Pro Spring Security

Securing Spring Framework 5 and Boot 2-based Java Applications

Second Edition

Carlo Scarioni
Massimo Nardone

Apress®

Pro Spring Security

Securing Spring Framework 5 and Boot 2-based Java Applications

Second Edition

Carlo Scarioni
Massimo Nardone

Apress®

Pro Spring Security: Securing Spring Framework 5 and Boot 2-based Java Applications

Carlo Scarioni
Surbiton, UK

Massimo Nardone
HELSINKI, Finland

ISBN-13 (pbk): 978-1-4842-5051-8
ISBN-13 (electronic): 978-1-4842-5052-5
https://doi.org/10.1007/978-1-4842-5052-5

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail editorial@apress.com; for reprint, paperback, or audio rights, please email bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484250518. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

I would like to dedicate this book to the memory of my beloved late mother, Maria Augusta Ciniglio. Thanks, Mom, for all the great things you have taught me, for making me a good person, for making me study to become a computing scientist, and for the great memories you left me. You will be loved and missed forever. I love you, Mom. RIP.

—Massimo

Table of Contents

About the Authors

Carlo Scarioni is a passionate software developer, motivated by learning and applying innovative and interesting software development tools, techniques, and methodologies.

His professional objectives are the following: to be in a technology-oriented enterprise where the technical staff is the soul of the company; to be on an important IT team; to be able to design and develop state-of-the-art software; to be able to apply new knowledge every day, in innovative ways, and with a great degree of freedom; to architect, design, and develop software that uses the best practices of the field; and to play with the latest technologies, learn every day, and participate in the research and innovation of software products. His specialties are TDD, object-oriented principles and design patterns, Java/JEE, Spring, application servers, SQL and NoSQL (MongoDB), multithreading, messaging, Enterprise Integration Patterns, Ruby, and RoR. His certifications are Sun Certified Enterprise Architect (Part I), Sun Certified Java Programmer, Sun Certified Business Component Developer, SpringSource Certified Professional, and IBM SOA Certified Associate.

Massimo Nardone has more than 24 years of experience in security, web/mobile development, cloud, and IT architecture. His true IT passions are security and Android. He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years. He holds a Master of Science degree in Computing Science from the University of Salerno, Italy.

His technical skills include security, Android, cloud, Java, MySQL, Drupal, Cobol, Perl, web and mobile development, MongoDB, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, etc.

He has worked as a visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (in the PKI, SIP, SAML, and Proxy areas). He currently works as an Executive Security Consultant, OT Security Leader, and Chief Information Security Office (CISO) for IBM, and he is member of ISACA Finland Chapter Board.

Massimo is the coauthor of *Pro JPA in Java EE 8* (Apress, 2018), *Beginning EJB in Java EE 8* (Apress, 2018), and *Pro Android Games* (Apress, 2015); he also reviewed more than 45 IT books for different publishers.

About the Technical Reviewer

Iuliana Cosmina is a Spring-certified Web Application Developer and a Spring-certified Spring Professional, as defined by Pivotal, the makers of Spring Framework, Boot, and other tools. She is the author of books with Apress on core Spring certification and Spring-certified web development. She is a software architect at Bearing Point Software and is an active coder and software contributor on GitHub, Stack Overflow, and more.

Acknowledgments

This book is definitely the work of more than one person. The people involved in the preparation of this book have brought so much experience and quality to the final version that the end product is many times better than if I had done all the work myself. Their input ranged from improving text style, to introducing better ways to present concepts, to performing code reviews and suggesting general improvements that have made this book a much better reading experience.

I am talking, of course, about the great people at Apress who have been with me along the full journey of writing this book. I'm talking about Steve Anglin, who initiated me into the project, kept an eye from afar on the progress of the book, and tried to make sure I kept on track as much as possible. I'm talking about Kevin Shea, who was my main editorial contact and made sure that I stayed on schedule with the book and helped with advice and support. I'm talking about Tom Welsh, who had the great responsibility of reading every chapter as I was writing it; he gave me great input on each section, including helping with my use of English grammar as well as ways to make the different parts more attractive to potential readers. I am talking about Manuel Jordan, who not only read every single chapter in a very detailed way, but also took on the laborious job of evaluating and executing every single line of code and making sure that the book provides code samples that can be reproduced by the readers in their own environments. His input is greatly appreciated, and it is the difference between having a full book or a half a book. There were, of course, many more people at Apress involved in the full review phases of the book, and I want to say "thank you for your help" to all of them.

I would like to also thank the creators, committers, and community of Spring and Spring Security for creating such an amazing piece of software and making it available to everyone. A big thank you to them for letting all developers share their knowledge and ways of work by freely distributing the source code of the different projects covered by the SpringSource umbrella. They make us all wiser and better developers.

Finally, I want to thank my wife for being with me all the time and motivating me to keep going forward.

—Carlo Scarioni

Many thanks go to my wonderful family—my wife, Pia, and my children, Luna, Leo, and Neve—for supporting me while I was working on this book. You are the most beautiful reason of my life.

I want to thank my beloved late mother, Maria Augusta Ciniglio, who always supported and loved me so much. I will love and miss you forever, my dearest mom.

Thanks to my beloved father, Giuseppe, and my brothers, Mario and Roberto, for your endless love and for being the best dad and brothers in the world.

I also want to dedicate this book to my late, dearest cousin, Gerardo "Amerigo" Nardone. You will be missed.

Thanks a lot to Steve Anglin and Matthew Moodie for giving me the opportunity to work as writer on this book, as well as to Mark Powers for doing such a great job during the editorial process and supporting me all the time, and of course the technical reviewers of this book for helping me to make a better book.

—Massimo Nardone

Introduction

Denying the impact of the Spring Framework in the Java world would be simply impossible. Spring has brought so many advantages to the Java developer that we could say it has made better developers of all of us. The good ones, the average ones. All of us.

The previous version of this book utilized Spring Security 3. Therefore, it is very important to note, in this new edition of the book, the most important changes from v3 to v5. Spring Security v5 is now part of Pivotal since SpringSource is no longer used.

Spring Framework 5 was published in September of 2017 and it can be considered the first major Spring Framework release since version 4 was released in December of 2013.

Spring's core building blocks of dependency injection and aspect-oriented programming are widely applicable to many business and infrastructure concerns, and certainly application security can benefit from these core functionalities. So this is Spring Security: an application-level security framework built on top of the powerful Spring Framework that deals mainly with the core security concepts of *authentication* and *authorization*, which are some of the fundamental functionalities of Spring Security v5.

Spring Security aims to be a full-featured security solution for your Java applications. Although its main focus is on web applications and the Java programming language, you will see that it goes beyond these two domains.

What we wanted to do in writing this book was to expose some of the internal works of Spring Security along with the standard explanations of how to use certain features. The idea is to teach beyond the basics of how to do something in particular, and instead focus on the plumbing inside the framework. We find that this is the best way of learning something: actually seeing how it is built in the core. That's not to say, of course, that the book doesn't cover basic setups and give quick, practical advice on using the framework, because it certainly does. The point is that instead of saying, "Use this to do that," we say, "This works like this… and this allows you to…." This is a point of view that only tools like Spring afford (because they are open source).

With that said, we suggest that the best way to use this book is to have the Spring Security source code checked out on your computer and go through the examples with both the code from the book and the code from Spring Security itself. This will not only help you understand each concept as it is introduced, but will also teach more than one good programming trick and good practice. We recommend this approach for studying any software whenever you have the chance. If the source code is out there, grab it. Sometimes a couple lines of code teach more than a thousand words. In this book, we will mainly introduce Spring Boot, analyze Spring Framework, and develop Java Web Applications with Spring Security v5.1.5, Java v11, and Servlet v4. Also, Spring Security v5 supports many different authentication mechanisms which are introduced and developed in this book, like Database (MongoDB and hsqldb), LDAP, X.509, OAuth 2/OpenID, WebSockets, SON Web Token (JWT), JAAS, and CAS. Web development frameworks like Grails and JRuby in the context of Rails and Scala are also introduced in this book.

Who This Book Is For

This book is written mainly for Java developers who use Spring in their work and need to add security to their applications in a way that leverages Spring's proven concepts and techniques. The book will also be helpful to developers who want to add web-layer security to their applications, even if those applications are not fully Spring-powered at their core. The book assumes you have knowledge of Java and some of its tools and libraries, such as Servlets and Maven. It also assumes that you know what you want to use security for and in what context you want to use it. This means, for example, we won't explain protocols like LDAP in much depth; instead, we'll concentrate on showing you how to integrate Spring Security with an LDAP user store. An in-depth knowledge of Spring is not essential because many of the concepts are introduced as we go along, but the more you understand about Spring, the more you are likely to get out of this book.

How This Book Is Structured

The book is divided into nine chapters that embody a progressive study of Spring Security. Starting from a summary of basic applications and an explanation of how the framework is structured, the content moves on to more advanced topics, such as using Spring Security in different JVM languages. The book follows a sequence that corresponds to the way this framework is normally used in real life.

The chapters in the book cover the following:

- **Chapter 1**: Introduces security in general and how to approach security problems at the application level
- **Chapter 2**: Introduces Spring Security v5, how to use it, when to use it, and all of its security functionalities
- **Chapter 3**: Introduces Spring Security with a simple example application that secures web access at the URL level
- **Chapter 4**: Provides a full introduction to the architecture of Spring Security, including the main components and how they interact with each other
- **Chapter 5**: Gives in-depth coverage of the web-layer security options available in Spring Security
- **Chapter 6**: Covers a wide array of authentication providers, including LDAP and JASS, which can be plugged into Spring Security
- **Chapter 7**: Covers access control lists (ACLs), which are used to secure individual domain objects, and how they fit into the general security concerns
- **Chapter 8**: Explains how to extend the core Spring Security functionality by making use of the many extension points supported by its modular architecture
- **Chapter 9**: Shows how to integrate Spring Security with different Java frameworks and some important JVM programming languages

Prerequisites

The examples in this book are all built with Java 11 and Maven 3.6.1. The latest Spring versions are used if possible. Spring Security 5.1.5 was the version used throughout the book. Tomcat Web Server v9 was used for the different web applications in the book, mainly through its Maven plugin, and the laptop used was a ThinkPad Yoga 360 with 8GB of RAM. All the projects were developed using the IntelliJ IDEA Ultimate 2019.2.

You are free to use your own tools and operating system. Because everything is Java based, you should be able to compile your programs on any platform without problems.

Downloading the Code

The code for the examples shown in this book is available via the Download Source Code button located at www.apress.com/9781484250518.

Contacting the Authors

You are more than welcome to send us any feedback regarding this book or any other subject we might help you with. You can contact Carlo Scarioni via his blog at http://cscarioni.blogspot.com, or you can send him email at carlo.scarioni@gmail.com. You can contact Massimo Nardone via email at massimonardonedevchannel@gmail.com.

CHAPTER 1

The Scope of Security

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

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

Of course, the growth of the Internet and the pursuit of reaching more people with our applications have opened more and more doors to cyber criminals trying to access these applications in illegitimate ways.

It is also true that good care is not always taken to ensure that a properly secured set of services is being 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 defense in depth (DiD) and its security layers. In general, DiD is a way to define how to develop the cybersecurity of the IT infrastructure by defining how all the defensive mechanisms are layered in order to protect and secure data and information. A failing DiD or too weak development might be a consequence of a cybersecurity attack on the IT infrastructure.

Let's understand a bit more about the mechanisms part of DiD. First of all, DiD is made of three major controls:

- **Administrative controls**: Policies, procedures, guidelines, awareness programs, etc.
- **Technical controls**: Firewalls, antivirus, intrusion prevention systems (IPS), etc.
- **Physical Controls**: Network and server rooms, video surveillance, etc.

C. Scarioni and M. Nardone, *Pro Spring Security*, https://doi.org/10.1007/978-1-4842-5052-5_1

Figure 1-1 shows the typical DiD mechanisms that define the IT infrastructure security layers.

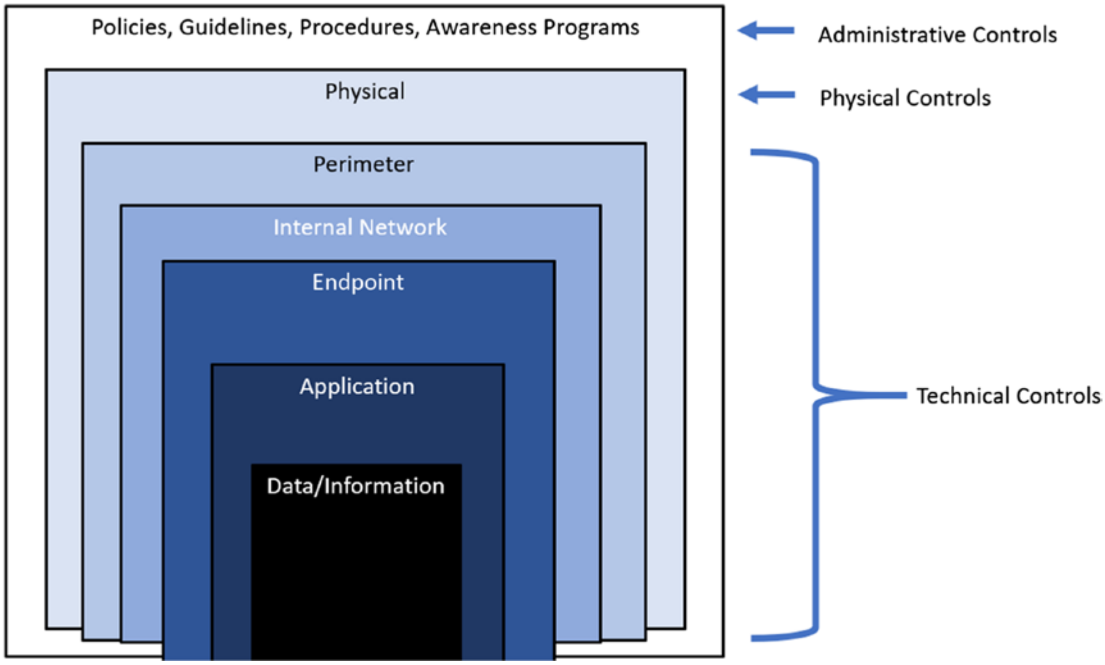

***Figure 1-1.** DiD mechanisms and IT infrastruxture layers*

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 clearly not enough in the vast majority of cases, as traffic at this level is normally allowed to enter the publicly open ports of your various exposed services with no restriction at all. Different attacks can be targeted at these open services, as attackers can execute arbitrary commands that could compromise your security constraints. There are tools like the popular `nmap` (http://nmap.org/) that can be used to scan a machine to find open ports. The use of such tools is an easy first step to take 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 perhaps, 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 main focus of this book will be on the application layer. The application security layer refers to all the constraints we establish in our applications to make sure that only the right people can do only 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.

In this book, we will cover 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 will help you understand better what the rest of the book will cover. They are the main concerns that Spring Security addresses to provide your applications with comprehensive protection against threats. In the following three subsections, we shall introduce

- Authentication
- Authorization
- ACLs

Authentication

The process of authentication 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 information it has stored—normally, in a database or LDAP[1] (Lightweight Directory Access Protocol) server. If the information provided by the user matches a record in the authentication server, the user is successfully authenticated in the system. The application will normally create an internal abstraction representing this authenticated user in the system. Figure 1-2 shows the authentication mechanism.

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

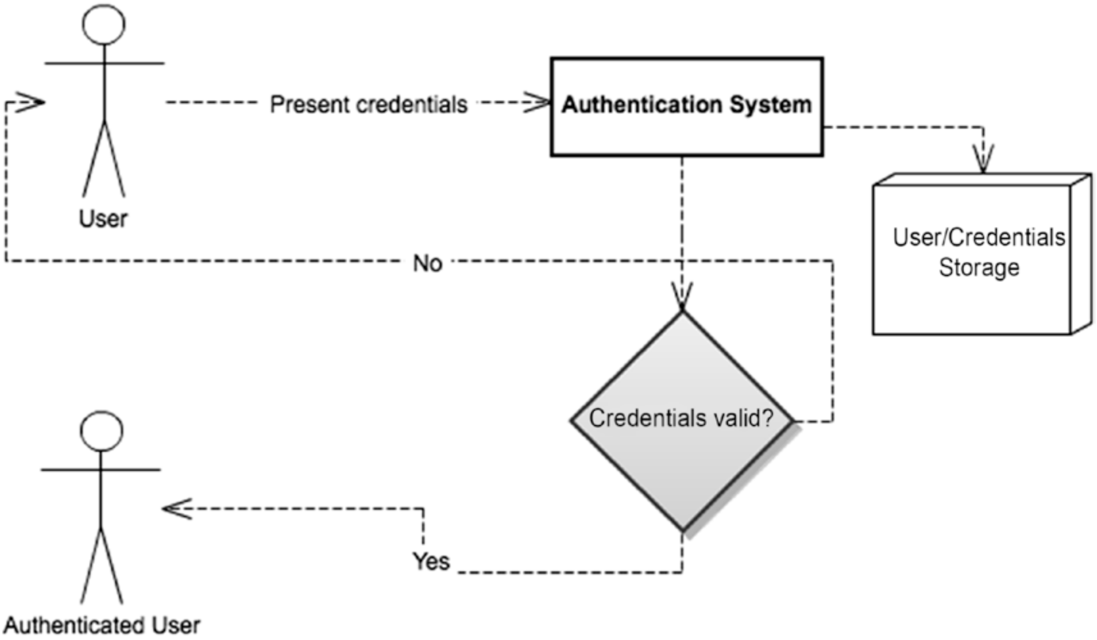

Figure 1-2. *Simple, standard authentication mechanism*

Authorization

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 to determine which actions the user is allowed to perform, and which resources she has access to, and make sure that 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.

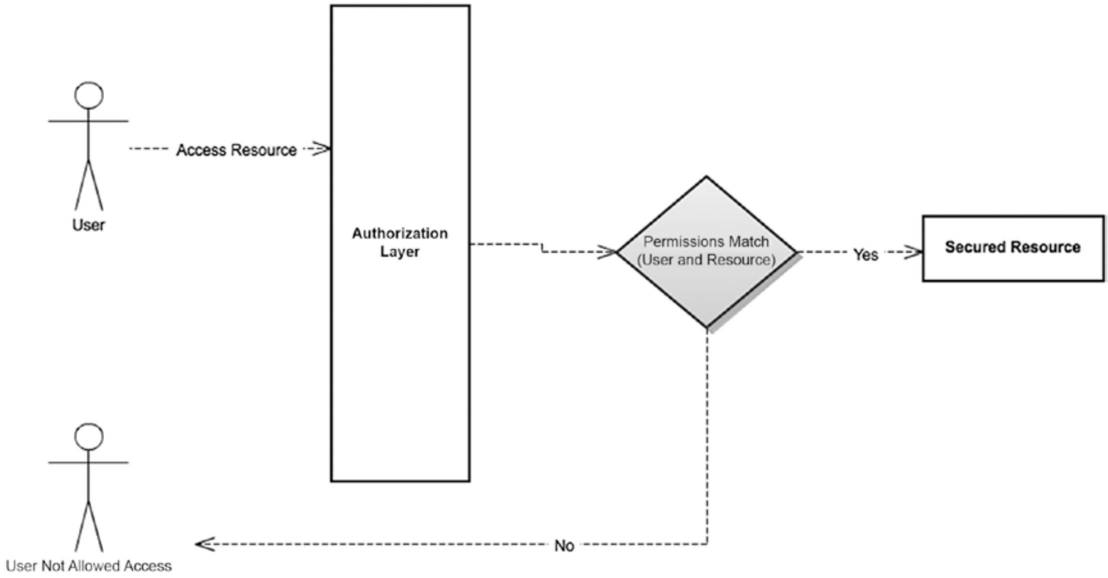

Figure 1-3. *Simple authorization process. The authenticated user tries to access a secured resource*

ACLs

Access control lists (ACLs) are part of the authorization process explained in the previous section. The key difference is that ACLs normally work at a finer grained level in the application. ACLs are simply 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

In this section, we shall introduce and explain some fundamental security concepts that you will be coming across frequently in the rest of the book:

- **User**: The first step in securing a system from malicious attackers is to identify legitimate users and allow access to them alone. User abstractions are created in the system and given their own identity. They are the users that will later be 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 role of admin will have the same access and permissions to the same resources. Roles serve simply as a way to group permissions to execute determined actions, making users with those roles inherit such permissions.
- **Resource**: By a *resource*, we mean, in this context, any part of the application that we want to access and that needs to be properly secured against unauthorized access—for example, a URL, a business method, or a particular business object.
- **Permissions**: Permissions refer to the access level needed to access a particular resource. For example, two users may be allowed to read a particular document, but only one of them is allowed to write to it. Permissions can apply either to individual users or to users that share a particular role.
- **Encryption**: This allows you to encrypt sensible information (normally passwords, but it can be something else, like cookies) so as 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 of such a password knows the original one. There are three main kinds of encryption algorithms:
    - **One-way encryption**: These algorithms, referred 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*. Here is the way to use it: A 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 to it. 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.

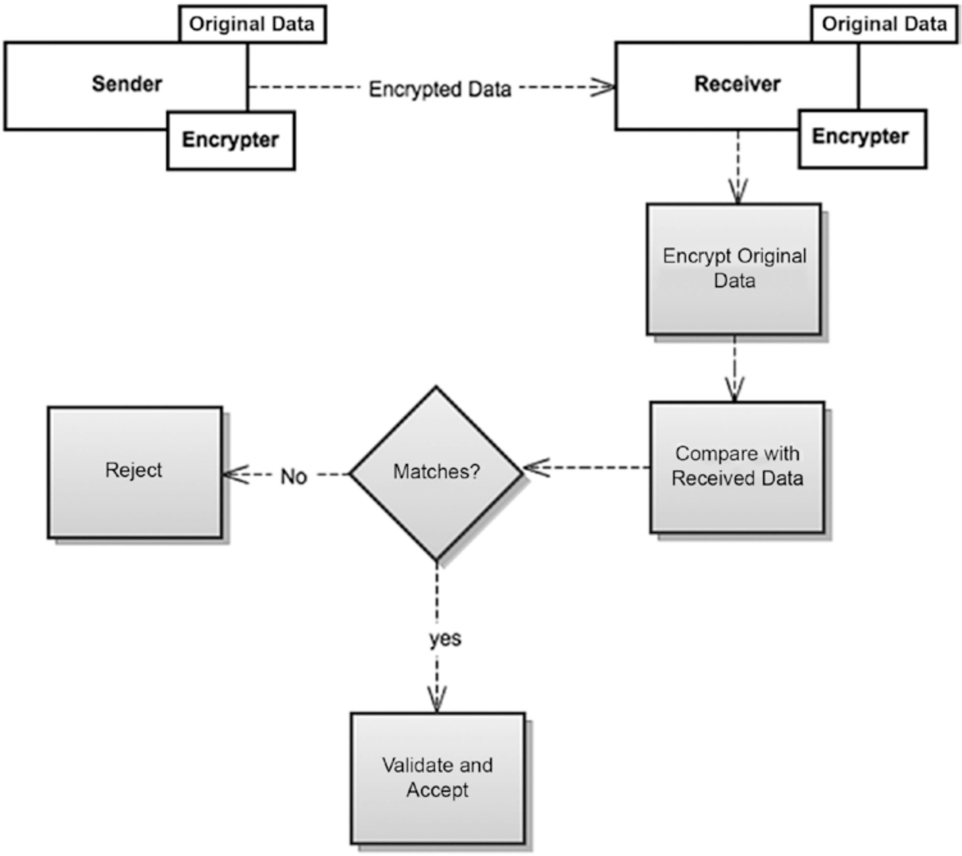

***Figure 1-4.** One-way encryption or hashing*

- **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 so that they can 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. Figure 1-5 shows symmetric encryption at work.

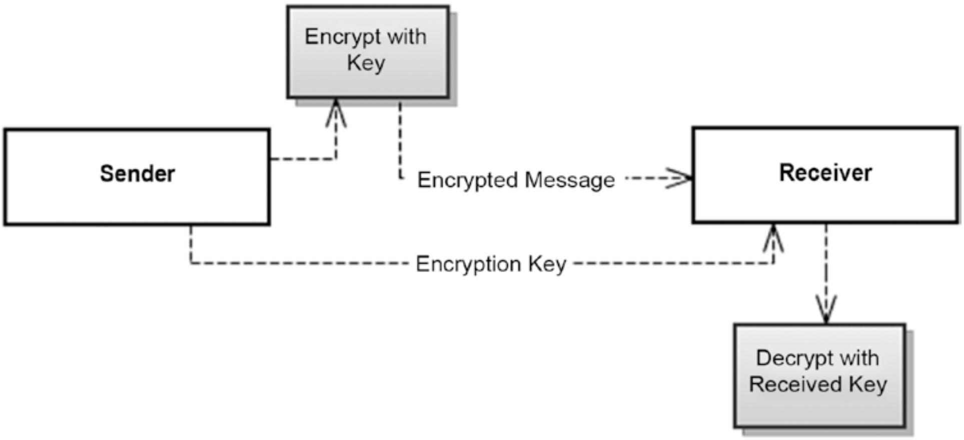

Figure 1-5. *Symmetric encryption. The two endpoints share the same encryption/decryption key*

- **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 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 is able to decrypt the message. So the normal scenario is the following:
    - 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 is able to decrypt the message because they don't have the receiver's private key.

Figure 1-6 shows the public key cryptography scheme.

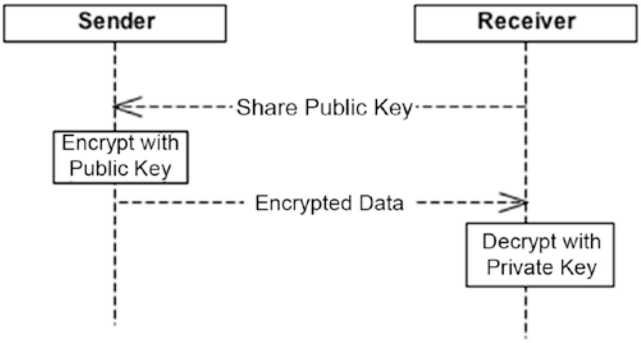

Figure 1-6. *Public key cryptography*

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 helper 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 the use of 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).

What to Secure

Not every part of the application requires a strong security model, or even any security at all. If, for example, one part of your application is supposed to serve static content to everyone interested in it, 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 or not they apply to your particular use case:

- **Identity management**: More than likely, your application will need to establish the identities of the different users that will be using it. 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 be sure to protect each user's identity information so that 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 will want to secure the communication of sensitive data using some kind of transport layer security, such as SSL.
- **Sensitive data protection**: Sensitive data will need to be protected against malicious attacks. This applies to the communication layer and to individual message transmission, as well as to credentials datastores. Encryption should be used in different layers to achieve the most secure application possible.

More 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, we will cover 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 gain a better understanding of all these concerns.

- **SQL (and other code) injection**: Validating user input is a very important part of application security. If data is not validated, an attacker could potentially write any kind of 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 consist of making the target system unresponsive to its intended users. This is normally done by saturating the server with requests so that it utilizes all the server's resources and makes it unresponsive to legitimate requests.
- **Cross-site scripting and output sanitation**: A kind of 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.

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)**: This API offers support for cryptographic algorithms, including hash-digest and digital-signature support.
- **Java Cryptographic Extensions (JCE)**: This API mainly provides facilities for the encryption and decryption of strings and also secret key generation for symmetric algorithms.
- **Java Certification Path API (CertPath)**: This API provides comprehensive functionality for integrating the validation and verification of digital certificates into an application.

- **Java Secure Socket Extension (JSSE)**: This API provides a standardized set of features to offer support for SSL and TLS protocols, both client and server, in Java.
- **Java Authentication and Authorization Service (JAAS)**: This API provides service for authentication and authorization in Java applications. It provides a pluggable system where authentication mechanisms can be plugged in independently to applications.

Please refer to this link for the entire list of Java 11 Security APIs: https://docs.oracle.com/en/java/javase/11/security/java-security-overview1.html#GUID-2EF0B3B8-9F3A-41CF-A7DA-63DB52180084.

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

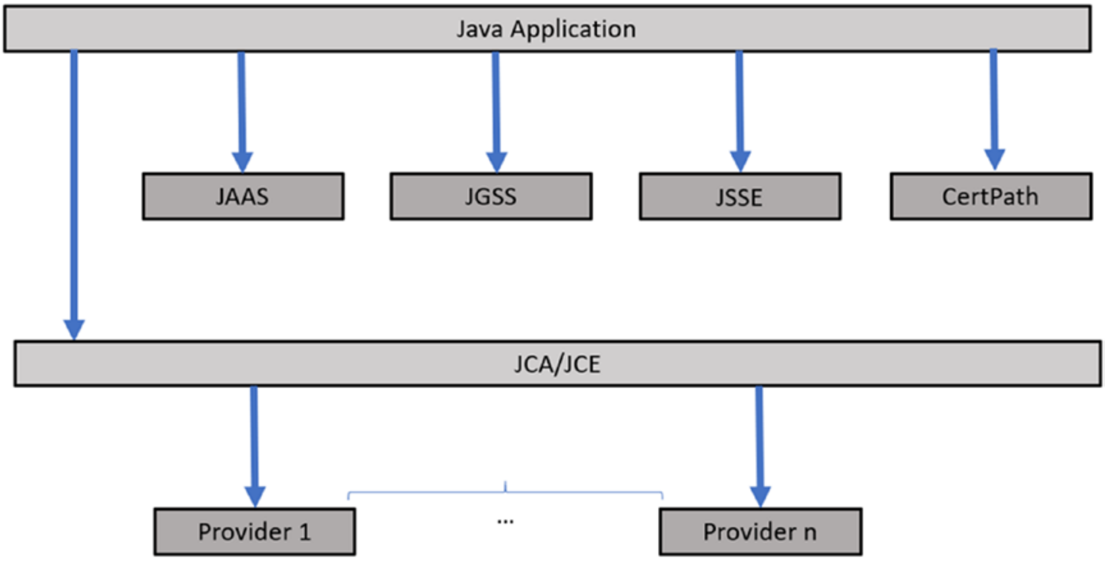

***Figure 1-7.** Java platform security architecture and elements*

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.