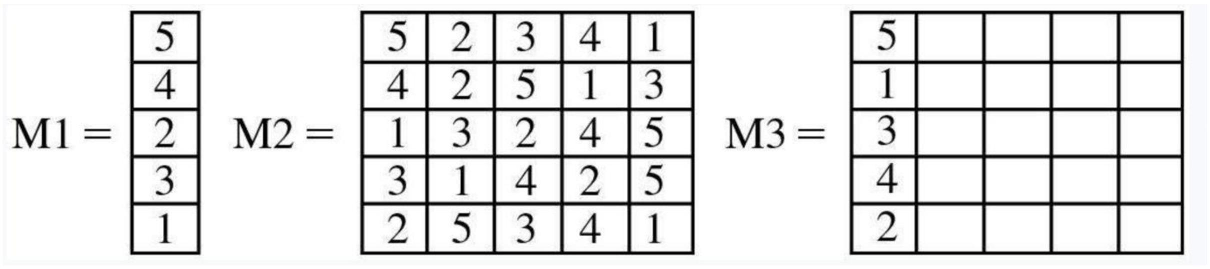
a. It should be clear that it is possible to construct **M3** to satisfy the preceding condition. As an example, fill in **M3** for the following simple case:



$$\text{M1} = \begin{array}{|c|}\hline 5 \\ \hline 4 \\ \hline 2 \\ \hline 3 \\ \hline 1 \\ \hline\end{array} \quad \text{M2} = \begin{array}{|c|c|c|c|c|}\hline 5 & 2 & 3 & 4 & 1 \\ \hline 4 & 2 & 5 & 1 & 3 \\ \hline 1 & 3 & 2 & 4 & 5 \\ \hline 3 & 1 & 4 & 2 & 5 \\ \hline 2 & 5 & 3 & 4 & 1 \\ \hline\end{array} \quad \text{M3} = \begin{array}{|c|c|c|c|c|}\hline 5 & & & & \\ \hline 1 & & & & \\ \hline 3 & & & & \\ \hline 4 & & & & \\ \hline 2 & & & & \\ \hline\end{array}$$

Convention: The *i*th element of **M1** corresponds to $k = i$. The *i*th row of **M2** corresponds to $x = i$; the *j*th column of **M2** corresponds to $p = j$. The *i*th row of **M3** corresponds to $z = i$; the *j*th column of **M3** corresponds to $k = j$. We can look at this in another way. The ith row of **M1** corresponds to the ith column of **M3**. The value of the entry in the *i*th row selects a row of **M2**. The entries in the selected **M3** column are derived from the entries in the selected **M2** row. The first entry in the **M2** row dictates where the value 1 goes in the **M3** column. The second entry in the **M2** row dictates where the value 2 goes in the **M3** column, and so on.

b. Describe the use of this set of tables to perform encryption and decryption between two users.
c. Argue that this is a secure scheme.

9. **2.9** Construct a figure similar to Figure 2.9 that includes a digital signature to authenticate the message in the digital envelope.

Chapter 3

Chapter 3

User Authentication

3.1 Digital User Authentication Principles

NIST SP 800-63-3 (*Digital Identity Guidelines*, June 2017) defines a digital identity as a unique representation of a subject engaged in an online transaction. Then digital user authentication is the process of determining the validity of one or more authenticators used to claim a digital identity. Authentication establishes that a subject has control of those authenticators. Systems can use the authenticated identity to determine if the authenticated individual is authorized to perform particular functions, such as database transactions or system resource access. In many cases, the authentication and transaction, or other authorized function, take place across an open network such as the Internet. Equally, authentication and subsequent authorization can take place locally, such as across a local area network. Table 3.1, from NIST SP 800-171 (*Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organizations*, February 2020), provides a useful list of security requirements for identification and authentication services.

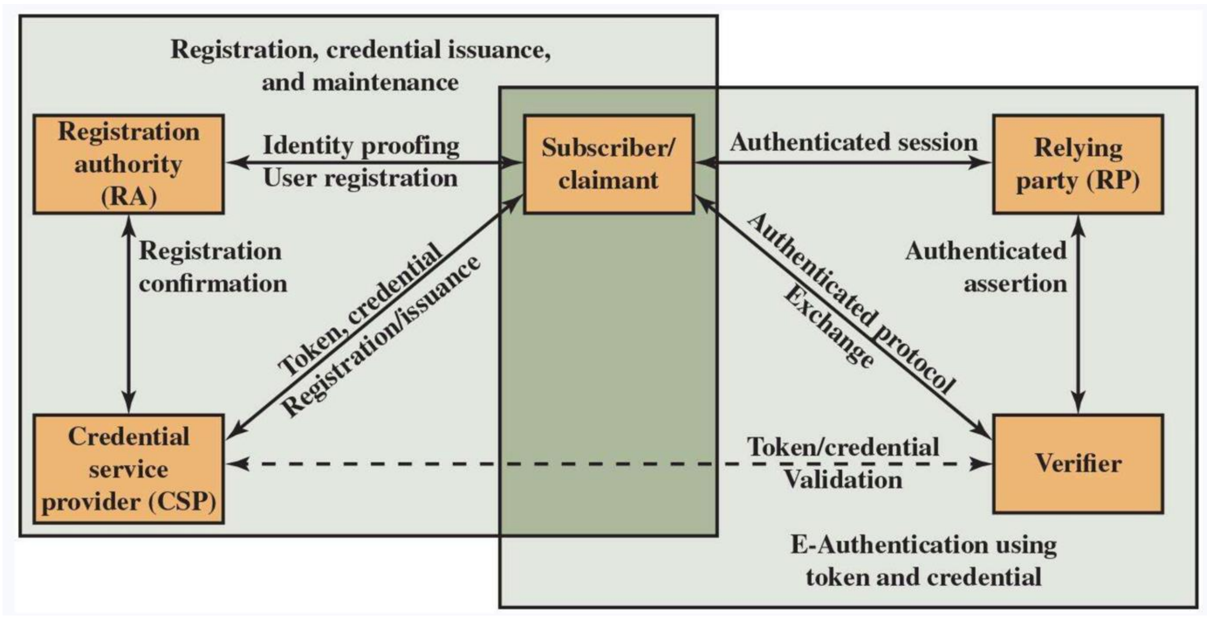
Table 3.1 **Identification and Authentication Security Requirements (NIST SP 800-171)**

Basic Security Requirements
1 Identify information system users, processes acting on behalf of users, or devices.
2 Authenticate (or verify) the identities of those users, processes, or devices as a prerequisite to allowing access to organizational information systems.
Derived Security Requirements:
3 Use multifactor authentication for local and network access to privileged accounts and for network access to non-privileged accounts.
4 Employ replay-resistant authentication mechanisms for network access to privileged and non-privileged accounts.
5 Prevent reuse of identifiers for a defined period.
6 Disable identifiers after a defined period of inactivity.
7 Enforce a minimum password complexity and change of characters when new passwords are created.
8 Prohibit password reuse for a specified number of generations.
9 Allow temporary password use for system logons with an immediate change to a permanent password.
10 Store and transmit only cryptographically-protected passwords.
11 Obscure feedback of authentication information.

A Model for Digital User Authentication

NIST SP 800-63-3 defines a general model for user authentication that involves a number of entities and procedures. We discuss this model with reference to Figure 3.1.

Figure 3.1 **The NIST SP 800-63-3 E-Authentication Architectural Model**


Registration, credential issuance, and maintenance
Registration authority (RA)
Identity proofing
User registration
Subscriber/ claimant
Authenticated session
Relying party (RP)
Registration confirmation
Token, credential Registration/issuance
Authenticated protocol Exchange
Authenticated assertion
Credential service provider (CSP)
Token/credential Validation
Verifier
E-Authentication using token and credential

The initial requirement for performing user authentication is that the user must be registered with the system. The following is a typical sequence for registration. An applicant applies to a **registration authority (RA)** to become a subscriber of a credential service provider (CSP). In this model, the RA is a trusted entity that establishes and vouches for the identity of an applicant to a CSP. The CSP then engages in an exchange with the subscriber. Depending on the details of the overall authentication system, the CSP issues some sort of electronic credential to the subscriber. The credential is a data structure that authoritatively binds an identity and additional attributes to a token possessed by a subscriber and can be verified when presented to the verifier in an authentication transaction. The token could be an encryption key or an encrypted password that identifies the subscriber. The token may be issued by the CSP, generated directly by the subscriber, or provided by a third party. The token and credential may be used in subsequent authentication events.

Once a user is registered as a subscriber, the actual authentication process can take place between the subscriber and one or more systems that perform authentication and, subsequently, authorization. The party to be authenticated is called a claimant, and the party verifying that identity is called a verifier. When a claimant successfully demonstrates possession and control of a token to a verifier through an authentication protocol, the verifier can verify that the claimant is the subscriber named in the corresponding credential. The verifier passes on an assertion about the identity of the subscriber to the relying party (RP). That assertion includes identity information about a subscriber, such as the subscriber name, an identifier assigned at registration, or other subscriber attributes that were verified in the registration process. The RP can use the authenticated information provided by the verifier to make access control or authorization decisions.

An implemented system for authentication will differ from or be more complex than this simplified model, but the model illustrates the key roles and functions needed for a secure authentication system.

Means of Authentication

There are four general means of authenticating a user's identity, which can be used alone or in combination:

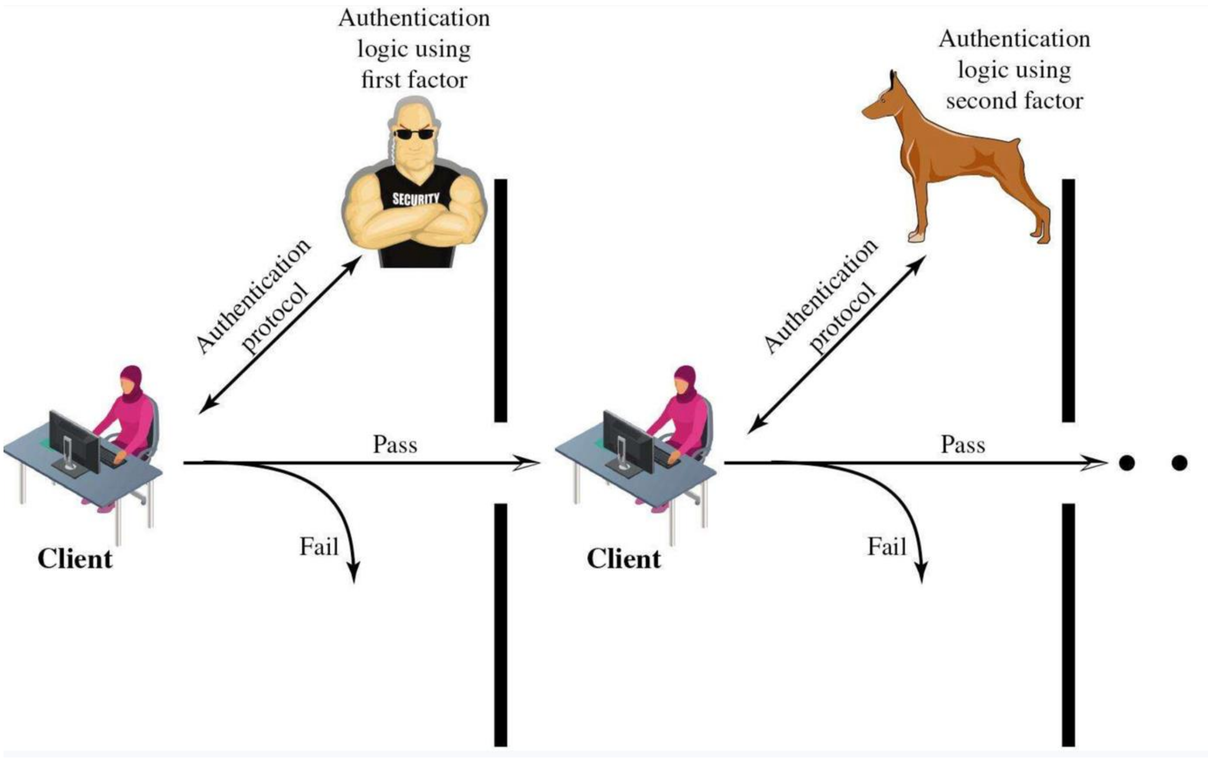
- **Something the individual knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- **Something the individual possesses:** Examples include electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token*.
- **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
- **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Furthermore, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience.

Multifactor Authentication

Multifactor authentication (MFA) refers to an authentication process where the user presents two or more pieces of evidence (or factors) to verify their identity, as shown in Figure 3.2. This usually involves a combination of the methods we discuss in this chapter, based on something they know, possess, are or do. For example, many online banking systems require the user to first enter a password (something they know) and then enter a code sent by SMS to their mobile phone (something they possess). Using multiple factors creates a system that is stronger and more resistant to compromise than using any single factor. Because of this, NIST SP 800-63B requires the use of multiple authentication methods for higher authentication assurance levels. MFA may also be highly recommended or required by governments for some industries.

Figure 3.2 **Multifactor Authentication**



The most common form of MFA used by larger organizations to verify the identity of their staff uses a physical token, often a **one-time password (OTP)** device. These require the user to provide a login and password and then enter the OTP code from their token. For the general public, MFA authentication most commonly uses a mobile device to either receive a code sent via SMS or voice call, or to generate an OTP code using an authenticator app.

Whichever approach is taken, any organization planning to introduce MFA needs to ensure their systems can support the chosen approach. This may be problematic for older legacy systems, which may require costly additional changes to provide this support. [LBJR11] details their conclusions after surveying a range of organizations at that time on their use of MFA. They found that the choice of MFA depended on which sector the organization was in. Defense and areas of government had higher acceptance than the health sector. They found that user resistance to using MFA was not an issue once it was adopted and that tokens were more common than biometrics. MFA adoption was usually part of a broader security architecture, and that government mandates and public perception also encouraged MFA adoption. They concluded that more guidance was needed to assist organizations adopting MFA and on how best to structure their secure use when user computers may be compromised with malware.

While using MFA improves the security of the authentication process, it is still vulnerable to social engineering or phishing attacks in particular, as well as to attacks using malware to intercept the authentication codes. We discuss these concerns further in Chapter 6.

Assurance Levels for User Authentication

An organization can choose between a range of user authentication technologies, based on the degree of confidence in the identity proofing and authentication processes. The choice between these will depend on the security risk assessment for an organization, which we discuss in Chapter 14. NIST SP 800-63-3 defines three separate levels for each of Identity Assurance Level (IAL) and Authenticator Assurance Level (AAL).

The following three IALs reflect the options that an organization can select, based on their risk assessment and the potential harm caused by an attacker making a successful false claim of an identity:

- **IAL1:** Has no requirement to link the applicant to a specific real-life identity. Any attributes provided are self-asserted. An example of where this level is appropriate is a consumer registering to participate in a discussion on an organization's website discussion board.
- **IAL2:** Provides evidence that supports the existence of the claimed identity and uses either remote or physically-present identity proofing to verify that the applicant is appropriately associated with this real-world or pseudonymous identity. This level is appropriate for a wide range of organizations, which require an initial identity assertion.
- **IAL3:** Requires physical presence for identity proofing. Identifying attributes must be verified by an authorized and trained representative of the CSP. This level is appropriate to enable clients or employees to access restricted services of high value or for which improper access is very harmful. For example, a law enforcement official accesses a law enforcement database containing criminal records, where unauthorized access could raise privacy issues and/or compromise investigations.

The following three AALs define options an organization can select, based on their risk assessment and the potential harm caused by an attacker taking control of an authenticator and accessing their systems:

- **AAL1:** Provides some assurance via a secure authentication protocol that the claimant controls authenticator(s) bound to the subscriber's account. A typical authentication technique at this level would be a user-supplied ID and password.
- **AAL2:** Provides high confidence that the claimant controls authenticator(s) bound to the subscriber's account. Proof of possession and control of two distinct authentication factors is required through secure authentication protocol(s) using approved cryptographic techniques.
- **AAL3:** Provides very high confidence that the claimant controls authenticator(s) bound to the subscriber's account. Authentication is based on proof of possession of a key through an approved cryptographic protocol and must use a hardware-based authenticator and an authenticator that provides verifier impersonation resistance.

NIST SP 800-63-3 also provides specific guidance to the selection of suitable Identity and Authenticator Assurance Levels as part of the organization's security risk assessment.

3.2 Password-Based Authentication

A widely used line of defense against intruders is the password system. Virtually all multiuser systems, network-based servers, Web-based e-commerce sites, and other similar services require that a user provide not only a name or identifier (ID) but also a password.

The system compares the password to a previously stored password for that user ID, maintained in a system password file. The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:

- The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.
- The ID determines the privileges accorded to the user. A few users may have administrator or “superuser” status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
- The ID is used in what is referred to as discretionary access control, as we discuss in Chapter 4. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

The Vulnerability of Passwords

In this subsection, we outline the main forms of attack against password-based authentication and briefly outline a countermeasure strategy. The remainder of this section goes into more detail on the key countermeasures.

Typically, a system that uses password-based authentication maintains a password file indexed by user ID. One technique that is typically used is to store not the user's password but a one-way hash function of the password, as described subsequently.

We can identify the following attack strategies and countermeasures:

- **Offline dictionary attack:** Typically, strong access controls are used to protect the system's password file. However, experience shows that determined hackers can frequently bypass such controls and gain access to the file. The attacker obtains the system password file and compares the password hashes against hashes of commonly used passwords. If a match is found, the attacker can gain access with that ID/password combination. Countermeasures include controls to prevent unauthorized access to the password file, intrusion detection measures to identify a compromise, and rapid reissuance of passwords should the password file be compromised.
- **Specific account attack:** The attacker targets a specific account and submits password guesses until the correct password is discovered. The standard countermeasure is an account lockout mechanism, which locks out access to the account after a number of failed login attempts. Typical practice is no more than five access attempts.
- **Popular password attack:** A variation of the preceding attack is to use a popular password and try it against a wide range of user IDs. A user's tendency is to choose a password that is easily remembered; this unfortunately makes the password easy to guess. Countermeasures include policies to inhibit the selection by users of common passwords and scanning the IP addresses of authentication requests and client cookies for submission patterns.
- **Password guessing against single user:** The attacker attempts to gain knowledge about the account holder and system password policies and uses that knowledge to guess the password. Countermeasures include training in and enforcement of password policies that make passwords difficult to guess. Such policies address secrecy, the minimum length of the password, the character set, prohibition against using well-known user identifiers, and the length of time before the password must be changed.
- **Workstation hijacking:** The attacker waits until a logged-in workstation is unattended. The standard countermeasure is automatically logging the workstation out after a period of inactivity. Intrusion detection schemes can be used to detect changes in user behavior.
- **Exploiting user mistakes:** If the system assigns a password, then the user is more likely to write it down because it is difficult to remember. This situation creates the potential for an adversary to read the written password. A user may intentionally share a password, to enable a colleague to share files, for example. Also, attackers are frequently successful in obtaining passwords by using social engineering tactics that trick the user or an account manager into revealing a password. Many computer systems are shipped with preconfigured passwords for system administrators. Unless these preconfigured passwords are changed, they are easily guessed. Countermeasures include user training, intrusion detection, and simpler passwords combined with another authentication mechanism.
- **Exploiting multiple password use:** Attacks can also become much more effective or damaging if different network devices share the same or a similar password for a given user. Countermeasures include a policy that forbids the same or similar password on particular network devices.

- **Electronic monitoring:** If a password is communicated across a network to log on to a remote system, it is vulnerable to eavesdropping. Simple encryption will not fix this problem because the encrypted password is, in effect, the password and can be observed and reused by an adversary.

Despite the many security vulnerabilities of passwords, they remain the most commonly used user authentication technique, and this is unlikely to change in the foreseeable future [HERL12]. Among the reasons for the persistent popularity of passwords are the following:

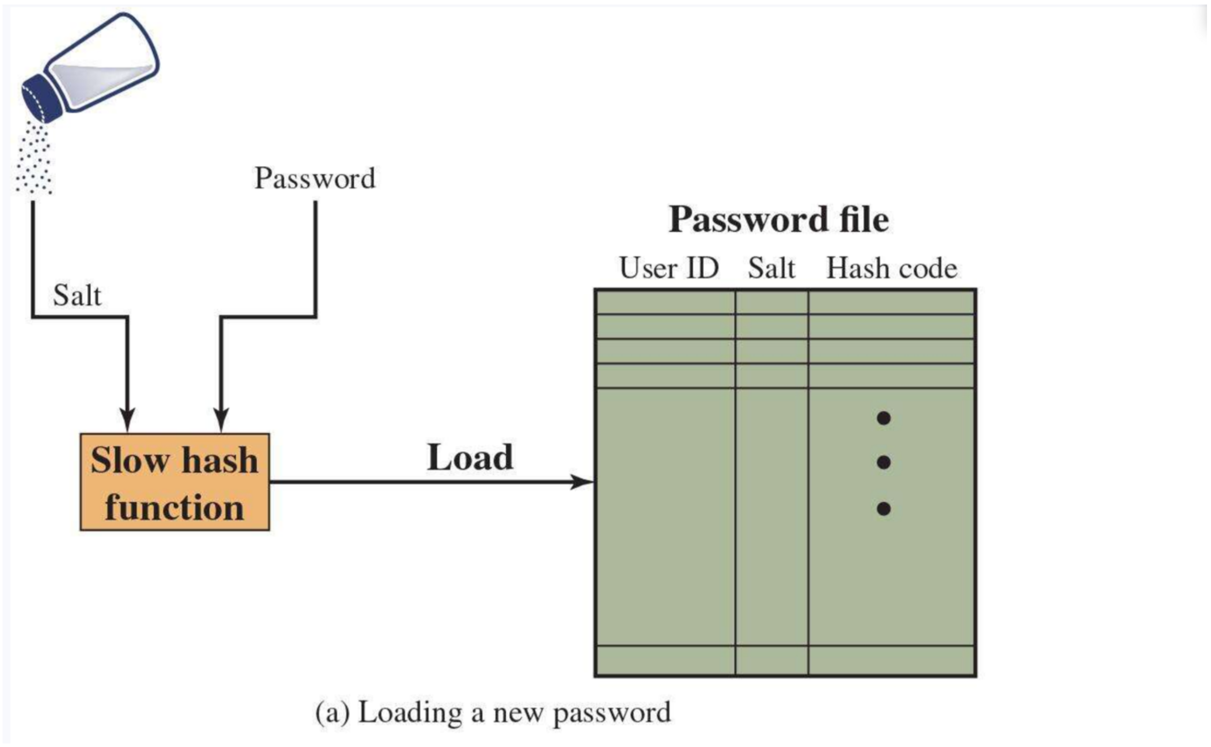
1. Techniques that utilize client-side hardware, such as fingerprint scanners and smart card readers, require the implementation of the appropriate user authentication software to exploit this hardware on both the client and server systems. Until there is widespread acceptance on one side, there is reluctance to implement on the other side, so we end up with a who-goes-first stalemate.
2. Physical tokens, such as smart cards, are expensive and/or inconvenient to carry around, especially if multiple tokens are needed.
3. Schemes that rely on a single sign-on to multiple services, using one of the non-password techniques described in this chapter, create a single point of security risk.
4. Automated password managers that relieve users of the burden of knowing and entering passwords have poor support for roaming and synchronization across multiple client platforms, and their usability has not been adequately researched.

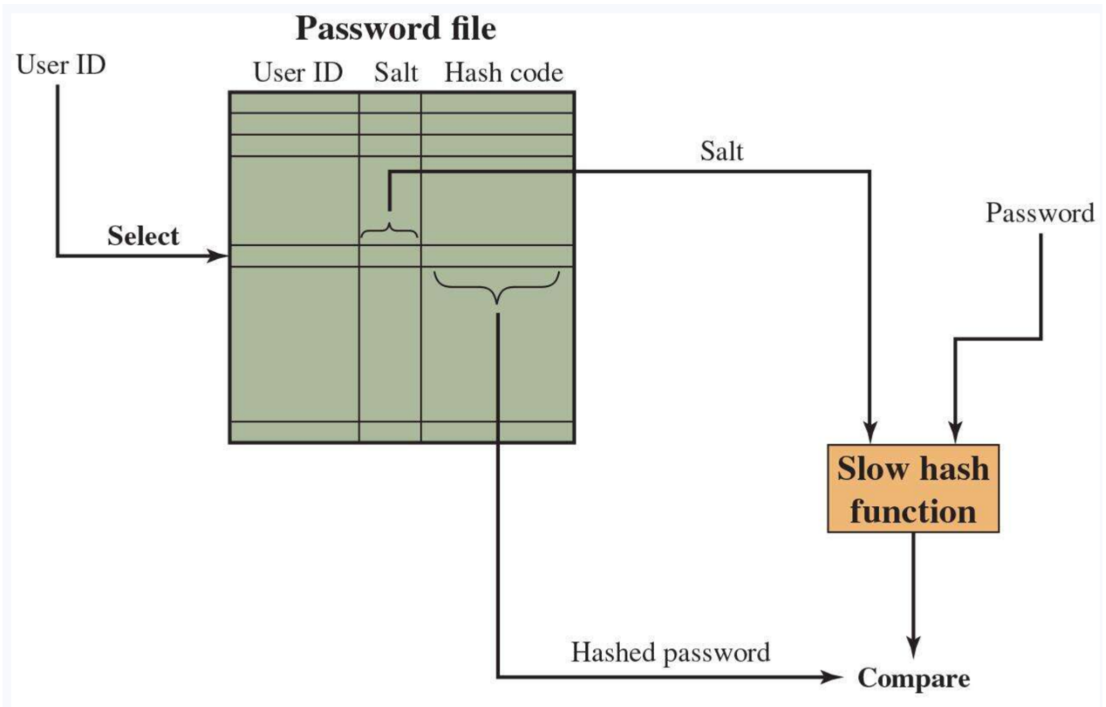
Thus, it is worth our while to study the use of passwords for user authentication in some detail.

The Use of Hashed Passwords

A widely used password security technique is the use of hashed passwords and a salt value. This scheme is found on virtually all UNIX variants as well as on a number of other operating systems. The following procedure is employed (see Figure 3.3a). To load a new password into the system, the user selects or is assigned a password. This password is combined with a fixed-length salt value [MORR79]. In older implementations, this value is related to the time at which the password is assigned to the user. Newer implementations use a pseudorandom or random number. The password and salt serve as inputs to a hashing algorithm to produce a fixed-length hash code. The hash algorithm is designed to be slow to execute in order to thwart attacks. The hashed password is then stored, together with a plaintext copy of the salt, in the password file for the corresponding user ID. The hashed password method has been shown to be secure against a variety of cryptanalytic attacks [WAGN00].

Figure 3.3 **UNIX Password Scheme**





(b) Verif ying a password

When a user attempts to log on to a UNIX system, the user provides an ID and a password (see Figure 3.3b). The operating system uses the ID to index into the password file and retrieve the plaintext salt and the encrypted password. The salt and user-supplied password are used as input to the encryption routine. If the result matches the stored value, the password is accepted.

The **salt** serves three purposes:

- It prevents duplicate passwords from being visible in the password file. Even if two users choose the same password, those passwords will almost certainly have different salt values. Hence, the hashed passwords of the two users will differ.
- It greatly increases the difficulty of offline dictionary attacks. For a salt of length *b* bits, the number of possible passwords is increased by a factor of 2^b increasing the difficulty of guessing a password in a dictionary attack.
- It becomes nearly impossible to find out whether a person with passwords on two or more systems has used the same password on all of them.

To see the second point, consider the way that an offline dictionary attack would work. The attacker obtains a copy of the password file. Suppose first that the salt is not used. The attacker's goal is to guess a single password. To that end, the attacker submits a large number of likely passwords to the hashing function. If any of the guesses matches one of the hashes in the file, then the attacker has found a password that is in the file. But faced with the UNIX scheme, the attacker must take each guess and submit it to the hash function once for each salt value in the dictionary file, multiplying the number of guesses that must be checked.

There are two threats to the UNIX password scheme. First, a user can gain access on a machine using a guest account or by some other means and then run a password guessing program, called a password cracker, on that machine. The attacker should be able to check many thousands of possible passwords with little resource consumption. In addition, if an opponent is able to obtain a copy of the password file, then a cracker program can be run on another machine at leisure. This enables the opponent to run through millions of possible passwords in a reasonable period.

UNIX Implementations

Since the original development of UNIX, many implementations have relied on the following password scheme. Each user selects a password of up to 8 printable characters in length. This is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to an encryption routine. The hash routine, known as crypt(3), is based on DES. A 12-bit salt value is used. The modified DES algorithm is executed with a data input consisting of a 64-bit block of zeros. The output of the algorithm then serves as input for a second encryption. This process is repeated for a total of 25 encryptions. The resulting 64-bit output is then translated into an 11-character sequence. The modification of the DES algorithm converts it into a one-way hash function. The crypt(3) routine is designed to discourage guessing attacks. Software implementations of DES are slow compared to hardware versions, and the use of 25 iterations multiplies the time required by 25.

This particular implementation is now considered woefully inadequate. For example, [PERR03] reports the results of a dictionary attack using a supercomputer. The attack was able to process over 50 million password guesses in about 80 minutes. Furthermore, the results showed that for about $10,000, anyone should be able to do the same in a few months using one uniprocessor machine. Despite its known weaknesses, this UNIX scheme is still often required for compatibility with existing account management software or in multivendor environments.

There are other much stronger hash/salt schemes available for UNIX. The recommended hash function for many UNIX systems, including Linux, Solaris, and FreeBSD (a widely used open source UNIX), is based on the MD5 secure hash algorithm (which is similar to, but not as secure as, SHA-1). The MD5 crypt routine uses a salt of up to 48 bits and effectively has no limitations on password length. It produces a 128-bit hash value. It is also far slower than crypt(3). To achieve the slowdown, MD5 crypt uses an inner loop with 1000 iterations.

Probably the most secure version of the UNIX hash/salt scheme was developed for OpenBSD, another widely used open source UNIX. This scheme, reported in [PROV99], uses a hash function based on the Blowfish symmetric block cipher. The hash function, called Bcrypt, is quite slow to execute. Bcrypt allows passwords of up to 55 characters in length and requires a random salt value of 128 bits to produce a 192-bit hash value. Bcrypt also includes a cost variable; an increase in the cost variable causes a corresponding increase in the time required to perform a Bcyrpt hash. The cost assigned to a new password is configurable, so administrators can assign a higher cost to privileged users.

Password Cracking of User-Chosen Passwords

Traditional Approaches

The traditional approach to password guessing, or password cracking as it is called, is to develop a large dictionary of possible passwords and to try each of these against the password file. This means that each password must be hashed using each available salt value and then compared with stored hash values. If no match is found, the cracking program tries variations on all the words in its dictionary of likely passwords. Such variations include the backward spelling of words, additional numbers or special characters, or a sequence of characters.

An alternative is to trade off space for time by precomputing potential hash values. In this approach the attacker generates a large dictionary of possible passwords. For each password, the attacker generates the hash values associated with each possible salt value. The result is a mammoth table of hash values known as a **rainbow table**. For example, [OECH03] showed that using 1.4 GB of data, they could crack 99.9% of all alphanumeric Windows password hashes in 13.8 seconds. This approach can be countered using a sufficiently large salt value and a sufficiently large hash length. Both the FreeBSD and OpenBSD approaches should be secure from this attack for the foreseeable future.

To counter the use of large salt values and hash lengths, password crackers exploit the fact that some people choose easily guessable passwords. A particular problem is that users, when permitted to choose their own passwords, tend to choose short ones. [BONN12] summarizes the results of a number of studies over the past few years involving over 40 million hacked passwords, as well as their own analysis of almost 70 million anonymized passwords of Yahoo! users, and found a tendency toward a length of six to eight characters and a strong dislike of non-alphanumeric characters in passwords.

The analysis of the 70 million passwords in [BONN12] estimates that passwords provide fewer than 10 bits of security against an online trawling attack, and only about 20 bits of security against an optimal offline dictionary attack. In other words, an attacker who can manage 10 guesses per account, typically within the realm of rate-limiting mechanisms, will compromise around 1% of accounts, just as they would against random 10-bit strings. Against an optimal attacker performing unrestricted brute force and wanting to break half of all available accounts, passwords appear to be roughly equivalent to 20-bit random strings. It can be seen then that using offline search enables an adversary to break a large number of accounts, even if a significant amount of iterated hashing is used.

Password length is only part of the problem. Many people, when permitted to choose their own password, pick a password that is guessable, such as their own name, their street name, a common dictionary word, and so forth. This makes the job of password cracking straightforward. The cracker simply has to test the password file against lists of likely passwords. Because many people use guessable passwords, such a strategy should succeed on virtually all systems.

One demonstration of the effectiveness of guessing is reported in [KLEI90]. From a variety of sources, the author collected UNIX password files containing nearly 14,000 encrypted passwords. The result, which the author rightly characterizes as frightening, was that in all, nearly one-fourth of the passwords were guessed. The following strategy was used:

1. Try the user’s name, initials, account name, and other relevant personal information. In all, 130 different permutations for each user were tried.

2. Try words from various dictionaries. The author compiled a dictionary of over 60,000 words, including the online dictionary on the system itself and various other lists as shown.
3. Try various permutations on the words from step 2. This included making the first letter uppercase or a control character, making the entire word uppercase, reversing the word, changing the letter "o" to the digit "zero," and so on. These permutations added another 1 million words to the list.
4. Try various capitalization permutations on the words from step 2 that were not considered in step 3. This added almost 2 million additional words to the list.

Thus, the test involved nearly 3 million words. Using the fastest processor available, the time to encrypt all these words for all possible salt values was under an hour. Keep in mind that such a thorough search could produce a success rate of about 25%, whereas even a single hit may be enough to gain a wide range of privileges on a system.

Attacks that use a combination of brute-force and dictionary techniques have become common. A notable example of this dual approach is John the Ripper, an open-source password cracker first developed in 1996 and still in use [OPEN13].

Modern Approaches

Sadly, this type of vulnerability has not lessened in the past 25 years or so. Users are doing a better job of selecting passwords, and organizations are doing a better job of forcing users to pick stronger passwords, a concept known as a complex password policy, as discussed subsequently. However, password-cracking techniques have improved to keep pace. The improvements are of two kinds. First, the processing capacity available for password cracking has increased dramatically. Now used increasingly for computing, graphics processors allow password-cracking programs to work thousands of times faster than they did just a decade ago on similarly priced PCs that used traditional CPUs alone. A PC running a single AMD Radeon HD7970 GPU, for instance, can try on average 8.2 x 10^9 password combinations each second, depending on the algorithm used to scramble them [GOOD12a]. Only a decade ago, such speeds were possible only when using pricey supercomputers.

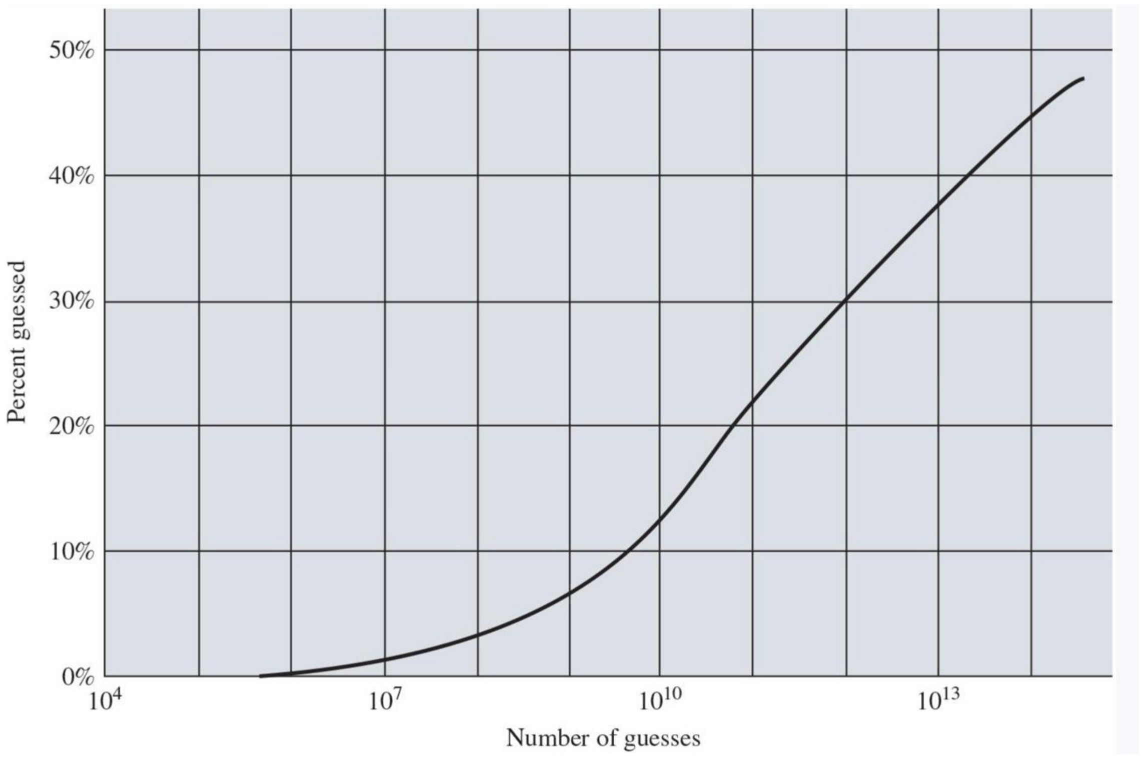
The second area of improvement in password cracking is in the use of sophisticated algorithms to generate potential passwords. For example, [NARA05] developed a model for password generation using the probabilities of letters in natural language. The researchers used standard Markov modeling techniques from natural language processing to dramatically reduce the size of the password space to be searched.

But the best results have been achieved by studying examples of actual passwords in use. To develop techniques that are more efficient and effective than simple dictionary and brute-force attacks, researchers and hackers have studied the structure of passwords. To do this, analysts need a large pool of real-word passwords to study, which they now have. The first big breakthrough came in late 2009, when an SQL injection attack against online games service RockYou.com exposed 32 million plaintext passwords used by its members to log in to their accounts [TIMM10]. Since then, numerous sets of leaked password files have become available for analysis.

Using large datasets of leaked passwords as training data, [WEIR09] reports on the development of a probabilistic context-free grammar for password cracking. In this approach, guesses are ordered according to their likelihood, based on the frequency of their character-class structures in the training data, as well as the frequency of their digit and symbol substrings. This approach has been shown to be efficient in password cracking [KELL12, ZHAN10].

[MAZU13] reports on an analysis of the passwords used by over 25,000 students at a research university with a complex password policy. The analysts used the password-cracking approach introduced in [WEIR09]. They used a database consisting of a collection of leaked password files, including the RockYou file. Figure 3.4 summarizes a key result from the paper. The graph shows the percentage of passwords that have been recovered as a function of the number of guesses. As can be seen, over 10% of the passwords are recovered after only 10^{10} guesses. After 10^{13} guesses, almost 40% of the passwords are recovered.

Figure 3.4 **The Percentage of Passwords Guessed after a Given Number of Guesses**



Password File Access Control

One way to thwart a password attack is to deny the opponent access to the password file. If the hashed password portion of the file is accessible only by a privileged user, then the opponent cannot read it without already knowing the password of a privileged user. Often, the hashed passwords are kept in a separate file from the user IDs, referred to as a **shadow password file**. Special attention is paid to making the shadow password file protected from unauthorized access. Although password file protection is certainly worthwhile, there remain vulnerabilities:

- Many systems, including most UNIX systems, are susceptible to unanticipated break-ins. A hacker may be able to exploit a software vulnerability in the operating system to bypass the access control system long enough to extract the password file. Alternatively, the hacker may find a weakness in the file system or database management system that allows access to the file.
- An accident of protection might render the password file readable, thus compromising all the accounts.
- Some of the users have accounts on other machines in other protection domains, and they use the same password. Thus, if the passwords could be read by anyone on one machine, a machine in another location might be compromised.
- A lack of, or weakness in, physical security may provide opportunities for a hacker. Sometimes, there is a backup to the password file on an emergency repair disk or archival disk. Access to this backup enables the attacker to read the password file. Alternatively, a user may boot from a disk running another operating system such as Linux and access the file from this OS.
- Instead of capturing the system password file, another approach to collecting user IDs and passwords is through sniffing network traffic.

Thus, a password protection policy must complement access control measures with techniques to force users to select passwords that are difficult to guess.

Password Selection Strategies

When not constrained, many users choose a password that is too short or too easy to guess. At the other extreme, if users are assigned passwords consisting of eight randomly selected printable characters, password cracking is effectively impossible. But it would be almost as impossible for most users to remember their passwords. Fortunately, even if we limit the password universe to strings of characters that are reasonably memorable, the size of the universe is still too large to permit practical cracking. Our goal, then, is to eliminate guessable passwords while allowing the user to select a password that is memorable. Four basic techniques are in use:

- User education
- Computer-generated passwords
- Reactive password checking
- Complex password policy

Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords. This **user education** strategy is unlikely to succeed at most installations, particularly where there is a large user population or a lot of turnover. Many users will simply ignore the guidelines. Others may not be good judges of what is a strong password. For example, many users (mistakenly) believe that reversing a word or capitalizing the last letter makes a password unguessable.

Nonetheless, it makes sense to provide users with guidelines on the selection of passwords. Perhaps the best approach is the following advice: A good technique for choosing a password is to use the first letter of each word of a phrase. However, do not pick a well-known phrase like “An apple a day keeps the doctor away” (Aaadktda). Instead, pick something like “My dog’s first name is Rex” (MdfniR) or “My sister Peg is 24 years old” (MsPi24yo). Studies have shown users can generally remember such passwords, but they are not susceptible to password guessing attacks based on commonly used passwords.

Computer-generated passwords also have problems. If the passwords are quite random in nature, users will not be able to remember them. Even if the password is pronounceable, the user may have difficulty remembering it and so be tempted to write it down. In general, computer-generated password schemes have a history of poor acceptance by users. FIPS 181 (*Automated Password Generator*, 1993) defines one of the best-designed automated password generators. The standard includes not only a description of the approach but also a complete listing of the C source code of the algorithm. The algorithm generates words by forming pronounceable syllables and concatenating them to form a word. A random number generator produces a random stream of characters used to construct the syllables and words.

A **reactive password checking** strategy is one in which the system periodically runs its own password cracker to find guessable passwords. The system cancels any passwords that are guessed and notifies the user. This tactic has a number of drawbacks. First, it is resource intensive if the job is done right. Because a determined opponent who is able to steal a password file can devote full CPU time to the task for hours or even days, an effective reactive password checker is at a distinct disadvantage. Furthermore, any existing passwords remain vulnerable until the reactive password checker finds them. A good example is the openware Jack the Ripper password cracker,[11] which works on a variety of operating systems.

A promising approach to improved password security is a **complex password policy**, or **proactive password checker**. In this scheme, a user is allowed to select their own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it. Such checkers are based on the philosophy that, with sufficient guidance from the system, users can select memorable passwords from a fairly large password space that are not likely to be guessed in a dictionary attack.

The trick with a proactive password checker is to strike a balance between user acceptability and strength. If the system rejects too many passwords, users will complain that it is too hard to select a password. If the system uses some simple algorithm to define what is acceptable, it provides guidance to password crackers to refine their guessing technique. In the remainder of this subsection, we will look at possible approaches to proactive password checking.

Rule Enforcement

The first approach is a simple system for rule enforcement. For example, the earlier NIST SP 800-63-2 suggests the following alternative rules:

- Password must have at least sixteen characters (basic16).
- Password must have at least eight characters including an uppercase and a lowercase letter, a symbol, and a digit. It may not contain a dictionary word (comprehensive8).

Although NIST then considered basic16 and comprehensive8 equivalent, [KELL12] found that basic16 is superior against large numbers of guesses. Combined with a prior result that basic16 is also easier for users [KOMA11], this suggests basic16 is the better policy choice. The more recent NIST SP 800-63B (*Digital Identity Guidelines: Authentication and Lifecycle Management*, June 2017) now requires users passwords to have at least 8 characters, with longer passwords or pass phrases encouraged. However, it also cautions that requiring passwords to be too long or complex can make it harder for users to remember them and lead to users taking steps that are counter-productive for security.

Although this approach is superior to simply educating users, it may not be sufficient to thwart password crackers. This scheme alerts crackers to which passwords *not* to try but may still make it possible to do password cracking.

The process of rule enforcement can be automated by using a proactive password checker, such as the openware pam_passwdqc,[12] which enforces a variety of rules on passwords and is configurable by the system administrator.

Password Checker

Another possible procedure, also required by NIST SP 800-63B, is simply to compile a large dictionary of possible “bad” passwords. When a user selects a password, the system checks to make sure that it is not on the disapproved list. There are two problems with this approach:

- **Space:** The dictionary must be very large to be effective.
- **Time:** The time required to search a large dictionary may itself be large. In addition, to check for likely permutations of dictionary words, either those words must be

included in the dictionary, making it truly huge, or each search must also involve considerable processing.

Bloom Filter

A technique [SPAF92a, SPAF92b] for developing an effective and efficient proactive password checker that is based on rejecting words on a list has been implemented on a number of systems, including Linux. It is based on the use of a Bloom filter [BLOO70] that uses a set of hash functions to map words in the password dictionary into a compact hash table.

Suppose we have a dictionary of 1 million words, and we wish to have a 0.01 probability of rejecting a password not in the dictionary. If we choose six hash functions, we need a hash table of 9.6×10^6bits or about 1.2 MB of storage using a Bloom filter. In contrast, storage of the entire dictionary would require on the order of 8 MB. Thus, we achieve a compression of almost a factor of 7. Furthermore, password checking involves the straightforward calculation of six hash functions and is independent of the size of the dictionary, whereas with the use of the full dictionary, there is substantial searching.[13]

3.3 Token-Based Authentication

Objects that a user possesses for the purpose of user authentication are called tokens. In this section, we first examine two types of tokens that are widely used; these are cards that have the appearance and size of bank cards (see Table 3.2). We next introduce hardware tokens, and then discuss the increasingly common use of mobile phones for authentication with either an SMS message, or the use of a software app.

Table 3.2 **Types of Cards Used as Tokens**

Card Type	Defining Feature	Example
Embossed	Raised characters only, on front	Old credit card
Magnetic Stripe	Magnetic bar on back, characters on front	Gift card
Memory	Electronic memory inside	Prepaid phone card
Smart Contact Contactless	Electronic memory and processor inside Electrical contacts exposed on surface Radio antenna embedded inside	Biometric ID card, Credit card

Memory Cards

Memory cards can store but not process data. The most common such card is a bank card with a magnetic stripe on the back. A magnetic stripe can store only a simple security code, which can be read (and unfortunately reprogrammed) by an inexpensive card reader. There are also memory cards that include an internal electronic memory.

Memory cards can be used alone for physical access, such as to a hotel room. For authentication, a user provides both the memory card and some form of password or personal identification number (PIN). A typical application is a gift card. The memory card, when combined with a PIN or password, provides significantly greater security than a password alone. An adversary must gain physical possession of the card (or be able to duplicate it) as well as gain knowledge of the PIN. Among the potential drawbacks, NIST SP 800-12 (*An Introduction to Computer Security: The NIST Handbook*, October 1995) notes the following:

- **Requires special reader:** This increases the cost of using the token and creates the requirement to maintain the security of the reader's hardware and software.
- **Token loss:** A lost token temporarily prevents its owner from gaining system access. Thus, there is an administrative cost in replacing the lost token. In addition, if the token is found, stolen, or forged, then an adversary need only determine the PIN to gain unauthorized access.
- **User dissatisfaction:** Although users may have no difficulty in accepting the use of a memory card as a gift card, its use for computer access may be deemed inconvenient.

Smart Cards

A wide variety of devices qualify as smart tokens. These can be categorized along four dimensions that are not mutually exclusive:

- **Physical characteristics:** Smart tokens include an embedded microprocessor. A smart token that looks like a bank card is called a smart card. Other smart tokens can look like calculators, keys, or other small, portable objects.
- **User interface:** Manual interfaces include a keypad and display for human/token interaction.

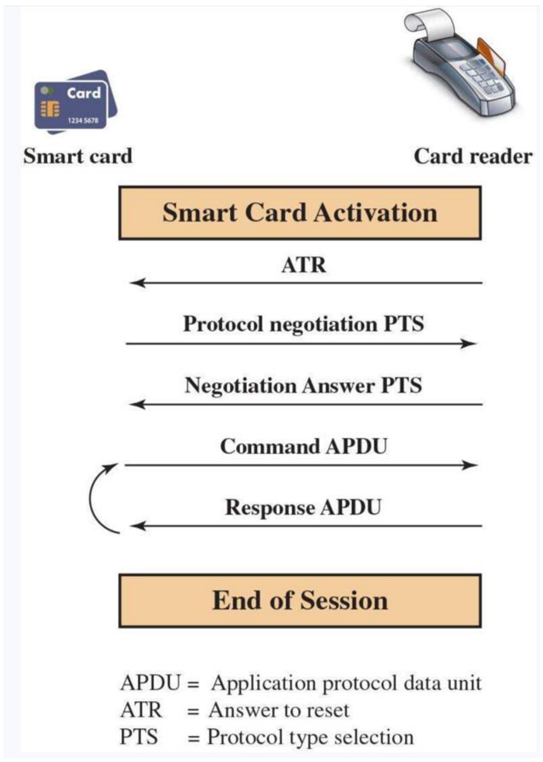
- **Electronic interface:** A smart card or other token requires an electronic interface to communicate with a compatible reader/writer. A card may have one or both of the following types of interface:
  — **Contact:** A contact smart card must be inserted into a smart card reader with a direct connection to a conductive contact plate on the surface of the card (typically gold plated). Transmission of commands, data, and card status takes place over these physical contact points.
  — **Contactless:** A contactless card requires only close proximity to a reader. Both the reader and the card have an antenna, and the two communicate using radio frequencies. Most contactless cards also derive power for the internal chip from this electromagnetic signal. The range is typically one-half to three inches for non-battery-powered cards, ideal for applications such as building entry and payment that require a very fast card interface.
- **Authentication protocol:** The purpose of a smart token is to provide a means for user authentication. We can classify the authentication protocols used with smart tokens into three categories:
  — **Static:** With a static protocol, the user authenticates with the token and then the token authenticates the user to the computer. The latter half of this protocol is similar to the operation of a memory token.
  — **Dynamic password generator:** In this case, the token generates a unique password periodically (e.g., every minute). This password is then entered into the computer system for authentication, either manually by the user or electronically via the token. The token and the computer system must be initialized and kept synchronized so the computer knows the password that is current for this token.
  — **Challenge-response:** In this case, the computer system generates a challenge, such as a random string of numbers. The smart token generates a response based on the challenge. For example, public-key cryptography could be used and the token could encrypt the challenge string with the token's private key.

For user authentication, the most important category of smart token is the smart card, which has the appearance of a credit card, has an electronic interface, and may use any of the type of protocols just described. The remainder of this section discusses smart cards.

A **smart card** contains within it an entire microprocessor, including processor, memory, and I/O ports. Some versions incorporate a special co-processing circuit for cryptographic operation to speed the task of encoding and decoding messages or generating digital signatures to validate the information transferred. In some cards, the I/O ports are directly accessible by a compatible reader by means of exposed electrical contacts. Other cards rely instead on an embedded antenna for wireless communication with the reader.

A typical smart card includes three types of memory. Read-only memory (ROM) stores data that do not change during the card's life, such as the card number and the cardholder's name. Electrically erasable programmable ROM (EEPROM) holds application data and programs, such as the protocols that the card can execute. It also holds data that may vary with time. For example, in a telephone card, the EEPROM holds the remaining talk time. Random access memory (RAM) holds temporary data generated when applications are executed.

Figure 3.5 illustrates the typical interaction between a smart card and a reader or computer system. Each time the card is inserted into a reader, a reset is initiated by the reader to initialize parameters such as clock value. After the reset function is performed, the card responds with an answer to reset (ATR) message. This message defines the parameters and protocols that the card can use and the functions it can perform. The terminal may be able to change the protocol used and other parameters via a protocol type selection (PTS) command. The card's PTS response confirms the protocols and parameters to be used. The terminal and card can now execute the protocol to perform the desired application.



Electronic Identity Cards

An application of increasing importance is the use of a smart card as a national identity card for citizens. A national electronic identity (eID) card can serve the same purposes as other national ID cards, and similar cards such as a driver's license, for access to government and commercial services. In addition, an eID card can provide stronger proof of identity and be used in a wider variety of applications. In effect, an eID card is a smart card that has been verified by the national government as valid and authentic.

One of the more recent and most advanced eID deployments is the German eID card *neuer Personalausweis* [POLL12]. The card has human-readable data printed on its surface, including the following:

- **Personal data:** Such as name, date of birth, and address; this is the type of printed information found on passports and drivers' licenses.
- **Document number:** The alphanumerical nine-character unique identifier of each card.
- **Card access number (CAN):** A six-digit decimal random number printed on the face of the card. This is used as a password, as explained subsequently.
- **Machine readable zone (MRZ):** Three lines of human- and machine-readable text on the back of the card. This may also be used as a password.

Eid Functions

The card has the following three separate electronic functions, each with its own protected dataset (see Table 3.3):

- **ePass:** This function is reserved for government use and stores a digital representation of the cardholder's identity. This function is similar to, and may be used for, an electronic passport. Other government services may also use ePass. The ePass function must be implemented on the card.
- **eID:** This function is for general-purpose use in a variety of government and commercial applications. The eID function stores an identity record that authorized service can access with cardholder permission. Citizens choose whether they want this function activated.
- **eSign:** This optional function stores a private key and a certificate verifying the key; it is used for generating a digital signature. A private sector trust center issues the certificate.

Table 3.3 **Electronic Functions and Data for eID Cards**

Function	Purpose	PACE Password	Data	Uses
ePass (mandatory)	Authorized offline inspection systems read the data.	CAN or MRZ	Face image; two fingerprint images (optional); MRZ data	Offline biometric identity verification reserved for government access
	Online applications read the data or access functions as authorized.	eID PIN	Family and given names; artistic name and doctoral	Identification; age verification; community ID

eID (activation optional)	Offline inspection systems read the data and update the address and community ID.	CAN or MRZ	degree; date and place of birth; address and community ID; expiration date	verification; restricted identification (pseudonym); revocation query
eSign (certificate optional)	A certification authority installs the signature certificate online.	eID PIN	Signature key; X.509 certificate	Electronic signature creation
	Citizens make electronic signature with eSign PIN.	CAN		

CAN = card access number

MRZ = machine readable zone

PACE = password authenticated connection establishment

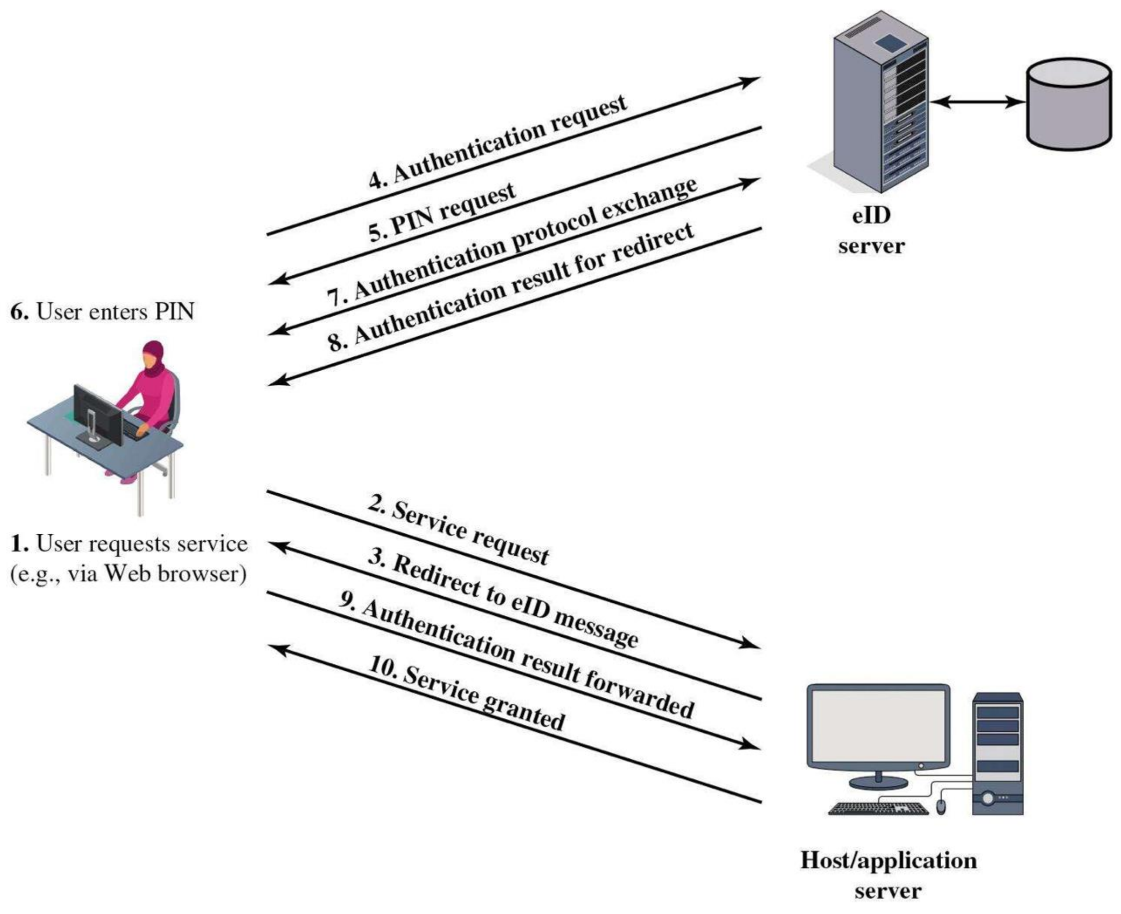
PIN = personal identification number

The ePass function is an offline function. That is, it is not used over a network, but rather is used in a situation where the cardholder presents the card for a particular service at that location, such as going through a passport control checkpoint.

The eID function can be used for both online and offline services. An example of an offline use is an inspection system. An inspection system is a terminal for law enforcement checks, for example, by police or border control officers. An inspection system can read the identifying information of the cardholder as well as biometric information stored on the card, such as facial image and fingerprints. The biometric information can be used to verify that the individual in possession of the card is the actual cardholder.

User authentication is a good example of online use of the eID function. Figure 3.6 illustrates a Web-based scenario. To begin, an eID user visits a website and requests a service that requires authentication. The website sends back a redirect message that forwards an authentication request to an eID server. The eID server requests that the user enter the PIN number for the eID card. Once the user has correctly entered the PIN, data can be exchanged between the eID card and the terminal reader in encrypted form. The server then engages in an authentication protocol exchange with the microprocessor on the eID card. If the user is authenticated, the results are sent back to the user system to be redirected to the Web server application.

Figure 3.6 **User Authentication with eID**


4. Authentication request
5. PIN request
7. Authentication protocol exchange
8. Authentication result for redirect
eID server
6. User enters PIN
1. User requests service (e.g., via Web browser)
2. Service request
3. Redirect to eID message
9. Authentication result forwarded
10. Service granted
Host/application server

For the preceding scenario, the appropriate software and hardware are required on the user system. Software on the main user system includes functionality for requesting and accepting the PIN number and for message redirection. The hardware required is an eID card reader. The card reader can be an external contact or contactless reader or a contactless reader internal to the user system.

Password Authenticated Connection Establishment (PACE)

Password Authenticated Connection Establishment (PACE) ensures that the contactless RF chip in the eID card cannot be read without explicit access control. For online applications, access to the card is established by the user entering the six-digit PIN, which should be known only to the holder of the card. For offline applications, either the MRZ printed on the back of the card or the six-digit card access number (CAN) printed on the front is used.

Hardware Authentication Tokens

We now discuss other types of hardware authentication tokens that can be used in the authentication process. These hold one or more embedded keys unique to the device and the ability to perform cryptographic operations using them in the authentication process.

One of the simplest hardware tokens is a **one-time password (OTP)** device. This has an embedded secret key that is used as a seed to generate an OTP, which it displays. The user enters the current OTP as part of an identity verification process on some system. That system separately computes the expected OTP and confirms whether the user has entered the correct value. Each OTP may only be used once. The OTP may be a constantly changing value, based on the current time, or it may be generated from a counter or other value, known as a nonce, that updates each time an OTP is used. These devices typically use a block cipher or hash function to cryptographically combine the secret key and the time or nonce value to create the OTP. They also typically include some form of tamper-resistant module to securely store the embedded secret key.

One of the most widely implemented OTP algorithms is "Time-based one-time password (TOTP)" that uses HMAC with a hash function such as SHA-1 and is specified in RFC 6238. We discuss HMAC and the SHA family of hash functions in Chapter 21. This algorithm is used in a range of hardware tokens, as well as by many of the mobile authenticator apps that we discuss shortly. The TOTP password is computed from the current Unix format time value as

$$T = \text{floor}\,((\text{Current Unix time} - \text{Time0})/\text{Step})$$

$$\text{TOTP}\ (\text{Key}, T) = \text{Truncate}\big(\text{HMAC-SHA-1}\ (\text{Key}, T)\big)$$

Where the following values are selected and shared with the service being authenticated when the authentication service is initialized

Key = a randomly generated or derived secret value at least as large as the hash function output

Time0 = the initial time value in seconds (default 0)

Step = the time step, or window, in seconds, used before the password is updated (default 30)

Truncate is a function that converts an HMAC-SHA-1 value into a TOTP value and is specified in RFC 4226, which RFC 6238 extends. It truncates the hash value down to 32 bits and then computes a decimal value with the desired number of digits, by default 6, from this. This value is then displayed to the user as the current one-time password value to use. Systems can choose to use one of the more recent SHA-256 or SHA-512 hash functions if desired, rather than the original SHA-1, if supported by the token. They can also use a password with 7 or 8 digits rather than the minimum of 6, if supported.

Systems using a time based OTP system have a window of some seconds or minutes before generating a new value, as shown in the TOTP calculation above. They need to allow for possible clock drift between the token and the verifying system. This is usually done by allowing OTP values for times within a small window of the current system time. If an OTP is supplied for a time slightly different to the system clock, then the system can record the drift amount, in order to compensate in future interactions. If the token and system drift too far out of synchronization, then a resynchronize process must be used to reestablish synchronization.

Systems that use a nonce need to allow for failed authentication attempts, that mean the token is generating an OTP that is one or a few steps later than that last successfully used to authenticate. If such a later OTP is detected, the system updates their nonce to match. As with time based systems, if the token and system nonce values are too far out of synchronization, then a resynchronize process must be used.

One disadvantage of tokens that display the code for the user to enter, is that another person can glance at the display and see the code. Hence care is needed when using such tokens. As an alternative, the token may use a communications link with the authenticating system, rather than a separate display. This may require the user to insert the token into a USB port on the system. Or it may use a near-field communication (NFC) or low energy Bluetooth (BLE) wireless connection. Such tokens often combine the OTP function with support for cryptographic operations. They contain one or more embedded keys and can perform some public or private key cryptographic operation, such as signing a challenge value. They may also provide more general support for securing keys and supporting a range of cryptographic operations beyond just authentication.

NIST SP 800-63B distinguishes between single-factor and multifactor devices. A single-factor device functions as we describe above. A multifactor device only provides the authentication service after some additional local authentication step on the device. This may involve the user supplying a PIN or password via a keyboard or sent using a USB or other communications interface, or with the use of a biometric reader on the device.

In recent years there has been growing support for the FIDO2 authentication protocol. This consists of the W3C Web Authentication “WebAuthn” standard and the FIDO Alliance “Client to Authenticator Protocol 2 (CTAP2).” These specify a standard for the communications between authenticator token or app and an application or server using the authentication service. It typically uses a user agent, such as a web browser with a WebAuthn client, as an intermediary between the authenticator and the authenticating service. Two prominent members of the FIDO Alliance, Google and Microsoft, both use FIDO2 to allow the use of authenticator tokens or apps as a second authentication factor.

Disadvantages of using a token include the token being lost or stolen, an attacker compromising the user’s computer system used by installing malware, which subverts the authentication process, or an attacker using social engineering to convince the user to either reveal the authentication code, or to approve an authentication request. [JAKR21] and [FLPZ21] both analyze several MFA authentication protocols, including FIDO used with a token, looking at their security against a range of attacks. They conclude that the authentication protocols are secure against some attacks, but may be compromised by others. They also suggest some changes to the protocols, which could improve security.

Authentication Using a Mobile Phone

Mobile (or cell) phones are increasingly used as an authentication token. The authentication process can use a code sent to the phone via either SMS or a voice message, which the user must then enter to verify their identity when logging in to a system. Or it may involve an app on the phone, which can function as a one-time password generator, or engage in an active exchange with the system where the user is verifying their identity. The main advantage of using a mobile phone as an authentication token is that it is something many users already own, carry with them, and use for a wide variety of applications. However, there are some disadvantages, as we discuss below.

Sending an authentication code to a mobile phone with an SMS or voice message is one of the simplest approaches to using a mobile phone as an authentication token. This form of authentication has been used for banking, government service access, and other uses for more than a decade. It has the advantage of not requiring any additional app on the phone. Indeed it can be used with even a basic feature phone without the ability to install additional apps.

However, there are a number of disadvantages with this approach [JOVE20]. First, you need mobile coverage to receive the SMS or voice message. This method does not work if the user’s phone does not have service. This may occur when travelling overseas for example. Mobile phones may also be lost or stolen. If this occurs, then the user will lose access until they can acquire a new phone, and transfer their phone number to it. And if the phone is stolen by an attacker who has also gained access to the user’s account name and password, they may be able to access the user’s accounts on systems for a period. Attackers may also try to transfer the user’s phone number to a new phone under their control, using a SIM swap attack. This widely used attack exploits deficiencies in phone companies customer support services used to transfer a phone number when a user changes their phone. While phone companies should check the validity of such transfer requests, there are numerous examples when these checks have been inadequate, a number has been transferred without the correct authorization, and then used to gain access to the legitimate user’s accounts [LKMN20]. If the number is successfully transferred, the attacker receives all calls and SMS messages, including authentication messages, sent to that phone number. Another attack exploits the recovery options that allow a user to reset the phone number used for authentication. If the attacker has access to the user’s emails, they may be able to change the phone number used for these messages and then gain access to the user’s accounts. It is also possible, though technically harder, for an attacker to intercept the SMS or voice message using either a fake mobile tower, or by attacking the underlying SS7 signaling protocol used by phone companies to manage calls and messages.

Because of these various limitations and security concerns, NIST SP 800-63B classifies authentication using SMS messages as restricted, with other alternatives encouraged. However it continues to be widely used for the reasons described above. And because, even with these concerns, it is still significantly more secure than just using a password alone.

Alternatively, authentication may involve the use of an app installed on the user’s mobile phone. The app usually implements a one-time password generator, as an alternative to a dedicated hardware token. There are widely accepted apps that provide this functionality from Microsoft, Google, and others. These apps mostly implement the “Time-based one-time password (TOTP)” algorithm that we described earlier. They have the advantage of not requiring a network connection when used.

These authentication apps may be used with multiple accounts. If an authentication app is used, it is important that the user generate and securely store backup codes for the app. These allow the app to be reinstalled on a new device if the original is lost or stolen. The use of an authentication app is generally regarded as more secure than sending an authentication code by SMS or phone message.

Disadvantages of this approach include the phone being lost or stolen, an attacker compromising the phone by installing malware, which subverts the authentication process, an attacker sending large numbers of authentication requests, known as “prompt-bombing” in the hope the user will just approve the access, or an attacker using social engineering to convince the user to reveal the authentication code. These latter attacks highlight the need for user awareness training, so that they only approve authentication requests they have initiated from their systems, as we discuss in Chapter 17. [OZBI20] describes how the authors recovered the secret key values from seven common authentication apps installed on an Android phone, despite the security protections Android provides for apps. This highlights that if an attacker has physical access to the device, its security may be compromised. [DLRS14] explores a range of attacks on mobile device authenticator apps, including infection of the PC the user authenticates with, infection of the mobile device, and attacks on the initialization and recovery mechanisms used by the authentication apps. Again, as with the use of SMS messages for authentication, although there are risks in using an authentication app, which should be mitigated, there use provides considerably greater security than just using a password alone.

Because of these concerns, NIST SP 800-63B specifies some requirements when mobile phones are used as authenticators. These include that the user uses the phone unlock mechanism before the authenticator app provides the one-time code, to provide additional protection for such devices.

We discuss a number of security concerns further in Section 24.2, as they are specific instances of the more general mobile device security issues.

3.4 Biometric Authentication

A biometric authentication system attempts to authenticate an individual based on their unique physical characteristics. These can be classified as either a static biometric, or a **dynamic biometric**, as we noted in Section 3.1. Static biometrics include static characteristics such as fingerprints, hand geometry, facial characteristics, and retinal and iris patterns. Dynamic biometrics include dynamic characteristics such as voiceprint and signature. In essence, biometrics is based on pattern recognition. Compared to passwords and tokens, biometric authentication is both technically more complex and expensive. It requires a suitable sensor for the chosen biometric characteristic. These range from relatively common sensors such as a camera for face or iris, or a microphone for voice, to specific sensors needed for characteristics such retina or fingerprint recognition. The need for such dedicated sensors has traditionally limited the general acceptance of biometric authentication. However with most mobile devices now including a camera and microphone, and increasingly also a fingerprint sensor, we see growing use of biometric authentication with these devices.

Physical Characteristics Used in Biometric Applications

A number of different types of physical characteristics are either in use or under study for user authentication. The most common are the following:

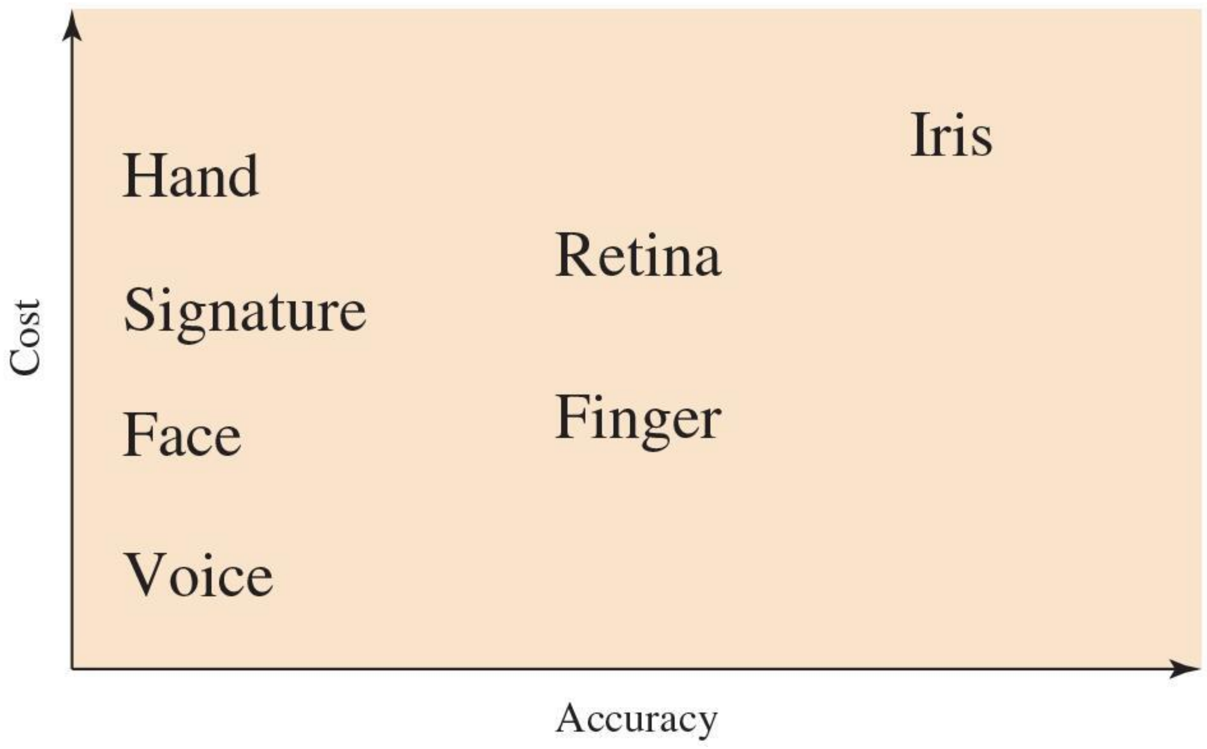
- **Facial characteristics:** Facial characteristics are the most common means of human-to-human identification; thus, it is natural to consider them for identification by computer. The most common approach is to define characteristics based on relative location and shape of key facial features, such as eyes, eyebrows, nose, lips, and chin shape. An alternative approach is to use an infrared camera to produce a face thermogram that correlates with the underlying vascular system in the human face.
- **Fingerprints:** Fingerprints have been used as a means of identification for centuries, and the process has been systematized and automated particularly for law enforcement purposes. A fingerprint is the pattern of ridges and furrows on the surface of the fingertip. Fingerprints are believed to be unique across the entire human population. In practice, automated fingerprint recognition and matching systems extract a number of features from the fingerprint for storage as a numerical surrogate for the full fingerprint pattern.
- **Hand geometry:** Hand geometry systems identify features of the hand, including shape and lengths and widths of fingers.
- **Retinal pattern:** The pattern formed by veins beneath the retinal surface is unique and therefore suitable for identification. A retinal biometric system obtains a digital

image of the retinal pattern by projecting a low-intensity beam of visual or infrared light into the eye.

- **Iris:** Another unique physical characteristic is the detailed structure of the iris.
- **Signature:** Each individual has a unique style of handwriting, and this is reflected especially in the signature, which is typically a frequently written sequence. However, multiple signature samples from a single individual will not be identical. This complicates the task of developing a computer representation of the signature that can be matched to future samples.
- **Voice:** Whereas the signature style of an individual reflects not only the unique physical attributes of the writer but also the writing habit that has developed, voice patterns are more closely tied to the physical and anatomical characteristics of the speaker. Nevertheless, there is still a variation from sample to sample over time from the same speaker, complicating the biometric recognition task.

Figure 3.7 gives a rough indication of the relative cost and accuracy of these biometric measures. The concept of accuracy does not apply to user authentication schemes using smart cards or passwords. For example, if a user enters a password, it either matches exactly the password expected for that user or not. In the case of biometric parameters, the system instead must determine how closely a presented biometric characteristic matches a stored characteristic. Before elaborating on the concept of biometric accuracy, we need to have a general idea of how biometric systems work.

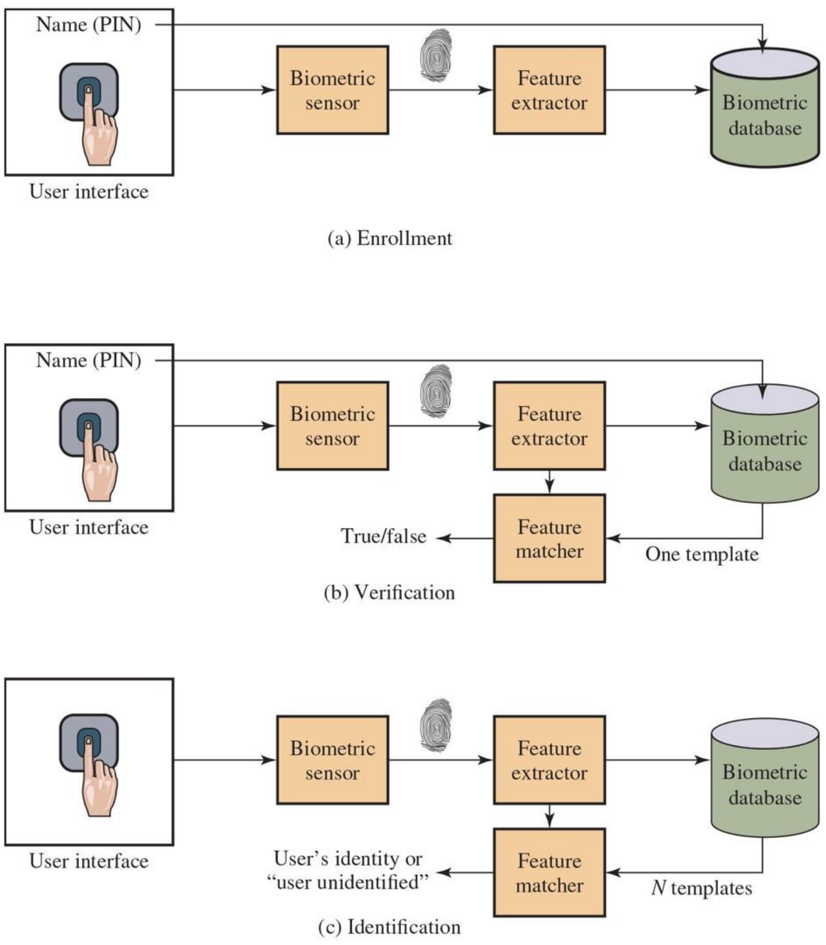
Figure 3.7 **Cost versus Accuracy of Various Biometric Characteristics in User Authentication Schemes**



Operation of a Biometric Authentication System

Figure 3.8 illustrates the operation of a biometric system. Each individual who is to be included in the database of authorized users must first be **enrolled** in the system. This is analogous to assigning a password to a user. For a biometric system, the user presents a name and, typically, some type of password or PIN to the system. At the same time, the system senses some biometric characteristic of this user (e.g., fingerprint of right index finger). The system digitizes the input and then extracts a set of features that can be stored as a number or set of numbers representing this unique biometric characteristic; this set of numbers is referred to as the user's template. The user is now enrolled in the system, which maintains for the user a name (ID), perhaps a PIN or password, and the biometric value.

Figure 3.8 A Generic Biometric System



Enrollment creates an association between a user and the user's biometric characteristics. Depending on the application, user authentication involves either verifying that a claimed user is the actual user or identifying an unknown user.

Depending on the application, user authentication on a biometric system involves either **verification** or identification. Verification is analogous to a user logging on to a system by using a memory card or smart card coupled with a password or PIN. For biometric verification, the user enters a PIN and also uses a biometric sensor. The system extracts the corresponding feature and compares that to the template stored for this user. If there is a match, then the system authenticates this user.

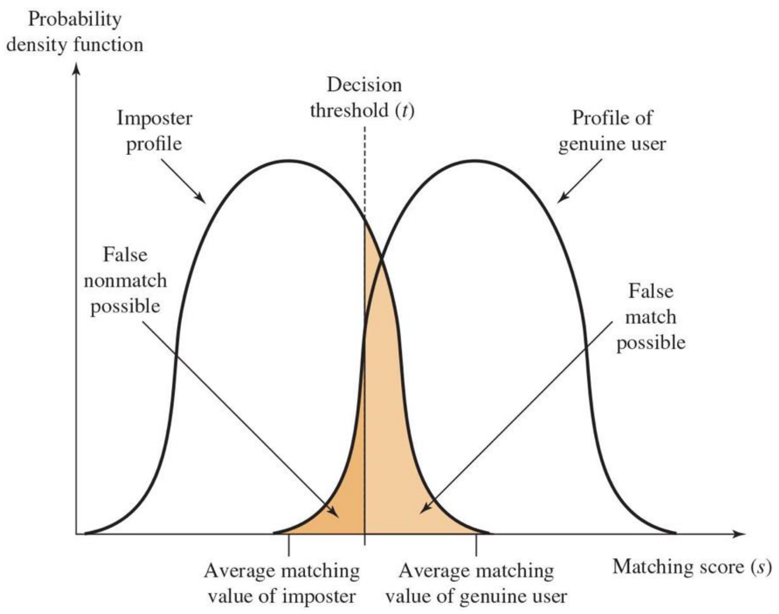
For an identification system, the individual uses the biometric sensor but presents no additional information. The system then compares the presented template with the set of stored templates. If there is a match, then this user is identified. Otherwise, the user is rejected.

Biometric Accuracy

In any biometric scheme, some physical characteristic of the individual is mapped into a digital representation. For each individual, a single digital representation, or template, is stored in the computer. When the user is to be authenticated, the system compares the stored template to the presented template. Given the complexities of physical characteristics, we cannot expect that there will be an exact match between the two templates. Rather, the system uses an algorithm to generate a matching score (typically a single number) that quantifies the similarity between the input and the stored template. To proceed with the discussion, we define the following terms. The false match rate is the frequency with which biometric samples from different sources are erroneously assessed to be from the same source. The false nonmatch rate is the frequency with which samples from the same source are erroneously assessed to be from different sources.

Figure 3.9 illustrates the dilemma posed to the system. If a single user is tested by the system numerous times, the matching score *s* will vary, with a probability density function typically forming a bell curve, as shown. For example, in the case of a fingerprint, results may vary due to sensor noise, changes in the print due to swelling or dryness, finger placement, and so on. On average, any other individual should have a much lower matching score but will also exhibit a bell-shaped probability density function. The difficulty is that the range of matching scores produced by two individuals, one genuine and one an imposter, compared to a given reference template, are likely to overlap. In Figure 3.9, a threshold value is selected such that if the presented value $s \geq t$ a match is assumed, and for $s < t$ a mismatch is assumed. The shaded part to the right of *t* indicates a range of values for which a false match is possible, and the shaded part to the left indicates a range of values for which a false nonmatch is possible. A false match results in the acceptance of a user who should not be accepted, and a false mismatch triggers the rejection of a valid user. The area of each shaded part represents the probability of a false match or nonmatch, respectively. By moving the threshold left or right, the probabilities can be altered, but note that a decrease in the false match rate results in an increase in the false nonmatch rate, and vice versa.

Figure 3.9 **Profiles of a Biometric Characteristic of an Imposter and an Authorized User**



In this depiction, the comparison between the presented feature and a reference feature is reduced to a single numeric value. If the input value (s) is greater than a preassigned threshold (t), a match is declared.

For a given biometric scheme, we can plot the false match rate versus the false nonmatch rate, called the operating characteristic curve. Figure 3.10 shows idealized curves for two different systems. The curve that is lower and to the left performs better. The dot on the curve corresponds to a specific threshold for biometric testing. Shifting the threshold along the curve up and to the left provides greater security and the cost of decreased convenience. The inconvenience comes from a valid user being denied access and being required to take further steps. A plausible trade-off is to pick a threshold that corresponds to a point on the curve where the rates are equal. A high-security application may require a very low false match rate, resulting in a point farther to the left on the curve. For a forensic application, in which the system is looking for possible candidates to be checked further, the requirement may be for a low false nonmatch rate.

Figure 3.10 **Idealized Biometric Measurement Operating Characteristic Curves (log-log scale)**

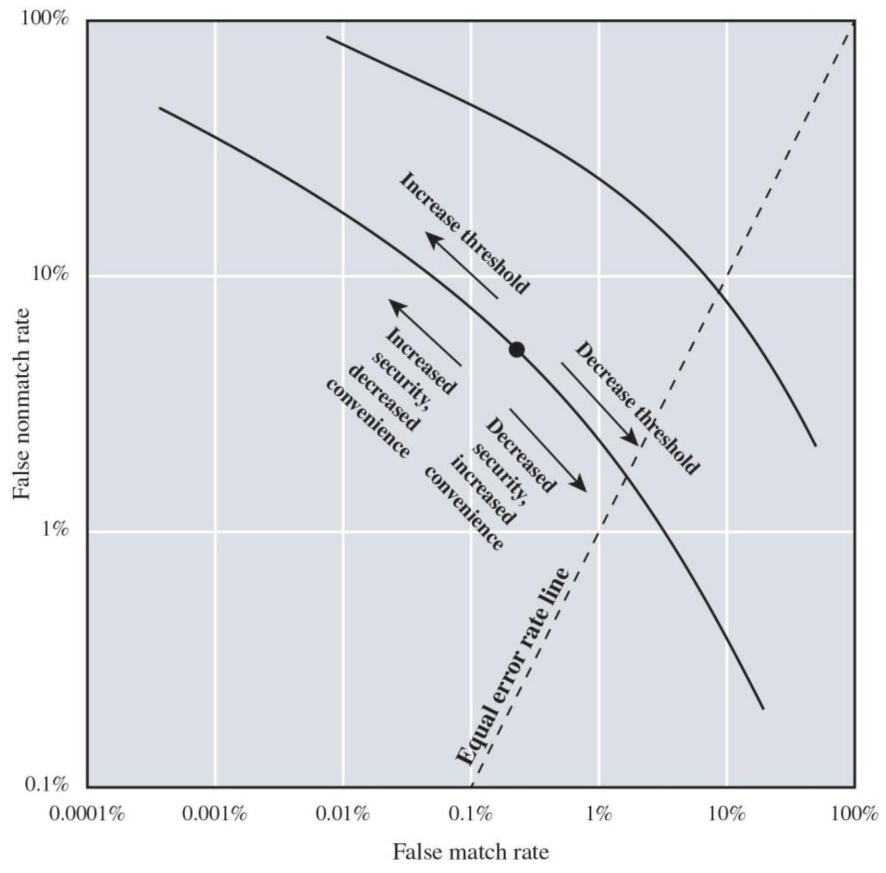
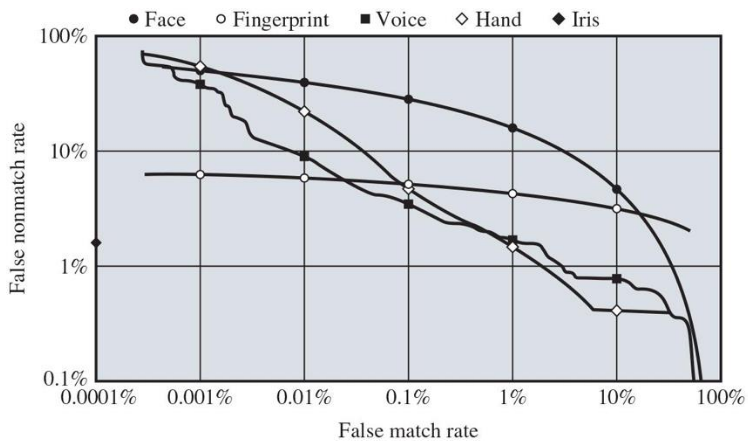


Figure 3.11 shows characteristic curves developed from actual product testing. The iris system had no false matches in over 2 million cross-comparisons. Note that over a broad range of false match rates, the face biometric is the worst performer.

Figure 3.11 **Actual Biometric Measurement Operating Characteristic Curves**



To clarify differences among systems, a log-log scale is used.

Source: From [MANSO1]. Mansfield, T., Gavin Kelly, David Chandler, Jan Kane. Biometric Product Testing Final Report. National Physics Laboratory, United Kingdom, March 2001. United Kingdom National Archives, Open Government Licence v3.0.

3.5 Remote User Authentication

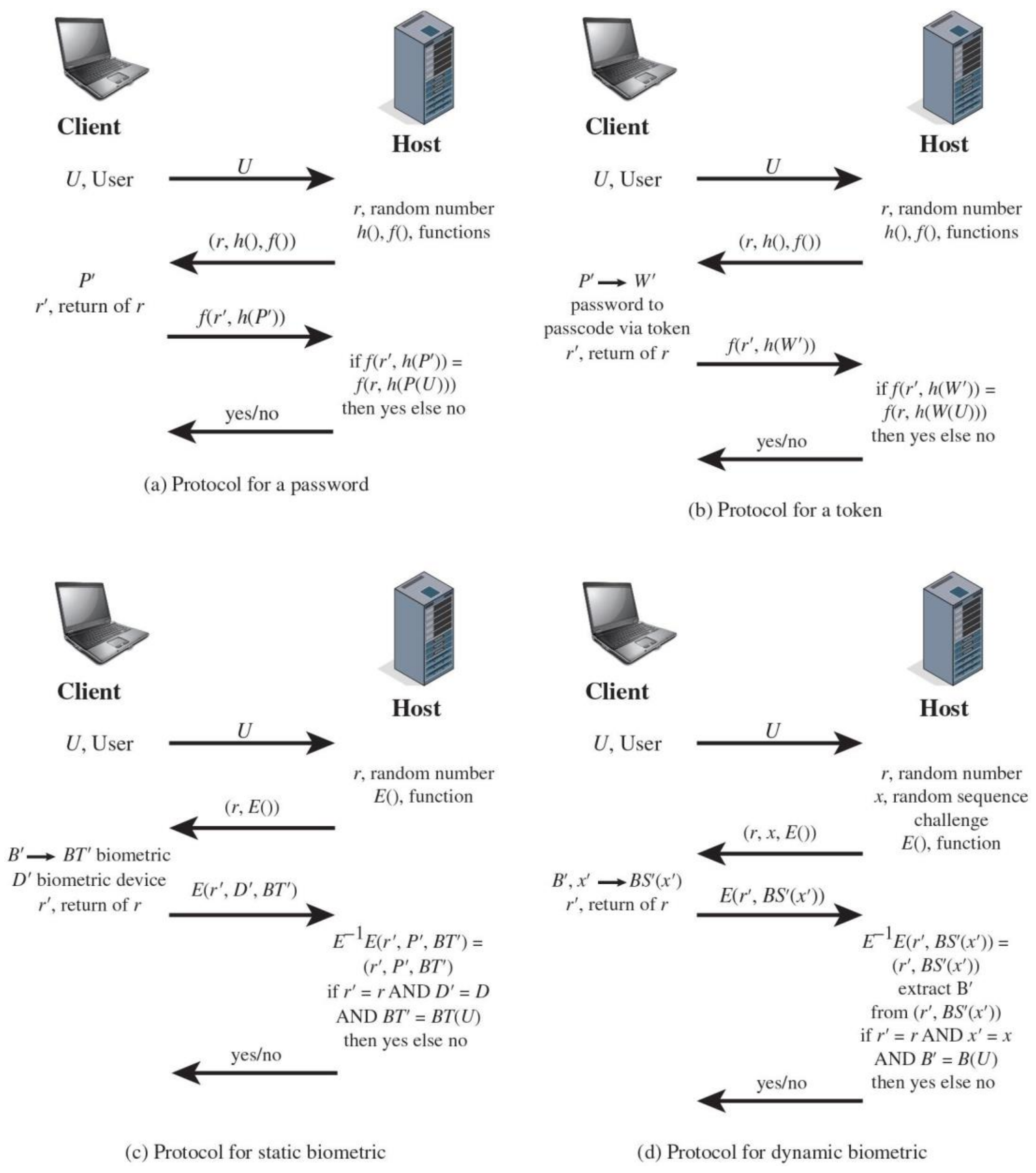
The simplest form of user authentication is local authentication, in which a user attempts to access a system that is locally present, such as a stand-alone office PC or an ATM machine. The more complex case is that of remote user authentication, which takes place over the Internet, a network, or a communications link. Remote user authentication raises additional security threats, such as an eavesdropper being able to capture a password or an adversary replaying an authentication sequence that has been observed.

To counter threats to remote user authentication, systems generally rely on some form of **challenge-response protocol**. In this section, we present the basic elements of such protocols for each of the types of authenticators discussed in this chapter.

Password Protocol

Figure 3.12a provides a simple example of a challenge-response protocol for authentication via password. Actual protocols are more complex, such as Kerberos, to be discussed in Chapter 23. In this example, a user first transmits their identity to the remote host. The host generates a random number *r*, often called a nonce, and returns this nonce to the user. In addition, the host specifies two functions, h( ) and *f*( ), to be used in the response. This transmission from host to user is the challenge. The user's response is the quantity *f*(r′, *h*(P′)), where r′ = r and P′ is the user's password. The function *h* is a hash function, so the response consists of the hash function of the user's password combined with the random number using the function *f*.

Figure 3.12 Basic Challenge-Response Protocols for Remote User Authentication



Source: Based on [OGOR03].

The host stores the hash function of each registered user's password, depicted as h(P(U)) for user *U*. When the response arrives, the host compares the incoming $f(r', h(P'))$ to the calculated $f(r, \mathrm{h}(P(\mathrm{U})))$. If the quantities match, the user is authenticated.

This scheme defends against several forms of attack. The host stores not the password but a hash code of the password. As discussed in Section 3.2, this secures the password from intruders into the host system. In addition, not even the hash of the password is transmitted directly; a function in which the password hash is one of the arguments is transmitted instead. Thus, for a suitable function *f*, the password hash cannot be captured during transmission. Finally, the use of a random number as one of the arguments of *f* defends against a replay attack, in which an adversary captures the user's transmission and attempts to log on to a system by retransmitting the user's messages.

Token Protocol

Figure 3.12b provides a simple example of a token protocol for authentication. As before, a user first transmits their identity to the remote host. The host returns a random number and the identifiers of functions f( )and *h*( ) to be used in the response. At the user end, the token provides a passcode ***W***. The token either stores a static passcode or generates a one-time random passcode. For a one-time random passcode, the token must be synchronized in some fashion with the host. In either case, the user activates the passcode by entering a password P'. This password is shared only between the user and the token and does not involve the remote host. The token responds to the host with the quantity $f(r', h(W'))$. For a static passcode, the host stores the hashed value *h* (W(U)); for a dynamic passcode, the host generates a one-time passcode (synchronized to that generated by the token) and takes its hash. Authentication then proceeds in the same fashion as for the password protocol.

Static Biometric Protocol

Figure 3.12c is an example of a user authentication protocol using a static biometric. As before, the user transmits an ID to the host, which responds with a random number *r* and, in this case, the identifier for an encryption *E*( ). On the user side is a client system that controls a biometric device. The system generates a biometric template BT′ from the user's biometric B' and returns the ciphertext E(r′, *D′*, *BT′*) , where *D′* identifies this particular biometric device. The host decrypts the incoming message to recover the three transmitted parameters and compares these to locally stored values. For a match, the host must find r′ = r. Also, the matching score between *BT′* and the stored template must exceed a predefined threshold. Finally, the host provides a simple authentication of the biometric capture device by comparing the incoming device ID to a list of registered devices at the host database.

Dynamic Biometric Protocol

Figure 3.12d is an example of a user authentication protocol using a dynamic biometric. The principal difference from the case of a stable biometric is that the host provides a random sequence as well as a random number as a challenge. The sequence challenge is a sequence of numbers, characters, or words. The human user at the client end must then vocalize (speaker verification), type (keyboard dynamics verification), or write (handwriting verification) the sequence to generate a biometric signal BS′(x′). The client side encrypts the biometric signal and the random number. At the host side, the incoming message is decrypted. The incoming random number r′ must be an exact match to the random number that was originally used as a challenge *(r)*. In addition, the host generates a comparison based on the incoming biometric signal BS′(x′), the stored template *BT(U)* for this user, and the original signal *x*. If the comparison value exceeds a predefined threshold, the user is authenticated.

3.6 Security Issues for User Authentication

As with any security service, user authentication, particularly remote user authentication, is subject to a variety of attacks. Table 3.4, from [OGOR03], summarizes the principal attacks on user authentication, broken down by type of authenticator. Much of the table is self-explanatory. In this section, we expand on some of the table's entries.

Table 3.4 **Some Potential Attacks, Susceptible Authenticators, and Typical Defenses**

Attacks	**Authenticators**	**Examples**	**Typical Defenses**
Client attack	Password	Guessing, exhaustive search	Large entropy; limited attempts
	Token	Exhaustive search	Large entropy; limited attempts; theft of object requires presence
	Biometric	False match	Large entropy; limited attempts
Host attack	Password	Plaintext theft, dictionary/exhaustive search	Hashing; large entropy; protection of password database
	Token	Passcode theft	Same as password; 1-time passcode
	Biometric	Template theft	Capture device authentication; challenge response
Eavesdropping, theft, and copying	Password	"Shoulder surfing"	User diligence to keep secret; administrator diligence to quickly revoke compromised passwords; multifactor authentication
	Token	Theft, counterfeiting hardware	Multifactor authentication; tamper resistant/evident token
	Biometric	Copying (spoofing) biometric	Copy detection at capture device and capture device authentication
	Password	Replay stolen password response	Challenge-response protocol

Replay	Token	Replay stolen passcode response	Challenge-response protocol; 1-time passcode
	Biometric	Replay stolen biometric template response	Copy detection at capture device and capture device authentication via challenge-response protocol
Trojan horse	Password, token, biometric	Installation of rogue client or capture device	Authentication of client or capture device within trusted security perimeter
Denial horse	Password, token, biometric	Lockout by multiple failed authentications	Multifactor with token

Client attacks are those in which an adversary attempts to achieve user authentication without access to the remote host or to the intervening communications path. The adversary attempts to masquerade as a legitimate user. For a password-based system, the adversary may attempt to guess the likely user password. Multiple guesses may be made. At the extreme, the adversary sequences through all possible passwords in an exhaustive attempt to succeed. One way to thwart such an attack is to select a password that is both lengthy and unpredictable. In effect, such a password has large entropy; that is, many bits are required to represent the password. Another countermeasure is to limit the number of attempts that can be made in a given time period from a given source.

A token can generate a high-entropy passcode from a low-entropy PIN or password, thwarting exhaustive searches. The adversary may be able to guess or acquire the PIN or password but must additionally acquire the physical token to succeed.

Host attacks are directed at the user file at the host where passwords, token passcodes, or biometric templates are stored. Section 3.2 discusses the security considerations with respect to passwords. For tokens, there is the additional defense of using one-time passcodes so that the passcodes are not stored in a host passcode file. Biometric features of a user are difficult to secure because they are physical features of the user. For a static feature, biometric device authentication adds a measure of protection. For a dynamic feature, a challenge-response protocol enhances security.

Eavesdropping in the context of passwords refers to an adversary's attempt to learn the password by observing the user, finding a written copy of the password, or some similar attack that involves the physical proximity of user and adversary. Another form of eavesdropping is keystroke logging (keylogging), in which malicious hardware or software is installed so that the attacker can capture the user's keystrokes for later analysis. A system that relies on multiple factors (e.g., password plus token or password plus biometric) is resistant to this type of attack. For a token, an analogous threat is **theft** of the token or physical copying of the token. Again, a multifactor protocol resists this type of attack better than a pure token protocol. The analogous threat for a biometric protocol is **copying** or imitating the biometric parameter so as to generate the desired template. Dynamic biometrics are l

Replay attacks involve an adversary repeating a previously captured user response. The most common countermeasure to such attacks is the challenge-response protocol.

In a **Trojan horse** attack, an application or physical device masquerades as an authentic application or device for the purpose of capturing a user password, passcode, or biometric. The adversary can then use the captured information to masquerade as a legitimate user. A simple example of this is a rogue bank machine used to capture user ID/password combinations.

A **denial-of-service** attack attempts to disable a user authentication service by flooding the service with numerous authentication attempts. A more selective attack denies service to a specific user by attempting logon until the threshold that causes lockout to this user because of too many logon attempts is reached. A multifactor authentication protocol that includes a token thwarts this attack because the adversary must first acquire the token.

3.7 Practical Application: An Iris Biometric System

As an example of a biometric user authentication system, we look at an iris biometric system that was developed for use by the United Arab Emirates (UAE) at border control points [DAUG04, TIRO05, NBSP08]. The UAE relies heavily on an outside workforce and has increasingly become a tourist attraction. Accordingly, relative to its size, the UAE has a very substantial volume of incoming visitors. On a typical day, more than 6,500 passengers enter the UAE via seven international airports, three land ports, and seven sea ports. Handling a large volume of incoming visitors in an efficient and timely manner thus poses a significant security challenge. Of particular concern to the UAE are attempts by expelled persons to re-enter the country. Traditional means of preventing reentry involve identifying individuals by name, date of birth, and other text-based data. The risk is that this information can be changed after expulsion. An individual can arrive with a different passport with a different nationality and changes to other identifying information.

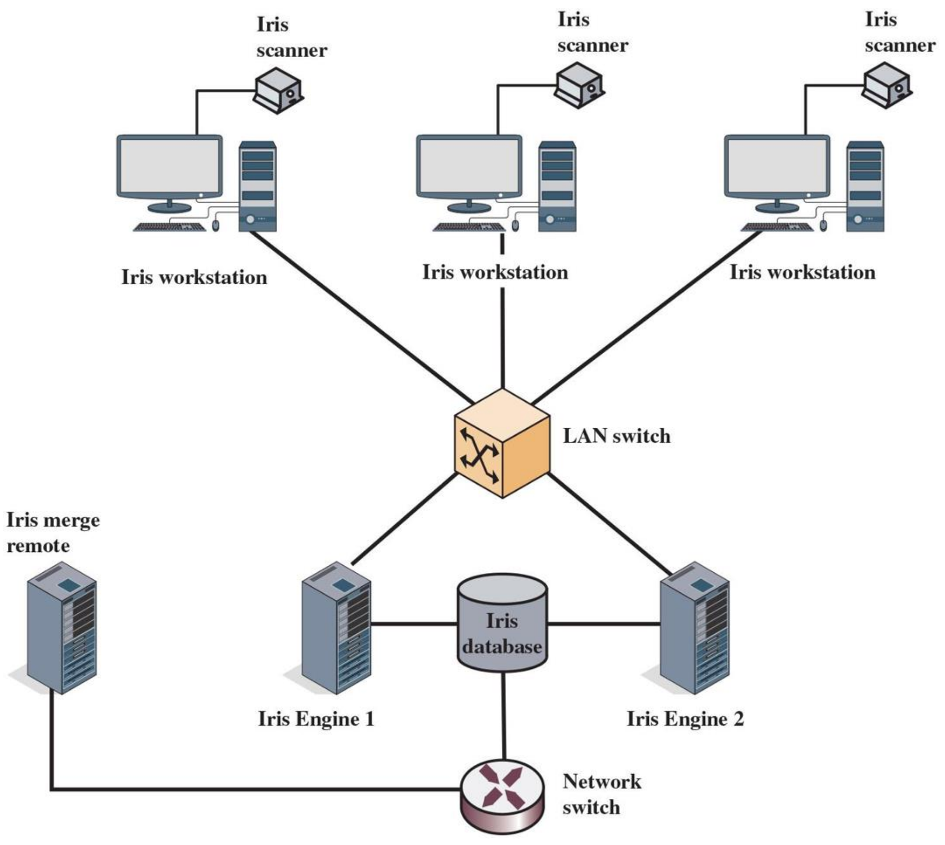
To counter such attempts, the UAE decided on using a biometric identification system and identified the following requirements:

- Identify a single person from a large population of people
- Rely on a biometric feature that does not change over time
- Use biometric features that can be acquired quickly
- Be easy to use
- Respond in real-time for mass transit applications
- Be safe and non-invasive
- Scale into the billions of comparisons and maintain top performance
- Be affordable

Iris recognition was chosen as the most efficient and foolproof method. No two irises are alike. There is no correlation between the iris patterns of even identical twins or the right and left eye of an individual.

System implementation involves enrollment and identity checking. All expelled foreigners are subjected to an iris scan at one of the multiple enrollment centers. This information is merged into one central database. Iris scanners are installed at all 17 air, land, and sea ports into the UAE. An iris-recognition camera takes a black-and-white picture 5 to 24 inches from the eye, depending on the camera. The camera uses non-invasive, near-infrared illumination that is similar to a TV remote control, barely visible and considered extremely safe. The picture first is processed by software that localizes the inner and outer boundaries of the iris and the eyelid contours in order to extract just the iris portion. The software creates a so-called phase code for the texture of the iris, similar to a DNA sequence code. The unique features of the iris are captured by this code and can be compared against a large database of scanned irises to make a match. Over a distributed network (see Figure 3.13) the iris codes of all arriving passengers are compared in real time exhaustively against an enrolled central database.

Figure 3.13 **General Iris Scan Site Architecture for UAE System**



Note that this is computationally a more demanding task than verifying an identity. In this case, the iris pattern of each incoming passenger is compared against the entire database of known patterns to determine if there is a match. Given the current volume of traffic and size of the database, the daily number of iris cross-comparisons is well over 9 billion.

As with any security system, adversaries are always looking for countermeasures. Expatriates who were banned from the UAE started using eye drops in an effort to fool the government's iris recognition system when they tried to re-enter the country. Therefore, UAE officials had to adopt new security methods to detect if an iris had been dilated with eye drops before scanning. A new algorithm and computerized step-by-step procedure has been adopted to help officials determine if an iris is in normal condition or if an eye-dilating drop has been used.

3.8 Case Study: Security Problems for ATM Systems

Redspin, Inc., an independent auditor, released a report describing a security vulnerability in ATM (automated teller machine) usage that affected a number of small to mid-size ATM card issuers. This vulnerability provides a useful case study illustrating that cryptographic functions and services alone do not guarantee security; they must be properly implemented as part of a system.

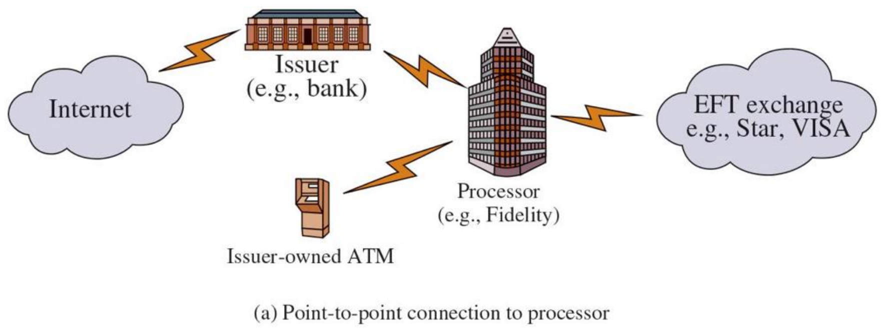
We begin by defining the terms used in this section as follows:

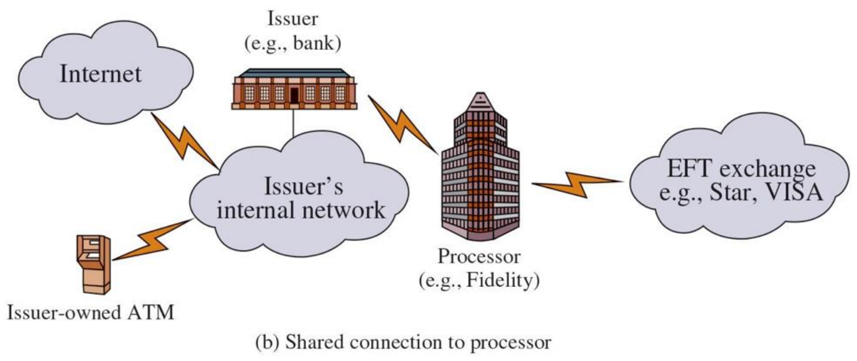
- **Cardholder:** An individual to whom a debit card is issued. Typically, this individual is also responsible for payment of all charges made to that card.
- **Issuer:** An institution that issues debit cards to cardholders. This institution is responsible for the cardholder's account and authorizes all transactions. Banks and credit unions are typical issuers.
- **Processor:** An organization that provides services such as core data processing (PIN recognition and account updating), electronic funds transfer (EFT), and so on to issuers. EFT allows an issuer to access regional and national networks that connect point of sale (POS) devices and ATMs worldwide. Examples of processing companies include Fidelity National Financial and Jack Henry & Associates.

Customers expect 24/7 service at ATM stations. For many small to mid-sized issuers, it is more cost-effective for contract processors to provide the required data processing and EFT/ATM services. Each service typically requires a dedicated data connection between the issuer and the processor, using a leased line or a virtual leased line.

Prior to about 2003, the typical configuration involving issuer, processor, and ATM machines could be characterized by Figure 3.14a. The ATM units linked directly to the processor rather than to the issuer that owned the ATM, via leased or virtual leased line. The use of a dedicated link made it difficult to maliciously intercept transferred data. To add to the security, the PIN portion of messages transmitted from ATM to processor was encrypted using DES (Data Encryption Standard). Processors have connections to EFT (electronic funds transfer) exchange networks to allow cardholders access to accounts from any ATM. With the configuration of Figure 3.14a, a transaction proceeds as follows. A user swipes their card and enters their PIN. The ATM encrypts the PIN and transmits it to the processor as part of an authorization request. The processor updates the customer's information and sends a reply.

Figure 3.14 **ATM Architectures**





Most small to mid-sized issuers of debit cards contract processors to provide core data processing and electronic funds transfer (EFT) services. The bank's ATM machine may link directly to the processor or to the bank.

In the early 2000s, banks worldwide began the process of migrating from an older generation of ATMs using IBM's OS/2 operating system to new systems running Windows. The mass migration to Windows has been spurred by a number of factors, including IBM's decision to stop supporting OS/2 by 2006, market pressure from creditors such as MasterCard International and Visa International to introduce stronger Triple DES, and pressure from U.S. regulators to introduce new features for disabled users. Many banks, such as those audited by Redspin, included a number of other enhancements at the same time as the introduction of Windows and triple DES, especially the use of TCP/IP as a network transport.

Because issuers typically run their own Internet-connected local area networks (LANs) and intranets using TCP/IP, it was attractive to connect ATMs to these issuer networks and maintain only a single dedicated line to the processor, leading to the configuration illustrated in Figure 3.14b. This configuration saves the issuer expensive monthly circuit fees and enables easier management of ATMs by the issuer. In this configuration, the information sent from the ATM to the processor traverses the issuer's network before being sent to the processor. It is during this time on the issuer's network that the customer information is vulnerable.

The security problem was that with the upgrade to a new ATM OS and a new communications configuration, the only security enhancement was the use of triple DES rather than DES to encrypt the PIN. The rest of the information in the ATM request message was sent in the clear. This included the card number, expiration date, account balances, and withdrawal amounts. A hacker tapping into the bank's network, either from an internal location or from across the Internet, potentially would have complete access to every single ATM transaction.

The situation just described leads to two principal vulnerabilities:

- **Confidentiality:** The card number, expiration date, and account balance can be used for online purchases or to create a duplicate card for signature-based transactions.
- **Integrity:** There is no protection to prevent an attacker from injecting or altering data in transit. If an adversary is able to capture messages en route, the adversary can masquerade as either the processor or the ATM. Acting as the processor, the adversary may be able to direct the ATM to dispense money without the processor ever knowing that a transaction has occurred. If an adversary captures a user’s account information and encrypted PIN, the account is compromised until the ATM encryption key is changed, enabling the adversary to modify account balances or effect transfers.

Redspin recommended a number of measures that banks can take to counter these threats. Short-term fixes include segmenting ATM traffic from the rest of the network either by implementing strict firewall rule sets or by physically dividing the networks altogether. An additional short-term fix is to implement network-level encryption between routers that the ATM traffic traverses.

Long-term fixes involve changes in the application-level software. Protecting confidentiality requires encrypting all customer-related information that traverses the network. Ensuring data integrity requires better machine-to-machine authentication between the ATM and processor and the use of challenge-response protocols to counter replay attacks.

3.9 Key Terms, Review Questions, and Problems

Key Terms

1. **applicant**
2. **biometric**

3. **challenge-response protocol**
4. **claimant**
5. **credential**
6. **credential service provider (CSP)**
7. **dynamic biometric**
8. **enroll**
9. **hashed password**
10. **identification**
11. **memory card**
12. **multifactor authentication (MFA)**
13. **nonce**
14. **one-time password (OTP)**
15. **password**
16. **rainbow table**
17. **registration authority (RA)**
18. **relying party (RP)**
19. **salt**
20. **shadow password file**
21. **smart card**
22. **static biometric**
23. **subscriber**
24. **token**
25. **user authentication**
26. **verification**
27. **verifier**

Review Questions

1. **3.1** In general terms, what are four means of authenticating a user's identity?
2. **3.2** List and briefly describe the principal threats to the secrecy of passwords.
3. **3.3** What are two common techniques used to protect a password file?

4. **3.4** List and briefly describe four common techniques for selecting or assigning passwords.
5. **3.5** Explain the different ways a simple memory card, a smart card, a hardware token, and a mobile device can be used for authentication.
6. **3.6** List and briefly describe the principal physical characteristics used for biometric identification.
7. **3.7** In the context of biometric user authentication, explain the terms, enrollment, verification, and identification.
8. **3.8** Define the terms *false match rate* and *false nonmatch rate*, and explain the use of a threshold in relationship to these two rates.
9. **3.9** Describe the general concept of a challenge-response protocol.

Problems

1. **3.1** Explain the suitability or unsuitability of the following passwords:
    a. **a.** YK 334
    b. **b.** mfmitm (for "my favorite movie is tender mercies")
    c. **c.** Natalie1
    d. **d.** Washington
    e. **e.** Aristotle
    f. **f.** tv9stove

g. **g.** 12345678
h. **h.** dribgib

2. **3.2** An early attempt to force users to use less predictable passwords involved computer-supplied passwords. The passwords were eight characters long and were taken from the character set consisting of lowercase letters and digits. They were generated by a pseudorandom number generator with 2^{15} possible starting values. Using the technology of the time, the time required to search through all character strings of length 8 from a 36-character alphabet was 112 years. Unfortunately, this is not a true reflection of the actual security of the system. Explain the problem.
3. **3.3** Assume passwords are selected from 4-character combinations of 26 alphabetic characters. Assume an adversary is able to attempt passwords at a rate of one per second.

   a. Assuming no feedback to the adversary until each attempt has been completed, what is the expected time to discover the correct password?

   b. Assuming feedback to the adversary that flags an error as each incorrect character is entered, what is the expected time to discover the correct password?

4. • **3.4** Assume source elements of length *k* are mapped in some uniform fashion into a target elements of length *p*. If each digit can take on one of the *r* values, then the number of source elements is r^k and the number of target elements is the smaller number r^p. A particular source element x_i is mapped to a particular target element y_j.
   a. What is the probability that the correct source element can be selected by an adversary on one try?
   b. What is the probability that a different source element x_k ($x_i \neq x_k$)that results in the same target element, *yj,* could be produced by an adversary?
   c. What is the probability that the correct target element can be produced by an adversary in one try?
5. **3.5** A phonetic password generator picks two segments randomly for each six-letter password. The form of each segment is CVC (consonant, vowel, consonant),

   where V = <a, e, I, o, u> and $C = \overline{V}$.

   a. What is the total password population?
   b. What is the probability of an adversary guessing a password correctly?
6. **3.6** Assume passwords are limited to the use of the 95 printable ASCII characters and that all passwords are 10 characters in length. Assume a password cracker with an encryption rate of 6.4 million encryptions per second. How long will it take to test exhaustively all possible passwords on a UNIX system?
7. **3.7** Because of the known risks of the UNIX password system, the SunOS-4.0 documentation recommends that the password file be removed and replaced with a publicly readable file called /etc/publickey. An entry in the file for user A consists of a user's identifier ID_A, the user's public key PU_a, and the corresponding private key PR_a. This private key is encrypted using DES with a key derived from the user's login password P_a. When A logs in, the system decrypts $E(P_a, PR_a)$ to obtain PR_a.
8. **3.8** The inclusion of the salt in the UNIX password scheme increases the difficulty of guessing by a factor of 4096. But the salt is stored in plaintext in the same entry as the corresponding ciphertext password. Therefore, those two characters are known to the attacker and need not be guessed. Why is it asserted that the salt increases security?

9. **3.9** Assuming you have successfully answered the preceding problem and understand the significance of the salt, here is another question. Wouldn’t it be possible to thwart completely all password crackers by dramatically increasing the salt size to, say, 24 or 48 bits?
10. **3.10** For the biometric authentication protocols illustrated in Figure 3.12, note that the biometric capture device is authenticated in the case of a static biometric but not authenticated for a dynamic biometric. Explain why authentication is useful in the case of a stable biometric but not needed in the case of a dynamic biometric.
11. **3.11** A relatively new authentication proposal is the Secure Quick Reliable Login (SQRL) described here: https://www.grc.com/sqrl/sqrl.htm. Write a brief summary of how SQRL works and indicate how it fits into the categories of types of user authentication listed in this chapter.

Chapter 4 Access Control

1. **4.1 Access Control Principles**
    1. Access Control Context
    2. Access Control Policies
2. **4.2 Subjects, Objects, and Access Rights**
3. **4.3 Discretionary Access Control**
    1. An Access Control Model

    2. Protection Domains
4. **4.4 Example: Unix File Access Control**
    1. Traditional UNIX File Access Control
    2. Access Control Lists in UNIX
5. **4.5 Mandatory Access Control**
    1. Bell-LaPadula (BLP) Model
6. **4.6 Role-Based Access Control**
    1. RBAC Reference Models
7. **4.7 Attribute-Based Access Control**
    1. Attributes
    2. ABAC Logical Architecture
    3. ABAC Policies
8. **4.8 Identity, Credential, and Access Management**
    1. Identity Management
    2. Credential Management
    3. Access Management
    4. Identity Federation
9. **4.9 Trust Frameworks**
    1. Traditional Identity Exchange Approach
    2. Open Identity Trust Framework
10. **4.10 Case Study: RBAC System for a Bank**
11. **4.11 Key Terms, Review Questions, and Problems**

Learning Objectives

After studying this chapter, you should be able to:

- Explain how access control fits into the broader context that includes authentication, authorization, and audit.
- Define four major categories of access control policies.
- Distinguish among subjects, objects, and access rights.
- Describe the UNIX file access control model.
- Discuss the principal concepts of role-based access control.
- Discuss the principal concepts of mandatory access control.
- Summarize the RBAC model.
- Discuss the principal concepts of attribute-based access control.
- Explain the identity, credential, and access management model.

- Understand the concept of identity federation and its relationship to a trust framework.

Two definitions of **access control** are useful in understanding its scope.

1. NISTIR 7298 (*Glossary of Key Information Security Terms*, July 2019) defines access control as the process of granting or denying specific requests to (1) obtain and use

information and related information processing services and (2) enter specific physical facilities.
2. RFC 4949, *Internet Security Glossary,* defines access control as a process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy.

We can view access control as a central element of computer security. The principal objectives of computer security are to prevent unauthorized users from gaining access to resources, to prevent legitimate users from accessing resources in an unauthorized manner, and to enable legitimate users to access resources in an authorized manner. The Open Web Application Security Project's 2021 report [OWAS21] on the 10 most critical Web application security risks listed broken access control in first place, indicating that much greater care is needed in its correct implementation. Table 4.1, from NIST SP 800-171 (*Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organizations*, February 2020), provides a useful list of security requirements for access control ser

Basic Security Requirements	
1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).
2	Limit system access to the types of transactions and functions that authorized users are permitted to execute.
Derived Security Requirements	
3	Control the flow of CUI in accordance with approved authorizations.
4	Separate the duties of individuals to reduce the risk of malevolent activity without collusion.
5	Employ the principle of least privilege, including for specific security functions and privileged accounts.
6	Use non-privileged accounts or roles when accessing nonsecurity functions.
7	Prevent non-privileged users from executing privileged functions and capture the execution of such functions in audit logs.
8	Limit unsuccessful logon attempts.
9	Provide privacy and security notices consistent with applicable CUI rules.
10	Use session lock with pattern-hiding displays to prevent access and viewing of data after period of inactivity.
11	Terminate (automatically) a user session after a defined condition.
12	Monitor and control remote access sessions.
13	Employ cryptographic mechanisms to protect the confidentiality of remote access sessions.
14	Route remote access via managed access control points.
15	Authorize remote execution of privileged commands and remote access to security-relevant information.
16	Authorize wireless access prior to allowing such connections.
17	Protect wireless access using authentication and encryption.
18	Control connection o

CUI = controlled unclassified information