"Kubernetes Primer" This is a good introduction to the Kubernetes container management system. (CoreOS, 2017)

https://coreos.com/resources/index.html#ufh-i-339012759-kubernetes-primer

"Continuous Delivery" Continuous delivery is often used as another name for continuous deployment. This blog includes a series of articles on this topic that expand on the material in this chapter. (J. Humble, 2015)

https://continuousdelivery.com/

PRESENTATIONS, VIDEOS, AND LINKS

https://iansommerville.com/engineering-software-products/microservices-architecture

EXERCISES

6.1 What are the advantages of using services as the fundamental component in a distributed software system?

6.2 Based on the functional breakdown of the authentication features shown in Figure 6.1, create a corresponding breakdown for two-factor authentication and password recovery.

6.3 Explain why microservices should have low coupling and high cohesion.

6.4 What are the principal problems with multi-tier software architectures? How does a microservices architecture help with these problems?

6.5 Explain the differences between synchronous and asynchronous microservices interactions.

6.6 Explain why each microservice should maintain its own data. Explain how data in service replicas can be kept consistent?

6.7 What is a timeout and how is it used in service failure management? Explain why a circuit breaker is a more efficient mechanism than timeouts for handling external service failures.

6.8 Explain what is meant by a "resource." How do RESTful services address resources and operate on them?

6.9 Consider the Upload service for photographs to be printed as shown in Figure 6.5. Suggest how this might be implemented and then design a RESTful interface for this service, explaining the function of each of the HTTP verbs. For each operation, identify its input and output.

6.10 Why should you use continuous deployment in a microservices architecture? Briefly explain each of the stages in the continuous deployment pipeline.

7

Security and Privacy

Software security should always be a high priority for product developers and users. If you don't prioritize security, you and your customers will inevitably suffer losses from malicious attacks. The aim of the attacks may be to steal data or hijack a computer for some criminal purpose. Some attacks try to extort money from a user by encrypting data and demanding a fee for the decryption key, or by threatening a denial of service attack on their servers.

In the worst case, these attacks could put product providers out of business. If providers are delivering a product as a service and it is unavailable or if customer data are compromised, customers are liable to cancel their subscriptions. Even if they can recover from the attacks, this will take a lot of time and effort that would have been better spent working on their software.

Figure 7.1 shows the three main types of threat that computer systems face. Some attacks may combine these threats. For example, a ransomware attack is a threat to the integrity of a system, as it damages data by encrypting them. This makes normal service impossible. Therefore, it is also a threat to the availability of a system.

Security is a system-wide problem. Application software is dependent on an execution platform that includes an operating system, a web server, a language run-time system, and a database. We also depend on frameworks and code generation tools to reuse software that others have developed. Figure 7.2 is a diagram of a system stack that shows the infrastructure systems that your software product may use.

Attacks may target any level in this stack, from the routers that control the network to the reusable components and libraries used by your product. However, attackers usually focus on software infrastructure—the operating system, web browsers, messaging systems, and databases. Everyone uses these, so they are the most effective targets for external attacks.

Figure 7.1 Types of security threat

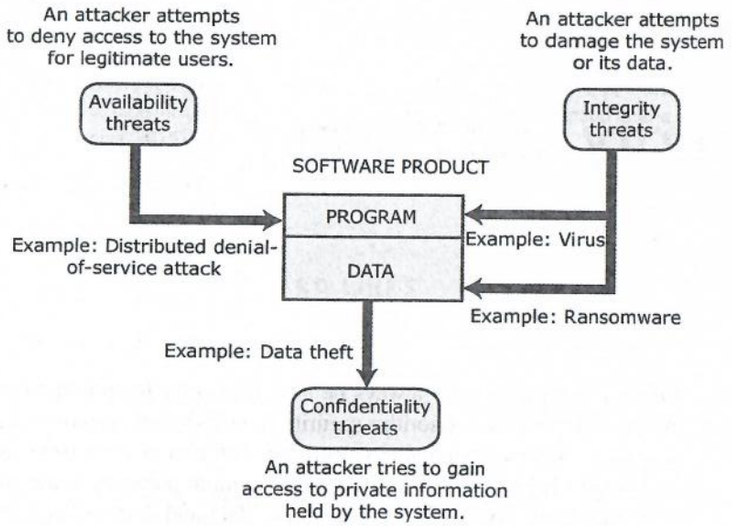


Figure 7.2 System infrastructure stack

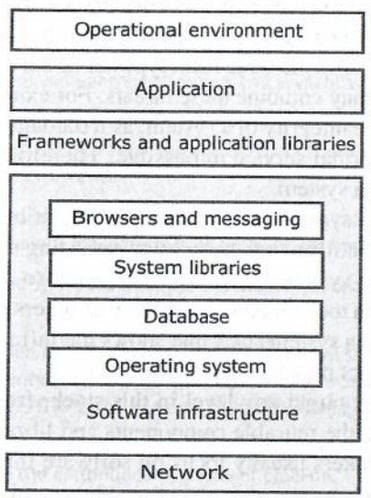


Table 7.1 Security management

Procedure	Explanation
Authentication and authorization	You should have authentication and authorization standards and procedures that ensure that all users have strong authentication and that they have properly set up access permissions. This minimizes the risk of unauthorized users accessing system resources.
System infrastructure management	Infrastructure software should be properly configured, and security updates that patch vulnerabilities should be applied as soon as they become available.
Attack monitoring	The system should be regularly checked for possible unauthorized access. If attacks are detected, it may be possible to put resistance strategies in place that minimize the effects of the attack.
Backup	Backup policies should be implemented to ensure that you keep undamaged copies of program and data files. These can then be restored after an attack.

Maintaining the security of your software infrastructure is a system management rather than a software development issue. You need management procedures and policies to minimize the risk of a successful attack that could ultimately compromise your application system (Table 7.1). Software, such as browsers and operating systems, needs to be updated to ensure that security flaws are fixed. They must be correctly configured so that there are no loopholes that attackers can use to gain access.

Operational security focuses on helping users to maintain security. Attacks on users are very common. Generally, the aim of the attacker is to trick users into disclosing their credentials or accessing a website that includes malware such as a key-logging system. To maintain operational security, you need procedures and practices that advise users how to use your system securely and regular reminders to users of these procedures.

If you offer your product as a cloud-based service, you should include features that help users manage operational security and deal with security problems that may arise. For example:

1. Auto-logout addresses the common problem of users forgetting to log out from a computer used in a shared space. This feature reduces the chances of an unauthorized person gaining access to the system.

2. User command logging makes it possible to discover actions taken by users that have deliberately or accidentally damaged some system resources. This feature helps to diagnose problems and recover from them and also deters malicious legitimate users, as they know that their behavior will be logged.
3. Multifactor authentication reduces the chances of an intruder gaining access to the system using stolen credentials.

Security is a huge topic and my aim here is to introduce some important aspects that are relevant to product developers. This chapter gives you a basic understanding of the issues, but I don't cover detailed security implementation.

7.1 Attacks and defenses

Many types of attack may affect a software system. They depend on the type of system, the way it has been implemented, the potential vulnerabilities in the system, and the environment where the system is used. I focus here on some of the most common types of attack on web-based software.

The targets of the attacks on a computer system may be the system provider or the users of the system. Distributed denial-of-service attacks (see Section 7.1.4) on servers aim to disable access to a system so that users are locked out and the system provider loses revenue. Ransomware attacks disable individual systems in some way and demand a ransom from users to unlock their computers. Data theft attacks may target personal data that can be sold or credit card numbers that can be used illegally.

A fundamental requirement for most attacks is for attackers to be able to authenticate themselves to your system. This usually involves stealing the credentials of a legitimate user. The most common way of doing this is to use social engineering techniques where users click on an apparently legitimate link in an email. This may take them to a lookalike site where they enter their credentials, which are then available to the attacker. Alternatively, the link may take them to a website that installs malware, such as a key logger, that records the user's keystrokes and sends them to the attackers.

7.1.1 Injection attacks

Injection attacks are a type of attack where a malicious user uses a valid input field to input malicious code or database commands. These malicious

instructions are then executed, causing some damage to the system. Code can be injected that leaks system data to the attackers. Common types of injection attack include buffer overflow attacks and SQL poisoning attacks.

Buffer overflow attacks are possible when systems are programmed in C or C++. These languages do not automatically check that an assignment to an array element is within the array bounds. You can declare a buffer as an array of a specific size, but the run-time system does not check whether an input exceeds the length of that buffer.

An attacker who understands how the system memory is organized can carefully craft an input string that includes executable instructions. This overwrites the memory and, if a function return address is also overwritten, control can then be transferred to the malicious code.

Modern software products are not usually developed in C or C++, so this type of attack is unlikely to be a major problem for web-based and mobile software products. Most programming languages check for buffer overflows at run time and reject long, malicious inputs. Operating systems and libraries are often written in C or C++, however. If inputs are passed directly from your system to an underlying system function, buffer overflow is a possibility.

SQL poisoning attacks are attacks on software products that use an SQL database. They take advantage of a situation where a user input is part of an SQL command. For example, the following SQL command is intended to retrieve a database record for a single account holder:

```
SELECT * FROM AccountHolders WHERE accountnumber = '34200645'
```

This statement should return those records in the table called Accountholders where the accountnumber field matches '34200645'. The single quotes identify a string to be matched against the named field.

Normally, the account number is input on a form. Let's assume you use a function called getAccountNumber to retrieve this. You can then create this SQL command:

```
accNum = getAccountNumber ()
SQLstat = "SELECT * FROM AccountHolders WHERE accountnumber = '"
+ accNum + "';"
database.execute (SQLstat)
```

This creates a valid SQL statement by catenating the SELECT part with the input variable accNum and adding a semicolon to end the SQL statement. Single quotes must still be included, as the value of accNum is substituted. This generated SQL statement can then be run against the database.

Now imagine that a malicious user inputs the account number as "10010010' OR '1' = '1". When this is inserted into the SQL query, it becomes

```
SELECT * from AccountHolders WHERE accountnumber = '10010010' OR '1' = '1';
```

The final condition is obviously always true, so the query is equivalent to

```
SELECT * from AccountHolders
```

Therefore, details of all account holders are returned and displayed to the malicious user.

SQL poisoning attacks are possible only when the system does not check the validity of the inputs. In this case, if we know that account numbers are eight digits, then the input function getAccountNumber should include an input check for characters other than digits. This would then reject the injected SQL code.

Validating all user inputs is the key to combating injection attacks. I explain how input validation can be implemented in Chapter 8.

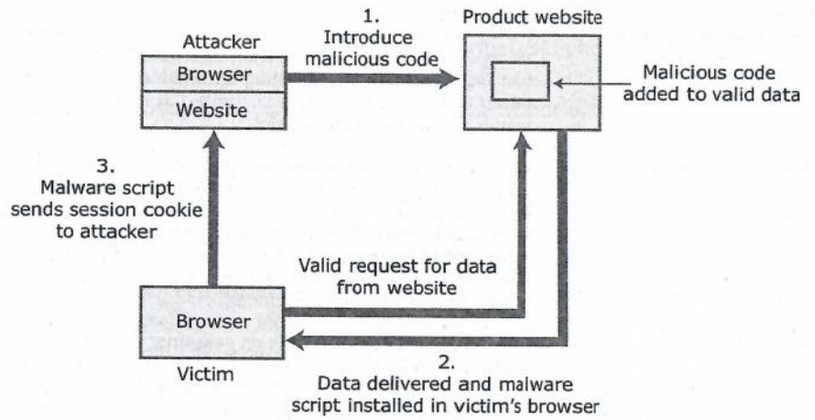
7.1.2 Cross-site scripting attacks

Cross-site scripting attacks are another form of injection attack. An attacker adds malicious Javascript code to a web page that is returned from a server to a client, and this script is executed when the page is displayed in the user's browser. The malicious script may steal customer information or direct customers to another website that may try to capture personal data or display advertisements. Cookies may be stolen, which makes a session hijacking attack possible.

The various kinds of cross-site scripting attacks are described in the XSS scripting tutorial that I include in the Recommended Reading section. They all take the same general form, shown in Figure 7.3, which shows an attack to steal a session cookie.

There are three actors in the scenario shown in Figure 7.3: an attacker, a legitimate website that provides user services, and a victim of the attack, who accesses the legitimate website.

Figure 7.3 Cross-site scripting attack



In the simplest type of cross-site scripting attack, the attacker replaces some legitimate information on the website with a malicious script. When the victim accesses that information, a web page is generated that includes the malicious script as well as the legitimate information requested by the victim. This is sent to the victim's browser, where the malicious code is executed. In this example, the malware steals the session cookie. This gives the attacker access to user information on the product website.

As with other types of injection attack, cross-site scripting attacks may be avoided by input validation. Attackers often add the malicious script to the database using a legitimate form. If this is checked for invalid inputs, then the malicious script can be rejected. Another line of defense is to check input from the database before adding it to a generated page. Finally, it is possible to use the HTML "encode" command, which states that information added to a web page is not executable but should be treated as data by the client's browser.

7.1.3 Session hijacking attacks

When a user authenticates with a web application, a session is created. A session is a time period during which the user's authentication is valid. The user doesn't have to re-authenticate for subsequent system interactions. The session is closed when the user logs out from a system. Alternatively, the session may be closed when the system "times out" because there have been no user inputs for a period of time.

Table 7.2 Actions to reduce the likelihood of session hijacking

Action	Explanation
Traffic encryption	Always encrypt the network traffic between clients and your server. This means setting up sessions using https rather than http. If traffic is encrypted, it is harder to monitor to find session cookies.
Multifactor authentication	Always use multifactor authentication and require confirmation of new actions that may be damaging. For example, before a new payee request is accepted, you could ask the user to confirm their identity by inputting a code sent to their phone. You could also ask for password characters to be input before every potentially damaging action, such as transferring funds.
Short timeouts	Use relatively short timeouts on sessions. If there has been no activity in a session for a few minutes, the session should be ended and future requests directed to an authentication page. This reduces the likelihood that an attacker can access an account if a legitimate user forgets to log off when they have finished work.

The authentication process involves placing a token on the user's computer or mobile device. This is called a session cookie. It is sent from the server to the client at the beginning of a session. The session cookie is used by the server to keep track of user actions. Each time the user makes an http request, the session cookie is sent to the server so that it can link this to previous actions.

Session hijacking is a type of attack where an attacker acquires a valid session cookie and uses it to impersonate a legitimate user. There are several ways an attacker can find out the session cookie value, including cross-site scripting attacks and traffic monitoring. In a cross-site scripting attack, the installed malware sends session cookies to the attackers. Traffic monitoring involves attackers capturing the traffic between the client and the server. The session cookie can then be identified by analyzing the data exchanged. Traffic monitoring is relatively easy if unsecured Wi-Fi networks are used and unencrypted data are exchanged.

Session hijacking may be active or passive. In active session hijacking, the attacker takes over a user session and carries out user actions on a server. So, if a user is logged on to a bank, the attacker can set up a new payee account and transfer money to this account. Passive session hijacking occurs when the attacker simply monitors the traffic between the client and the server, looking for valuable information such as passwords and credit card numbers.

Table 7.2 shows various actions you can take to reduce the likelihood of a session hijacking attack.

7.1.4 Denial-of-service attacks

Denial-of-service attacks are attacks on a software system that are intended to make that system unavailable for normal use. They might be used by malicious attackers who disagree with the policies or actions of the product vendor. Alternatively, attackers might threaten a product provider with a denial of service attack and demand payment not to carry out the threat. They set the level of "ransom" lower than the amount they expect the product provider to lose if their system is out of service.

Distributed denial-of-service (DDOS) attacks are the most common type of denial-of-service attacks. These involve distributed computers that have usually been hijacked as part of a botnet, sending hundreds of thousands of requests for service to a web application. There are so many of these that legitimate users are denied access.

Combating a DDOS attack is a system-level activity. Most cloud providers have specialist software available that can detect and drop incoming packets and thus help restore your services to normal operation.

Other types of denial-of-service attacks target application users. For example, user lockout attacks take advantage of a common authentication policy that locks out a user after a number of failed authentication attempts. Users often use their email address as their login name, so if an attacker has access to a database of email addresses, he or she can try to log in using these addresses. The aim is to lock users out rather than gain access and so deny the service to these users.

There have been so many security breaches that it is relatively easy to get lists of email addresses, and these are often used as user identifiers. If you don't lock accounts after failed validation, then you run the risk of attackers being able to log in to your system. If you do, you may be denying access to legitimate users.

You can take two actions to reduce the damage that such an attack may cause:

1. *Temporary lockouts* If you lock out a user for a short time after failed authentication, the user can regain access to your system after a few minutes. This makes it much more complex for attackers to continue their attack, as they have to continually repeat previous login attempts.
2. *IP address tracking* You may log the IP addresses normally used by users to access your system. If there are failed login attempts from a different IP address, then you can lock out further attempts from that address but allow logins from the user's usual IP addresses.

Sometimes attackers are simply vandals whose aim is to crash an application with no monetary motive. They try to do this by inputting very long strings into forms in the hope that these will not be detected. These attacks are relatively simple to circumvent by using input validation and by handling all exceptions that arise when unexpected input is detected.

7.1.5 Brute force attacks

Brute force attacks are attacks on a web application where the attacker has some information, such as a valid login name, but does not have the password for the site. The attacker creates different passwords and tries to log in with each of these. If the login fails, the attacker then repeatedly tries again with a different password.

Attackers may use a string generator that generates every possible combination of letters and numbers and use these as passwords. You may think this would take a long time, but all strings of six characters or fewer can be generated in a few seconds. You can check this using one of the password checkers on the web.[1] The time required to generate passwords depends on the length of the password, so long passwords are more secure.

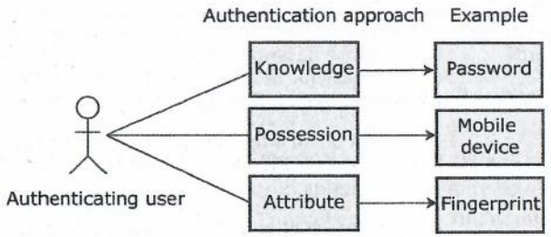
To speed up the process of password discovery, attackers take advantage of the fact that many users choose easy-to-remember passwords. They start by trying passwords from the published lists of the most common passwords. They then usually try a dictionary attack, using all the words in a dictionary. People find it difficult to remember random character strings, so they choose real words that have some significance for them.

Because brute force attacks involve successive retries, many sites block users after a small number of attempts. The problem with this, as I explained in Section 7.1.4, is that this action blocks out legitimate users. An attacker who has a list of user logins and blocks access to all of them can cause widespread disruption.

Brute force attacks rely on users setting weak passwords, so you can circumvent them by insisting that users set long passwords that are not in a dictionary and are not common words. Two-factor authentication, explained in the next section, is also an effective way of deterring these attacks.

[1]For example: https://howsecureismypassword.net/

Figure 7.4 Authentication approaches



7.2 Authentication

Authentication is the process of ensuring that a user of your system is who they claim to be. You need authentication in all software products that maintain user information so that only the providers of that information can access and change it. You also use authentication to learn about your users so that you can personalize their experience of using your product.

Authentication in software products is based on one or more of three approaches—namely, user knowledge, user possession, and user attributes (Figure 7.4).

Knowledge-based authentication relies on users providing secret, personal information when registering to use the system. Each time a user logs on, the system asks for some or all of this information. If the information provided matches the registered information, the authentication is successful. Passwords are the most widely used method of knowledge-based authentication. An alternative, which is often used with passwords, is personal questions that the authenticating user must answer, such as "name of first school" or "favorite film."

Possession-based authentication relies on the user having a physical device that can be linked to the authenticating system. This device can generate or display information that is known to the authenticating system. The user then inputs this information to confirm that they possess the authenticating device.

The most commonly used version of this type of authentication relies on the user providing their mobile phone number when registering for an account. The authenticating system sends a code to the user's phone number. The user has to input this code to complete the authentication.

An alternative approach, which is used by some banks, is based on a special-purpose device that can generate one-time codes. The device calculates

Table 7.3 Weaknesses of password-based authentication

Weakness	Explanation
Insecure passwords	Users choose passwords that are easy to remember. However, it is also easy for attackers to guess or generate these passwords, using either a dictionary or a brute force attack.
Phishing attacks	Users click on an email link that points to a fake site that tries to collect their login and password details.
Password reuse	Users use the same password for several sites. If there is a security breach at one of these sites, attackers then have passwords that they can try on other sites.
Forgotten passwords	Users regularly forget their passwords, so you need to set up a password recovery mechanism to allow these to be reset. This can be a vulnerability if users' credentials have been stolen and attackers use that mechanism to reset their passwords.

a code based on some aspect of the user input. The user inputs this code and it is compared with the code generated by the authenticating system, using the same algorithm as that encoded in the device.

Attribute-based authentication is based on a unique biometric attribute of the user, such as a fingerprint, which is registered with the system. Some mobile phones can authenticate in this way; others use face recognition for authentication. In principle, this is a very secure approach to authentication, but there are still reliability issues with the hardware and recognition software. For example, fingerprint readers often don't work if the user has hot, damp hands.

Each of these approaches to authentication has advantages and disadvantages. Therefore, to strengthen authentication, many systems now use multi-factor authentication, which combines approaches. Service providers, such as Google, offer two-stage authentication; after inputting a password, the user has to input a code sent to the mobile phone. Using a phone provides another level of security, as the phone has to be unlocked using a code, fingerprint, or in some other way.

If your product is delivered as a cloud service, the most practical authentication approach is knowledge-based authentication based on a password, possibly backed up with other techniques. Everyone is familiar with this authentication method. Unfortunately, password-based authentication has well-known weaknesses, as listed in Table 7.3.

You can reduce the risks of password-based authentication by forcing users to set strong passwords. However, this increases the chances that they will

forget their password. You may also ask that individual letters rather than the whole password are input, which means the whole password is not revealed to key-logging malware. You may augment password-based authentication with knowledge-based authentication and require users to answer questions as well as input a password.

The level of authentication that you need depends on your product. If you do not store confidential user information but use authentication to recognize your users, then knowledge-based authentication may be all you need. If you hold confidential user details, however, such as financial information, you should not use knowledge-based authentication on its own. People are now used to two-stage authentication, so you should use phone-based authentication as well as passwords and possibly personal questions.

Implementing a secure and reliable authentication system is expensive and time-consuming. Although toolkits and libraries, such as OAuth, are available for most of the major programming languages, there is still a lot of programming effort involved. Unlike some other aspects of a product, you can't release partial implementations of an authentication system with the aim of extending them in later releases.

For this reason, it is best to think of authentication as a service, even if you are not using a service-oriented approach to build your product. An authentication service can be outsourced using a federated identity system. If you build your own system, you can use a "safer" programming language, such as Java, with more extensive checking and static analysis tools to develop your authentication service. This increases the chances of finding vulnerabilities and programming errors. Your authentication service can also be used for other products that you may develop.

7.2.1 Federated identity

You have almost certainly used websites that offer the opportunity to "Login with Google" or "Login with Facebook." These sites rely on what is called a "federated identity" approach, where an external service is used for authentication.

The advantage of federated identity for users is that they have a single set of credentials that are stored by a trusted identity service. Instead of logging into a service directly, you provide your credentials to a known service that confirms your identity to the authenticating service. You don't have to keep track of different user IDs and passwords. Because your credentials are stored in fewer places, the chances of a security breach where these are revealed is reduced.

Figure 7.5 Federated identity

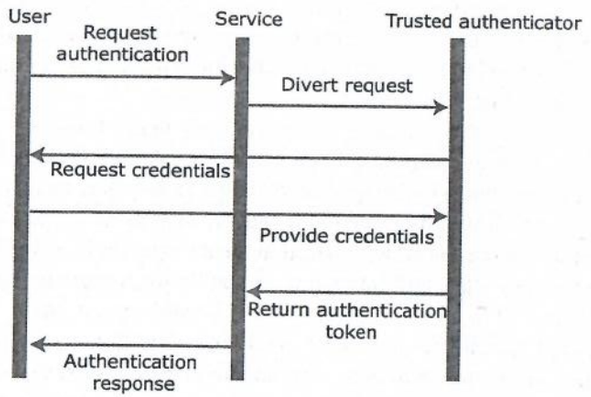


Figure 7.5 is a simplified description of the sequence of actions in a federated identity system.

Consider a product that offers a "Login with Google" option. A user who clicks on this is diverted to the Google identity service. This service validates the user's identity using their Google account credentials. It then returns a token to the diverting site to confirm that the user is a registered Google user. If the user is already logged into a Google service, such as Gmail, then the identity is already registered and there is no need for the user to input any further information.

There are two advantages of using federated identities for authentication:

1. You don't have to maintain your own database of passwords and other secret information. System attackers often try to gain access to this database, so if you maintain your own, you have to take stringent security precautions to protect it. Implementing and maintaining an authentication system are expensive processes for small product companies. Large companies, such as Google and Facebook, have the resources and the expertise to do this.
2. The identity provider may give additional information about users that can be used to personalize your service or to target advertising at users. Of course, when you set up a federated identity system with a major provider, then you have to ask users whether they are willing to share their information with you. There is no guarantee they will agree to this.

Identity verification using Google or Facebook as a trusted service is acceptable for consumer products that are aimed at individual customers. For business products, you can still use federated identity, with authentication based on the business's own identity management system.

If you use a product such as Office 365, you can see how this works. You identify yourself initially to Office 365 using your business email address. The identity management system discovers the business domain from your address and looks up the business's own identity management server. You are diverted to this server, where you input your business credentials, and a token is then sent to the Office 365 system that validates your identity.

Some people dislike federated identity services because of privacy concerns. User information has to be shared with the third-party identity service as a condition of using the service. If Google is the identity service, it therefore knows what software you are using. It can update the data that it holds about you with this information to improve its targeting of personalized advertisements.

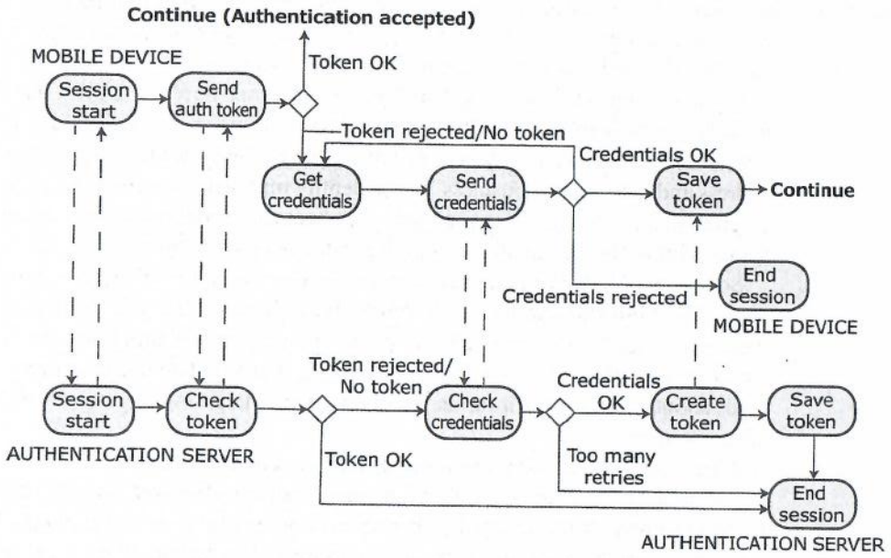
There are various ways to implement federated authentication, but most of the major companies that offer federated authentication services use the OAuth protocol. This standard authentication protocol has been designed to support distributed authentication and the return of authentication tokens to the calling system.

However, OAuth tokens do not include information about the authenticated user. They only indicate that access should be granted. This means it is not possible to use OAuth authentication tokens to make decisions on user privileges—for example, what resources of the system they should have access to. To get around this problem, an authentication protocol called OpenID Connect has been developed that provides user information from the authenticating system. Most of the major authentication services now use this, except Facebook, which has developed its own protocol on top of OAuth.

7.2.2 Mobile device authentication

The ubiquity of mobile devices (tablets and phones) means that companies that offer a cloud-based product usually offer users a mobile app to access their service. You can, of course, use exactly the same approach to authentication on a mobile device as you do on a browser. This will probably annoy your users, however, and dissuade them from using your app. Mobile keyboards are fiddly and prone to errors; if you insist on strong passwords, as you should, there's a good chance that users will mistype them.

Figure 7.6 Mobile device authentication

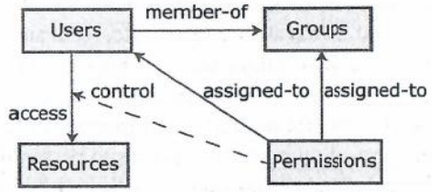


As an alternative to using a login/password pair, a commonly used approach to mobile authentication is to install an authentication token on the mobile device. When the app starts, the token is sent to the service provider to identify the user of the device. This approach to authentication is shown in Figure 7.6.

Users register through the app vendor's website and create an account where they define their authentication credentials. When they install the app, users authenticate themselves using these authentication credentials. These credentials are sent over a secure connection to the authentication server. This server then issues an authentication token that is installed on the user's mobile device. Subsequently, when the app starts, it sends this token to the authentication server to confirm the user's identity. For added security, the authentication token may expire after some period of time so that users have to periodically re-authenticate themselves to the system.

A potential weakness in this approach is that if a device is stolen or lost, then someone who is not the device owner can get access to your product. You can guard against this by checking that the device owner has set up a device passcode or biometric identification that should protect the device from unauthorized access. Otherwise, you require the user to re-authenticate with your app every time it starts up.

Figure 7.7 Elements of an access control policy



Issuing individual users digital certificates and using certificate-based authentication is a variant of token-based authentication where the token is a digital certificate (see the Recommended Reading section). This is a more secure approach than simple authentication tokens because certificates are issued by trusted providers and their validity can be checked. The same certificate can be used to provide single sign-on across a range of applications. There is a significant overhead in managing certificates, however. You have to either do this yourself or outsource the management to a security service.

You must always encrypt authentication information when it is being sent from the client device to the authentication server. You do this using an https rather than an http connection between the client and the server. I explain secure transmission in Section 7.4.

7.3 Authorization

Authentication involves a user proving their identity to a software system. Authorization is a complementary process in which that identity is used to control access to software system resources. For example, if you use a shared folder on Dropbox, the folder's owner may authorize you to read the contents of that folder but not to add new files or overwrite files in the folder.

When a business wants to define the type of access that users get to resources, this is based on an access control policy. This policy is a set of rules that define what information (data and programs) is controlled, who has access to that information, and the type of access that is allowed (Figure 7.7).

For example, an access control policy may specify that nurses and doctors have access to all medical records stored on the system. Doctors may modify information on a record, but nurses may only add new information. Patients may read their own records and may issue a request for correction if they find what they believe is an error.

If you are developing a product for individual use, you probably don't need to include access control features. Your access control policy is simply that the individual user is allowed to create, read, and modify all of their own information. If you have a multiuser business system or share information in individual accounts, however, then access control is essential.

Explicit access control policies are important for both legal and technical reasons. Data protection rules limit access to personal data, and this must be reflected in the defined access control policy. If this policy is incomplete or does not conform to the data protection rules, then there may be subsequent legal action in the event of a data breach. Technically, an access control policy can be a starting point for setting up the access control scheme for a system. For example, if the access control policy defines the access rights of students, then when new students are registered, they all get these rights by default.

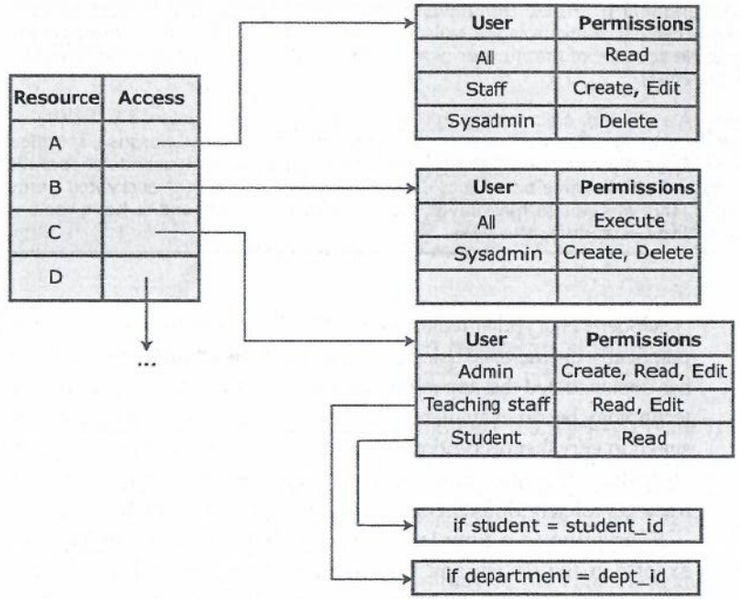
Access control lists (ACLs) are used in most file and database systems to implement access control policies. ACLs are tables that link users with resources and specify what those users are permitted to do. For example, for this book I would like to be able to set up an ACL to a book file that allows reviewers to read that file and annotate it with comments. However, they are not allowed to edit the text or to delete the file.

If ACLs are based on individual permissions, then these lists can become very large. However, you can dramatically cut their size by allocating users to groups and then assigning permissions to the group (Figure 7.8). If you use a hierarchy of groups, then you can add permissions or exclusions to subgroups and individuals.

Figure 7.8 shows examples of ACLs in a university associated with resources A, B, and C. Resource A is a public document that anyone can read. However, it can only be created and edited by staff in the institution and can only be deleted by system administrators. Resource B is an executable program. Anyone can execute it, but only system administrators can create and delete it. Resource C is a student information system. Administrative staff can create, read, and edit records in the system. Teaching staff can read and edit records of students in their department. Students can only read their own record. To ensure that student information is retained, no one has permission to delete student data.

Unless you have a very specialized product, it is not worth developing your own access control system for authorization. Rather, you should use the ACL mechanisms in the underlying file or database system. However, you may decide to implement your own control panel for the ACL system that reflects the data and file types used in your product. This makes it easier to set up and revoke access permissions and reduces the chances of authorization errors.

Figure 7.8 Access control lists



7.4 Encryption

Encryption is the process of making a document unreadable by applying an algorithmic transformation to it. The encryption algorithm uses a secret key as the basis of this transformation. You can decode the encrypted text by applying the reverse transformation. If you choose the right encryption algorithm and secret key, then it is virtually impossible for anyone else to make the text readable without the key.

This encryption and decryption process is shown in Figure 7.9.

Figure 7.9 Encryption and decryption

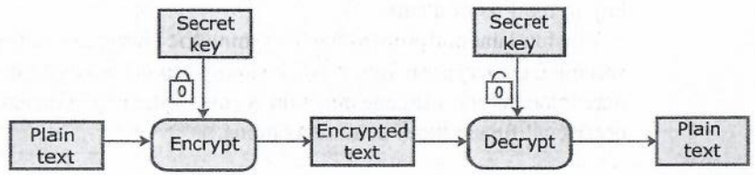


Table 7.4 Technology and encryption

During World War II, the German military used an encryption system based on an electromechanical coding machine called Enigma. They believed it to be practically uncrackable because of the number of combinations that would have to be tested to break the code. However, Alan Turing, a pioneering British computer scientist, designed two early computers, one electro-mechanical (Bombe) and one electronic (Colossus), specifically to crack the Enigma encryption. These computers could carry out thousands of operations per second, and it became possible to decode a large percentage of encrypted German messages. This was said to have saved thousands of Allied lives and to have hastened the defeat of Nazi Germany by the Allies.

Modern encryption techniques enable you to encrypt data so that they are practically uncrackable using currently available technology. However, history has demonstrated that apparently strong encryption may be crackable when new technology becomes available (Table 7.4). Quantum computers are particularly suited to very fast decryption of text that is encrypted using current encryption algorithms. If commercial quantum systems become available, we will have to use a completely different approach to encryption on the Internet.

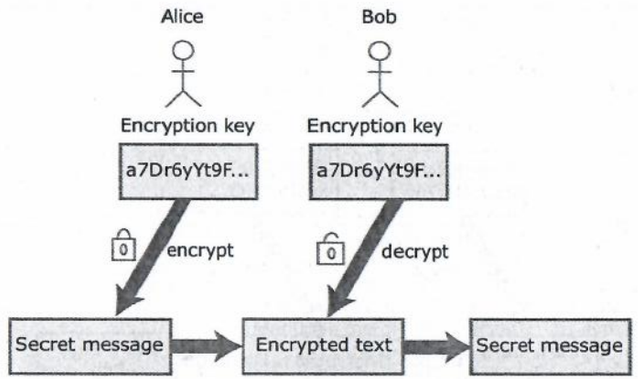
Encryption is a complex topic; most engineers, including me, are not experts in the design and implementation of encryption systems. Consequently, I don't give advice on what encryption schemes to use, how to manage encryption keys, and so on. What I aim to do here is give an overview of encryption and make you aware of what you have to think about when making decisions about it.

7.4.1 Symmetric and asymmetric encryption

Symmetric encryption, illustrated in Figure 7.10, has been used for hundreds of years. In a symmetric encryption scheme, the same encryption key is used for both encoding and decoding the information that is to be kept secret. If Alice and Bob wish to exchange a secret message, both must have a copy of the encryption key. Alice encrypts the message with this key. When Bob receives the message, he decodes it using the same key to read its contents.

The fundamental problem with a symmetric encryption scheme is securely sharing the encryption key. If Alice simply sends the key to Bob, an attacker may intercept the message and gain access to the key. The attacker can then decode all future secret communications.

Figure 7.10 Symmetric encryption



An alternative approach, called asymmetric encryption (Figure 7.11), does not require secret keys to be shared. An asymmetric encryption scheme uses different keys for encrypting and decrypting messages. Each user has a public and a private key. Messages may be encrypted using either key but can only be decrypted using the other key.

As the name suggests, public keys may be published and shared by the key owner. Anyone can access and use a published public key. However, a message can only be decrypted by the user's private key, so is readable by only the intended recipient. For example, in Figure 7.11, Alice encrypts a

Figure 7.11 Asymmetric encryption

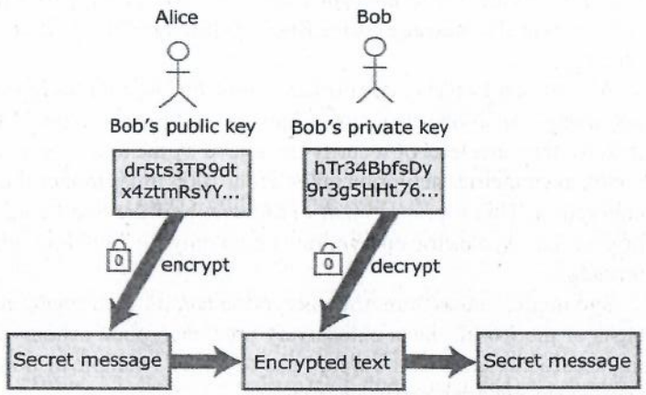
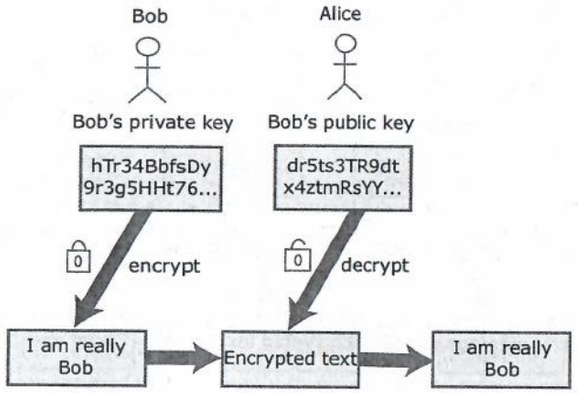


Figure 7.12 Encryption for authentication



secret message using Bob's public key. Bob decrypts the message using his private key, which only he knows. The message cannot be decrypted with Bob's public key.

Asymmetric encryption can also be used to authenticate the sender of a message by encrypting it with a private key and decrypting it with the corresponding public key. Let's assume Alice wants to send a message to Bob and she has a copy of his public key. However, she is not sure whether or not the public key that she has for Bob is correct, and she is concerned that the message may be sent to the wrong person. Figure 7.12 shows how private/public key encryption can be used to verify Bob's identity. Bob uses his private key to encrypt a message and sends it to Alice. If Alice can decrypt the message using Bob's public key, then Alice has the correct key.

As there isn't a secure key exchange problem, an obvious question is "Why not always use asymmetric rather than symmetric encryption?" The reason is that, for the same level of security (measured by the time required to crack the code), asymmetric encryption takes about 1000 times longer than symmetric encryption. This is proportional to the length of the text being encoded so, in practice, asymmetric encryption is used only for encoding relatively short messages.

Symmetric and asymmetric encryption can be used together. This is the basis of the world's most extensively used encryption scheme for exchanging secure messages on the web. I use this as an example of how to combine symmetric and asymmetric encryption.

Table 7.5 Elements of digital certificates

Certificate element	Explanation
Subject information	Information about the company or individual whose website is being visited. Applicants apply for a digital certificate from a certificate authority who checks that the applicant is a valid organization.
Certificate authority information	Information about the certificate authority (CA) who has issued the certificate.
Certificate information	Information about the certificate itself, including a unique serial number and a validity period, defined by start and end dates.
Digital signature	The combination of all of the above data uniquely identifies the digital certificate. The signature data are encrypted with the CA's private key to confirm that the data are correct. The algorithm used to generate the digital signature is also specified.
Public key information	The public key of the CA is included along with the key size and the encryption algorithm used. The public key may be used to decrypt the digital signature.

7.4.2 TLS and digital certificates

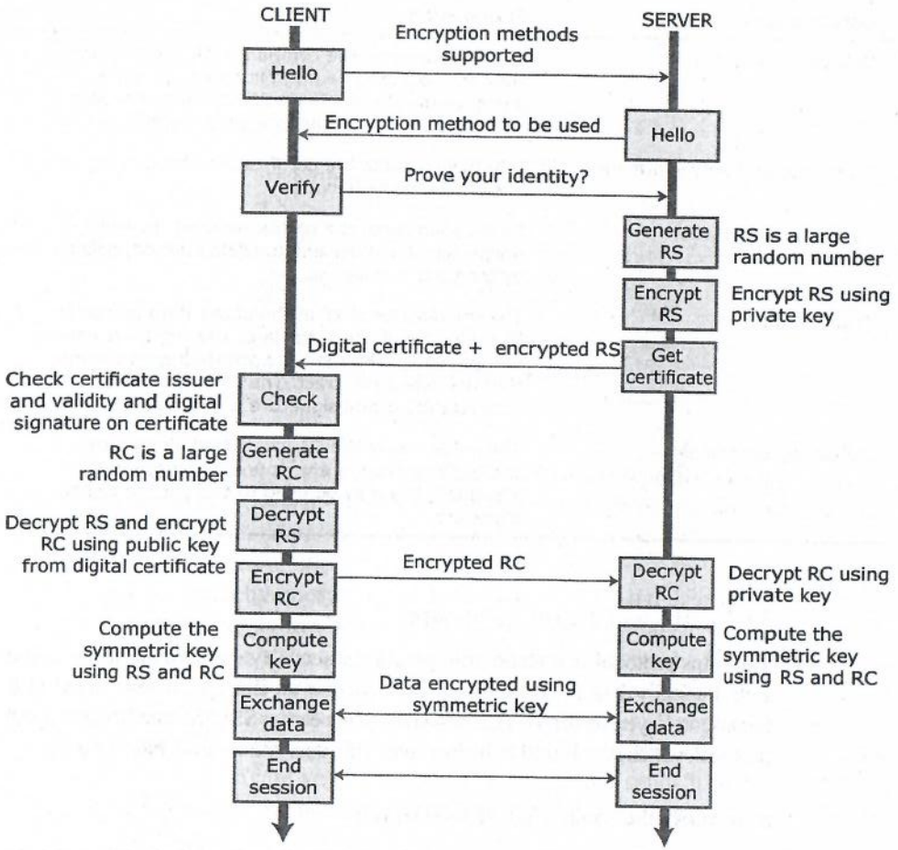
The https protocol is a standard protocol for securely exchanging texts on the web. Basically, it is the standard http protocol plus an encryption layer called TLS (Transport Layer Security). TLS has replaced the earlier SSL (Secure Socket Layer) protocol, which was found to be insecure. This encryption layer has two uses:

- to verify the identity of the web server;
- to encrypt communications so that they cannot be read by an attacker who intercepts the messages between the client and the server.

TLS encryption depends on a digital certificate that is sent from the web server to the client. Digital certificates are issued by a certificate authority (CA), which is a trusted identity verification service. Organizations that buy a digital certificate have to provide information to the CA about their identity, and this identity information is encoded in the digital certificate. Therefore, if a certificate is issued by a recognized CA, the identity of the server can be trusted. Web browsers and apps that use https include a list of trusted certificate providers.

Table 7.5 shows the information that is included in a digital certificate.

Figure 7.13 Using symmetric and asymmetric encryption in TLS



The CA encrypts the information in the certificate using their private key to create a unique signature. This signature is included in the certificate along with the public key of the CA. To check that the certificate is valid, you can decrypt the signature using the CA's public key. The decrypted information should match the other information in the certificate. If not, the certificate has been forged and should be rejected.

When a client and server wish to exchange encrypted information, they communicate to set up a TLS connection. They then exchange messages, as shown in Figure 7.13, to establish the encryption key that both the client and the server will use.

The digital certificate that the server sends to the client includes the server's public key. The server also generates a long random number, encrypts it using its private key, and sends this to the client. The client can then decrypt this using the server's public key and, in turn, generates its own long random number. It encrypts this number using the server's public key and sends it to the server, which decrypts the message using its private key. Both client and server then have two long random numbers.

The agreed encryption method includes a way of generating an encryption key from these numbers. The client and server independently compute the key that will be used to encrypt subsequent messages using a symmetric approach. All client—server traffic is then encrypted and decrypted using that computed key. There is no need to exchange the key itself.

7.4.3 Data encryption

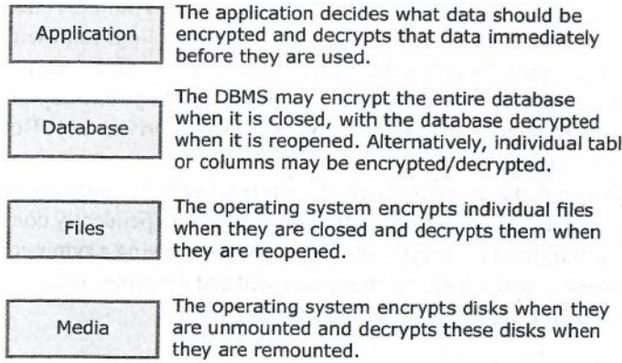
As a product provider, you inevitably store information about your users and, for cloud-based products, user data. User information may include personal information such as addresses, phone numbers, email addresses, and credit card numbers. User data may include documents that the users have created or business databases.

For example, say your product is a cloud-based system for labs that allows them to store and process information on tests of new pharmaceuticals. The database includes information about experiments, the participants in these experiments, and the test results. Theft of these data may compromise the privacy of the participants in the test, and disclosure of the test results may affect the financial position of the testing company.

Encryption can be used to reduce the damage that may occur from data theft. If information is encrypted, it is impossible, or very expensive, for thieves to access and use the unencrypted data. Therefore, you should encrypt user data whenever it is practicable to do so. The practicality of encryption depends on the encryption context:

1. *Data in transit* The data are being moved from one computer to another. Data in transit should always be encrypted. When transferring the data over the Internet, you should always use the https rather than the http protocol to ensure encryption.
2. *Data at rest* The data are being stored. If data are not being used, then the files where the data are stored should be encrypted so that theft of these files will not lead to disclosure of confidential information.

Figure 7.14 Encryption levels



3. *Data in use* The data are being actively processed. There are problems in using encryption with data that are in use. Encrypting and decrypting the data slow down the system response time. Furthermore, implementing a general search mechanism with encrypted data is impossible because of the difficulties in matching search terms with encrypted data.

Encryption of data is possible at four different levels in the system (Figure 7.14). Generally, more protection is afforded at the higher levels in this stack, as the data are decrypted for a shorter period of time.

Media-level encryption is where an entire disk is encrypted. This provides some limited protection and can be used to protect the data on laptops and portable media if they are lost or stolen. This level is not really relevant to product developers.

File-level encryption involves encrypting entire files and is relevant if you maintain some information in files rather than store everything in a DBMS. Generally, this means you have to provide your own encryption system for your system files. You should not trust the encryption used by cloud providers, such as Dropbox, as they hold the keys and so can access your data.

Most database management systems provide some support for encryption:

1. *Database file encryption* The files in which the database holds its data are encrypted. When the DBMS requests data from a file, it is decrypted as it is transferred to the system's memory and encrypted when it is written back to the file.

2. *"Column-level" encryption* Specific columns in a relational database system are encrypted. For example, if your database holds personal information, you should encrypt the column that holds the user's credit card number. The column need only be decrypted when the number is retrieved—for example, sent in a transaction to the credit card company.

Application-level encryption allows you, as a product developer, to decide what and when data should be encrypted. You implement an encryption scheme within your product to encrypt and decrypt confidential data. Each user of your product chooses a personal encryption key. The data are encrypted in the application that generates or modifies the data rather than relying on database encryption. Consequently, all stored data are always encrypted. You should not store the encryption keys used.

Unfortunately, application-level encryption has several drawbacks:

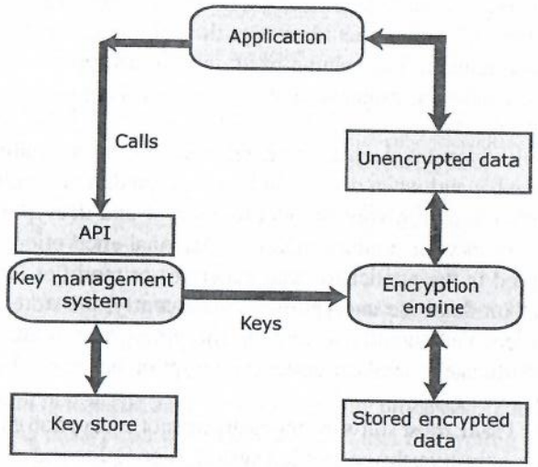
1. As I said, most software engineers are not encryption experts. Implementing a trustworthy encryption system is complex and expensive, and there is a real possibility that you will make mistakes. This means your system may not be as secure as you intended.
2. Encryption and decryption can significantly affect the performance of your application. The time needed for encryption and decryption slows down the system. Users may either reject your software or not use the encryption feature.
3. In addition to encryption, you need to provide key management functionality, which I cover in the next section. Normally, this involves writing extra code to integrate your application with a key management system.

If you decide to implement encryption in your application, crypto libraries are available for most programming languages. For symmetric encryption, the AES and Blowfish algorithms are very secure, but you should always develop or bring in specialist expertise to help you choose the encryption approach that is most appropriate for your product.

7.4.4 Key management

A general problem in any encryption system is key management. This is the process of ensuring that encryption keys are securely generated, stored, and accessed by authorized users. Businesses may have to manage tens of thousands

Figure 7.15 Using a KMS for encryption management



of encryption keys. Because of the huge number of encryption keys and digital certificates that have to be managed, it is impractical to do key management manually. You need to use some kind of automated key management system (KMS).

Key management is important because if you get it wrong, unauthorized users may be able to access your keys and so decrypt supposedly private data. Even worse, if you lose encryption keys, then your encrypted data may be permanently inaccessible.

A KMS is a specialized database designed to securely store and manage encryption keys, digital certificates, and other confidential information. It may provide functionality such as key generation—for example, a public key/private key pair, access control that governs which people and applications can access keys, and key transfer that securely transfers the keys from the KMS to other network nodes.

Figure 7.15 shows the elements of an encryption system with access coordinated using a KMS.

Businesses may be required by accounting and other regulations to keep copies of all of their data for several years. For example, in the United Kingdom, tax and company data have to be maintained for at least six years, with a longer retention period for some types of data. Data protection regulations may require that these data be stored securely, so the data should be encrypted.

To reduce the risks of a security breach, however, encryption keys should be changed regularly. This means that archival data may be encrypted with a different key from the current data in your system. Therefore, a KMS must maintain multiple timestamped versions of keys so that system backups and archives can be decrypted if required.

Some elements of KMS functionality may be provided as a standard OS facility, such as Apple's MacOS Keychain, but this is really only suitable for personal or perhaps small business use. More complex KMS products and services are available for large businesses. Amazon, Microsoft, and Google provide KMSs that are specifically designed for cloud-based products.

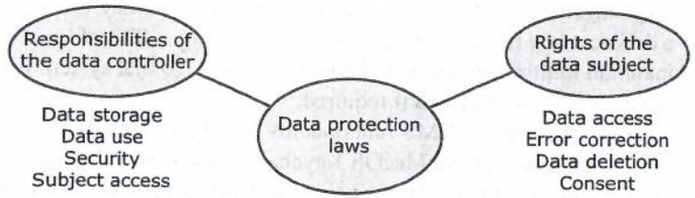
7.5 Privacy

Privacy is a social concept that relates to the collection, dissemination, and appropriate use of personal information held by a third party, such as a company or a hospital. The importance of privacy has changed over time, and individuals have their own views on what degree of privacy is important. Culture and age also affect peoples' views on what privacy means. For example:

- Some people may be willing to reveal information about their friends and colleagues by uploading their contacts list to a software system; others do not wish to do so.
- Younger people were early adopters of the first social networks, and many of them seem to be less inhibited about sharing personal information on these platforms than older people.
- In some countries, the level of income earned by an individual is seen as a private matter; in others, all tax returns are openly published.

To maintain privacy, you need to have a secure system. However, security and privacy are not the same thing. Facebook is a secure system with few breaches of its security. There have been several privacy breaches, however, because the features of the system prevent or make it difficult for users to control who sees their personal information. In a medical information system, if an external attacker gains access to the medical records, this is a security failure. If the information in the system is used to send unwanted marketing information about care homes, this is a privacy failure.

Figure 7.16 Data protection laws



People have different opinions on privacy, so it is impossible to establish objective "privacy standards" based on a definition of "sensitive personal information." Few people would argue against maintaining the privacy of health information. But what about location information—should this be private or not? Knowing an individual's location can enhance the user experience in many products. This information can be misused, however, so some people don't want to disclose their locations or won't allow other companies to use their location information.

In many countries, the right to individual privacy is protected by data protection laws. These laws limit the collection, dissemination, and use of personal data to the purposes for which they were collected. For example, a travel insurance company may collect health information to assess their level of risk. This is legal and permissible. However, it would not be legal for those companies to use this information to target online advertising of health products, unless their users had given specific permission for this.

Figure 7.16 shows the areas that may be covered by data protection laws. These laws differ from country to country, and some country's laws do not cover all areas. The European Union's data protection regulations (GDPR) are among the most stringent in the world, and I base my discussion here on these regulations. The legislation does not only apply to European companies. The GDPR applies to all companies that hold data about EU citizens, irrespective of where these companies are based. Therefore, U.S., Indian, and Chinese companies that allow EU citizens to create accounts must follow the GDPR.

Data protection laws typically refer to data subjects and data controllers. The data subject is the individual whose data are being managed, and the data controller is the manager of the data. The term "data owner" is ambiguous, so it is not usually used. Data subjects have the right to access the stored data and to correct mistakes. They must give their consent for the use of their data and may ask for relevant data to be deleted. The data controller is responsible

Table 7.6 Data protection principles

Data protection principle	Explanation
Awareness and control	Users of your product must be made aware of what data are collected when they are using your product, and must have control over the personal information that you collect from them.
Purpose	You must tell users why data are being collected and you must not use those data for other purposes.
Consent	You must always have the consent of a user before you disclose their data to other people.
Data lifetime	You must not keep data for longer than you need to. If a user deletes an account, you must delete the personal data associated with that account.
Secure storage	You must maintain data securely so that it cannot be tampered with or disclosed to unauthorized people.
Discovery and error correction	You must allow users to find out what personal data you store. You must provide a way for users to correct errors in their personal data.
Location	You must not store data in countries where weaker data protection laws apply unless there is an explicit agreement that the stronger data protection rules will be upheld.

for storing data securely in a location covered by data protection legislation. The controller must provide subject access to the data and should use it only for the purpose for which it was collected.

Data protection laws are based on a set of privacy principles that reflect good privacy practice (Table 7.6).

There are three business reasons why you should pay attention to information privacy:

1. If you are offering a product directly to consumers and you fail to conform to privacy regulations, then you may be subject to legal action by product buyers or by a data regulator. If your conformance is weaker than the protection offered by data protection regulations in some countries, you cannot sell your product in these countries.
2. If your product is a business product, business customers require privacy safeguards so that they are not put at risk of privacy violations and legal action by users.

3. If personal information is leaked or misused, even if this is not seen as a violation of privacy regulations, your reputation may be seriously damaged. Customers may stop using your product because of this.

The information that your software needs to collect depends on the functionality of your product and on the business model you use. You should not collect personal information that you do not need. Say you are developing a service-oriented learning environment. You need to collect information about the learners using the system, the services they use, the learning modules they access, and their performance in assessments. You do not need information on the ethnic background of users, their family circumstances, or what other software they use.

To maintain the privacy of user data, you should establish a privacy policy that defines how personal and sensitive information about users is collected, stored, and managed. The general data protection principles, shown in Table 7.6, should serve as a framework for the development of a privacy policy for your product.

Software products use data in different ways, so your privacy policy has to define the personal data that you will collect and how you will use those data. Product users should be able to review your privacy policy and change their preferences regarding the information that you store. For example, users should be able to state whether or not they want to receive marketing emails from you. Your privacy policy is a legal document and it should be auditable to check that it is consistent with the data protection laws in countries where your software is sold.

Unfortunately, too many software companies bury their privacy policy in a long "terms and conditions" document that, in practice, nobody reads. They therefore get away with collecting user data that are not needed for their product and using these data in ways that users would not expect. This is not illegal, but it is unethical. The GDPR now requires software companies to provide a summary of their privacy policy, written in plain language rather than legal jargon.

Some software business models are based on providing free access to the software and using the users' data in some way to generate revenue. The data may be used to target advertising at users or to provide services that are paid for by other companies. If you use this model, you should make clear that you collect data for this purpose and that your service depends on monetizing user data in some way. You should always allow users to opt out of the use of their data by other companies.

Privacy becomes particularly challenging when your product includes sharing features that allow users to see what other users are doing and how they use your product. Facebook is the prime example of this. There have been many controversies over Facebook privacy and the ways in which the company uses user data and provides privacy controls for users. Facebook provides extensive privacy controls, but these are not all located in the same place and they are sometimes difficult to find. Consequently, many Facebook users inadvertently reveal personal information that they might prefer to keep private.

If your product includes social network functionality so that users can share information, you should ensure that users understand how to control the information they share. Ideally, you should offer a "privacy dashboard," where all privacy controls are in one place and are clear to users. If the functionality of your system depends on mining user information, you should make clear to users that setting privacy controls may limit the functionality that your system offers.

KEY POINTS

- Security is a technical concept that relates to a software system's ability to protect itself from malicious attacks that may threaten its availability, the integrity of the system and its data, and the theft of confidential information.
- Common types of attack on software products are injection attacks, cross-site scripting attacks, session hijacking attacks, denial-of-service attacks, and brute force attacks.
- Authentication may be based on something a user knows, something a user has, or some physical attribute of the user.
- Federated authentication involves devolving responsibility for authentication to a third party, such as Facebook or Google, or to a business's authentication service.
- Authorization involves controlling access to system resources based on the user's authenticated identity. Access control lists are the most commonly used mechanism to implement authorization.
- Symmetric encryption involves encrypting and decrypting information using the same secret key. Asymmetric encryption uses a key pair—a private key and a public key. Information encrypted using the public key can only be decrypted using the private key.
- A major issue in symmetric encryption is key exchange. The TLS protocol, which is used to secure web traffic, gets around this problem by using asymmetric encryption for transferring the information required to generate a shared key.