't attached anything uniquely hers to the message.

Note that Alice did not sign anything—that is, she hasn't used her private key to digitally put her identity on the message or digest. Let's watch BlackHat convince Bob he has Alice's newsletter, when in reality BlackHat has intercepted Alice's newsletter and substituted his forgery. In Figure 13-6, BlackHat forges Alice's newsletter, and makes a message digest from the forgery (a gray newsletter depicts BlackHat's forgery). Then, in Figure 13-7, BlackHat intercepts Alice's genuine newsletter and message digest and substitutes his forged "Alice's NEWZ" and digest.

In Figure 13-8, Bob independently calculates a message digest from the forged newsletter following the same procedure shown in Figure 13-5. Because his calculated message digest is equivalent to the one he received, he accepts the

Figure 13-6 BlackHat forges a newsletter and makes a message digest from the forgery.

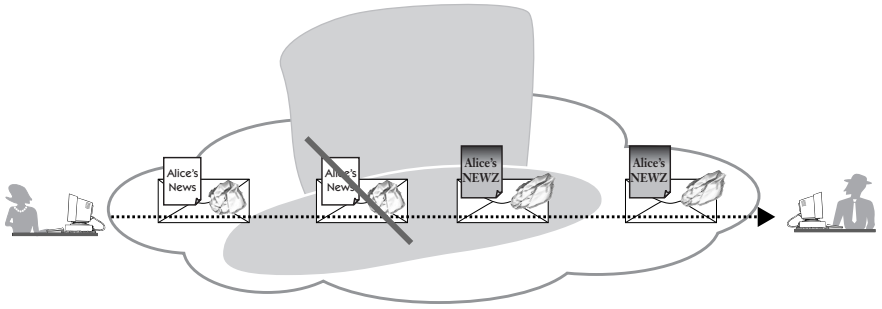

Figure 13-7 BlackHat intercepts Alice's genuine newsletter and digest and substitutes his forged newsletter and digest.

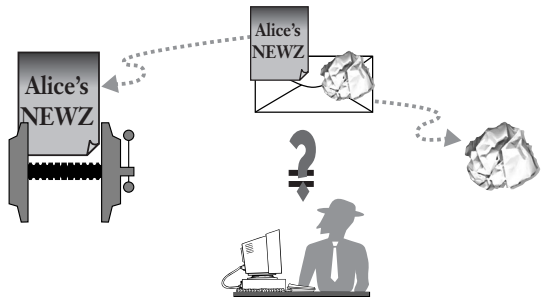

Figure 13-8 Bob verifies that the newsletter hasn't changed since the attached message digest was made.

newsletter as unaltered since it left the newsletter creator. Bob's logic isn't wrong; the message and the message digest are a matched pair. But as shown in Figure 13-8, there's no signed (private key encrypted) message digest to help Bob detect that Alice did not create the newsletter he received. Unfortunately, Bob buys CrashingDotCom.

Let's recap how BlackHat tricked Bob. Bob uses the newsletter/digest pair to determine whether the newsletter was modified after it left the sender. But BlackHat becomes the sender by substituting his own newsletter/digest pair.

Signing the Message Digest

The difference between an unsigned and a signed message digest

To prevent BlackHat from successfully forging her newsletter, Alice signs a message digest of her newsletter. Figure 13-9 illustrates the difference between a signed and an unsigned digest. Figure 13-9(a) shows Alice making and sending a digest. Alice and Bob tried this in Figures 13-4 and 13-5, but it was successfully attacked. Figure 13-9(b) shows Alice making and sending a signed digest.

Verifying a signed message digest

First, let's see how Bob verifies that the newsletter he receives is Alice's authentic newsletter. Then we'll examine how the signed message digest detects BlackHat's forgeries. Figure 13-10 shows Bob verifying that the message has not been modified since Alice made and signed the digest. (For convenience, the upper-left corner of Figure 13-10 shows Figure 13-5.) Figure 13-10 is similar to Figure 13-5 except that Bob first decrypts the signed digest with Alice's public key. Bob then verifies the newsletter's genuineness in the same way as shown previously.

As shown in Figure 13-9(b), Alice attached her identity to the newsletter digest. Bob knows that the newsletter is authentically from Alice because only Alice's public key will verify (public key decrypt) a message digest equal to the one Bob independently calculates from the newsletter he received.

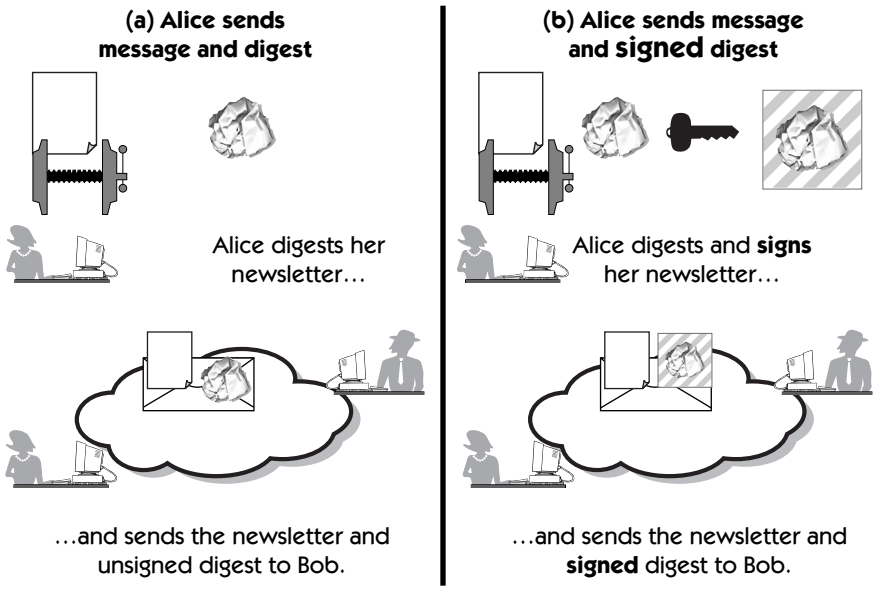

Figure 13-9 The difference between a message digest and a signed message digest. An unsigned message digest doesn't detect BlackHat's forgery.

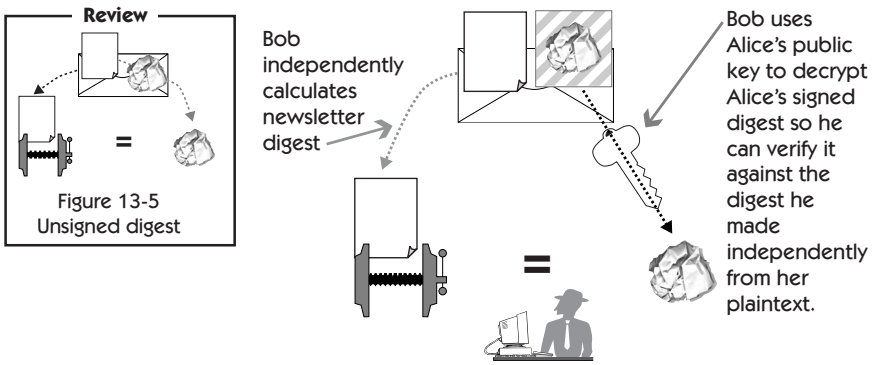

Figure 13-10 Bob verifies Alice's newsletter.

Detecting BlackHat's Forgery

Signing digest detects BlackHat's forgery.

The signed newsletter digest prevents BlackHat from fooling Bob with his forgery. Let's illustrate what happens when BlackHat tries to send his forged newsletter to Bob.

Let's first review some basics in Figure 13-11. In particular, notice that encryption with one key and decryption with its matching key is like multiplying by 1. In Figure 13-11(a), a plaintext message is encrypted and decrypted, and a message digest is signed (encrypted) and verified (decrypted) with Alice's private/public key pair. In Figure 13-11(b), a message digest is signed (encrypted) with BlackHat's private key and verified (decrypted) with Alice's public key. Note that because BlackHat's private key is not a companion to Alice's public key, the result of encryption/decryption is like multiplying by anything *other* than 1. In other words, BlackHat can't make a digest, sign it with his private key, and expect that decrypting with Alice's public key will produce his digest. With this in mind, we're ready to see why BlackHat can't successfully complete his forgery.

Figures 13-12(a) and (b) compare Bob verifying Alice's authentic newsletter and rejecting BlackHat's forgery. In both (a) and (b), Bob receives a plaintext newsletter and an encrypted newsletter digest. He calculates a newsletter digest from the plaintext newsletter. Then he decrypts the encrypted newsletter digest with Alice's public key.

In Figure 13-12(a), Bob's calculated digest and the digest he decrypts with Alice's public key are equal; Bob accepts the newsletter as genuine. But in Figure 13-12(b) they are not equal, and Bob rejects the newsletter. In (b) they are not equal because, as you saw in Figure 13-11(b), signing (encrypting with BlackHat's private key) and verifying (decrypting with Alice's public key) is like

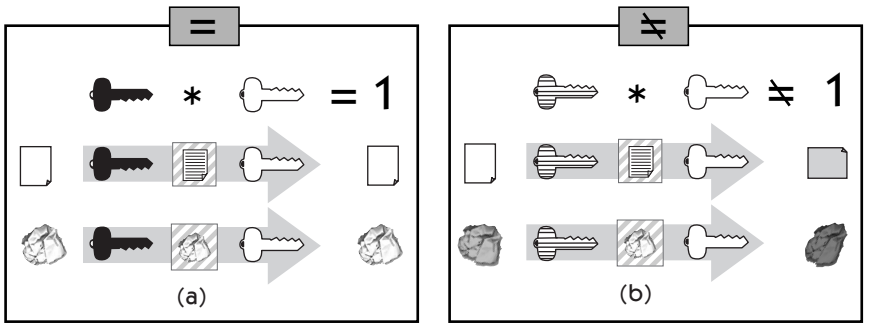

Figure 13-11 (a) Encrypting and decrypting with Alice's private/public key pair is like multiplying by 1, but (b) encrypting with BlackHat's private key and decrypting with Alice's public key is like multiplying by anything except 1.

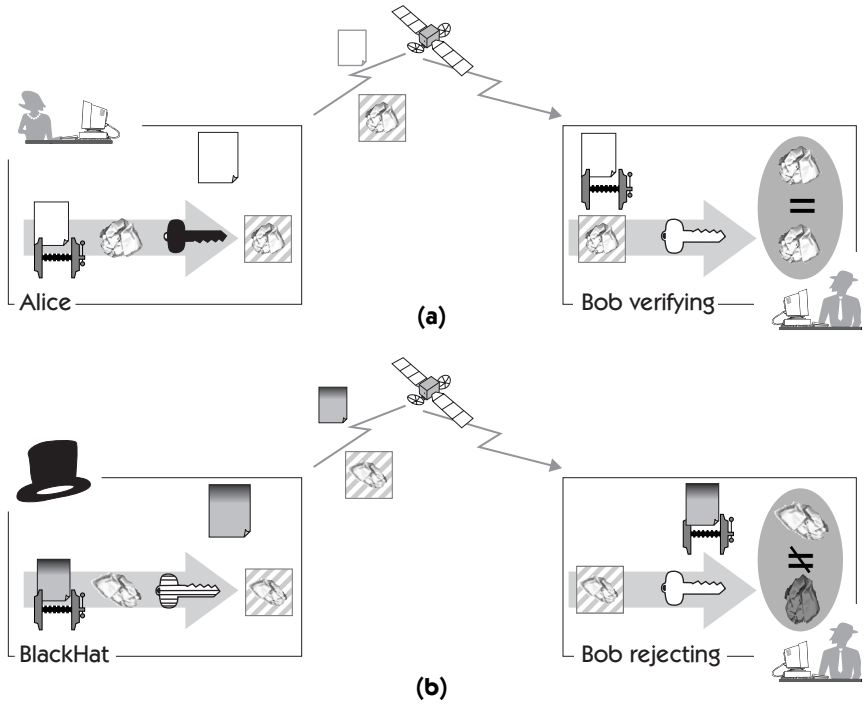

Figure 13-12 Bob decrypts a digest using Alice's public key. (a) Bob retrieves exactly what Alice signed; (b) Bob retrieves something different from what BlackHat signed because BlackHat's private key is not paired with Alice's public key.

multiplying by anything but 1. Because BlackHat doesn't have Alice's private key, he can't make a signed (encrypted) newsletter digest that correctly verifies (decrypts) with Alice's public key, as seen in Figure 13-12(b).

Replay Attacks

BlackHat could fool Bob by sending a copy of an old Alice newsletter and digest. For example, suppose that BlackHat recorded Alice's November 2000 newsletter. In February 2001 he intercepts Alice's newsletter and substitutes the one from November. This is called a *replay* attack. To defeat this attack, Alice can timestamp her newsletters.

Supplement: Unsuccessfully Imitating a Message Digest

Although the following attack won't work, it's instructive to examine because digests are designed to protect against this attack.[3]

Because BlackHat can't easily attack Alice's signed message digest, he contemplates the attack shown in Figure 13-13. In the left portion of the figure, BlackHat makes a newsletter that digests to the same value as Alice's newsletter. Again, Alice's genuine newsletter is white and BlackHat's is gray. In the right top portion of the figure, BlackHat removes Alice's newsletter—but not Alice's signed message digest—and substitutes his forgery. Bob decrypts Alice's signed digest, independently calculates a hash from BlackHat's forged newsletter, and accepts it as authentically Alice's. BlackHat was successful because his fraudulent newsletter and Alice's newsletter made equivalent message digests.

BlackHat's full attack is shown in Figure 13-14. As before, Alice makes a newsletter and a signed newsletter digest. BlackHat intercepts Alice's newsletter and substitutes his own newsletter; note that he does not substitute a new signed newsletter digest.

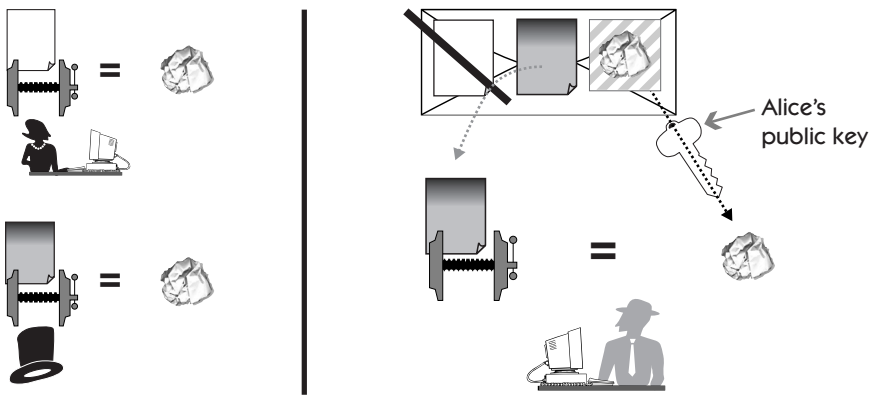

Figure 13-13 BlackHat makes a newsletter digest equivalent to Alice's newsletter digest.

3. This is an optional section. You need not read or understand this section in order to understand the rest of the book.

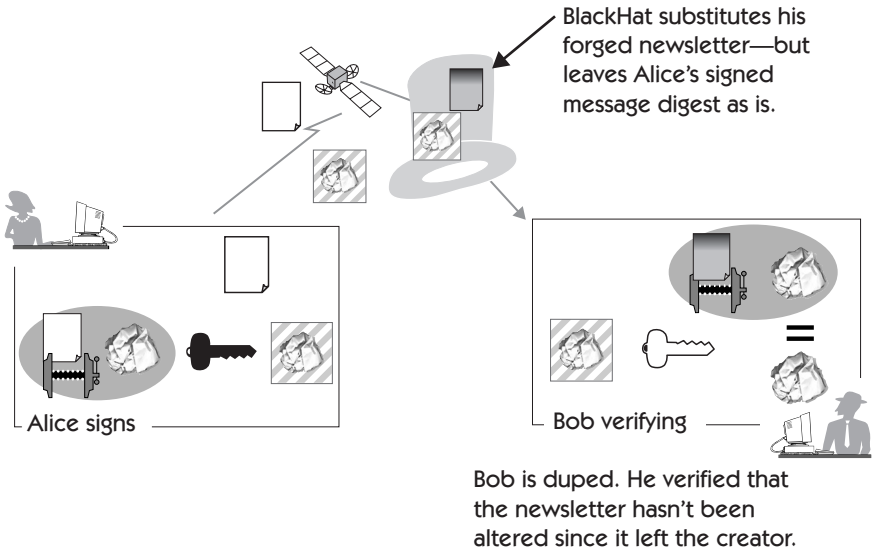

Figure 13-14 BlackHat forges a newsletter. Because BlackHat's newsletter digest is exactly equivalent to Alice's newsletter digest, Bob mistakenly accepts BlackHat's forgery.

Message digest assurances prevent this attack.

As before, Bob receives BlackHat's forged newsletter and Alice's signed newsletter digest. He independently calculates a newsletter digest except that now he doesn't know that the newsletter is BlackHat's forgery. As before, Bob decrypts Alice's signed message digest. He compares the two digests and accepts BlackHat's forgery. The shaded ovals show that BlackHat tricked Bob (and Alice) by defeating the uniqueness of Alice's message digest program; he was able to make a newsletter whose digest was equal to Alice's newsletter digest. Unfortunately, Bob has no way to detect this forgery, and he accepts BlackHat's newsletter as Alice's genuine newsletter.

Fortunately, cryptographic message digest methods ensure that BlackHat can't make an equivalent digest. That's the subject of Chapter 14.

Review

Because public key encryption and decryption are slow, cryptographers invented a condensed representation of a message, called a message digest or cryptographic hash. Message digests are used as short proxies for usually much larger messages and are designed to detect intentional modification to a message.

By signing the newsletter digest, Alice attaches her identity to the digest just as if she had signed her newsletter. She signs the message digest because it's more efficient than signing the underlying message.

Alice sends and Bob verifies using the following basic procedure.

1. Alice makes a message digest from a plaintext message.
2. Alice signs the message digest and sends the signed digest and plaintext message to Bob.
3. Bob independently re-creates the message digest from the plaintext.
4. Bob decrypts the signed message digest with Alice's public key.
5. Bob verifies that the message is authentic if the message digest he created is identical to the decrypted message digest he received from Alice.