CHAPTER 9

AUTHENTICATION TOKENS

DARK HELMET: So the combination is 1-2-3-4-5. [Lifts helmet and yells] That's the stupidest combination I ever heard in my life! That's the kinda thing an idiot would have on his luggage!

— Mel Brooks et al., *Spaceballs*

IN THIS CHAPTER

Password sniffing has been especially troublesome in remote access and networking environments. This led to several technical strategies and commercial products to generate unsniffable passwords.

- Passwords using hardware tokens
- Making passwords unsniffable with one-time passwords
- One-time password tokens using internal counters or clocks
- Tokens with PINs
- Configuring one-time passwords
- Attacks on one-time passwords

9.1 TOKENS: SOMETHING YOU HAVE

The essence of token-based authentication is that you must have the token in your possession in order to authenticate yourself to the computer. The computer will not recognize you without the token, regardless of whether the token was lost, lent, or stolen. This is the way tokens have worked for thousands of years. Important people have used personal seals to authenticate important documents, and the recipients of the documents would treat the seal's impression as evidence that the person really produced the document. The seal served as an authentication token for documents.

Key-operated mechanical locks are probably the most common authentication tokens used today. Section 2.2 shows how a key carries a base secret (Figure 2.2), and authentication tokens likewise carry a base secret. In fact, the familiar benefits and shortcomings of keys often also apply to computer-oriented authentication tokens.

Everyone has a personal trove of stories about keys: losing them, finding them, locking them in the car, and so on. Much of this translates into the behavior of computer authentication tokens. Here is a summary of the fundamental properties of tokens from a security standpoint:

- A person must physically possess the token in order to use it.
- A good token is hard to duplicate.
- A person can lose a token and unintentionally lose access to a critical resource.
- People can detect stolen tokens by taking inventory of the tokens they should have in their possession.

Numerous authentication products use tokens, for both computer and noncomputer applications. One reason for their popularity is that they take away much of the burden of memorization. A token can reliably carry a much more complicated base secret than most people can memorize. Few people have the memory or motivation necessary to memorize a really strong password. Tokens can carry very complicated base secrets without burdening the owner with any memorization. At most, a token might require a PIN.

Tokens generally fall into two categories: passive and active. In both cases the token incorporates a base secret, and one must copy the base secret in order to make a working copy of a particular token. A *passive token* is simply a storage device for the base secret. Examples include mechanical keys, ATM cards, most employee badges, and some specialized devices like “datakeys.” An *active token* can generate different outputs under different circumstances. For example, an active token can take part in a challenge response authentication protocol, or provide other crypto functions that use the token’s base secret. Traditionally, active tokens have been either commercial one-time password tokens or smart cards, though other models have evolved that plug into existing ports on desktop and laptop computers.

PASSIVE TOKENS

Passive tokens work by presenting their base secret to the authentication mechanism: a lock responds to the notches on a key, or a card with a magnetic stripe transmits its stored data via a card reader. ATM cards, for example, emit the bank account number stored on a particular card. Aside from these, there are numerous other examples: "datakeys" from Datakey, Inc., that look like keys but contain a read-only memory chip, plastic cards with patterns of punched holes, cards with data encoded in optical bar codes, and wire coils that emit a specific signal when passed near the appropriate reader.

The wire coil systems are often referred to as proximity cards since they are recognized when they are near the card reader. Examples include the XyLoc from Ensure Technologies and the Sage-ID from Tri-Sage. Some models work only when they are right next to the reader, while others work at a distance of a few feet or even a hundred feet. The cards that work at the greater distances usually contain a battery; they are often referred to as "active cards" although they only emit a fixed code. One vendor provides a workstation security system that activates a lock screen whenever the proximity card leaves the workstation's vicinity.

The most common passive tokens are, of course, plastic cards with a magnetic stripe ("mag stripe cards"). Today, they appear everywhere, including ATM cards, credit cards, drivers' licenses, and on employee badges to operate electronic door locks. The cards, their sizes, and their data formats are fully standardized and available from countless manufacturers. One estimate claims that mag stripe readers cost as little as $10, and mag stripe writers as little as $20 (purchased in quantity, U.S. currency). *see Note 1.*

The problem with passive tokens is that they are often too easy to copy. People often assume that tokens won't be copied because they require special equipment like a key copier or a mag stripe reader. In fact, such technologies are often victims of their own success. Consider mechanical keys: we find key copying equipment everywhere since it's a simple and profitable activity, and a determined attacker can even copy a key without using a copying machine. Even mag stripe equipment is now cheap and relatively common. In late 1999, a clerk in a fancy department store allegedly took advantage of mod-

ern miniaturization by copying customers' mag stripes with a compact reader she had attached to her Palm Pilot. *see Note 2.*

Many, if not most, of today's mag stripe cards deal with the problems of theft and copying by incorporating a PIN. Many companies that use card-controlled locks will demand a PIN before opening important doors, like those from the street. And, of course, ATMs require both the card and its PIN before they perform banking functions. The combination of card and PIN produces two separate authentication factors: something you have plus something you know. This reduces the risk of masquerade, since the attacker must copy the card and also uncover the PIN.

This is not to say that mag stripe card security is foolproof. We've already seen that attackers can make copies of cards; thieves have been equally resourceful at acquiring the PINs to go with those cards. A trivial strategy is to simply stand next to a victim using the ATM and note the PIN as it is typed in (an example of Attack A-9, Shoulder surfing). Some banks would transmit PINs from remote ATMs over conventional phone lines with no additional protection, and thieves would simply eavesdrop on the transmission to recover both PINs and card numbers. Other banks would store the PIN on the card in some form, and thieves would use a variety of techniques to either recover the PIN or to substitute a different account number if the card's PIN is already known.

Some attacks involved PIN recovery by bank employees. About 1% of all bank employees are discharged every year for disciplinary reasons like petty theft, so the risk of embezzlement by staff is very real. Cases have been reported in which bank staff sold PINs to the local crime syndicate. Dishonest bank programmers have even been caught modifying the bank's PIN assignment software to make the PINs easier for their partners in crime to guess. Such problems can't be fought by changing technologies; the bank must use strategies like separation of duty instead. *see Note 3.*

ACTIVE TOKENS

An *active token* doesn't need to emit its base secret to authenticate its owner. Instead, it uses the secret to do something else—like generate a one-time password. Tokens for computer authentication don't have to implement one-time passwords, but this is what tradi-

tional token products do (SafeWord, SecureID, CryptoCard, etc.). More sophisticated tokens connect directly to a workstation and can often perform a variety of crypto functions in addition to authentication. Figure 9.1 illustrates some active tokens.

Active tokens can use crypto techniques for authentication that are immune from attacks via sniffing and replay. Clearly, sniffing is a problem when we authenticate across a network. Token-based authentication will essentially generate a different set of messages each time its owner tries to authenticate, and it does the attacker no good to try to replay a previous set of messages. Moreover, the attacker can't predict what a correct set of authentication messages should be, assuming that the token's authentication protocol is well designed and implemented.

Traditional one-time password tokens evolved in the days of time-shared remote access. Originally, these tokens were the size of pocket calculators and often included a keypad, but more recent tokens also show up on key rings. These tokens generate authentication data to be typed in on a keyboard, so they can work with any interactive computer system and don't require special hardware. However, this also limits the complexity of their authentication pro-

FIGURE 9.1: *Seven examples of active tokens.* These tokens come in a broad range of shapes, sizes, and technologies. *Clockwise from upper left*, a SafeWord Silver 2000 one-time password token from Secure Computing; another SafeWord token—the SafeWord Gold 3000; an iKey USB token from Rainbow Technologies; *right*, a Fortezza crypto card—an NSA-developed PCMCIA card; an iButton ring from Dallas Semiconductor designed to be worn as jewelry; a SecurID one-time password token from RSA Security; a typical credit card-sized smart card produced by Schlumberger.

tocols, since they rely on the user to transcribe authentication data between the token and the computer.

Other active tokens plug directly into a computer and handle more complicated authentication protocols. Today, these come in a variety of packages, as shown in Figure 9.1. The principal example of such tokens is the *smart card*, a device the size of a credit card that contains a small processing system and memory. A few active tokens have also been implemented as "PC cards," which are also known as PCMCIA cards, named after the Personal Computer Memory Card International Association that established the standard. Recently, several vendors have introduced "USB tokens" that connect directly into the Universal Serial Bus: an emerging standard for connecting peripherals like printers to personal computers, desktop and laptop alike. Other tokens, like the iButton, require a special-purpose reader.

Smart cards have been used extensively in Europe, so the technology has seen a good deal of real-world use. European credit cards have been smart cards since the 1980s. Another early application was as value storage cards. For example, a public transit authority could use smart cards to hold a passenger's prepaid fares. When a passenger like Tim entered a train or bus, he would put his smart card in a reader, and the reader would deduct the cost of the fare from his card. When the card ran low, Tim went to a special machine, inserted some money, and the machine added value to the card.

Internally, the card would be programmed with a protocol that worked only in conjunction with computers belonging to the transit commission. This usually involved secret data shared between all valid cards and the transit commission. Even if Tim had a smart card reader on his home computer, the card wouldn't let him add or deduct from the fares stored there.

Recently, credit card vendors in the United States have started offering smart cards intended to provide security for e-commerce transactions. As of this writing, many of these cards work by sharing a base secret between the cards and the credit card vendor, and the secure transaction protocol works only in conjunction with the card issuer's Web site. To some extent, this is because it is so hard to share secret keys safely across enterprises; this reflects funda-

mental differences between the indirect and off-line design patterns discussed in Chapter 4. Public key cryptography may provide broader applications for smart cards, so further discussion of them appears in Section 15.3.

Smart cards and other plug-in tokens share certain security strengths and weaknesses. All of them usually present the attacker with a security perimeter that can be difficult to physically penetrate, since the internal logic is usually embedded in a single integrated circuit. However, these devices aren't immune to attack. Unlike conventional password tokens that may contain their own keypad and display, these tokens must rely entirely on other equipment for input and output. Attackers can exploit that if they subvert the software that talks to the token and possibly trick the card, its user, or the proprietor. Moreover, some devices are not well designed to resist probing, and a knowledgeable attacker can extract critical data from them while using modest resources. *see Note 4.*

As of this writing, the major limitation facing plug-in active tokens is that they depend on connections that aren't present in many, often most, desktop computers. Obviously, Cathy can't log on if her workstation won't take her smart card. Tokens built as PCMCIA cards can usually be plugged into laptops, but few desktop systems routinely include a PCMCIA interface.

The connection problem might tilt some proprietors in favor of USB tokens: although such tokens may cost more to manufacture than smart cards, they cost less than the combination of a smart card plus its reader. From a security standpoint, USB tokens appear to offer the same theoretical promise of hardware-enforced security as smart cards. Experiments have also found, however, that a knowledgeable and properly equipped attacker can penetrate the hardware protection of many USB products. *see Note 5.*

9.2 Network Password Sniffing

The story of Ali Baba, which opened Chapter 1, begins with overhearing the password to the thieves' cave: a sniffing attack. In Section 1.5 we see passwords recovered through shoulder surfing and even by copying them out of RAM within a computer. But today's users on modern wide area networks face a much broader threat. Today, attackers often sniff unprotected passwords out of messages

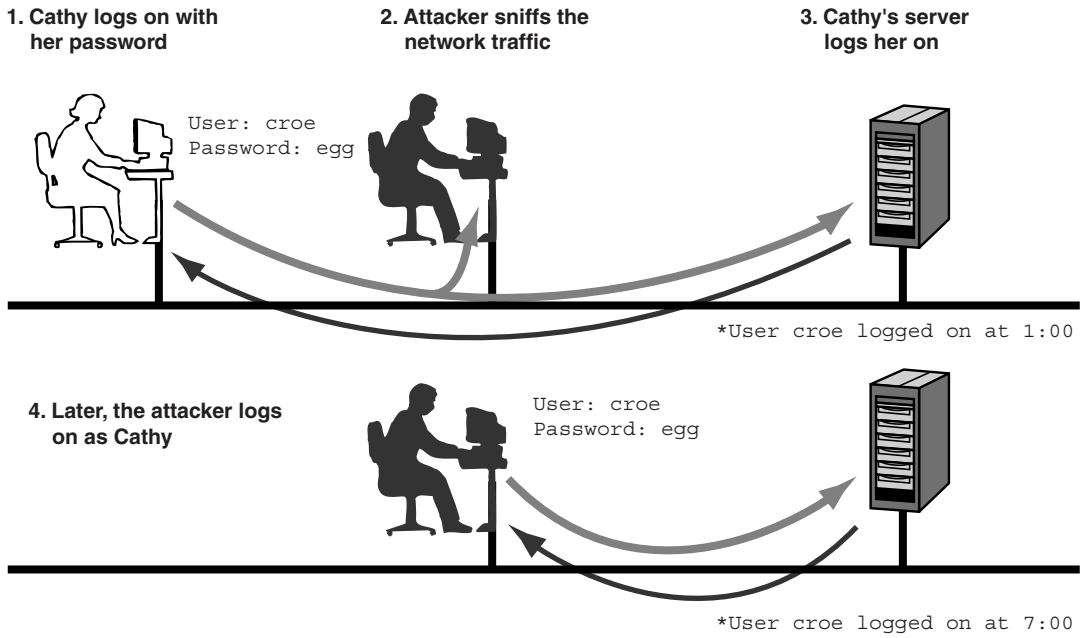

FIGURE 9.2: *Sniffing passwords on a network.* First, Cathy types in her "reusable" password to log in to the server. Next, an attacker sniffs it while it travels across the network, and uses it to masquerade as Cathy.

in transit between computers. Some attackers install sniffer software within computers belonging to an Internet service provider. The sniffer software snoops on all traffic passing through that part of the Internet. This provides the attacker with a lot of passwords to exploit.

Figure 9.2 shows the network sniffing strategy. Cathy Roe logs on to a service by typing her user name and password. As that information travels across the network to her service, an attacker sees it and copies it down. Neither Cathy nor her service can reliably detect the attacker, and Cathy logs on successfully. Later, the attacker uses Cathy's user name and password to log on himself. The service can't tell the difference between the attacker and Cathy, so the attacker logs on successfully.

A-54

Sniffing the password data from the masses of other network traffic isn't particularly difficult. There are several indicators that will tell an automated sniffer program that a message is part of someone logging on. Obviously, the program can look for prompts like "User

name" and "Password." Also, the program can simply look for the beginning of a connection. In the days when many people used line-oriented terminals, it was simple to look for the start of a Telnet connection and copy the first several dozen characters sent back and forth.

Sniffing is one of those problems (if not *the* problem) that distinguishes internal password environments from external ones, as discussed in Section 6.2. Sniffing makes authentication on public networks much trickier than on small private networks and LANs. While reusable passwords might pass safely across a small LAN in some environments, it is an invitation for attack on the Internet.

An obvious solution is to encrypt the password: to transform it into a nontextual form that the attacker can't transform back into text. If the password isn't in a readable textual form, then the attacker can't type it in when he tries to log on as Cathy.

D-54

Unfortunately, the solution isn't as simple as it looks. Classic encryption requires a secret key that Cathy shares with her server. Setting up and sharing that secret key is at least as complicated as managing Cathy's password. In a sense, straightforward encryption gives her two passwords to worry about instead of one (the SSL protocol neatly addresses this problem in Chapter 13).

Instead of straight encryption, we could compute a one-way hash of the password and transmit that instead. This doesn't require a shared secret key. If the attacker intercepts the hashed password, he can't easily turn it back into the original textual password, and he can't use the hashed password directly to log in.

However, the hashed password opens us up to a replay attack: the attacker could modify his system to simply provide an encoded password when he tries to masquerade as someone else. If the service simply expects a hashed password, the attacker can simply replay Cathy's hashed password. The server won't be able to tell if the password was generated from the original textual version or was simply replayed. The hashed password is really a *password equivalent* that provides only a modest amount of security over using the plaintext password itself.

A-55

Imagine if the 40 thieves had used a hashed password: the word would probably have just sounded like gibberish instead of a genuine word in Hindustani. Regardless, Ali Baba and his brother could

still have been able to intercept it and use it to open the cave's door (assuming, of course, they could pronounce it accurately).

On the other hand, what if the cave's door used a whole series of passwords, changing the password each time one was used? The incantation "Open, Sesame" would work only once. When Ali Baba tried it again later, it would not have worked, the thieves' treasure would have been safe, and Ali Baba's story would have been forgotten.

If we do this with computer services, then the sniffed password does the attacker no good. This is the basic concept behind *one-time password* mechanisms. The service expects a new password each time the person logs on. The protocol used by the particular one-time password mechanism establishes how to generate the correct password.

D-55

But it's hard enough to remember a single password, and most people would find it impossible to figure out the next password to use each time they had to log on. That's where the tokens come in: commercial one-time password systems use hardware or software to generate the user's next password.

9.3 ONE-TIME PASSWORDS

There are two general strategies for generating one-time passwords. *Counter-based* tokens combine the base secret with a synchronized counter to generate one-time passwords. *Clock-based* tokens use a synchronized clock to generate one-time passwords. One vendor has even developed a product that combines both techniques. All these techniques rely on a random base secret stored in the one-time password token. New passwords combine the base secret with some arbitrary value: a counter, a clock, or both.

COUNTER-BASED ONE-TIME PASSWORDS

In 1994, an international team of thieves gained unauthorized access to the Citibank Cash Management System, which allows major Citibank customers to transfer money from their Citibank accounts to other financial institutions around the world. As with many crimes, the details may never be known for sure, but certain things seem likely based on reports about the incident. In particu-

lar, the reports suggest that thieves never needed to burglarize Citibank's cash management computer center in Parsippany, New Jersey. Instead, they accessed the system remotely using stolen passwords.

The scheme was apparently masterminded by 24-year-old Vladimir Levin, the head systems operator at A O Saturn, a computer company in St. Petersburg, Russia. Levin and his accomplices acquired and shared passwords that authorized over 40 transactions, totaling over $10 million. The thieves authenticated themselves as legitimate account managers and moved money between bank accounts in Finland, Russia, Germany, the Netherlands, the United States, Israel, and Switzerland. Some team members, like Alexei Lashmanov, 28, provided access to particular accounts to receive stolen funds, while others actually extracted money from unlucky accounts.

see Note 6.

The team's efforts finally unraveled in October 1994 after five months of activity. The final package of transactions, totaling $2.8 million, ended as an accomplice was arrested in San Francisco while attempting to open a bank account to receive the stolen funds. Citibank managed to cancel or reverse all except $400,000 of the thieves' transactions.

To prevent a repetition of the problems, Citibank installed a counter-based one-time password system for authenticating cash management accounts. Each account manager was issued a token and could not perform transactions without it. This by itself eliminated a major problem with passwords: there was no way someone could induce an account manager to share an account's password without giving up the token as well. In addition, it prevented people with trusted access to cash management messages (like systems operators) from intercepting and reusing an account's password.

Counter-based tokens incorporate an internal counter and use it to generate a fresh password each time its owner needs one. Typically, the owner presses a button on the front of the token; the button increments the counter and the token displays the new password on a built-in display (Figure 9.3). If an attacker intercepts a password and tries to reuse it, the system won't recognize the password as valid (Figure 9.4). Furthermore, the attacker can't guess the next password by examining the sequence: the token com-

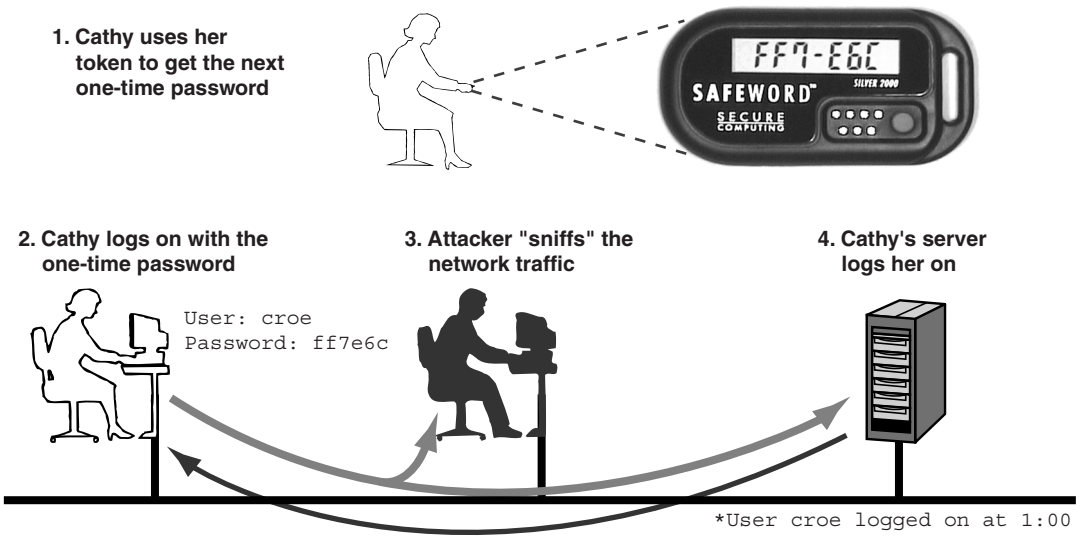

FIGURE 9.3: *Using a one-time password token.* Cathy pushes the button on her password token to display her next one-time password. She types it in to the password dialog. Her server accepts the login. An attacker sniffs the one-time password, but can't use it to log in (see Figure 9.4). The card shown here is a SafeWord Silver 2000 token from Secure Computing Corporation.

putes each new password by encrypting the counter with an internally stored base secret.

SafeWord, a product of Secure Computing Corporation, is a counter-based token. As shown in Figure 9.5, the token increments its internal counter, combines it with the owner's base secret, and generates a one-way hash value from them. The token displays the result in its window. The ActivCard token, a product of PC Dynamics, uses a variant of this technique discussed in the next section.

The exact process for producing the one-way hash depends on the particular product in question. There are a variety of techniques for producing a hash, as noted in Section 8.5. SafeWord, for example, uses 56-bit DES in a MAC construction to produce each one-time password.

The one-time password token generates a correct password as long as both the token and the destination computer have matching copies of the token's internal counter and base secret. Initially, an administrator inserts these values into the token using a special wand that connects through a cluster of seven contacts on the front. The token in Figure 9.3 shows these contacts on the lower right. The

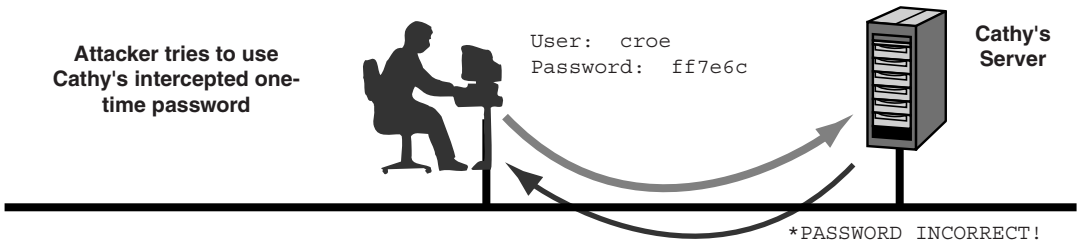

FIGURE 9.4: *An attacker trying to replay a one-time password.* The attacker has sniffed a one-time password and tries to use it to log on as croe. The server accepts the password only once, so the attacker fails.

SafeWord administration software generates the 56-bit base secret randomly and saves the result in the token's account record along with an initial 20-bit counter value.

Besides the initial password values (the base secret and the initial counter value), the token programmer can also configure the token in particular ways. For example, tokens can often display passwords in several different formats that vary the number of digits or the character set used in passwords. The token uses the format selected when it is programmed. More elaborate tokens like the SafeWord Platinum card (Figure 4.3) can store two or more different sets of initial counter values. This allows the card's owner to log in to different systems that use different secrets. The owner uses the keypad on the front of the card to choose among the systems. Some tokens may also be programmed to require a PIN before emitting a one-time password.

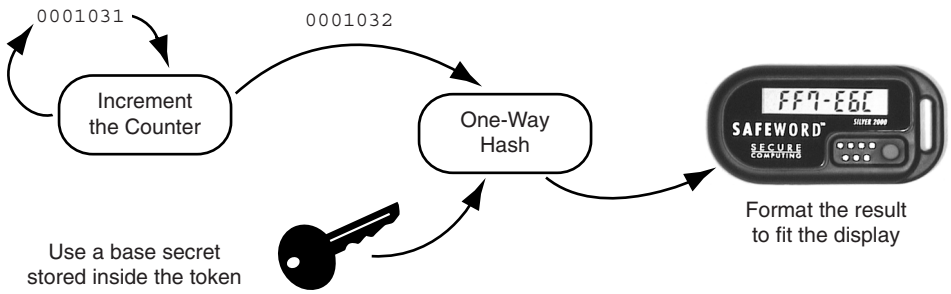

FIGURE 9.5: *Computing a counter-based one-time password.* The owner pushes the button on the token. The token increments the internal counter, combines it with the owner's unique base secret, and computes a one way hash result. The token formats the result and displays the new one-time password.

Counter synchronization is an important design issue for counter-based systems. Ideally, the token's owner should only press the password button (and thus increment the token's counter) when actually logging on to the system. Otherwise, the token's counter increments by one while the corresponding counter in the server remains the same. This can cause a mismatch between the server and the token. To resynchronize, most tokens assume that the person will try to log in again immediately and use the token's next password in sequence. Then the server examines the pair of passwords to verify that they could have been generated by the owner's token.

In short, this strategy uses a double-length password that would be significantly harder for an attacker to guess at random. The SafeWord server always saves the previous password it received from the token and, if the user tries to log on twice in a row, the server checks to see if the pair of passwords could in fact have been generated by that particular user's token. To do this, the server increments the counter and tests the corresponding password against the first incorrect password, repeating several times and looking for a match. If the server finds a match, it increments the counter again and compares the result against the second password from the token. If the server can match two passwords in a row like that, it authenticates the user and updates the counter to reflect the second of the pair of passwords. According to the vendor, the process is designed to keep the likelihood of successful guesses below one in a million.

see Note 7.

Note that the "one in a million" likelihood yields an average attack space of 19 bits, and the attacks must be performed interactively. Off-line attacks might be performed against the base secret by examining the stream of one-time passwords. Cracking the 56-bit DES key off-line would, of course, involve an average attack space of 54 bits.

The ActivCard token uses a slightly different procedure to resynchronize the counter it uses with a counter on the server. ActivCard passwords incorporate a low-order digit from the counter in each password it generates. If the digit doesn't match the counter on the server's side, the server will try a higher counter value with the same low digit, within a predefined range of values. If the result suc-

ceeds, the server accepts the authentication and updates its internal counter. Otherwise, the authentication fails and the counter remains unchanged.

see Note 8.

An interesting feature of the counter-based approach is that the system does not behave well if someone makes a copy of a token. If people use both the original token and a copy of it to log on, some attempts will fail and others will cause resynchronization. Although both tokens will work some of the time, the system will not behave correctly. In some cases it may be possible to identify this failure pattern to detect duplicated tokens.

CLOCK-BASED ONE-TIME PASSWORDS

The SecurID token, a product of RSA Security, uses internal clocks instead of counters. SecurID tokens combine an internal clock value with a 64-bit secret key to generate a time-based password each time the user needs one (Figure 9.6). The token reads its internal clock, which reports the number of seconds since the beginning of 1986. Then it computes a one-way hash that combines the clock value with the owner's base secret. The token displays the result, which the owner then uses as the password when logging in.

The server performs essentially the same process by combining its own clock value with a copy of the token's secret key. To account for the time required to transmit the password, as well as clock drift between token and server, the server accepts any password derived from a clock value within an established "time window." In some

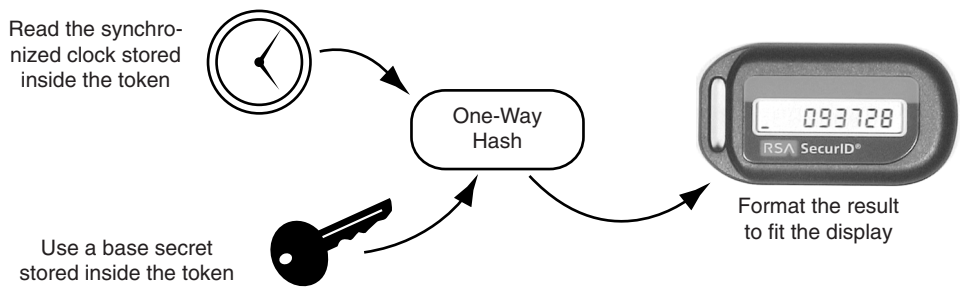

FIGURE 9.6: *Computing a clock-based one-time password.* The owner activates the token. The token reads the internal clock, combines the time with the owner's unique base secret, and computes a one-way hash result. The token formats the result and displays the new one-time password. The token shown on the right is a recent SecurID model sold by RSA Security.

environments, the server's time window has been as short as 30 seconds, though today's network oriented systems typically use a three-minute window.

The technical details of SecurID password generation are proprietary, but they have been leaked to the security community and widely distributed. If we assume that the hash function is computationally secure, the 64-bit key should present attackers with a 63-bit average attack space. As of this writing, the cryptographic community has begun open discussions of the strengths and weaknesses of the SecurID hash function. As yet, no serious weaknesses have been found that would cast doubt on its effectiveness as a one-time password token. Its principal vulnerabilities are shared with all one-time password tokens, as discussed in Section 9.4. *see Note 9.*

In an ideal world, the clock-based tokens would not have to deal with resynchronization like counter-based tokens. SecurID tokens and servers in modern systems are synchronized with Universal Coordinated Time (UCT). For most purposes, UCT is the same as Greenwich Mean Time (GMT); while there are subtle technical differences, they don't affect tokens. In practice, however, the clocks built into the tokens inevitably drift away from UCT as they age. Although the design could account for this problem by increasing the window size, a larger window size also increases the threat of password interception and replay by an attacker.

Modern SecurID servers incorporate a strategy to adapt to clock drift in individual tokens. First, the server maintains clock drift information for each token and updates the information whenever the token successfully logs in. If the owner does not log in for a long time and the token drifts too far out of sync, the server falls back on a two-password resynchronization strategy similar to those used by counter-based tokens. If the first password falls outside the synchronization window, the SecurID server will reject it so that the owner has to provide a second password. If both passwords reflect the same clock drift factor, then authentication succeeds and the server updates the token's clock drift factor. *see Note 10.*

An interesting feature of clock-based tokens is that a token's owner could in theory use the same token to log on to several different servers in completely different sites. Each site would need the token's base secret, clock drift information, and the corresponding

user name. Then each site could authenticate the token's owner without requiring separate tokens or base secrets. However, such an arrangement makes it easier for an attacker to sniff the password sent to one server and replay it to connect to a different one.

The ActivCard token noted in the previous section combines both a counter and an internal clock. ActivCard provides a much broader range of timer increments than SecurID: a single tick may be as long as one hour. This could provide an unacceptably wide window within which an attacker might capture and replay a password, except that the ActivCard password also incorporates a counter. This yields different passwords within a single clock tick. Each ActivCard one-time password contains eight decimal digits. Six digits represent the actual password and the other two provide resynchronization information. As noted in the previous section, one of the digits contains the low-order digit of the counter. The server also expects to find the low-order digit of the timer as one of the digits. The server uses the digit from the timer to adjust for clock drift.

The ActivCard strategy uses a different resynchronization strategy than that used by SecurID, but it also yields longer passwords with no corresponding improvement in safety. In general, a longer password should reduce the risk of successful guessing or of other attacks. Instead, ActivCard uses the extra two digits to simplify resynchronization on the server side.

9.4 Attacks on One-Time Passwords

Although there are subtle differences between the various one-time password technologies, all face certain major attacks. All are at risk of attacks that divert an authenticated connection, like phone line redirection (Attack A-47 in Section 8.2) and IP address theft (Attack A-49 in Section 8.4). Below, we examine two additional attacks. The first is a man in the middle attack in which an attacker interferes with the victim's authentication process and then replays the one-time password. The second attack is an attack on IP connections that yields a similar result to an IP address theft. Although some observers have occasionally suggested other types of attacks, particularly against SecurID, the other attacks generally involve more sophistication, risk, and effort than the attacks described here. *see Note 11.*

MAN IN THE MIDDLE ATTACK

In this attack, the attacker eavesdrops on someone who is trying to log on to a server with a one-time password token. When the person sends the one-time password, the attacker intercepts the password, interrupts the communications path between the person and the server, and then uses the password himself to log on. The victim will most likely assume the problem was caused by network errors and is unlikely to suspect that an attack occurred. The person may try to log on again, and the server may accept this attempt if people are allowed to be logged on more than once at a given time. If the server rejects the attempt, the user might still not suspect foul play. Meanwhile, the attacker is successfully masquerading as the legitimate user.

A-56

This is a very sophisticated attack. The attacker must have control of the right portions of the network to be able to monitor the victim's traffic and interrupt the communications path to the server when necessary. The attack requires impeccable timing, particularly if the victim uses a clock-based one-time password token.

In this situation, an "asynchronous" challenge response mechanism (Chapter 10) has a security advantage over the "synchronous" tokens discussed in this chapter. A challenge is generally associated with a particular host or connection on the network, since it is supposed to represent the attempt to establish an authenticated session. If the same user attempts to complete the process of logging on from a different host, then the server is going to issue a new challenge. Thus, the attacker can't intercept a response and use it to log on from a different site or connection. Note that this does not resist a more active attack, like IP address theft.

D-56

IP HIJACKING

The first known case of the *IP hijacking* attack took place at the same time as the IP spoofing attacks described earlier (Attack A-53 in Section 8.6). The IP hijacking attack relied on special software that, once installed in a system, allowed an attacker to steal an established connection.

The original attack, which took place near Christmas 1994, began with an IP spoofing attack, which was used to transmit "r" com-

mands to the victim's computers. The spoofed commands gave the attacker enough privileges to install software into the victim's operating system—software to perform the IP hijacking attack. IP hijacking became the first widely available, practical example of a connection hijacking attack. *see Note 12.*

IP hijacking works by attacking a connection at one of its endpoints: when a victim tries to connect to another host, the attacker steals the connection at the victim's end. The hijacking software is a special software package that gets installed as part of the operating system, and runs with the processor operating in privileged mode. The hijacking software then lets the attacker take over any connection that is currently set up. Ideally for the attacker, the connection should be a Telnet connection to a command shell, or a similar connection set up by an "r" command.  *A-57*

It is hard to defend against IP hijacking. The attack relies on software that becomes part of the operating system, and this can make it difficult to block, until we detect the presence of the hijacking software. This represents our principal defense: to check the integrity of the host's operating system in order to detect the presence of suspicious software. A well-known package for tracking the integrity of critical computer files is *tripwire*, developed at Purdue.  *D-57* *see Note 13.*

As of this writing, IP hijacking software only exists for Unix. It isn't clear how many systems the existing attack scripts can be used against, or how easily the scripts might be adapted to other systems. So while the attack is available, and in some sense even common, it might not actually be particularly prevalent.

9.5 Incorporating a PIN

The problem with tokens in general is that they can be stolen and then used to perform a crime. Most people are very familiar with this in the context of auto theft prevention: "Lock your car and take the keys." An unattended token may represent too much of a temptation, even if outright theft is unlikely.  *A-58*

The general solution to this problem is the Personal Identification Number, or PIN. Most people are familiar with PINs because they are used with virtually all automated teller machines (ATMs). A PIN is like a reusable password, but it is almost always used as part of a two-factor authentication mechanism: the PIN is "something you  *D-58*

know" while a password token or ATM card serves as "something you have." Typically, any token that includes a key pad may be configured to require a PIN in order to generate a one-time password. *see Note 14.*

Although some systems allow longer PINs, many PINs are limited to four decimal digits. This means that determined attackers should be able to guess a PIN by systematically trying every possible value. Sometimes the attacker can apply cultural information to improve the chances of guessing the correct PIN, especially if users are allowed to choose their own PINs. Some systems are also vulnerable to more sophisticated guessing attacks if their PINs are based on special mathematical relationships to the authentication process.  *A-59* *see Note 15.*

Below, we examine three techniques for implementing PINs: PIN appended to an external password, PIN as an internal password, and PIN as part of the base secret. If our goal is to prevent attackers from guessing the PINs, then the second and third techniques are the most promising. The second technique uses the PIN as a password to grant access to the token. The third technique uses the PIN to generate the base secret and detect guessing at the server.

PIN APPENDED TO AN EXTERNAL PASSWORD

In this approach, we combine the PIN with other information, like the output of a one-time password token, and send them together as an external password to a remote system. This is also called a "soft PIN," since it works with tokens that don't have keypads. When faced with a password prompt and a password token, the user types the password displayed on the token, and then types in the token's PIN. This approach is used by SecurID and other token vendors to provide a PIN with a token that lacks a keypad.

This approach does keep attackers from using any token they might manage to steal, or from applying any PIN they happen to sniff from an authentication. However, there is a risk that an attacker will manage to sniff a particular PIN (or perhaps all PINs used at a site) and then steal the corresponding token. This is a sophisticated attack, since it involves coordinated physical and sniffing attacks. For some sites, this is an acceptable risk to take in exchange for lower-cost tokens that omit the keypad.

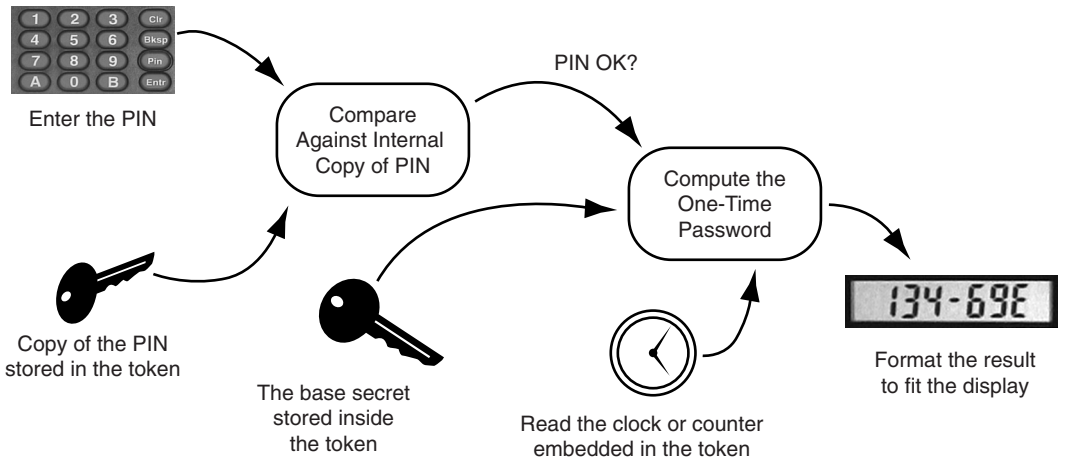

FIGURE 9.7: *Using a PIN to unlock a token.* Many tokens use the PIN to enable the computation of the one-time password. The token compares the PIN typed in against an internal copy. If repeated PIN entries are wrong, the token takes steps to resist a PIN guessing attack.

PIN AS AN INTERNAL PASSWORD

Figure 9.7 shows how to use the PIN as a password to grant access to the token. We call the PIN an "internal password" since it is keyed directly into the token and is only seen by the token. Bad PINs either disable the token or introduce delays. If the PIN's value is stored in the token, then the token can detect incorrect PINs and try to foil the guessing attack.

PINs implemented this way can also provide the basis for a duress signal, as described in Section 1.4. For example, many SafeWord tokens will recognize the entered PIN as a duress signal if the last digit entered is one less than in the user's correct PIN. The one-time password generated by the duress PIN then signals the server that the user is under duress. This implementation actually provides a lot of flexibility for implementing the duress signal: the signal could simply change the password generation slightly, or it could select an entirely different base secret to use, or it could even send a fixed response. *see Note 16.*

It is probably safer to handle the duress PIN inside the token than to send it combined with an external password. In the latter case, it's possible that the attackers already know the victim's legitimate PIN value; this would make it risky to use the duress PIN.

An obvious strategy for foiling a guessing attack is to simply disable the token after a fixed number of guesses. Once disabled, the owner needs to take the token to a system administrator for reprogramming. Until then, the token will not allow the owner to log on. SafeWord tokens, for example, may be configured to disable themselves after as few as three bad PINs or as many as fifteen are entered.

D-59

However, this strategy can seriously interfere with the system's usability. Keep in mind that one-time password mechanisms are more complex than traditional reusable password systems, and this increases the likelihood of usage errors. Legitimate users are more likely to encounter problems that are hard to diagnose because of the added complexity. This in turn increases the likelihood that someone will key in an incorrect PIN if he or she is having trouble logging on.

Another strategy to foil a guessing attack is to take longer and longer to report the incorrect PIN to the person each time he types in the wrong PIN. This is reasonably practical with hardware tokens, since the goal is to prevent the attacker from using the token until the legitimate owner can report it missing. The increasing delay reduces the risk of misuse without necessarily disabling the token. If the bad PINs were actually mistakes made by the token's legitimate owner, the increasing delay is a less painful security measure than the alternative of disabling the token. This is particularly true if the token provides authentication to users on travel: it would be extremely difficult to get a token reprogrammed while its owner is traveling.

D-60

These two strategies are well suited for implementation in special-purpose hardware tokens like those shown in Figure 9.1. However, the technique is not as safe if the one-time password token is implemented in software, as described in Section 10.3. If the token is able to verify whether the PIN is correct all by itself without contacting a separate server, then there is enough information embedded in the software token for an attacker to recover the PIN. Then the attacker can analyze the personal credentials embedded in the software token and determine the PIN's value. Security researchers have demonstrated this in practice.

A-60

see Note 17.

A capable software token designer will take steps to resist such an attack, but the steps cannot succeed for long. For example, the software token can store the PIN in a hashed form, and also use the valid PIN value to encode the base secret. But the attacker can quickly mount a systematic attack against the hashed PIN, since most PINs are only a few digits long. For the moment, nobody so far has reported the existence of PIN-cracking programs to perform such an attack. However, it remains a realistic if sophisticated risk against software tokens.

PIN AS PART OF THE BASE SECRET

The third approach to PIN handling is to incorporate the PIN into the base secret. This approach, shown in Figure 9.8, solves the problem faced by software tokens. The token does not store a copy of the PIN at all. In fact, the token does not even store a complete copy of the base secret. Instead, the actual base secret is constructed from a combination of the PIN and the token's partial base secret. If the attacker guesses the PIN incorrectly, he generates the wrong base secret value. The error doesn't appear until he actually tries to log on to the server, at which point the server detects and logs an incorrect one-time password. The attacker can't verify the PIN based on information within the software token, he must try to log on to the service, and that will produce a record of his attempt.

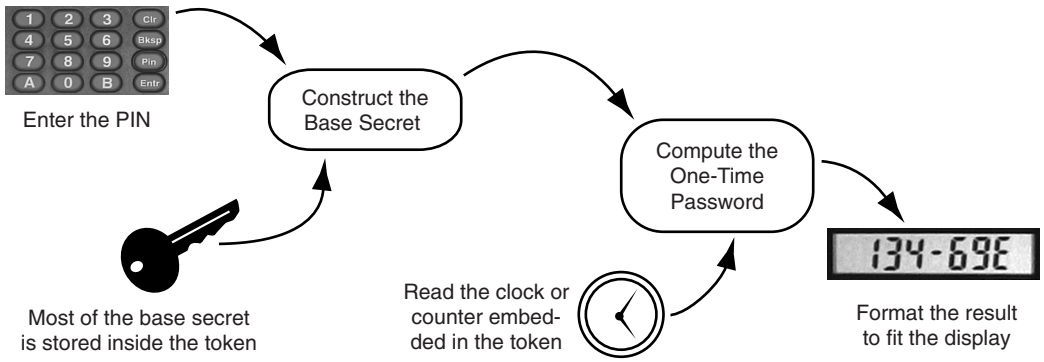

FIGURE 9.8: *Base secret partially derived from the PIN.* In this approach the token does not contain a copy of the PIN in any form and cannot detect incorrect PINs. The token can only generate the correct base secret if the owner provides the correct PIN. If the owner types the wrong PIN, the owner gets an invalid password and cannot log on. This helps the server detect PIN guessing attacks.

A major strength of this last method is that it forces the attacker to go to the server in order to verify his PIN guesses. This opens up his attack to detection, which gives the defenders a chance to identify the attack and resist it. Unfortunately, this is not the only way the attacker can verify a PIN guess. If the attacker can intercept several legitimate one-time passwords from the right software token, he can use those passwords to validate his PIN guesses. Fortunately, this is a relatively sophisticated attack: it requires the attacker to make a copy of the victim's soft token and intercept several uses of that token. Then the attacker needs a program to exhaustively try each possible PIN in conjunction with the software token's credentials to see if they would generate the intercepted passwords. While this is possible in theory, it requires a lot of work on the part of the attacker. Furthermore, each vendor uses a different strategy for generating passwords and for protecting data within their software tokens. A successful attack on one vendor's product would not apply to other vendors' products.

A-61

One can also implement a duress PIN as part of the base secret. In this case, however, the token device itself would not be aware of the fact that a duress PIN was used. The duress PIN would change the cryptographic results, and the server would itself have to perform the extra checking to see if an incorrect password is really a duress signal.

PINs are not passwords: they have never been intended to resist lengthy, systematic attacks. They are intended to be used with special hardware to reduce the risk of token theft. Hardware tokens, or the hardware elements of an ATM, protect the PINs against theft and against systematic attacks. PINs become vulnerable when used in less-protected environments.

9.6 ENROLLING USERS

Enrollment is the process of telling the authentication system about a person the system should be able to authenticate. When tokens are used for authentication, the system must be able to associate tokens with individual people, and associate base secrets with the tokens that carry them. Thus, enrollment involves two separate activities: token programming and user management. In fact, this distinction appears in many products. Most vendors provide a token

programming application that generates base secrets, programs them into tokens, and exports a table of new user records for importation into the server authentication software.

Token programming in general involves the following steps:

1. Prompt the operator for the token's serial number and the user name of the owner.
2. Generate a random number to use for the base secret.
3. If the token uses a PIN, generate a random number to use as the PIN.
4. Program the token with this information.
5. Create a database entry containing the user name, token serial number, and the base secret for on-line authentication. The PIN should not appear in this database.
6. If the token uses a PIN, print a separate "PIN report" for each token that identifies the user name, token serial number, and its initial PIN. In high-security applications, PIN reports should be printed on special forms that hide the PIN's value until the form is split and opened.

Once the operator has finished programming a group of tokens, the token programming software will export a database identifying all of the newly programmed tokens. The operator must now import that database into the server. Once the data is installed, the newly programmed tokens can be used.

The token programming software contains the necessary procedures to communicate with the token programming device (Figure 9.9). Each brand of token has its own electrical characteristics, and establishes its own expectations as to how it must be programmed. The token programming software formats the data as needed to program a token and transmits that data to the token programming device.

Although administrators may generally be trustworthy people, there are cases in which the proprietor must address the risk of some administrators' misusing the card programming system. A bank does not want its employees issuing themselves copies of customers' ATM cards, complete with PINs, but such things have indeed happened, as we saw in Section 9.1. A subverted administrator can

A-62

FIGURE 9.9: *Token programming device.* This programming device or "initializer" accepts an authentication token in the slot on top. Internally, the device contains probes that touch a set of contacts on the card. Each card must be programmed individually. The programming process loads a base secret into the token and configures it for any site-specific features, like preferred password formatting. This is an older programming device built by CryptoCard.

cause serious problems in a token-based authentication system. If the administrator has a number of blank tokens at his disposal, he can program them in different ways to circumvent server security. Obviously, he can create bogus new users and assign tokens to them. He might also be able to program blank tokens to mimic existing tokens belonging to authorized users. This could allow the administrator to masquerade as other users.

The solution is to establish two or more separate duties within the card programming activity: require the participation of at least two separate people whenever we add a token to the system. The following list gives examples of activities that the token programming process could keep separate. Sites can implement some of these activities by establishing and staffing the appropriate procedures and responsibilities. Some activities rely on technical features of

D-62

vendors' token programming software. In practice, different vendors' products provide different capabilities for separation of duty, and these capabilities can vary from one release to the next. Here are examples of things that could be separated:

- Assign one person to manage the inventory of unused tokens and another person to perform the programming. Require strict accounting for all tokens used. This makes it harder to create bogus cards without detection.
- Assign one person to program tokens and a separate person to import the list of new users into the site server. This prevents a single person from independently creating a new user.
- Use card programming software that keeps a strict log of every time it programs a token. This is most effective if the token programming software can extract the serial number from each token. The log should be included every time the program exports a list of newly programmed tokens.
- The card programming software should not disclose the base secrets it generates. If the software generates an export file that must move to a different computer, then the base secrets should be encrypted or otherwise hidden. For example, the S/Key password system in Section 10.1 uses one-way hashes.
- The card programming software should protect the programming log from modification. This may involve cryptographic hashes if the log is transferred between computers.

Separation of duty is not the perfect solution. It places a significant burden on the technical features of the authentication system and requires clear understanding by the site's security administrators. Not every site can afford the time, effort, and inconvenience it takes. Sites rarely implement such measures unless they suffer a security incident that is costly enough to justify them. Financial institutions implement very sophisticated procedures to ensure separation of duty when generating PINs, but this happened after many years of costly experience with PIN fraud.

9.7 SUMMARY TABLES

TABLE 9.1: *Attack Summary*

Attack 	Security Problem	Prevalence	Attack Description
A-54. Network password sniffing	Masquerade as someone else	Common	Monitor traffic on a network link, intercept any plaintext passwords seen, and exploit them
A-55. Exploit password equivalent	Masquerade as someone else	Common	Intercept a hashed or otherwise encoded password and use in forged network messages where hashed, not typed, passwords are expected
A-56. Interception and replay	Masquerade as someone else	Sophisticated	Intercept a one-time password and replay it while blocking the legitimate user from successfully logging on
A-57. IP hijacking	Masquerade as someone else	Common or Sophisticated	Intercept an established connection and reattach it to a program controlled by the attacker
A-58. Token theft	Masquerade as someone else	Physical	Steal the token belonging to an authorized user
A-59. PIN guessing	Masquerade as someone else	Trivial	Steal a token and manually try every possible value for the token's PIN
A-60. Extract PIN from software token	Masquerade as someone else	Sophisticated	Copy the victim's software token, analyze its contents to identify the encoded PIN, do brute force attack to determine the PIN value, then use it with the software token

TABLE 9.1: *Attack Summary (Continued)*

Attack 	Security Problem	Prevalence	Attack Description
A-61. Test PIN against intercepted passwords	Masquerade as someone else	Sophisticated	Intercept several of the victim's one-time passwords, copy the victim's software token, extract the partial base secret encoded with the PIN, do brute force analysis to find a PIN that matches the generated passwords, then use the PIN
A-62. Subverted token administrator	Masquerade as someone else	Trivial	Trusted person who programs tokens also programs extra tokens used for penetrating legitimate accounts

TABLE 9.2: *Defense Summary*

 Defense	Foils Attacks	Description
D-54. Encoded password	A-54. Network password sniffing	Encrypt or hash a password when it must traverse a public or other untrustworthy network
D-55. One-time password token	A-54. Network password sniffing A-55. Exploit password equivalent A-61. Test PIN against intercepted passwords	Generates a new password for each attempt to log on. An attacker cannot log on by trying to intercept and reuse a password, since passwords only work once
D-56. Challenge-response one-time passwords	A-56. Interception and replay	Use challenge-response one-time passwords instead of synchronous one-time passwords
D-57. Check the host OS integrity	A-57. IP hijacking	Check the software components of the host OS to see if they have been modified to insert subverted software

TABLE 9.2: *Defense Summary (Continued)*

Defense	Foils Attacks	Description
D-58. PINs on tokens	A-58. Token theft	Require the owner to enter a PIN before the token will generate a valid one-time password
D-59. Lock-up after incorrect PINs entered	A-59. PIN guessing	Disable the token after the user enters too many incorrect PINs, so that attackers can't find the PIN through exhaustive trial and error
D-60. Increasing delay for incorrect PINs	A-59. PIN guessing	If the wrong PIN is entered, delay before accepting another attempt. Increase the length of delay with each incorrect PIN entered.
D-61. PIN forms part of the token's base secret	A-59. PIN guessing A-60. Extract PIN from software token	Incorporate the PIN into the base secret so that the token will not contain the correct base secret unless the correct PIN is entered. Do not detect the wrong PIN at the token.
D-62. Separation of duty in token programming procedure	A-62. Subverted token administrator	Require the participation of two or more people in the process of programming and enabling tokens for authentication