© The Author(s), under exclusive license to APress Media, LLC, part of Springer Nature 2024

M. Nardone, C. Scarioni, *Pro Spring Security*

https://doi-org.ezproxy.sfpl.org/10.1007/979-8-8688-0035-1_1

1. The Scope of Security

Massimo Nardone[1] and Carlo Scarioni[2]

(1) HELSINKI, Finland

(2) Surbiton, UK

Security. It is an incredibly overloaded word in the IT, OT, and IoT world. It means so many different things in many different contexts, but in the end, it is all about protecting sensitive and valuable resources against malicious usage.

IT has many layers of infrastructure and code that can be subject to malicious attacks, and arguably, you should ensure that all these layers get the appropriate levels of protection.

In operational technology (OT), where generally the systems were isolated from the external networks and operated independently, the increasing connectivity and integration of OT systems with information technology (IT) networks and the Internet, the risk of cyberattacks targeting these systems has significantly grown. OT security aims to address these risks and protect against threats that could disrupt operations, cause physical damage, or impact public safety.

In the Internet of Things (IoT), security refers to the measures and practices implemented to protect the interconnected devices, networks, and data associated with IoT systems, such as networks of physical objects or "things" embedded with sensors, software, and connectivity to exchange data and perform various tasks. These objects range from household appliances and wearable devices to industrial machinery and infrastructure. Given the proliferation of IoT devices and their increasing integration into various domains, securing IoT systems is critical to mitigate potential risks and protect the privacy, integrity, and availability of their data and services.

The growth of the Internet and the pursuit of reaching more people with our applications have opened more doors to cyber criminals trying to access these applications illegitimately.

It is also true that good care is not always taken to ensure that a properly secured set of services is offered to the public. And sometimes, even when good care is taken, some hackers are still smart enough to overcome security barriers that, superficially, appear adequate.

The first step is to define a defense-in-depth strategy and security layers.

Defense in depth (also known as DiD) is a security strategy that involves implementing multiple layers of defense to protect a system or network from potential threats. It aims to provide a comprehensive and resilient security posture by incorporating various security measures at different levels, such as physical, technical, and administrative controls.

The defense-in-depth concept recognizes that no single security measure is fool-proof, and relying on a single layer of defense can leave vulnerabilities. By employing multiple layers, other layers can still provide protection even if one is breached or compromised.

In practice, a defense-in-depth strategy can include a combination of measures such as firewalls, intrusion detection systems, encryption, access controls, strong authentication mechanisms, security awareness training, regular system updates and patching, network segmentation, and physical security measures like locked doors and security cameras. These layers collectively create a more robust and resilient security infrastructure.

The goal of a defense-in-depth strategy is to increase the difficulty for attackers, making it harder for them to penetrate a system and move deeper into the network. Requiring attackers to overcome multiple barriers increases the likelihood of detection and mitigation, reducing the potential impact of a successful attack. Overall, it is a proactive approach to security that emphasizes multiple layers of protection, reducing the risk of successful attacks and minimizing the potential damage they can cause.

In general, a defense-in-depth strategy is a way to define how to develop the cybersecurity of the IT infrastructure by defining how all the defensive mechanisms are layered to protect and secure data and information. A failing or weak defense-in-depth strategy might result from a cybersecurity attack on the IT infrastructure.

Let’s try to understand a bit more about defense-in-depth mechanisms. First, there are three major controls.

- **Physical controls** are security measures that aim to protect the physical infrastructure and assets. They include surveillance cameras, access controls (such as locks and biometric systems), perimeter fencing, security guards, and intrusion detection systems.
- **Perimeter security** focuses on securing the boundary between the internal network and the external environment. It involves firewalls, intrusion prevention systems (IPS), and demilitarized zones (DMZs) to filter and monitor network traffic, control access, and prevent unauthorized entry.
- **Network security** measures aim to protect the internal network infrastructure. They include technologies such as network segmentation, virtual private networks (VPNs), intrusion detection systems (IDS), and IPS to detect and prevent unauthorized access,

monitor network traffic, and detect and respond to potential threats.

- **Identity and access management (IAM)** controls ensure that only authorized individuals can access systems and resources. This includes strong authentication mechanisms like passwords, two-factor authentication (2FA), multi-factor authentication, access control policies, and privilege management to enforce least privilege principles.
- **Application security** focuses on securing the software and applications used within an organization. This involves implementing secure coding practices, regular vulnerability assessments and penetration testing, web application firewalls (WAFs), and application-level authentication and authorization mechanisms.
- **Data encryption** protects data by transforming it into a secure format that can only be accessed with the correct decryption key. It is used to secure data at rest (stored data) and in transit (data transmitted over networks).
- **Security monitoring and incident response** involve continuous monitoring of systems and networks, which is crucial to detecting and responding to security incidents. This includes using security information and event management (SIEM) tools, log analysis, IDS, and incident response plans to promptly identify and mitigate potential threats.
- **Security awareness and training** includes educating employees and users about security best practices and potential threats is vital. Regular security awareness training helps individuals understand their role in maintaining a secure environment and enables them to identify and report suspicious activities.

By combining these major controls, organizations can establish a multi-layered defense-in-depth security approach that provides and increases overall resilience against various threats.

Figure 1-1 shows typical defense-in-depth mechanisms defining IT infrastructure security layers.

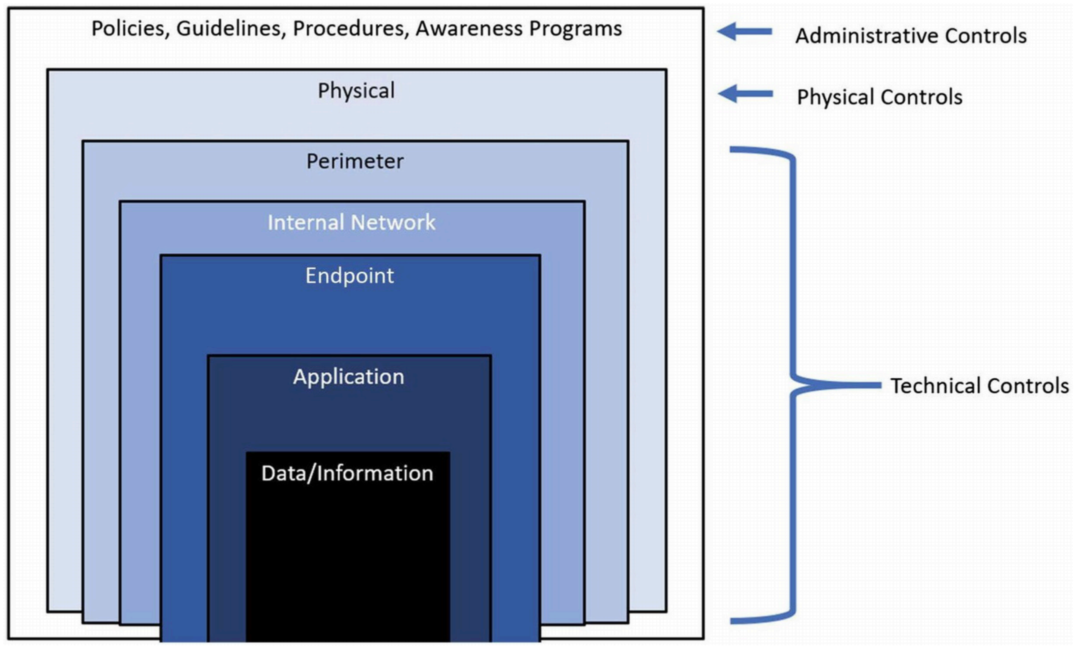

Figure 1-1 Defense-in-depth mechanisms and IT infrastructure layers

The three major security layers in an IT infrastructure are the network, the operating system (part of the endpoint security layer), and the application itself.

The Network Security Layer

The network security layer is probably the most familiar one in the IT world. When people talk about IT security, they normally think of network-level security—in particular, security that uses firewalls.

Even though people often associate security with the network level, this is only a very limited layer of protection against attackers. Generally speaking, it can do no more than defend IP addresses and filter network packets addressed to certain ports in certain machines in the network.

This is not enough in most cases, as traffic at this level is normally allowed to enter the publicly open ports of your various exposed services without restriction. Different attacks can be targeted at these open services, as attackers can execute arbitrary commands that could compromise your security constraints. Tools like the popular `nmap` (**`http://nmap.org/`**) can scan a machine to find open ports. Using such tools is an easy first step in preparing an attack because well-known attacks can be used against such open ports if they are not properly secured.

A very important part of the network-layer security, in the case of web applications, is the use of Secure Sockets Layer (SSL) to encode all sensitive information sent along the wire, but this is related more to the network protocol at the application level than to the network physical level at which firewalls operate.

The Operating System Layer

The operating system layer is probably the most important one in the whole security schema, as a properly secured operating system (OS) environment can at least prevent a whole host machine from going down if a particular application is compromised.

If an attacker is somehow allowed to have unsecured access to the operating system, they can basically do whatever they want—from spreading viruses to stealing passwords or deleting your whole server's data and making it unusable. Even worse, they could take control of your computer without you even noticing and use it to perform other malicious acts as part of a botnet. This layer can include the deployment model of the applications since you need to know your operating system's permission scheme to ensure that you don't give your applications unnecessary privileges over your machine. Applications should run as isolated as possible from the other components of the host machine.

The Application Layer

The primary focus of this book is on the application layer. The application security layer refers to all the constraints you establish in your applications to make sure that only the right people can do the right things when working through the application.

Applications, by default, are open to countless avenues of attack. An improperly secured application can allow an attacker to steal information from the application, impersonate other users, execute restricted operations, corrupt data, gain access to the operating system level, and perform many other malicious acts.

This book covers application-level security, which is the domain of Spring Security. Application-level security is achieved by implementing several techniques, and there are a few concepts that help you understand better what the book covers. They are the main concerns that Spring Security addresses to provide your applications with comprehensive protection against threats. The following three subsections introduce authentication, authorization, and ACLs.

Authentication

Authentication is the process of verifying the identity of a user or entity attempting to access an application. It ensures that the user is who they claim to be. Common authentication methods include the following.

- **Username and password**: Users provide a unique username and corresponding password.
- **Multi-factor authentication (MFA)**: Users provide multiple forms of identification, such as a password and a one-time verification code sent to their mobile device.
- **Biometric authentication**: Users verify their identity using unique physical characteristics, such as fingerprints, facial recognition, or iris scans.

The authentication process allows an application to validate that a particular user is who they claim they are. In the authentication process, a user presents the application with information about herself (normally, a username and a password) that no one else knows. The application takes this information and tries to match it against the information stored—normally, in a database or LDAP[1] (Lightweight Directory Access Protocol) server. If the information the user provides matches a record in the authentication server, the user is successfully authenticated. The application normally creates an internal abstraction representing this authenticated user in the system. Figure 1-2 shows the authentication mechanism.

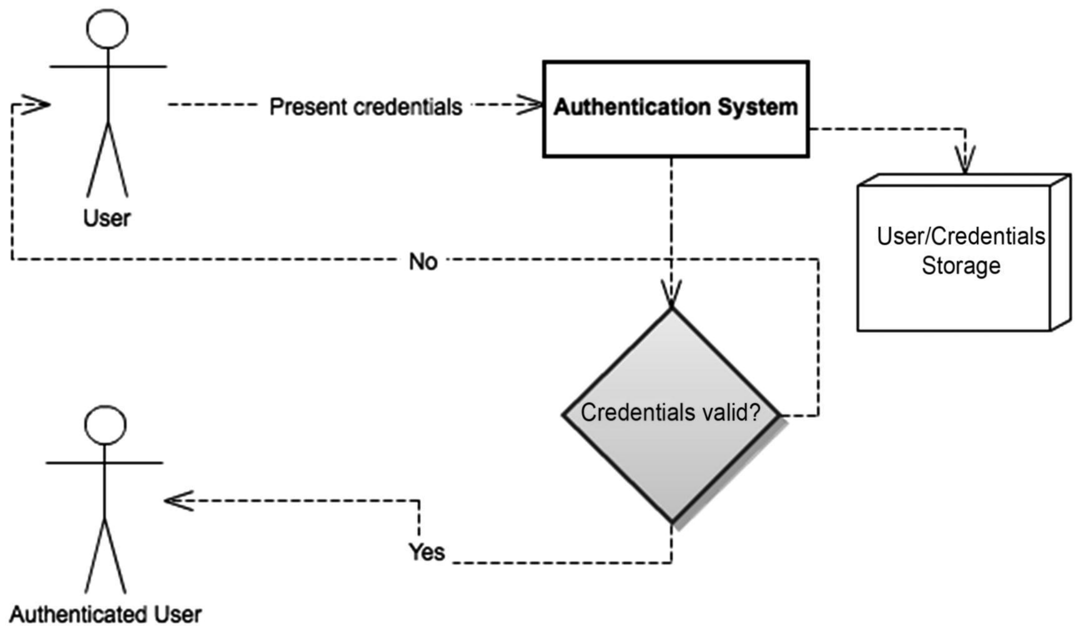

Figure 1-2 Simple, standard authentication mechanism

Authorization

Authorization determines what actions or resources a user can access within an application. Once a user is authenticated, authorization mechanisms control their permissions based on predefined rules and policies. This ensures that users can only access the features and data they are authorized to use. Authorization can be role-based, attribute-based, or rule-based.

- **Role-based access control (RBAC)**: Users are assigned roles, and permissions are granted based on those roles. For example, a manager role may access certain administrative features, while a regular user role may only access basic functionalities.
- **Attribute-based access control (ABAC)**: Access is granted based on specific attributes or characteristics of the user, such as job title, department, or location.
- **Rule-based access control**: Access control rules are defined based on predefined conditions or criteria. For example, granting access

during specific timeframes or based on certain data conditions.

When a user is authenticated, that only means that the user is known to the system and has been recognized by it. It doesn't mean that the user is free to do whatever she wants in said system. The next logical step in securing an application is determining which actions the user can perform and which resources she can access. If the user doesn't have the proper permissions, she cannot carry out that particular action. This is the work of the authorization process. In the most common case, the authorization process compares the user's set of permissions against the permissions required to execute a particular action in the application, and if a match is found, access is granted. On the other hand, if no match is found, access is denied. Figure **1-3** shows the authorization mechanism.

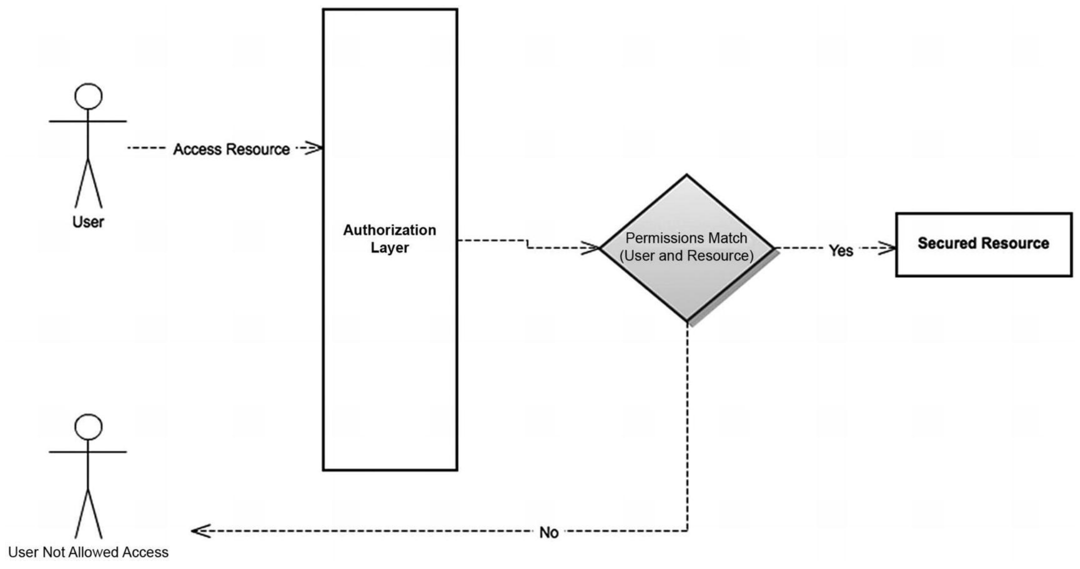

Figure 1-3 Simple authorization process: the authenticated user tries to access a secured resource

ACLs

An access control list (ACL) manages access rights and permissions to specific resources or objects within an application. It is typically used in conjunction with authorization. An ACL defines who has access to a particular resource and what actions they can perform on that resource. It consists of a list of users or groups and their corresponding permissions (read, write, execute, etc.) for specific resources.

ACLs are part of the authorization process explained in the previous section. The key difference is that ACLs normally work at a finergrained level in the application. ACLs are a collection of mappings between resources, users, and permissions. With ACLs, you can establish rules like "User John has administrative permission on the blog post X" or "User Luis has read permission on blog post X." You can see the three elements: user, permission, and resource. Figure 1-3 shows how ACLs work; they are just a special case of the general authorization process.

Authentication and Authorization: General Concepts

This section introduces and explains fundamental security concepts that you will come across frequently in the book.

- **User**: The first step in securing a system from malicious attackers is identifying legitimate users and allowing access to them alone. User abstractions are created in the system and given their own identity. They are the users that are later allowed to use the system.
- **Credentials**: Credentials are the way a user proves who they are. Normally, in the shape of passwords (certificates are also a common way of presenting credentials), they are data that only the owner of it knows.
- **Role**: In an application security context, a role can be seen as a logical grouping of users. This logical grouping is normally done so the grouped users share a set of permissions in the application to access certain resources. For example, all users with the admin role have the same access and permissions to the same resources. Roles are a way to group permissions to execute determined actions, making users with those roles inherit such permissions.
- **Resource**: Any part of the application you want to access that needs to be properly secured against unauthorized access—for example, a URL, a business method, or a particular business object.
- **Permissions**: The access level needed to access a particular resource. For example, two users may be allowed to read a particular

document, but only one can write to it. Permissions can apply to individual users or users that share a particular role.

- **Encryption**: It allows you to encrypt sensible information (normally passwords, but it can be something else, like cookies) to make it incomprehensible to attackers even if they get access to the encrypted version. The idea is that you never store the plain text version of a password but instead store an encrypted version so that nobody but the owner knows the original one.
  The following describes types of encryption algorithms.

- **One-way encryption**: These algorithms, referred to as *hashing algorithms*, take an input string and generate an output number known as the *message digest*. This output number cannot be converted back into the original string. This is why the technique is referred to as *one-way encryption*.

  For example, let's suppose the requesting client encrypts a string and sends the encrypted string to the server. The server may have access to the original information from a previous registration process, for example, and if it does, it can apply the same hash function. Then, it compares the output from this hashing to the value sent by the client. If they match, the server validates the information.

  Figure 1-4 shows this scheme. Usually, the server doesn't even need the original data. It can simply store the hashed version and then compare it with the incoming hash from the client.

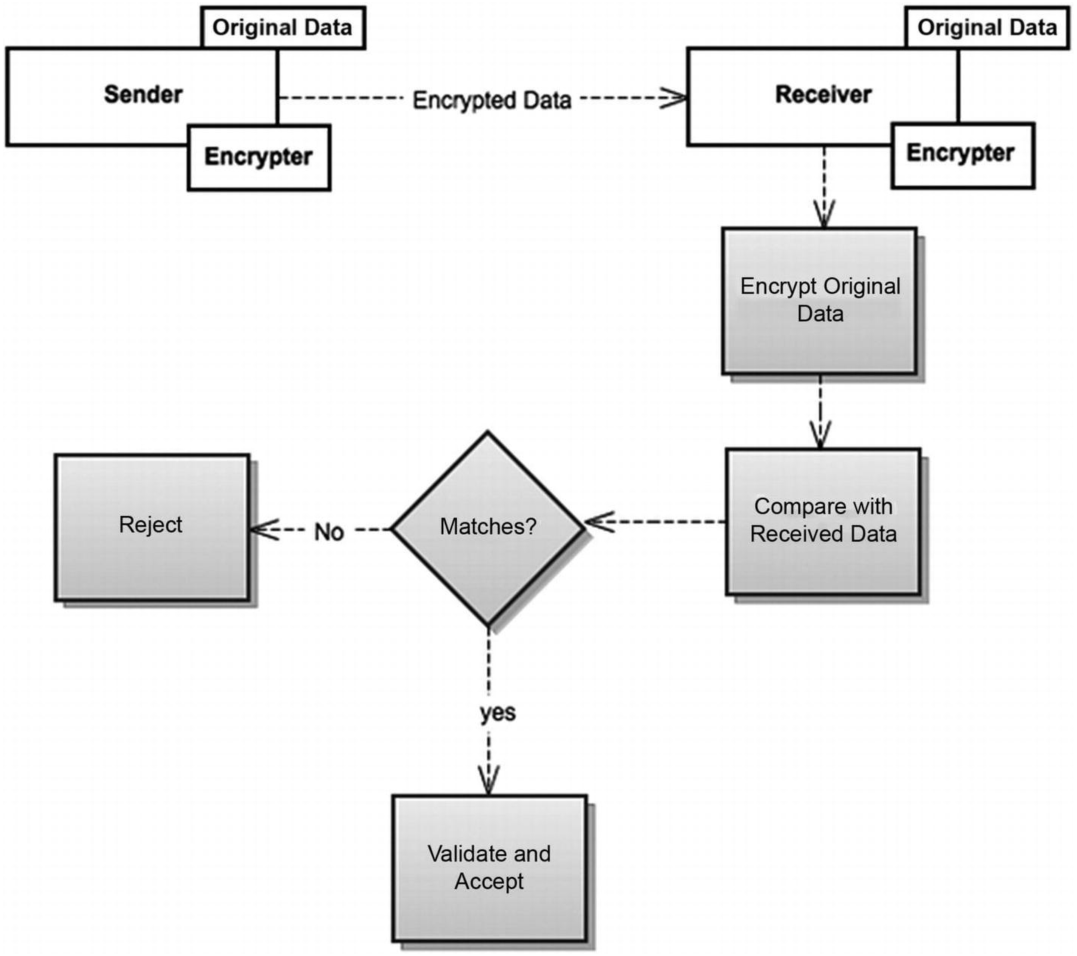

Figure 1-4 One-way encryption or hashing

- **Symmetric encryption**: These algorithms provide two functions: encrypt and decrypt. A string of text is converted into an encrypted form and then can be converted back to the original string. In this scheme, a sender and a receiver share the same keys to encrypt and decrypt messages on both ends of the communication. One problem with this scheme is how to share the key between the endpoints of the communication. A common approach is to use a parallel secure channel to send the keys.
  - **Key**: Symmetric encryption uses a single shared secret key for encryption and decryption. This means that both the sender and the recipient use the same key.
  - **Speed**: Symmetric encryption algorithms are generally faster and more efficient than asymmetric encryption algorithms.
  - **Use case**: Symmetric encryption is commonly used for securing large amounts of data, such as file encryption or secure communication between two parties who already share a secret key.
- Figure 1-5 shows symmetric encryption at work.

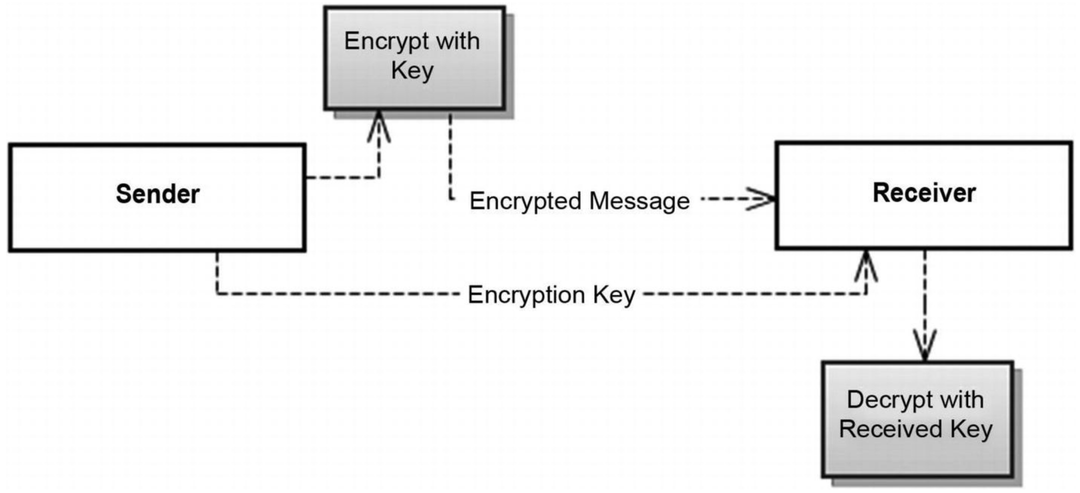

Figure 1-5 Symmetric encryption: the two endpoints share the same encryption/decryption key

- **Public key cryptography**: These techniques are based on asymmetric cryptography. In this scheme, a different key is used for encryption than for decryption. These two keys are referred to as the *public key*, which is used to encrypt messages, and the *private key*, which is used to decrypt messages. The advantage of this approach over symmetric encryption is that there is no need to share the decryption key, so no one but the intended receiver of the information can decrypt the message. The following describes a normal scenario.
  - The intended recipient of messages shares her public key with everyone interested in sending information to her.
  - A sender encrypts the information with the receiver's public key and sends a message.
  - The receiver uses her private key to decrypt the message.
  - No one else can decrypt the message because they don't have the receiver's private key.

The following defines the key, speed, and use case for asymmetric or PKI encryption.

- **Key**: Asymmetric encryption uses a pair of keys—a public key and a private key. The public key is freely available to anyone, while the owner keeps the private key secret.
- **Encryption and decryption**: The public key is used for encryption, while the private key is used for decryption. This means the

data encrypted with the public key can only be decrypted with the corresponding private key.

- **Security**: Asymmetric encryption provides a higher level of security because the private key is not shared or transmitted.
- **Use case**: Asymmetric encryption is commonly used for secure key exchange, digital signatures, and secure communication between parties who don't have a pre-shared secret key.

Figure 1-6 shows the public key cryptography scheme.

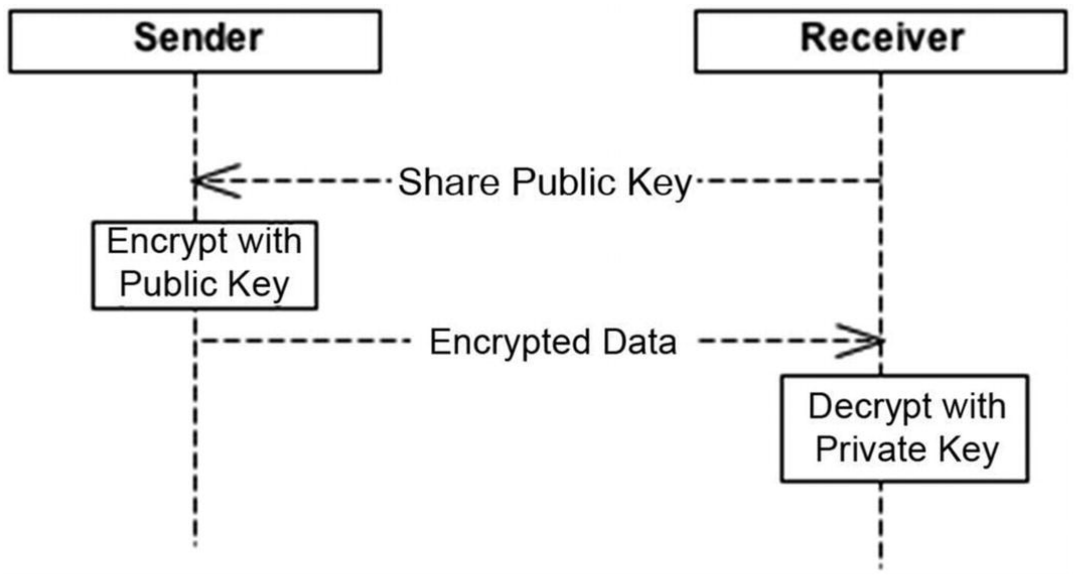

Figure 1-6 Public key cryptography

The use of encryption achieves, among other things, two other security goals.

- **Confidentiality**: Potentially sensitive information belonging to one user or group of users should be accessible only to this user or group. Encryption algorithms are the main helpers in achieving this goal.
- **Integrity**: Data sent by a valid user shouldn't be altered by a third entity on its way to the server or in its storage. This is normally accomplished through one-way cryptographic algorithms that make it almost impossible to alter an input and produce a corrupted message whose encrypted hash is the same as the original message (thus deceiving the receiver into thinking it is valid).

In practice, a combination of symmetric and asymmetric encryption is often used in *hybrid encryption*. In hybrid encryption, symmetric encryption encrypts the actual data, while the symmetric key is encrypted using the recipient's public key (asymmetric encryption). This approach combines the efficiency of symmetric encryption with the security and flexibility of asymmetric encryption.

What to Secure

Not every part of the application requires a strong security model or any security. If, for example, one part of your application is supposed to serve static content to everyone interested, you can simply serve this content. There probably are no security concerns to handle here.

Anyway, when starting to work on a new application, you should think about the security constraints that your application will have. You should think about concerns like those in the following list and whether they apply to your particular use case.

- **Identity management**: Your application will likely need to establish the users' identities. Usually, your application will do different things for different users, so you need a way to associate users with certain functionality. You also need to protect each user's identity information so it can't be compromised.
- **Secured connections**: In an Internet environment, where anyone in the world can potentially access your system and eavesdrop on other users accessing your system, you most likely want to secure the communication of sensitive data using some kind of transport layer security, such as SSL.
- **Sensitive data protection**: Sensitive data needs to be protected against malicious attacks. This applies to the communication layer, individual message transmission, and credentials data stores. Encryption should be used in different layers to achieve the most secure application possible.

Additional Security Concerns

There are many more security concerns than the ones explained so far. Because this is a Spring Security book and not a general application-security book, it covers only things related to Spring Security. However, we think it is important that you understand that there are many more security concerns than those addressed directly by Spring Security. The following is a quick overview of some of the most common ones. This is only intended to make you aware of their existence, and we recommend you consult a different source (such as a general software security textbook) to better understand all these concerns.

- **SQL (and other code) injection**: Validating user input is vital to application security. If data is not validated, an attacker could write any string as input (including SQL or server-side code) and send that information to the server. If the server code is not properly written, the attacker could wreak significant havoc because she could execute any arbitrary code on the server.
- **Denial-of-service attacks**: These attacks make the target system unresponsive to its intended users. This is normally done by saturating the server with requests to utilize all the server's resources and make it unresponsive to legitimate requests.
- **Cross-site scripting and output sanitation**: An injection can be done where the target is the client part of the application. The idea is that the attacker can make an application return malicious code inside the web pages returned and thus execute it in the user's browser. This way, the attacker invisibly executes actions using the real user's authenticated session.
- **Unauthorized access**: This occurs when an individual or entity gains unauthorized entry to a system, network, or data. It can result in data breaches, theft of sensitive information, or unauthorized manipulation of systems.
- **Malware and ransomware**: *Malware* refers to malicious software designed to disrupt, damage, or gain unauthorized access to systems. *Ransomware* is a specific type of malware that encrypts data and demands a ransom for its release. Both malware and ran-

somware can lead to data loss, financial loss, and operational disruptions.

- **Phishing and social engineering**: *Phishing* involves fraudulent attempts to obtain sensitive information, such as passwords or financial details, by disguising it as a trustworthy entity via emails, phone calls, or websites. *Social engineering* exploits human vulnerabilities to manipulate individuals into revealing confidential information or performing actions that can compromise security.
- **Data breaches**: These breaches occur when unauthorized individuals access sensitive or confidential data, such as personal information, credit card details, or intellectual property. Data breaches can result in financial loss, reputational damage, and legal consequences.
- **Insider threats**: These threats involve individuals with authorized access to systems or information who misuse their privileges for malicious purposes. This can include intentional data theft, sabotage, or unauthorized disclosure of sensitive information.
- **Weak authentication and password security**: Weak or easily guessable passwords, inadequate authentication mechanisms, and insufficient password management practices can leave systems vulnerable to unauthorized access and compromise.
- **Vulnerabilities and software exploits**: Software vulnerabilities, such as unpatched or outdated systems, can be exploited by attackers to gain unauthorized access, inject malware, or manipulate systems. It is crucial to promptly apply security patches and updates to mitigate these risks.
- **Cloud security**: Organizations utilizing cloud services must address specific security concerns, including data privacy, data segregation, access control, and cloud provider vulnerabilities.
- **IoT security**: The proliferation of IoT devices introduces new security challenges, including insecure device configurations, lack of encryption, and vulnerabilities in IoT networks. Compromised IoT devices can be used to launch attacks or gain unauthorized access to networks.

Addressing these IT security concerns requires a comprehensive and multi-layered approach, including implementing strong security controls, regular security assessments, user education and awareness, incident response planning, and adherence to security best practices.

Java Options for Security

Java and Java EE out-of-the-box security solutions are very comprehensive. They cover areas ranging from a low-level permission system through cryptography APIs to an authentication and authorization scheme.

The list of security APIs offered in Java is very extensive, as the following list of the main ones shows.

- **Java Cryptography Architecture (JCA)** supports cryptographic algorithms, including hash-digest and digital signature support.
- **Java Cryptographic Extensions (JCE)** mainly provides facilities for the encryption and decryption of strings and secret key generation for symmetric algorithms.
- **Java Certification Path API (CertPath)** provides comprehensive functionality for integrating the validation and verification of digital certificates into an application.
- **Java Secure Socket Extension (JSSE)** provides a standardized set of features to support SSL and TLS protocols, both client and server, in Java.
- **Java Authentication and Authorization Service (JAAS)** provides a service for authentication and authorization in Java applications. It provides a pluggable system where authentication mechanisms can be plugged in independently to applications.
- **Java Generic Security Services (Java GSS-API)** securely exchanges messages between communicating applications. “Introduction to JAAS and Java GSS-API Tutorials” is a series of tutorials demonstrating various aspects of using JAAS and Java GSS-API.

The JDK is divided into modules. The following modules contain security APIs.

- java.base
- java.security.jgss
- java.security.sasl
- java.smartcardio
- java.xml.crypto
- jdk.jartool
- jdk.security.auth
- jdk.security.jgss

For the entire list of Java release 20 security APIs, please refer to https://docs.oracle.com/en/java/javase/20/security/security-api-specification1.html.

Figure 1-7 shows the Java platform security architecture and elements.

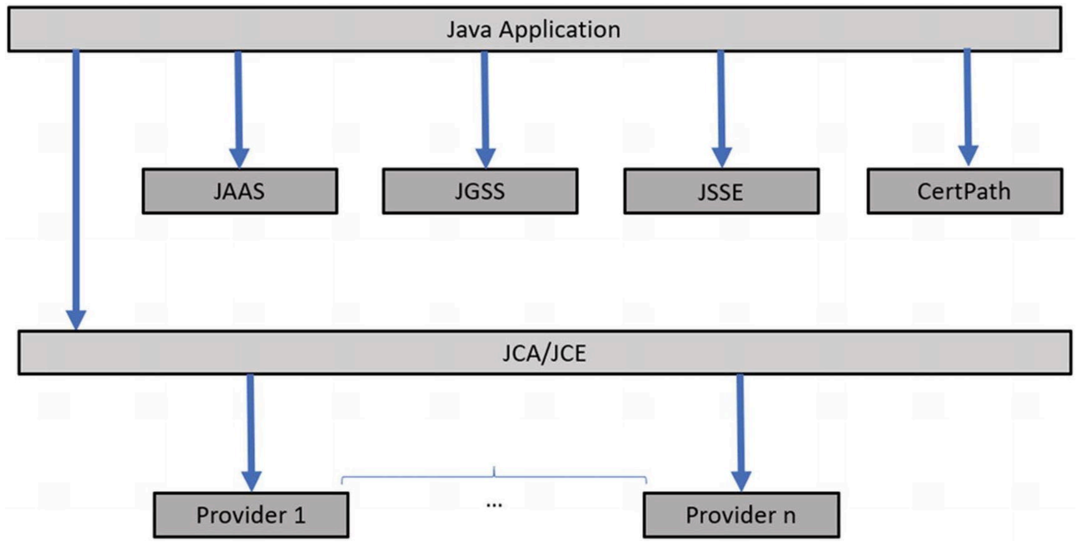

Figure 1-7 Java platform security architecture and elements

Spring Security's main concerns are in the authentication/authorization realm. So, it overlaps mainly with the JAAS Java API, although they can be used together, as you will see later in the book. Most of the other APIs are leveraged in Spring Security.

For example, CertPath is used in X509AuthenticationFilter, and JCE is used in the spring-security-crypto module.

Summary

This chapter introduced security from a general point of view down to defense-in-depth mechanisms. It explained in a very abstract way the main concerns in IT security, especially from an application point of view. It also briefly described the main Java APIs that support security at different levels.

You can see that this chapter was a very quick overview of security concerns. It is beyond the scope of this book to go any further than this on general topics, although some of them are studied in more depth when they apply to Spring Security. This is nothing like a comprehensive software security guide, and if you are interested in learning more about software security in general, you should consult the specialized literature. The next chapter introduces Spring Security.

Footnotes

[1] LDAP is explained in some detail in Chapter 8, where various authentication providers are covered.