7

Authentication

> "Who are you, Master?" he asked.
>
> "Eh, what?" said Tom sitting up, and his eyes glinting in the gloom. "Don't you know my name yet? That's the only answer. Tell me, who are you, alone, yourself and nameless."
>
> *Lord of the Rings*
> —J.R.R. TOLKIEN

Authentication is the process of *proving* one's identity. This is distinct from the assertion of identity (known, reasonably enough, as *identification*) and from deciding what privileges accrue to that identity (*authorization*). While all three are important, authentication is the trickiest from the perspective of network security.

Authentication is based on one, two, or three *factors*:

- Something you know
- Something you have
- Something you are

The first factor includes passwords, PINs, and the like. The second includes bank cards and authentication devices. The third refers to your biological attributes. Authentication solutions can involve one-, two-, or three-factor authentication. Most simple applications use single-factor authentication. More important ones require at least two. We recommend two-factor authentication using the first two when authenticating to a host from an untrusted environment like the Internet.

Machine-to-machine authentication is generally divided into two types: *cryptographic* and *other*. (Some would say "cryptographic" and "weak.")

The level of authentication depends on the importance of the asset and the cost of the method. It also includes convenience and perceived convenience to the user. Though hardware tokens can

Levels of Authentication—User-Chosen Passwords

User-chosen passwords are easily remembered, but contain surprisingly little entropy: people rarely pick good keys, and experience has shown that user education won't change this. Passwords can be classified as follows:

- **Cleartext:** Easily sniffed on unencrypted links. Used by *telnet*, *ftp*, and *rlogin*.
- **Hashed:** Subject to dictionary attacks. The dictionary may be pre-computed and read off a disk, speeding up the attack. LanManager passwords, used in Windows and Windows NT, fall into this category.
- **Hashed with salt:** Salting, or encrypting with a variable nonce key, frustrates pre-computed searches. UNIX password files have 4096 salting values. Dictionary attacks are slower than without salt, but still yield rich results.

be quite easy to use, we often hear that upper management will not tolerate anything more complex than a password. (We think this sells management short.) Imagine protecting a company's most valuable secrets with an often poorly chosen password!

What is an appropriate level of authentication? Should you use hand held authenticators for logins from the Internet? What about from the inside? What sort of authentication do you want for outgoing calls, or privileged (*root*) access to machines? For that matter, who will maintain the authentication databases?

7.1 Remembering Passwords

> Duh, uh, a, open, uh, sarsaparilla. Uh, open Saskatchewan. Uh, open septuagenarian. Uh, open, uh, saddle soap. Euh, open sesame.
>
> *Ali Baba Bunny*
> —EDWARD SELZER

We already discussed password attacks and defenses in Section 5.1. That section is concerned with choosing good passwords and protecting them from discovery or theft. As a means of personal authentication, passwords are categorized as "something you know." This is an advantage because no special equipment is required to use them, and also a disadvantage because what you know can be told to someone else, or captured, or guessed.

Levels of Authentication—Machine-Chosen Passwords

A computer is much better than people at choosing random keys (though there have been famous bugs here!) They can generate high entropy, but this can be hard to remember. The machine-chosen password can be

- **translated to a pronounceable string and memorized** It's hard to remember, for example, 56 random bits, but they can be changed into a string of pronounceable syllables or words. Can you remember a password like "immortelle monostely Alyce ecchymosis"? These four words, chosen at random from a 72,000 list of English words, encode roughly 64 bits of key, and would be very hard to discover in a dictionary attack. We are not sure we could spell ecchymosis twice the same way, and this password would take a while to type in. This approach would allow for some spelling correction, as we have a fixed list of possible words. Most approaches stick to syllables. Several password generators use this method. See Section 7.1.1
- **printed out** A list of one-time passwords could be printed out. If the paper is lost or observed, the keys can leak. OTP-based approaches use this.
- **stored in a portable computer** This is popular, and not a bad way to go if the computer is never stolen or hacked. Bear in mind that laptops are at high risk of being stolen, and that most computers do seem to be vulnerable to being hacked. Some programs, like PGP, encrypt the key with a second password, which takes us back to square one, dictionary attacks. But the attacker would need access to the computer first.
- **stored in a removable media** Keys and passwords can be stored in a USB "disk," a small, removable gadget that is available with many megabytes of flash memory. These are relatively inexpensive and can be expected to drop in price and jump in capacity over time. A single-signon solution that uses this approach would be wonderful. This solution is not as secure as others; users must physically protect their USB device carefully.
- **stored in security tokens** This is the most secure approach. The token has to be stolen and used. Because they hide the key from the user, it may cost a lot of money to extract that actual key from the device, which typically has strong, complicated hardware measures designed to frustrate this attack, and "zeroize" the key. But cost, inconvenience, and (in some cases) the need for special token readers are problems.

No security expert we know of regards passwords as a strong authentication mechanism. Nevertheless, they are unlikely to disappear any time soon, because they are simple, cheap, and convenient.

In fact, the number of passwords that the average person must remember is staggering (see Sidebar on page 141). The proliferation of password-protected Web sites, along with the adoption of passwords and PINs (*i.e.,* very short passwords with no letters) by just about every institution has created a state in which no user can behave in the "appropriate" way. Translation: There is no way to remember all of the passwords that one needs in order to function in the world today. As a result, people write them down, use the same password for multiple purposes, or generate passwords that are easily derivable from one another. It is also likely that when prompted for a password, people may inadvertently provide the password for a different service. It is worth noting that some passwords, such as your login password and the one to your online banking, exist to protect your stuff. Other passwords, such as that to a subscription Web site, exist to benefit others. There is little incentive for users to safeguard or care about passwords in the latter category.

Writing them down or storing them in a file risks exposure; forgetting them often leads to ridiculous resetting policies ("Sure, you forgot your password, no problem, I'll change it to your last name. Thank you for calling."); and giving the wrong password to the wrong server is clearly undesirable.

If the number of passwords that people are required to have is a problem, it is compounded by the inexplicable policy found in many IT policy manuals stating that users must change their password every n months. There is no better way to ensure that users pick easily memorizable (*i.e.,* guessable) passwords and write them down. We're not sure what the origin of this popular policy is, but studies have shown that requiring users to change their password on a regular basis leads to *less* security, not more [Adams and Sasse, 1999; Bunnell *et al.*, 1997]. Quoting from the CACM paper by Adams and Sasse:

> Although change regimes are employed to reduce the impact of an undetected security breach, our findings suggest they reduce the overall password security in an organization. Users required to change their password frequently produce less secure password content (because they have to be more memorable) and disclose their passwords more frequently. Many users felt forced into these circumventing procedures, which subsequently decreased their own security motivation.

So what is a person to do? There is no perfect solution to the multiple password dilemma. One piece of advice is to group all of the passwords by level of importance. Then, take all of the non-important passwords, such as those required for free subscription services on the Web, and use the same easy-to-remember, easy-to-guess, totally-useless-but-I-had-to-pick-something password. Then, pick the highest security, the most important group, and find a way to pick unique and strong passwords that you can remember for those (good luck). One of the approaches that we have is to keep a highly protected file with all of the passwords. The file is encrypted and never decrypted on a networked computer. Backup copies of the encrypted file can be kept all over the place. The file of passwords is encrypted using a very strong and long passphrase. That said, this is not an ideal solution, but we do not live in an ideal world.

Passwords Found in One's Head

Here are some of the passwords that one of the authors currently holds:

Worthless: internal recruiting Web pages, *New York Times* online, private Web area, yahoo.com, realtor.com

Slightly important: acm.org, usenix.org, buy.com, quicken.com, inciid.org, Ibaby.com, amazon.com, barnseandnoble.com, Marriott rewards, continental.com frequent flier account, EZPass PIN, e-toys, ticketmaster, Web interface to voice mail, combination lock on backyard fence, publisher royalties online access, hushmail.com e-mail account

Quite secure: employee services Web site, child care reimbursement program, Unix account login, former university account login, NT domain account login, online phone bill, home voice mail access code, work voice mail access code, cell phone voice mail access code, quicken password for each *linked site*, domain name registration account, drivers license online registration, dial-in password, OTP-based password, keyless access code for car

Top security: garage (2 doors + temporary nanny code), burglar alarm (regular code, master code, nanny's code, and a distress code), bank Web login, online broker, PCAnywhere password for remote control and file transfer, quicken PIN vault, 401k account online access and phone access, stock options account, dial-in password, online access to IRA from previous job, paypal account

A total of 53 passwords.

There are some alternatives to written passwords. None of them have really caught on in Web applications, but perhaps some applications could benefit from them. There has been a study of using images for authentication [Dhamija and Perrig, 2000], and a commercial product called Passface that relies on the recognition of faces for authentication. Authentication based on knowledge of a secret algorithm was proposed as far back as 1984 [Haskett, 1984]. There is also a paper on authenticating users based on word association [Smith, 1987], and more recent work has centered on graphical passwords whereby users remember pictures instead of strings [Jermyn *et al.*, 1999].

Several tools can be used to protect passwords by putting them all into a file or a database, and then encrypting the collection of passwords with a single passphrase. Examples of this are *Quicken*'s PIN vault, Counterpane's *password safe*,[1] and the *gnu keyring* for PalmOS,[2] which protects keys and passwords on PDAs. Use of these password-protecting mechanisms requires that the encrypted database is available when needed; that the user remember the master password; and that the master password is not susceptible to dictionary attack. It is reminiscent of the quote by the wise man at the beginning of Chapter 15 of [Cheswick *et al.*, 2003].

There's also a more subtle risk of such products: Who has access to the encrypted file? You may think that it's on your Palm Pilot, but you probably synchronize your Palm Pilot to your desktop machine; in a corporate environment, that desktop's disks may be backed up to a file server. Indeed, the synchronization file may live on a networked disk drive. Could a Bad Guy launch a dictionary attack on one of the *copies* of the file?

7.1.1 Rolling the Dice

It is well known that when it comes to picking textual passwords, regardless of the possible password space, humans tend to operate in a vary narrow range. This range is usually quickly tested by machine. The *diceware* project is designed to help humans utilize the entire password space. It is most useful for systems on which the passphrase is not likely to change, such as the passphrase that locks PGP's keys. As usual, there is a compromise between usability and security. *Diceware* produces very good passphrases, but users are forced to memorize a collection of strings. This, of course, results in written copies of the passphrases. Written passphrases are not necessarily the end of the world, but physically protecting the paper scraps is paramount.

The main difference between a passphrase in a system like PGP and the password you use to login into an account is that passphrases are used as keys that directly encrypt information. In the case of PGP, the user's passphrase represents a key that encrypts a user's private RSA key. Therefore, the entropy required for the passphrase needs to be high enough for the requirements of the symmetric cipher used in the encryption. In today's systems, this is about 90 bits [Blaze *et al.*, 1996].

Here's how *diceware* works: The program contains a list of $6^5 = 7776$ short words and simple abbreviations, with an average length of 4.2 letters. A list can be found at `http://world.std.com/~reinhold/diceware.wordlist.asc`. Alternative lists exist as well. In the word list, each word is associated with a five-digit number, where each digit is between 1 and 6 inclusive.

1. `http://www.counterpane.com/passsafe.html`.
2. `http://gnukeyring.sourceforge.net/`.

To generate a passphrase, obtain some real-world, physical playing dice, and decide how many words you would like to include. Obviously, picking more words provides higher assurance, at the expense of having to memorize a longer passphrase. Generating approximately 90 bits of entropy requires seven words in the passphrase. Using an online dice generator or a computer program that simulates randomness is not a good idea because deterministic processes cannot simulate randomness as well as real dice can. Next, roll the dice and write down the numbers in groups of 5. Then, use the five-digit numbers to look up the words in the list. Every group of 5 numbers has a corresponding short word, under six characters, in the list. For example, if you roll 3, 1, 3, 6, and 2, the five-digit number is 31362, and this corresponds to the word "go" in the word list.

To make passphrase selection even more secure, you can mix in special characters, such as punctuation marks. The right way to do that is to produce a dictionary matching numbers to characters and then roll the dice again. You could also devise a way to mix the case of the letters, but this will be at the cost of memorability. It is important to use dice to pick the characters because the randomness of the dice roll eliminates any bias you might have as a human. This is the main philosophy behind *diceware*. Any decision that affects the choice of passphrase should be determined randomly, because people have biases, which when understood can be programmed into a cracking tool.

7.1.2 The Real Cost of Passwords

Earlier, we said that passwords are a cheap solution. In fact, they're not nearly as cheap as you might think. There's a major hidden expense: dealing with users who have forgotten their passwords. In other words, what do you do when Pat calls up and says, "I can't log in"?

If all of your users are in the same small building as your system administrator, it's probably not a real problem for you. Pat can wander down to the systems cave (by tradition, systems administrators are not allowed to see daylight), and the administrator will recognize Pat and solve the problem immediately. Besides, it won't happen all that often; Pat probably uses that password every day to log in.

The situation is very different for ISPs. How do you authenticate the request? How do you know it's really Pat?

This isn't a trivial question; many hacks have been perpetrated by inadequate verification. A few years ago, the ACLU site on AOL was penetrated in exactly this fashion.[3] But setting up a proper help desk is expensive, especially when you consider the cost of training—repeated training, because turnover is high; and ongoing training, because new scams are invented constantly.

This is another instance of social engineering (see Section 5.2). But preventing it adds a lot of cost to "cheap" passwords. Note, too, that hybrid schemes, such as a token plus a PIN, can incur the same cost. A token or biometric scheme may be cheaper, if you factor in the true cost of the lost password help desk, but what is the cost of a lost PIN help desk? For that matter, what is the cost of the lost or broken token help desk? Furthermore, your biggest problem is telecommuters, because you have to mail them new tokens. Are their physical mailboxes secure?

3. See `http://news.com.com/2100-1023-211606.html?legacy=cnet` for details.

7.2 Time-Based One-Time Passwords

One can achieve a significant increase in security by using *one-time passwords*. A one-time password behaves exactly as its name indicates: It is used exactly once, after which it is no longer valid. This provides a very strong defense against eavesdroppers, compromised *telnet* commands, and even publication of login sessions.

There are a number of possible ways to implement one-time password schemes. The best-known involve the use of some sort of *handheld authenticator,* also known as a *dongle* or a *token*.

SecurID makes one common form of authenticator that uses an internal clock, a secret key, and a display. The display shows some function of the current time and the secret key. This output value, usually combined with a PIN, is used as the authentication message. The value changes about once per minute, and generally only one login per minute is allowed. (The use of cryptography to implement such functions is described in Chapter 18.) These "passwords" are never repeated.

The client takes the response from the SecurID token and sends it to the server, which consults an authentication server, identifying the user and the entered response. The authentication server uses its copy of the secret key and clock to calculate the expected output value. If they match, the authentication server confirms the identification to the server.

In practice, clock skew between the device and the host can be a problem. To guard against this, several candidate passwords are computed, and the user's value is matched against the entire set. A database accessible to the host keeps track of the device's average clock rate and skew to help minimize the time window. But this introduces another problem: A password could be replayed during the clock skew interval. A proper implementation should cache all received passwords during their valid lifetime; attempted reuses should be rejected and logged. This scheme may also be subject to a race attack (see Section 5.4.1) on the last digit of the password.

It is important to secure the link between the server and the authentication server, either with a private link or by using cryptographic authentication. The serving host has to know that it is talking to the real authentication server, and not an imposter. It is often less important that the communication be private, as the one-time password may have passed in the clear between the client and the server in the first place. Of course, it is never a good idea to leak information needlessly.

The database on the authentication server presents a few special problems. It is vital that the authentication server be available: It can hold the keys for many important services, sometimes for an entire company. This means that it is prudent to have several servers available for reliability, though usually not for capacity: An authentication transaction should not take very much time.

But replicated databases offer a sea of potential troubles. They usually must be kept synchronized, or old versions may offer access that has been revoked. Machines that are down when the database changes must be refreshed before they come back online. Updates must be propagated rapidly and safely: Imagine offering a false update to an authentication server. Furthermore, does your replication mechanism handle the cache of recently used passwords? Can an attacker who has sniffed a password on the way to one server launch a denial-of-service attack on the server, to force a replayed authentication to go to the backup?

When the situation allows, it may be safer to run a single, very reliable server than to try to get distributed databases working correctly and safely. We are not saying that replicated databases shouldn't be used, just that they be designed and used very carefully.

7.3 Challenge/Response One-Time Passwords

A different one-time password system uses a nonrepeating challenge from the server. The response is a function of the challenge and a secret known to the client. Challenge/response can be implemented in client software or in a hardware token, or even computed by the user:

```
challenge:  00193 Wed Sep 11 11:22:09 2002
response:   ab0dh1kd0jkfj1kye./
```

This response was quickly computed by a user, based on challenge text. In this case, the algorithm is secret, and there is no key. The algorithm must be easily learned and remembered, and then obscured. Most of the response here is meaningless chaff. It would take a number of samples for an eavesdropper to figure out the important features of the response and deduce the algorithm used. This approach weakens quickly as more samples are transmitted. (This example is from an experimental emergency password system developed by one of the authors.)

Challenge/response identification is derived from the *Identification Friend or Foe (IFF)* devices used by military aircraft [Diffie, 1988]. It, in turn, is derived from the traditional way a military sentry challenges a possible intruder.

In networking, challenge/response is used to avoid transmitting a known secret. An eavesdropper's job is more difficult. One can't simply read the password as it flies by; but a dictionary attack must be mounted to guess the secret. We can even make the dictionary attack less certain by returning only part of the computed challenge.

A number of Internet protocols can use challenge/response: *ppp* has CHAP, and *pop3* has APOP, for example. But the strongest user authentication we know of uses a hardware *token* to compute the response. We've been told that spy agencies sometimes use these.

Again, the user has a device that is programmed with a secret key. The user enters a PIN into the device (five consecutive failures clear the key) and then keys in the challenge. The token computes some function of the challenge and the key, and displays the result, which serves as the password.

This model offers several modest security advantages over the time-based password scheme. Because no clock is involved, there is no clock skew, and hence no need for a cache. The PIN is known only to the user and the token. It is not stored in a central database somewhere.

If the same user is trying to authenticate from several sessions simultaneously, each session will use a different challenge and response. This situation probably doesn't arise often, perhaps only when an account is shared, which is a bad idea anyway. But it totally and easily frustrates the race attacks described in Section 5.4.1.

Conversely, the device must have a keypad, and the user must transcribe the challenge manually. Some have complained about this extra step, or suggested that upper management would

never put up with it. We point out that this authentication is very strong (spies use it), and that not all managers have pointy hair.

Both of these schemes involve "something you have," a device that is subject to loss or theft. The usual defense is to add "something you know" in the form of some sort of *personal identification number (PIN)*. An attacker would need possession of both the PIN and the device to impersonate the user. (Note that the PIN is really a password used to log in to the handheld authenticator. Although PINs can be very weak, as anyone in the automatic teller machine card business can testify [Anderson, 1993, 2002], the combination of the two factors is quite strong.) The device usually shuts down permanently after a few invalid PINs are received, limiting the value of PIN-guessing attacks. In addition, either approach must have the key accessible to the host, unless an authentication server is used. The key database can be a weakness and must be protected.

Finally, note that these authentication tokens can be compromised if the attacker has access to the device. Expensive equipment can read data out of computer chips. How much money is your attacker willing to spend to subvert your system?

Many people carry a computer around these days. These algorithms, and especially the following, are easily implemented in a portable machine, such as a cell phone.

7.4 Lamport's One-Time Password Algorithm

Lamport proposed a one-time password scheme [Lamport, 1981] that can be implemented without special hardware. Assume there is some function F that is reasonably easy to compute in the forward direction but effectively impossible to invert. (The cryptographic hash functions described in Section A.7 are good candidates.) Further assume that the user has some secret—perhaps a password—x. To enable the user to log in some number of times, the host calculates $F(x)$ that number of times. Thus, to allow 1,000 logins before a password change, the host would calculate $F^{1000}(x)$, and store only that value.

The first time the user logs in, he or she would supply $F^{999}(x)$. The system would validate that by calculating

$$F(F^{999}(x)) = F^{1000}(x).$$

If the login is correct, the supplied password—$F^{999}(x)$—becomes the new stored value. This is used to validate $F^{998}(x)$, the next password to be supplied by the user.

The user's calculation of $F^n(x)$ can be done by a handheld authenticator, a trusted workstation, or a portable computer. Telcordia's implementation of this scheme [Haller, 1994], known as *S/Key*, goes a step further. While logged on to a secure machine, the user can run a program that calculates the next several login sequences, and encodes these as a series of short words. A printed copy of this list can be used while traveling. The user must take care to cross off each password as it is used. To be sure, this list is vulnerable to theft, and there is no provision for a PIN. S/Key can also run on a PC. (Similar things can be done with implementations of the IETF version, known as *One-Time Password (OTP)* [Haller and Metz, 1996].)

Because there is no challenge, Lamport's algorithm may be subject to a race attack (see Section 5.4.1).

7.5 Smart Cards

A smart card is a portable device that has a CPU, some input/output ports, and a few thousand bytes of nonvolatile memory that is accessible only through the card's CPU. If the reader is properly integrated with the user's login terminal or workstation, the smart card can perform any of the validation techniques just described, but without their weaknesses. Smart cards are "something you have," though they are often augmented by "something you know," a PIN.

Some smart cards have handheld portable readers. Some readers are now available in the PC card format.

Consider the challenge/response scheme. As normally implemented, the host would need to possess a copy of the user's secret key. This is a danger: The key database is extremely sensitive, and should not be stored on ordinary computers. One could avoid that danger by using public-key cryptographic techniques (see Section A.4), but there's a problem: The output from all known public key algorithms is far too long to be typed conveniently, or even to be displayed on a small screen. However, not only can a smart card do the calculations, it can also transmit them directly to the host via its I/O ports. For that matter, it could read the challenge that way, too, and simply require a PIN to enable access to its memory.

It is often assumed that smart cards are tamper-proof. That is, even if an enemy were to get hold of one, he or she could not extract the secret key. But the cards are rarely, if ever, that strong. Apart from destructive reverse-engineering—and that's easier than you think—there are a variety of nondestructive techniques. Some subject cards to abnormal voltages or radiation; others monitor power consumption or the precise time to do public key calculations.

7.6 Biometrics

Another method of authenticating attempts to measure something intrinsic to the user. This could be something like a fingerprint, a voice print, the shape of a hand, an image of the face, the way a person types, a pattern on the retina or iris, a DNA sequence, or a signature. Special hardware is usually required (though video cameras are now more common on PCs), which limits the applicability of biometric techniques to comparatively few environments. The attraction is that a biometric identifier can be neither given away nor stolen.

In practice, there are some limitations to biometrics. Conventional security wisdom says that authentication data should be changed periodically. While this advice may seem to contradict Section 7.1, there's a big difference between *forcing* someone to change their password and *permitting* them to. Changing your authenticator is difficult to do when it is a fingerprint.

Not all biometric mechanisms are user-friendly; some methods have encountered user resistance. Davies and Price [1989] cite a lip-print reader as one example. Moreover, by their very nature, biometrics do not provide exact answers. No two signatures are absolutely identical, even

from the same individual, and discounting effects such as exhaustion, mood, or illness. Some tolerance must be built into the matching algorithm. Would you let someone log in if you were 93% sure of the caller's identity?

Some systems use smart cards to store the biometric data about each user. This avoids the need for host databases, instead relying on the security of the card to prevent tampering. It is also possible to incorporate a random challenge from the host in the protocol between the smart card and the user, thus avoiding replay attacks.

Currently, we are unaware of any routine use of biometric data on the Internet. But as microphone-equipped machines become more common, usage may start to spread. Research in this area is under way; there is a scheme for generating cryptographic keys from voice [Monrose *et al.*, 2001]. One problem with such schemes is that you may be able to spoof someone after they leave a voice message on your machine. Perhaps in a future world, people will have to constantly disguise their voice unless they are logging into their machine.

The real problem with Internet biometrics is that the remote machine is not reading a fingerprint, it's reading a string of bits. Those bits are purportedly from a biometric sensor, but there's no way to be sure.

Attempts to find dynamic biometrics that are useful in a security context have failed. Research into keystroke dynamics—that is, the *way* people type—has shown that it is difficult to use this as an authentication metric [Monrose and Rubin, 2000].

Another problem with biometrics is that they do not change and are left all over the place. Every time you pick up a glass to drink, open a door, or read a book, you are leaving copies of your fingerprint around. Every time you speak, your voice can be recorded, and every time you see the eye doctor, he or she can measure your retina. There have been published reports of fake fingerprints created out of gelatin, and of face recognition software being fooled by life-size photographs.

7.7 RADIUS

Remote Authentication Dial In User Service (RADIUS) [Rigney *et al.*, 1997] is a protocol used between a network access point and a back-end authentication and authorization database. RADIUS is frequently used by ISPs for communication between modem-attached *Network Access Servers (NASs)* and a central authorization server. The centralized database lists all authorized users, as well as what restrictions to place on each account. There is no need for each NAS to have its own copy. Corporations with their own modem pools use RADIUS to query the corporate personnel database.

The RADIUS traffic between the querier and the server is cryptographically protected, but not very well. The protocol has also suffered from implementation errors affecting security (see CERT Advisory CA-2002-06). RADIUS has had many official and private extensions to it over the years. The architecture is not clean, and RADIUS is being replaced by a newer system called *Diameter.*

7.8 SASL: An Authentication Framework

Simple Authentication and Security Layer (SASL) [Myers, 1997; Newman, 1998, 1997] is an authentication framework that has been incorporated into several widely used protocols, including *imap*, *pop3*, *telnet*, and *ldap*. The intent of SASL is to create a standardized mechanism for supporting many different authentication mechanisms. SASL also provides the option to negotiate a security layer for further communications.

SASL by itself does not necessarily provide sufficient security. The security of SASL depends on the mechanisms that are chosen; perhaps using SASL over an SSL connection to authenticate users is a reasonable thing to do, but pretty much any authentication mechanism works in that scenario. Conversely, [Myers, 1997] suggests using MD4 [Rivest, 1992a], even though that hash function is believed to be weak. Furthermore, using SASL for authentication alone leaves the connection vulnerable to hijacking. If you are integrating SASL into a key exchange protocol, the extra overhead is probably not needed, as the key exchange protocol probably authenticates the user already.

The advantage of SASL is that it provides a standardized framework for an application that wishes to support multiple authentication techniques.

7.9 Host-to-Host Authentication

7.9.1 Network-Based Authentication

For better or worse, the dominant form of host-to-host authentication on the Internet today relies on the network. That is, the network itself conveys not only the remote user's identity, but is also presumed to be sufficiently accurate that one can use it as an authenticated identity. As we have seen, this is dangerous. Network authentication itself comes in two flavors: address-based and name-based. For the former, the source's numeric IP address is accepted. Attacks on this form consist of sending something from a fraudulent address. The accuracy of the authentication thus relies on the difficulty of detecting such impersonations—and detecting them can be very hard.

Name-based authentication is weaker still. It requires that not only the address be correct, but also the name associated with that address. This opens a separate avenue of attack for the intruder: corrupting whatever mechanism is used to map IP addresses to host names. The attacks on the DNS (see Section 2.2.2) attempt to exploit this path.

7.9.2 Cryptographic Techniques

Cryptographic techniques provide a much stronger basis for authentication. While the techniques vary widely (see Chapter 18 for some examples), they all rely on the possession of some "secret" or cryptographic key. Possession of this secret is equivalent to proof that you are the party known to hold it. The handheld authenticators discussed earlier are a good example.

If you share a given key with exactly one other party, and receive a message that was encrypted with that key, you *know* who must have sent it. No one else could have generated it. (To be sure, an enemy can record an old message and retransmit it later. This is known as a *replay attack*.)

You usually do not share a key with every other party with whom you wish to speak. The common solution to this is a *Key Distribution Center (KDC)* [Needham and Schroeder, 1978, 1987; Denning and Sacco, 1981]. Each party shares a key—and hence some trust—with the KDC. The center acts as an intermediary when setting up calls. While the details vary, the party initiating the call will contact the KDC and send an authenticated message that names the other party to the call. The KDC can then prepare a message for the other party, and authenticate it with the key the two of them share. At no time does the caller ever learn the recipient's secret key. Kerberos (see Section 18.1) is a well-known implementation of a KDC.

While cryptographic authentication has many advantages, a number of problems have blocked its widespread use. The two most critical ones are the need for a secure KDC, and the difficulty of keeping a host's key secret. For the former, one must use a dedicated machine, in a physically secure facility, or use a key exchange protocol based on public key cryptography. Anyone who compromises the KDC can impersonate any of its clients. Similarly, anyone who learns a host's key can impersonate that host and, in general, any of the users on it. This is a serious problem, as computers are not very good at keeping long-term secrets. The best solution is specialized cryptographic hardware—keep the key on a smart card, perhaps—but even that is not a guaranteed solution, because someone who has penetrated the machine can tell the cryptographic hardware what to do.

7.10 PKI

> "When I use a word," Humpty Dumpty said, in a rather scornful tone, "it means just what I choose it to mean, neither more nor less."
>
> *Through the Looking Glass*
> —LEWIS CARROLL

Public Key Infrastructure (PKI) is one of the most misunderstood concepts in security. There was a time when PKI was believed to be the magical pixie dust that would make any system secure. Different people mean different things when they use the term PKI. In general, PKI refers to an environment where principles (people, computers, network entities) possess public and private keys, and there is some mechanism whereby the public keys are known to others in a trustworthy fashion. Typically, the proof of one's public key is achieved via a *certificate*. In its broadest sense, a certificate is a signed statement from a trusted entity stating something about a public key or a principle.

It is important to distinguish between *identity* certificates and *authorization* certificates. Identity certificates, the ones you are more likely to come across, are certificates in which a trusted party binds an identity to a public key. Authorization certificates represent a credential that can be used by a principle to achieve some access, or to perform some function, based on their possession of a private key.

Identity certificates are arranged in a hierarchy, whereby a trusted party, usually called a *Certificate Authority (CA)* issues certificates to entities below it, and receives its own certificates from

trusted parties above it. The path ultimately leads to a root node, which is the reason why global PKI of identity certificates is a pipe dream—the most oversold and least realistic concept in security. Whom do you trust to be the root of trust in the world?

However, pki (lowercase PKI) that applies to a subset of the world *is* a realistic concept. Organizations such as companies, the military, and even universities tend to be hierarchical. The concept of public key infrastructure maps itself nicely to such organizations, and thus the technology is quite useful.