31 Network Security, Threats, Authentication, Authorization, and Securing Devices

WENBIN LUO

INTRODUCTION

Nowadays, almost all the information is stored electronically in computers, network servers, mobile devices, or other storage media. We cannot protect electronic information in a cabinet in the same way we protected physical documents in the past. We have to come up with new techniques to protect the information from unauthorized access, use, or manipulation. The easiest way to protect electronic data is to encrypt it so that people cannot figure out what it contains without knowing the secret keys. Over the last several decades, two major types of encryption techniques have been invented: private key encryption and public key encryption.

Private key encryption, which is also called symmetric encryption, scrambles the original data with a secret key. In order to unscramble a message or file, people need to have *the same* secret key. Otherwise, there is no way to recover the original data. Public key encryption, also called asymmetric encryption, usually scrambles the original data with a publically accessible key. In order to unscramble a message or file, people have to be able to supply *a different* private key, which is paired mathematically with the public key used in the encryption phase. Some special software tools are needed to generate the pair of public and private keys. We will demonstrate how to create your own public and private keys later, using a free open source tool called GNU Privacy Guard (GnuPG or GPG).

Cryptography is a good tool for security, and it is the basis for many security mechanisms. However, it should be mentioned that it is not the solution to all security problems. Building a reliable and highly secure system is a multifaceted and very complicated process, which requires us to look at various potential attacks from different angles. A system's actual security status is not the same as the technology employed to secure it. For example, passwords, which are the foundation on which much of computer security is built, pose one of the biggest practical problems [1]. Nowadays, people visit many different websites and are required to set passwords for many of them. In order to make things easier, many people use the same password for various accounts. In such cases, if their passwords are harvested by an attacker by intercepting the network traffic, keyboard logging, or simply guessing, then the attacker can get into all their other accounts that share the same password. Also, if an insider in one system compromises their password, their accounts in other systems are also compromised.

In this chapter, first, we will review private key encryption and public key encryption techniques. Then, we will give examples of how to perform private key encryption, public key encryption, and digital signing using GPG. Second, we will discuss some of the threats, which a computer system or user may face, and how they work. Third, we will examine some existing authentication techniques and take a look at various authorization methods, especially those used on a Linux system. Finally, the last topic will be on how to secure devices.

PRIVATE KEY ENCRYPTION

Private key encryption was invented earlier than public key encryption, and it is the standard form of encryption used nowadays. The purpose of private key encryption is to keep the data secret. Its advantage includes efficient algorithms for encryption and decryption. However, the limitation is that both senders and receivers must share the secret keys. The difficulty in sharing secret keys is how to safely transmit them before one can even start encrypting messages. Many security systems or mechanisms use public key encryption to transmit secret keys, and then use private key encryption to encrypt the actual data. The reason for that is to combine the beauty of public key encryption with the speed of private key encryption.

Figure 31.1 shows the building block of private key encryption. The meanings of symbols in this figure are as follows.

- E: encryption unit
- D: decryption unit
- M: plaintext, that is, the original message
- C: ciphertext, that is, the encrypted message
- K: the secret key shared between say an operator and a manager

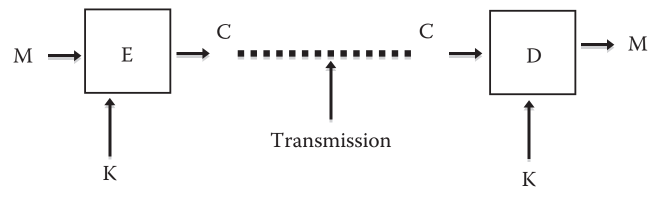

FIG. 31.1
The building block of private key encryption.

It should be noted that the encryption and decryption algorithms (E and D) are publicly known. A simple example of encryption and decryption is as follows. Assume that we have a one-byte secret key (K) "?" whose ASCII code in hexadecimal is "3f." The message (M) we would like to encrypt is a character "%," whose ASCII code in hexadecimal is "25." Assume that we use the following simple encryption algorithm (E): just do bit-wise exclusive-or operation of M and K as follows.

K:	3f=0 0 1 1 1 1 1 1
M:	25=0 0 1 0 0 1 0 1
C:	1a=0 0 0 1 1 0 1 0

The decryption algorithm (D) is similar to the encryption algorithm: just do bit-wise exclusive-or operation of C and K as follows.

K:	3f=0 0 1 1 1 1 1 1
C:	1a=0 0 0 1 1 0 1 0
M:	25=0 0 1 0 0 1 0 1

It is clear that the original message, M, is recovered. This simple scheme can be extended to encrypt and decrypt long messages or data with long keys. If the length of the key is the same as that of the message and each key is used once and only once during encryption, then the above system is called one-time pad, which is very secure. However, it should be mentioned that the above method is not secure and should not be used in practice when a single key is used repeatedly. A simple cryptanalysis could either recover the message, M, or reveal enough information about M from the C, without even knowing the secret key, K. This is especially true when M consists of human audio conversations because of the abundant redundancy in natural languages.

Stream Ciphers

Generally speaking, a private key cipher can be classified as either a stream cipher or a block cipher. Figure 31.2 shows how a stream cipher works. It takes as inputs a key (K) and a message (M). The key is usually of fixed size such as 128 or 256 bits and the message can be of arbitrary size. Given a fixed-size key, a stream cipher will need to generate a pseudorandom key (PK) of the same size as that of M, and then do bit-wise exclusive-or operation of M and PK to get the C.

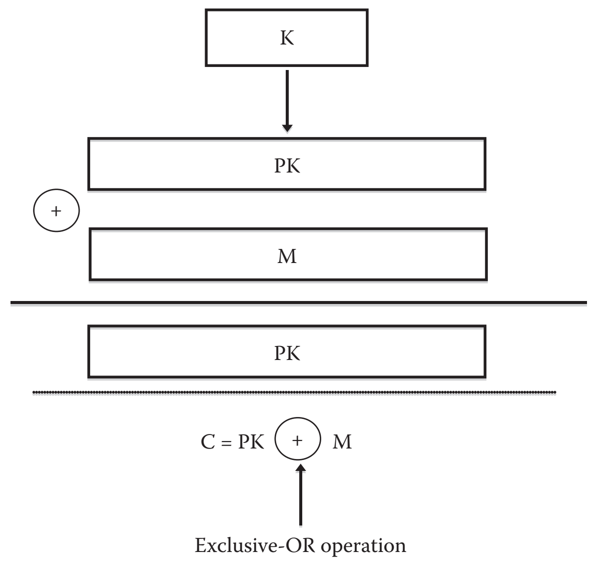

FIG. 31.2
A stream cipher.

The PK generation algorithm is the most important part of a stream cipher. The diagram of a stream cipher is shown in Figure 31.2.

To recover the original message from the C, the stream cipher needs to take the same key K from the user and generate the same PK based on K. After that, it applies bit-wise exclusive-or operation to C and PK to uncover the message M. RC4 is a widely used stream cipher, which was designed by Ron Rivest of RSA Security in 1987. It is used in popular protocols such as secure sockets layer (SSL) to protect Internet traffic and wired equivalent privacy (WEP) to secure wireless networks.

Block Ciphers

Block ciphers work differently from stream ciphers. Instead of generating a PK of the same size as a message, block ciphers divide the message into blocks of the same size, which is usually 64, 128, or 256 bits. Then it encrypts each individual block with a user-supplied secret key (K) of fixed size, such as 56, 128, 168, 192, or 256 bits. Figure 31.3 shows the diagram of a block cipher.

Similar to encryption, a block cipher starts decryption by dividing the C into blocks of the same size first. After that, it decrypts each individual block of C with the secret key. The core parts of a block cipher are encryption (E) and decryption (D) algorithms, which usually involve some mixing operations several times. Data encryption standard (DES) and its variant triple DES, and advance encryption standard (AES) are two of the most widely used block ciphers. DES uses substitution-box (s-box) to perform some mixing operations, whereas AES relies on mathematical operations in finite fields to achieve similar goals [2]. It should be mentioned that block ciphers could work in different modes,

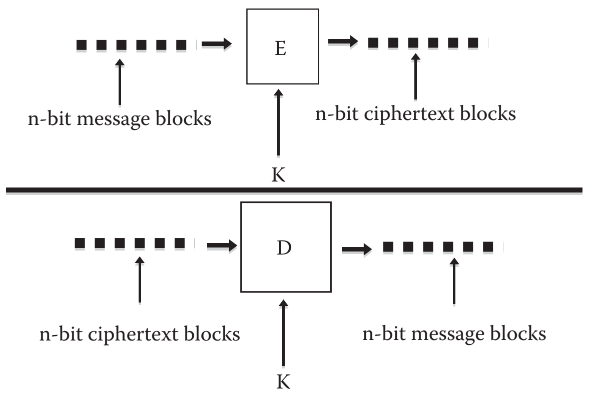

FIG. 31.3
A block cipher.

such as the electronic codebook (ECB) and the cipher-block chaining (CBC), in order to allow block ciphers to provide confidentiality or message integrity.

PUBLIC KEY ENCRYPTION

Public key encryption was invented in 1970s. Its encryption provides confidentiality. In addition, it can be used to create digital signatures, which provide data integrity. Unlike private key encryption, there are less efficient algorithms for public key encryption and decryption. However, the advantage of public key encryption is that it does not require secret keys shared in advance.

Public key cryptosystems use trapdoor functions to encrypt and decrypt. A trapdoor function is a function that is relatively easy to compute its value in one direction and very difficult to find its inverse without knowing some special secret information, called the "trapdoor." Trapdoor functions are widely used in cryptography. One of the best-known trapdoor functions is RSA, which was created as an algorithm for public key cryptography in 1978. RSA are the surname initials of its three inventors Ron Rivest, Adi Shamir, and Leonard Adleman at Massachusetts Institute of Technology (MIT). The security of RSA depends on the time complexity of factoring large integers. In other words, there are currently no efficient ways (with respect to time) to factor large integers on modern computers. The high-level overview of a public key cryptosystem, including RSA, is shown in Figure 31.4.

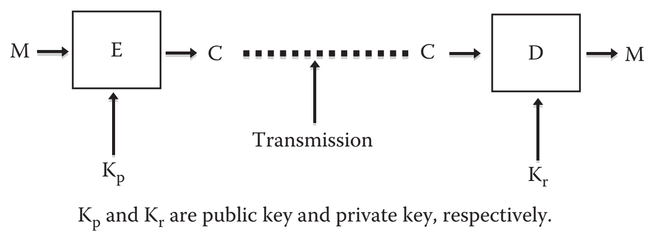
M
E
C
C
D
M
Transmission
K_p
K_r
K_p and K_r are public key and private key, respectively.

FIG. 31.4
A public key cryptosystem.

The mathematical proof of why RSA works can be found in many textbooks such as Stallings 2003 [2]. A public key cryptosystem can be used in two different ways: data encryption and digital signature. Assume that two users mentioned above, an operator and a manager, want to communicate with each other. Further, assume that each user has a pair of public and private keys (K_{pa}, K_{ra}) and (K_{pb}, K_{rb}), respectively. We will demonstrate how to generate a key pair later.

Data Encryption

Assume that the manager wants to encrypt an M and send the encrypted message, that is, C, to the operator. First, she needs to get the operator's public key K_{pb}, and then encrypt the message as follows:

$C = E(K_{pb}, M)$; where E is the encryption algorithm of the public key cryptosystem.

Once the encrypted C is sent it is decrypted using private key K_{rb} as follows:

$M = D(K_{rb}, C)$; where D is the decryption algorithm of the public key cryptosystem.

It should be mentioned that a user's public key (K_{pa} or K_{pb}) can and should be made publicly available to receive and read encrypted messages. However, a user's private key (K_{ra} or K_{rb}) should be kept secret. Otherwise, anyone with access to the private key can read his encrypted messages.

RSA versus Elliptic Curve Cryptography

Two mathematicians, Neal Koblitz and Victor Miller, independently discovered elliptic curve cryptography (ECC) in 1985. They found that a well-known structure called "elliptic curve" is suitable for employment in cryptography. The advantage of ECC in comparison to RSA is convincing: *less memory requirement* and *computation time*, which indicates less storage and greater speed. As a result, ECC can be used in smart cards, cellular phones, and pagers. As an example, key lengths of 160 bits as in ECC ensure the security of a RSA key of 1024 bits. The RSA procedure currently changes its key length to 2048 bits. As a result, ECC only has to increase its key length to 192 bits. The security of an ECC key of 512 bits is the same as that of a RSA key of 15,360 bits. U.S. National Security Agency (NSA) adopted ECC for protecting information classified as mission-critical by the U.S. government. It seems very likely that other government agencies and the commercial sectors will follow suit and adopt ECC for strong security across a wide variety of applications and devices.

Public Key Infrastructure

The public key infrastructure (PKI) is used to avoid interception and decryption of messages. Within a PKI, the trusted certificate authority, such as VeriSign, is responsible for signing a user or a server's public key. Of course, for this to work, everyone has to know the certificate authority's

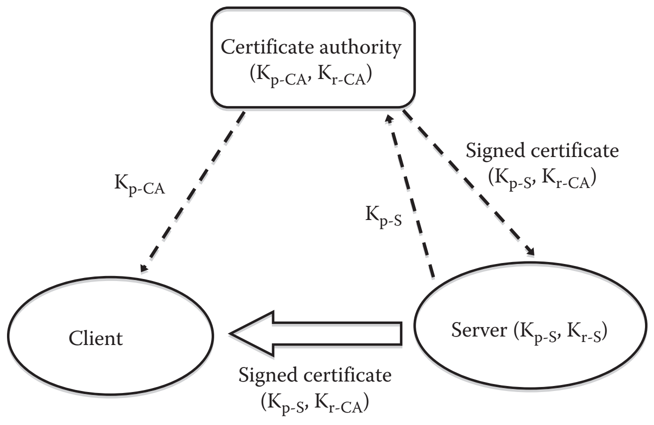
Certificate authority
($K_{p\text{-}CA}$, $K_{r\text{-}CA}$)
$K_{p\text{-}CA}$
$K_{p\text{-}S}$
Signed certificate
($K_{p\text{-}S}$, $K_{r\text{-}CA}$)
Client
Server ($K_{p\text{-}S}$, $K_{r\text{-}S}$)
Signed certificate
($K_{p\text{-}S}$, $K_{r\text{-}CA}$)

FIG. 31.5
Public key infrastructure.

public key in order to verify its signature. If you check your web browser, you will find hundreds of preinstalled certificate authorities' public keys, including VeriSign, Microsoft, Wells Fargo, Thawte, RSA Security, Cybertrust, and so on. Certificate authority can sign certificates, which identify others including other authorities. It leads to certificate chains. A high-level overview of a public key infrastructure is shown in Figure 31.5.

In the above figure, the signed certificate contains a server's public key, $K_{p\text{-}S}$, which is signed by a certificate authority's private key, $K_{r\text{-}CA}$. The authenticity of $K_{p\text{-}S}$ can be verified by any clients with the publicly known key $K_{p\text{-}CA}$. However, $K_{r\text{-}CA}$ cannot be revealed from the signed certificate.

Recent Developments

The security of RSA depends on the fact that factoring large integers is hard on a classical computer. Even though it is impractical for today's computers to crack the two public key cryptography schemes RSA and ECC, a quantum computer, if ever built, would be powerful enough to render both schemes useless. Peter Shor [3] showed that efficient randomized algorithms for factoring integers and finding discrete logarithms, another hard problem on a classical computer, exist on a quantum computer. In other words, a quantum computer could factor large integers in polynomial time. Therefore, the RSA cryptosystem will be broken on a quantum computer. For that reason, cryptographers are searching for new cryptographic techniques to resist quantum computer attacks [4]. Currently, researchers are working in at least the following four directions, which use different mathematical structures:

1. One direction is to use the theory of error-correcting codes. In an error-correcting code-based scheme, errors are added into a message to make it unreadable. Only the intended receiver with the right code could correct the error and recover the original message.
2. Another direction is to utilize the fact that solving randomly generated polynomial systems is supposed to be very hard. The corresponding scheme is called a multivariate public key system.
3. The third direction is to develop a hash-based signature scheme, which uses secure hash functions to compress a large data set into a unique identification number.
4. The last one is to build a lattice-based system. One such example is the NTRUEncrypt public key cryptosystem, which is based on the shortest vector problem in a lattice [5]. Its security relies on the difficulty of factoring certain polynomials in truncated polynomial rings. NTRUEncrypt public key cryptosystem was standardized in 2008 as IEEE Standard 1363.1-2008. Its speed and low memory usage make it very attractive for use in wireless devices and smart cards. More information on the system can be found at its company's website at http://securityinnovation.com/.

With the rapid advance of parallel and distributed computing technology combined with the more and more powerful classical computers being built, we have to keep increasing the key sizes of RSA and ECC in order to avoid brute force attack on both techniques, which might eventually make the application of RSA and ECC too complicated and less efficient, especially on wireless devices and smart cards. At present, NTRUEncrypt cryptosystem performs at least five times faster than some existing public key cryptosystems. In addition, it is believed to be immune to quantum computer attacks. As a result, it is a leading candidate for adoption if quantum computers render RSA and ECC unusable.

THREATS

According to privacyrights.org, the top three ways leading to data breach are hacking, stolen equipment, and lost equipment. Hacking alone causes more than half of all reported data breach cases. Over hundreds of millions of customer records have been lost or stolen in the last few years. Hacking refers to the process of making a system function in ways unexpected by the system owner or designer. The system could be operating systems, software applications, hardware systems, or their combinations. An attacker uses hacking to gain information from computer systems illegally or even make them stop functioning normally. Over the years, many hacking techniques have been reported, which include structured query language (SQL) injection, cross-site scripting, denial of service (DoS), buffer overflows, phishing, pharming, and many more [6,11].

In this section, we will briefly describe three common attacks noting that there exist many other treats, such as viruses, malwares, worms, and botnets. As computing is evolving constantly, the threats are changing too. Nowadays, an increasing number of applications are hosted on the cloud. Mobile devices and smart phones, such as various netbooks, iPhone, and Android-based handsets, are used widely, which changes the way we perform online banking, shopping, and other activities. They now become the new targets of cyber attacks.

SQL Injection

SQL is a database language for managing data in relational database management systems. SQL injection attacks a database system by passing malicious code via syntactically valid SQL statements to an SQL server for parsing and execution. The following line of code illustrates how SQL injection works.

```
SELECT * FROM USERS WHERE uname IS
'$username'
```

The above SQL statement is to pull out all the records whose field `uname` equals to the user input, stored in the variable `$username`, from the database table `USERS`. However, if a malicious user crafts a special string, such as "`'; DROP TABLE USERS; ––`", as user input, then the following SQL statements will be passed to an SQL server.

```
SELECT * FROM USERS WHERE uname IS ''; DROP
TABLE USERS; --'
```

It actually consists of the following two valid SQL statements. The string "`––'`" is an SQL comment, which will be ignored by an SQL server.

```
1. SELECT * FROM USERS WHERE uname IS '';
2. DROP TABLE USERS;
```

The first SQL statement is to pull out all records whose field `uname` is empty, whereas the second SQL statement is to delete the entire table `USERS`. In other words, all records in that table will be eliminated. To prevent SQL injection, it is very important for you to validate and filter all user inputs.

Phishing

Phishing refers to the act of luring people to surrender their private information via fake e-mails or instant messages, which usually direct users to a bogus site to update personal information, such as credit card numbers, passwords, etc. Once the information is entered at the bogus site, it is stolen. Phishing e-mails typically contain some grammatical mistakes or typos. Following is a fake e-mail, claiming from a legitimate bank. It asks users to confirm their account information. Please pay special attention to the URL link in the message, which, if clicked, will actually direct users to another website.

Followings are some commonly used phishing methods.

- Use carefully chosen domain names to confuse users, such as
  http://www.amazon.com.phishingSite.com
- Use a URL that redirects users to a phishing site, such as
  http://www.goodBank.com/url.php? http://www.phishingSite.com
- Use randomized links, such as
  http://www.IAmNotAPhishingSite.com/e83h736k47jd/, which actually points to a phishing site

Pharming

Pharming aims to redirect a legitimate website's URL to a bogus website by changing the hosts file on a victim's computer or by exploiting vulnerability in domain name system (DNS) server software. For example, most Linux systems use */etc/hosts* file to resolve computer names or URLs. An attacker can add the following line to a computer's */etc/hosts* file to redirect a good website to a bogus site as follows.

IPAddressOfPhishingSite http://www.goodBank.com

Next time, when a user using the computer tries to visit http://www.goodBank.com, the browser will be redirected to a phishing site with the IP address IPAddressOfPhishingSite. This type of attack is even more difficult to detect because the URL of a phishing site is the same as that of a legitimate website. It will bypass all URL checks. Similarly, an attacker can hack a DSN server to enable pharming attacks.

AUTHENTICATION

The goal of authentication is to establish who the user is. The basis for authentication can be something you know, such as password and crypto key, or hardware token, or biometrics (fingerprint). In this section, we will discuss different authentication techniques, including password authentication, certificate authentication, tokens, and biometrics.

Password Authentication

The basic idea for password authentication is that each user has a secret password, and a system checks users' passwords to authenticate users. The issues to be addressed here include: (1) How are users' passwords stored? (2) How does a system check users' passwords? (3) How easy is it for attackers to guess users' passwords? Password hacking includes dictionary attacks where users' passwords are compared against the words or their combinations in a dictionary. If a match is found, then basically a user's password is guessed correctly. In real applications, passwords are usually not stored in clear texts to prevent people from seeing the passwords directly. Instead, their hash values are stored in a password file. A hash function takes a password as input and produces a hash value as the output. A good hash function has the property that given an input, it is easy to calculate the output. However, given the output, it is very difficulty (ideally impossible) to recover the corresponding input. Various techniques have been proposed over years to make password guessing more difficult or infeasible.

One such method is called password salting, which stores a random salt along with a user password. A salt is usually a random bit string. For example, a 12 bit random salt slows down password guessing by a factor of 2^{12} because each password can now be combined with one of the 2^{12} possible salts. In order to guess the password, an attacker has to compute and find out the correct salt first.

Another technique is scratch list–based one-time password. It works as follows. A system generates a list, called scratch list, of random passwords, which is delivered to a user via an independent channel such as physical delivery by a trusted person, or the user picks it up by himself. The first time a user logs into the system, he uses the first password in the scratch list. After that, it was crossed over so that the second time he has to use the second password in the list to log into the system. And it continues that way. Since both the system and the user keep the same list, the system can authenticate users correctly.

When a user wants to log into a system remotely, his password has to be sent over a network such as Internet to the remote system. This can be very risky if the password is sent over the network without encryption. Someone can easily intercept the user's password during its transmission, and therefore, can illegally log into the system using it. Several techniques have been proposed to solve the above problem. One method is to encrypt users' passwords before transmission over the network, as used by secure shell. There are many secure shell clients available. Under Linux, Unix, and Mac OS X, a secure shell client is usually installed by default and a user can access it by simply typing the command "*ssh*" to launch it. If not installed, a user can always install a free version of it called *OpenSSH*. On Windows machines, one can use *PuTTY*, which implements the secure shell protocol.

Another technique employed to protect passwords is called challenge–response authentication. When a user initiates a connection to a system, the system presents the user with some string called *challenge*. Upon receiving the challenge, the user computes the output, called *response*, of a certain function with the challenge and the user's secret key as inputs, and then sends the response to the system. When the response arrives at the system, system checks to find out if the response has certain properties. If it does, then the system authenticates the user successfully. Otherwise, the authentication fails. As we can see, during challenge–response authentication, the user's secret password is not sent over the network. Therefore, challenge–response authentication is very suitable for authentication over a network [11].

Certificate Authentication

Certificate authentication involves PKI. A popular example of certificate authentication is the cryptographic protocols used by web browsers: transport layer security (TLS) and its predecessor, SSL. SSL/TLS provides authenticated key exchange, which is used for secure communications over networks. It requires certificates issued by a trusted certificate authority. Financial institutions and online shopping sites use SSL/TLS to secure customers' transactions. All major web browsers have installed hundreds of known certificate authorities' public keys. SSL protocol works in two steps: SSL handshake protocol and SSL record protocol. SSL handshake protocol results in a secret session key shared by a server and a web browser to encrypt all subsequent communications. SSL record protocol provides encryption and integrity for transmitted data.

A brief explanation of the SSL handshake protocol works as follows. A web browser starts the SSL handshake protocol by sending a request to a server. The server picks a secret random session key and encrypts it using the server's private key. Then, the server sends the web browser both the encrypted session key and its certificate, that is, its public key signed by a known certificate authority. When the web browser receives the encrypted session key and the server's certificate, it first verifies the authenticity of the server's certificate using the certificate authority's public key, which is already installed in the web browser. Once the web browser verifies the authenticity of the server's certificate, it then uses the server's public key to decrypt the encrypted session key to get the secret session-key. This way, only the server and the web browser share the session-key. As we can see, the server and the web browser establish the session-key using public key encryption. After that, they encrypt all the data sent between them using private key encryption with the session-key as the secret key. The reason that private key encryption is preferred over public key encryption to secure all subsequent communications is that private key encryption is in general much faster than public key encryption.

Token-Based Authentication

Token-based authentication devices generally work in the following two different ways: one-time password and challenge–response system. A one-time password device produces a new password each time a user wants to log into a system. Since the password changes every time the user logs in, it increases the difficulty for an attacker to guess the password correctly. In order for the device and the system to produce synchronized sequence of passwords, the device and the system need to share some initial data, which has to be set up securely. A challenge–response-based device receives a challenge from the system each time a user wants to log in. It then computes a response based on the challenge received and some initial data stored in the device. Next, the device sends the response back to the system to authenticate the user. When the system receives the response, it checks its validity to decide whether the user should be authenticated or not.

Biometrics-Based Authentication

Biometrics-based authentication uses a person's physical characteristics to authenticate him to a system. The typical physical characteristics include fingerprint, hand, retina, voice, face, written signature, and keyboard timing. The advantages of using physical characteristics are that they cannot be lost or forgotten. However, unlike passwords, biometrics cannot be changed. In addition, biometrics-based authentication depends on reliable algorithms to accurately identify users' physical characteristics. A false-positive, which identifies a wrong person's physical characteristics as real one, will allow unauthorized person to access the system. On the other hand, a false-negative, which fails to identify a legitimate user's physical characteristics as real one, will deny an authorized person to access the system.

AUTHORIZATION

Authorization determines what a user is allowed to do. It is the procedure of specifying consumers' access rights to resources such as electronic data, computer devices, and functionality provided by other applications or devices. Consumers include users, software programs, and various devices. The key part of authorization is to define access policy. On modern multiuser operating systems, such as Linux, authorizations are usually expressed as access policies in the form of file permissions or access control lists. In defining access policies, the principle of least privilege is strongly recommended. That is, consumers should only be authorized to access whatever they need to accomplish their tasks and no more. Before a system checks a consumer's access rights to the resource, it usually uses authentication to verify the consumer's identity. It should be mentioned that another separate procedure, accounting, is often applied to determine what a user actually did.

Basic File Permissions in Linux

In this chapter, Linux system will be used as an example to discuss issues on authorization. In this system, each object is represented as either a regular file or a special file. The following symbols are used to denote different file types in Linux as shown in Table 31.1.

In a Linux system, there are three basic permissions: *read* (*r*), *write* (*w*), and *execute* (*x*).

- *Read (r)*: It allows you to read, view, and print a file.
- *Write (w)*: It allows you to write, edit, and delete a file.
- *Execute (x)*: For a regular file such as a binary program or shell script, it allows you to execute the file. If it is a directory, it allows you to search the directory.

Let us take a look at the following line:

```
-rwxr--r-- 1 wluo staff 1245 2009-12-18 14:08
ABC.txt
```

The first triplet of characters is `rwx`, which indicates that the owner `wluo` of the file "`ABC.txt`" has the *read*, *write*, and *execute* permissions to it. The second triplet "`r––`" indicates that the group `staff` has *read* permission to the file. The "–" indicates that *read* and *execute* permissions are not granted to the group *staff*. The last triplet "`r––`" indicates that everyone else has the same permissions to the file as the group `staff`.

Advanced Permissions in Linux

Besides the three basic permissions *read*, *write*, and *execute*, Linux has the following advanced permissions:

- *SUID (also known as "Set User ID" or "setuid"):* If this permission is set on an executable file, the user who executes that file will have the same permissions as the owner of the file. An example is the *passwd* program in Linux, which is owned by the *root* user and is used by all users to change their user passwords. The SUID permission is set for the *passwd* program so that all users can run the program to change their passwords

TABLE 31.1
File Types in Linux

Symbol	*Meaning*	*Description*
—	Regular file	They contain normal data, such as text files, images, documents, and executable programs
D	Directory	Files that are lists of other files
L	Link	A mechanism to make a file or directory visible at multiple locations of a Linux system's file structure
B	Block device	A mechanism used for input and output. It refers to devices through which the system moves data in the form of blocks
C	Character device	A mechanism used for input and output. It refers to devices through which the system transmits data, one character at a time
S	Socket	A special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control
P	Named pipe	A special file type, which acts more or less like sockets and forms a way for processes to communicate with each other, without using network socket semantics

and write their new passwords into either*/etc/passwd* or */etc/shadow* file. Regular users do not have write permission to either */etc/passwd* or*/etc/shadow* file, which prevents them from unauthorized changes. In order for them to be able to change their passwords, the SUID permission for the *passwd* program has to be set as shown below by the ending *s* in the first triplet "*rws*" on a Ubuntu Linux system.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 29104 2008-12-08
01:14 /usr/bin/passwd
```

- *SGID (also known as "setgid"):* This permission can be applied in two ways. First, if it is set on an executable file, the user who executes that file will get the same permissions as the group owner of the file. Second, if it is applied on a directory, everything created in that directory afterward will get the group owner of the directory as its group owner. SGID permission is often set on a directory, shared by a team working on the same project, so that every team member can create files in that directory and share them among each other. The following commands show how to set a directory's SGID, and then confirm it as shown in the last line by the ending *s* in the second triplet "*rws*" on a Ubuntu Linx system.

```
$ chmod g+s fun
$ ls -ld fun
drwxr-sr-x 2 wluo wluo 4096 2010-01-02
11:30 fun
```

- *Sticky bit:* Sticky bits can be used to prevent files in a public writable directory from being deleted in an undesirable way. For example, Linux systems usually have the sticky bit set on the directory */tmp*, to which all users have write permissions as shown below by the ending *t* in the third triplet "*rwt*" on a Ubuntu Linx system. This way, it can prevent users from deleting each other's files either accidentally or maliciously because in this case only the owner of a file can delete it.

```
$ ls -ld /tmp
drwxrwxrwt 14 root root 4096 2010-01-
02 08:13 /tmp
```

Access Control Lists, Capabilities, and File Attributes in Linux

So far, we discussed the basic permission model on a Linux system. However, the default Linux permission system has a shortcoming, that is, it does not allow more than one entity to be set as user or group owner of a file. Access control lists (ACL) offers a way to overcome this shortcoming. One can add to the access control list multiple users or groups who also have rights to the file. Before the use of ACLs on a file system, option *acl* to the Linux file system configuration file */etc/fstab* must be added for all file systems. After that, server needs to be restarted, or remount all partitions (e.g., *mount -o remount/*) where ACLs have been applied. After ACLs has been enabled for a given file system, the *setfacl* command can be used for setting or *getfacl* command can be used to monitor the permissions that are set for a file. On a Ubuntu or Debian Linux system, the *setfacl* and *getfacl* commands are from the package *acl*, which can be installed by running the command *apt-get install acl*. In the following example on a Ubuntu Linux system, we first show the default ACL of a file. Then we grant an additional user *mail* read (r), write (w), and execute (x) access to the file.

In addition, on a modern Linux system, the kernel capabilities were introduced to provide some granular control to the capabilities of the *root* user, who is the administrator of a Linux system. Kernel capabilities make it possible to allow or disallow particular pieces of the root user's power. To control those capabilities, you can install a package called *lcap* by running the command *apt-get install lcap* on a Ubuntu or Debian Linux system. After the package is installed, one can use the command *lcap* to display or remove the Linux kernel capabilities.

Finally, files on a Linux system can be secured by working with file attributes. Once the owner of a file sets file attributes on it, they will be applied to all users who access the file. On a Linux system, one can use *chattr* and *lsattr* commands to change or display file attributes. There are many attributes available, which can assign files to control default system behaviors. As an example, several available file attributes and their meanings are as follows. To avoid a certain amount of disk I/O for laptop systems, one can set a file's *"A" attribute*. This way when the file is accessed, its access time record will not be modified. One can set a file's *"s" attribute* set so that when the file is deleted, its blocks are zeroed in the storage media to provide secure deletion. To allow future recovery of a file after its deletion, one set a file's *"u" attribute* so that the file's contents are saved automatically when it is deleted.

SECURING DEVICES

Users usually store images and the data in their hard drives without encryption. But it is extremely important to protect data via encryption. Due to the advancement in hardware and software technology, we can do on-the-fly data encryption, that is, everything written to the disk or removable media is automatically encrypted. On modern hardware, the overhead for encryption is negligible and no dedicated hardware is required to make that happen. In this section, we will first review some available tools for full disk encryption. Then, we will present a working example on how to implement disk encryption using a free open source tool in Linux.

Full Disk Encryption Tools

We will compare different full disk encryption tools in terms of cost, supported operating systems (OS), and features, as shown in Table 31.2 [13–29].

TABLE 31.2
Full Disk Encryption Tools

Product	*Supported OS*	*Cost*	*Description*
Windows BitLocker Drive Encryption http://www.microsoft.com/windows/windows-vista/features/bitlocker.aspx	Windows	Included with the Ultimate edition of Windows Vista and Windows 7	It enhances data protection by combining drive encryption with the integrity checking of early boot components. In addition, Windows 7 has a new feature called *BitLocker To Go*, which protects portable storage devices and external hard drives
IronKey https://www.ironkey.com/	Windows, Linux, and Mac OS X	From $79 to $299, depending on the size of the drive (1G–32G)	It claims to be the world's most secure flash drive. It has the following features: military-grade encryption, identity theft protection, secure web browsing, and secure password storage
BitArmor DataControl http://www.bitarmor.com/	Windows	N/A	Only one software installation for file encryption, full disk encryption, and protection for removable media, e-mail attachments, and CD/DVD
TrueCrypt http://www.truecrypt.org/	Windows, Linux, and Mac OS X	Free/open source	It provides pre-boot authentication and plausible deniability, that is, hidden volumes and hidden operating systems, in case an adversary forces you to reveal the password.
DiskCryptor http://diskcryptor.net/index.php/DiskCryptor_en	Windows	Free/open source	It claims to be the only truly free solution without TrueCrypt's limits on the use and modification of its source code. Based on its website, its goal is to become the best product in its category
StorageCrypt http://www.magic2003.net/	Windows	$30	It takes only few seconds to encrypt a 100 GB drive, and it is very easy to use
DriveCrypt Plus Pack http://www.securstar.com/	Windows	$125	It provides preboot authentication and hidden operating systems. In addition, it supports USB-token authentication at pre-boot level
Dekart Private Disk http://www.dekart.com/	Windows	$65	This product has a unique feature called disk firewall, which protects data from illegal copying, viruses, and spyware by controlling what applications can access the encrypted disk. Also, it can be launched directly from removable media, such as USB
FreeOTFE http://www.freeotfe.org/	Windows, with support for encrypted Linux volumes (Cryptoloop "losetup," dm-crypt, and LUKS)	Free/open source	It is easy to use, highly portable, and works on both PCs and PDAs. In addition, it provides Linux compatibility (Cryptoloop "losetup," dm-crypt, and LUKS supported)
PGP Desktop Professional http://www.pgp.com/	Windows and Mac OS X	$239	In addition to full disk encryption, it supports e-mail encryption and IM encryption as well.
McAfee Endpoint Encryption http://www.mcafee.com/us/enterprise/products/data_protection/data_encryption/endpoint_encryption.html	Windows	N/A	A product with full features, including seamless integration with existing infrastructure such as active directory, LDAP, PKI, and others
GuardianEdge Hard Disk Encryption http://www.guardianedge.com	Windows	N/A	It provides native active directory integration, power failure protection, and secure access to encrypted hard drives through a recovery mechanism when passwords are lost
BestCrypt http://www.jetico.com/encryption-bestcrypt-volume-encryption/	Windows	$120	It provides transparent encryption of all the data stored on fixed and removable disk devices and has features such as anti-keylogger, traveller mode, and multiply passwords
CREDANT Mobile Guardian http://www.credant.com/products.html	Windows	N/A	It provides six key layers of protection: user data encryption, application data encryption, external media encryption, common data encryption, system data encryption, and operating system protection

TABLE 31.2 (continued)
Full Disk Encryption Tools

Product	Supported OS	Cost	Description
SafeNet ProtectDrive http://www.safenet-inc.com/Products/Data_Protection/Disk_and_File_Encryption/ProtectDrive.aspx	Windows	N/A	It integrates with Microsoft's active directory, provides single point of management, and allows password-based removable media protection
Sophos Endpoint Security and Data Protection http://www.sophos.com/products/enterprise/endpoint/security-and-control/	Windows	N/A	It secures computers and sensitive data with antivirus, antispyware, firewall, network access control, data loss prevention, and encryption technologies in one solution.
dm-crypt http://www.saout.de/misc/dm-crypt/	Linux	Free/open source	Dm-crypt provides transparent encryption of block devices under Linux kernel 2.6 or later. Dm-crypt encrypted volumes can be accessed from Windows via FreeOTFE

Even though full disk encryption can protect computers and mobile devices very well, it is not the ultimate assurance against data leaks. It is only part of the solution to protect information overall. For example, if a user has an encrypted hard drive, but he needs to send sensitive files via e-mails or copy them onto USB devices. With only full disk encryption, the files attached to e-mails or copied to USB devices will be in unencrypted form. File-based encryption supplements full disk encryption, especially in shared machines or devices. Some full disk encryption tools in Table 31.2 provide this type of protection mechanism. For example, BitArmor DataControl uses *smart tag* technology to protect data as it travels from device to device. It provides persistent file encryption, full disk encryption, protection for USB drives, and encryption for e-mail attachments. Smart tags contain encryption and protection policies and stay with files at rest or in motion. This way, data are consistently protected at all times, which does not require any extra efforts from end users.

An Example: Disk Encryption in Linux

In this section, we will show very briefly how to use a free open source package *dm-crypt* to encrypt a file system on a Ubuntu Linux system. The package *dm-crypt* is a transparent disk encryption subsystem in Linux kernel versions 2.6 and later. It resides entirely in the kernel space and uses some front-end tools such as *cryptsetup* and *cryptmount* to facilitate the creation and activation of encrypted volumes. We will use the *cryptsetup* in this example.

Setting up the system for encryption: To create an encrypted file system in Ubuntu, we first install the userland tool *cryptsetup* to manipulate the functionality provided by *dm_crypt*.

Encrypting a file system: Once the system of encryption is installed and understood, one can start encrypting a file system that reside in disk partitions, or volumes of redundant array of inexpensive disks (RAID), or logical volumes and individual files. The *dm-crypt* can also be used to encrypt the whole disks, including removable media. In addition, *dm-crypt* can be used to encrypt the root file system.

Unmouting an encrypted file system: When a system is powered or no longer needed to access the encrypted file system, it has to be unmounted, which is basically the reverse process of the previous section. First, the file system needs to be unmounted using the *umount* command as.

```
$ sudo umount TopSecret
```

Remouting an encrypted file system: After encrypted file system is unmounted, it needs to remounted it again before the next usage. First, create a loop device by running the following command.

```
$ sudo losetup /dev/loop0 safetyBox
```

It should be mentioned that the all the procedures above could be automated with a Linux shell script to ease the task of setting up, unmounting, and remounting the encrypted device.

CONCLUSIONS AND SUMMARY

Network security studies the theories and tools to protect interconnected computer systems from unauthorized access, use, modification, or destruction. Network security continues to be a growing concern because the number of viruses, intrusions, and other attacks on computer systems and mobile devices increases every year. Without robust security systems, financial transactions, business transactions, as well as the larger infrastructure—the power grid and gas pipelines—would be unable to efficiently function. Personal information that exists in electronic formats is extremely vulnerable. With the wide use of computer networking and wireless communication systems for carrying data between different devices, network security tools are needed to secure the data during its transmission.

To protect data, one should make sure that only trusted software is installed in computers or mobile devices. For secure operations, it is important to check that the sites are SSL/TLS enabled and protected by strong cryptographic techniques. It is equally important to use good passwords and reset questions for your user accounts.

References

1. Anderson, R., *Security Engineering*, 2nd edn., Wiley, Hoboken, NJ, 2008. Free online chapters can be downloaded at http://www.cl.cam.ac.uk/~rja14/book.html (accessed on March 19, 2011).
2. Stallings, W., *Cryptography and Network Security*, 3rd edn., Prentice Hall, Upper Saddle River, NJ, 2003.
3. Shor, P., Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Journal of Computing* 26: 1484–1509, 1997.
4. Heger, M., Cryptographers take on quantum computers, *IEEE Spectrum* 46: 14, 2009.
5. Hoffstein, J., Pipher, J., and Silverman, J. H., *NTRU: A Ring Based Public Key Cryptosystem*, In *Lecture Notes in Computer Science 1423*, J.P. Buhler (ed.), Springer-Verlag, Berlin, Germany, pp. 267–288, 1998.
6. Bonneau, J., Anderson, J., Anderson, R., and Stajano, F., Eight friends are enough: Social graph approximation via public listings, *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, Nuremberg, Germany, April 1–3, 2009, pp. 13–18, 2009.
7. Conti, G., *Googling Security*, Addison-Wesley Professional, Indianapolis, IN, 2008.
8. Daswani, N., Kern, C., and Kesavan, A., *Foundations of Security: What Every Programmer Needs To Know*, Apress, New York, 2007.
9. Dingledine, R., Mathewson, N., and Syverson, P., Tor*:* The second-generation onion Router, *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, August 9–13, 2004, pp. 303–320, 2004.
10. Viega, J. and McGraw, G., *Building Secure Software*, Addison-Wesley Professional, Indianapolis, IN, 2001.
11. Viega, J. and Messier, M., *Secure Programming Cookbook for C and C*++, O'Reilly Media, Sebastopol, CA, 2003.
12. Tanenbaum, A., *Computer Networks*, 4th edn., Prentice Hall, Upper Saddle River, NJ, 2003.
13. Encryption, http://www.microsoft.com/windows/windows-vista/features/bitlocker.aspx (accessed on May 15, 2010).
14. IronKey, https://www.ironkey.com/ (accessed on June 3, 2010).
15. BitArmor DataControl, http://www.bitarmor.com/ (accessed on June 2010).
16. TrueCrypt, http://www.truecrypt.org/ (accessed on June 4, 2010).
17. DiskCryptor, http://diskcryptor.net/index.php/DiskCryptor_en (accessed on May 15, 2010).
18. StorageCrypt, http://www.magic2003.net/ (accessed on June 15, 2010).
19. Dekart Private Disk, http://www.dekart.com/ (accessed on June 15, 2010).
20. FreeOTFE, http://www.freeotfe.org/ (accessed on June 16, 2010).
21. PGP Desktop Professional, http://www.pgp.com/ (accessed on June 17, 2010).
22. McAfee Endpoint Encryption, http://www.mcafee.com/us/enterprise/products/data_protection/data_encryption/end-point_encryption.html (accessed on June 3, 2010).
23. GuardianEdge Hard Disk Encryption, http://www.guard-ianedge.com (accessed on June 3, 2010).
24. BestCrypt, http://www.jetico.com/encryption-bestcrypt-volume-encryption/ (accessed on June 7, 2010).
25. CREDANT Mobile Guardian, http://www.credant.com/products.html (accessed on June 7, 2010).
26. SafeNet ProtectDrive, http://www.safenet-inc.com/Products/Data_Protection/Disk_and_File_Encryption/ProtectDrive.aspx (accessed on June 8, 2010).
27. Sophos Endpoint Security and Data Protection, http://www.sophos.com/products/enterprise/endpoint/security-and-control/
28. dm-crypt http://www.saout.de/misc/dm-crypt/ (accessed on May 6, 2010).
29. Security Innovation, Co., http://securityinnovation.com/ (accessed on May 18, 2010).