



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITY OF PADUA

Department of Information Engineering

Bachelor's Degree in Computer Engineering

Authentication proxy:
delegating authentication towards SPID,
the italian Public Digital Identity System

Bachelor thesis

Supervisor

Prof. Matteo Comin

Graduating

Enrico Biancotto

ANNO ACCADEMICO 2022-2023

Enrico Biancotto: *Authentication proxy:
delegating authentication towards SPID,
the italian Public Digital Identity System*, Bachelor thesis, © March 2023.

Abstract

SPID, Public Digital Identity System, is the Italian solution born in March 2013 in order to provide a single unified digital identity card, for the citizens, to access public and private services.

It is a worldwide example of a successful public-private partnership, and it is recognised for the open-source nature of the project, it also recognised for strong adoption among citizens.

The goal of this thesis is to provide a complete analysis of the SPID system, from the technical point of view, to the implementation in a Java Spring web application for a private company.

We will see the main components of the system, the authentication process, the security and privacy aspects, and the main problems that the system has to face.

Sommario

SPID, il Sistema Pubblico di Identità Digitale, è la soluzione italiana nata a Marzo 2013 per fornire un accesso unificato tramite identità digitali ai servizi pubblici e privati, messo a disposizione per i cittadini italiani.

È un esempio mondiale di una collaborazione vincente tra il settore pubblico e il privato, e viene riconosciuto per la natura open-source del progetto e per la forte adozione tra i cittadini.

Lo scopo di questa tesi è di offrire una analisi completa sul sistema SPID, sia da un punto di vista tecnico, sia da un punto di vista applicativo, implementando un sistema di autenticazione in una applicazione web Java Spring per una azienda privata.

Andremo a vedere le componenti principali del sistema, il processo di autenticazione, gli aspetti di sicurezza e privacy, e i principali problemi che il sistema deve affrontare.

Contents

1	Introduction	1
1.1	The company	1
1.1.1	Triskel S.r.l.	1
1.2	Internship description	1
1.2.1	Internship planning	2
2	Background knowledge	4
2.1	Authentication vs authorization	4
2.2	Authentication protocols	5
2.2.1	SAML	5
2.2.2	OpenID Connect	7
2.3	Differences between SAML and OIDC	11
2.3.1	Shared Features	11
2.3.2	Threat modeling	11
2.3.3	SAML and OIDC Conclusions	13
2.4	SPID, the italian Public Digital Identity System	14
2.4.1	What is SPID	14
2.4.2	History of SPID	15
2.4.3	Future of SPID	17
2.5	Java Spring web application	17
2.5.1	Java Spring overview	18
2.6	Satosa authentication proxy	19
3	Solution development	21
3.1	Architectural overview	21
3.2	Solution development	22
3.2.1	Java Spring web application	22
3.2.2	SaToSa authentication proxy	25
3.2.3	Testing	26
3.3	Findings	28
3.3.1	Achievements	28
3.3.2	Future improvements	28
3.3.3	Personal evaluation	29
	Acronyms	30
	Bibliography	31

List of Figures

1.1	Triskel S.r.l. logo	1
1.2	Internship schedule	3
2.1	SAML IdP Initiated Authentication Flow, credits [8]	6
2.2	SAML SP Initiated Authentication Flow, credits [8]	6
2.3	OAuth 2.0 Client to Server Authentication Flow, credits [10]	8
2.4	OAuth 2.0 Server to Server Authentication Flow, credits [10]	9
2.5	OpenID Connect Authentication Flow, credits [10]	10
2.6	Example of SPID login page in a Service Provider	14
2.7	SPID Authentication Flow	14
2.8	Java Spring Overview	18
2.9	Java Spring Fullstack Example	19
2.10	SaToSa One To Many Example	20
2.11	SaToSa Many-to-One Example	20
3.1	Architecture overview	21
3.2	SecurityConfig file	24
3.3	SPID SP Access Button Example	25
3.4	SaToSa server configuration	26

Chapter 1

Introduction

In this chapter we present the company where the internship took place, the problem that was faced and the internship planning.

1.1 The company

The company where the internship took place is called **Didanet S.r.l.**, a company born in **Triskel S.r.l.**

1.1.1 Triskel S.r.l.

Triskel is a company specialized in consulting and implementation of advanced system solutions, it proposes itself on the market as a highly specialized and professional partner for the management and maintenance of all your IT structures.



Figure 1.1: Triskel S.r.l. logo

1.2 Internship description

The main internship objective was to enable the **login** with **SPID** credentials in a **Java Spring** web application.

SPID is the italian Public Digital Identity System, allowing italian citizens to login with a single set of credentials. Over time more private companies have started to use SPID as an authentication method, as we can see from italian public registry, [1].

As it became more clear over time, when a private company wants to provide login with SPID for it's users, it has economic benefits to use an **authentication proxy** to aggregate authentication for internal services, and then expose a unique **single sign-on** (SSO) service to the SPID federation.

The authentication proxy is a service that acts as a **middleman** between the Service Provider (SP) and the SPID Identity Provider (IdP). The SP is a web application that

accepts and receives authentication assertions by a IdP. The SP developed in this internship is a Java Spring web application developed in Java Spring 4.x

To enter the SPID federation a company must register to the **AgID** (*Agenzia per l'Italia Digitale*) and then register to the **SPID** SP registry. When registering, the SPID Service Provider must pass a suite of tests carefully designed by the AgID to ensure the correct implementation of the SPID requirements, so it is necessary to have a proper testing environment to check our service provider.

1.2.1 Internship planning

The path to pursue became more apparent during the requirements analysis, the main objective of the internship was well defined, but the solution was not, hence the need to analyze the SPID requirements in the first weeks of the internship.

The final internship structure became this:

1. **Requirements analysis:** in the beginning of the internship I devoted most of my time to:
 - (a) Analyse the problem and understand the overall architecture;
 - (b) Study SPID requirements, implement Security Assertion Markup Language (SAML) and OpenID Connect (OIDC) protocol in Java Spring web application;
 - (c) Feasibility study of existing *authentication proxy* solutions.

2. **Software Development:** development of the authentication proxy, tested with an existing SP Java web app.

Software development pace was divided into:

- (a) Integrate and test the possible solutions:
 - * Test the solutions against existing Proof Of Concept (PoC) Java Spring web applications, solutions including SATOSA[2], Keycloak[3], Apereo CAS[4], Shibboleth[5];
 - * Compare authentication flows from a user point of view;
 - * Test the maintainability of the solutions from a developer perspective, bearing in mind company's current knowledge base in authentication systems;
 - (b) **SATOSA implementation:** implementation of the SATOSA authentication proxy;
 - (c) **Testing:** test the authentication flow with a Java Spring SP against a local testing IdP for SPID, see *italia/spid-saml-check* [6] official repository;
3. **Winter break:** Christmas holidays;
 4. **PoC application:** this is the final part of the internship, where the authentication proxy was integrated with a Java Spring web application. This phase was divided into:
 - (a) **Integration:** development of the web application (back-end and front-end);

- (b) **Debug/Improvements:** improvement of the PoC to better fit the future web application needs;

Gantt chart In the following Gantt chart (Figure 1.2), I have outlined the timings of each step:

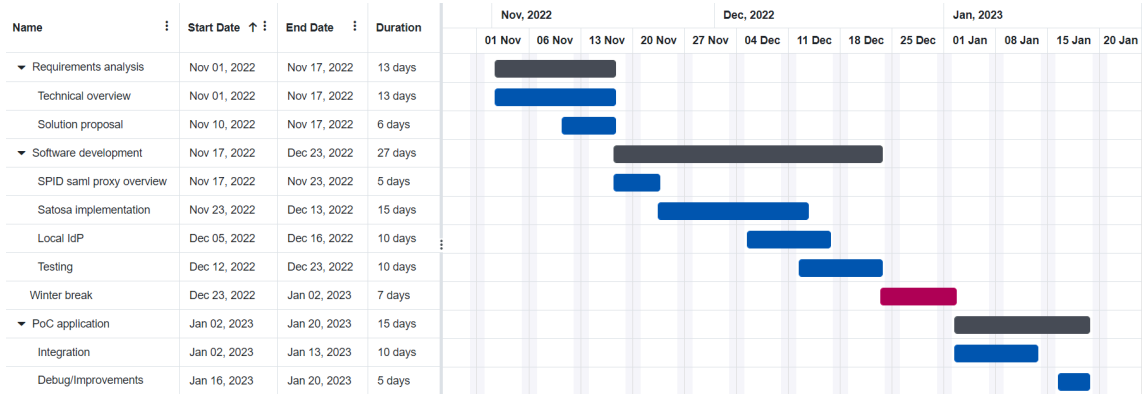


Figure 1.2: Internship schedule

Chapter 2

Background knowledge

This chapter presents the concepts needed to comprehend the content of this thesis fully. As we have stated in the previous chapter, the main objective of this thesis is to enable the login with SPID in a Java Spring web application, via an authentication proxy. To achieve this goal, we need to understand the technologies involved in the process and the concepts behind them.

2.1 Authentication vs authorization

In order to create an application to login with the SPID credentials, we need to understand what authentication and authorization are and what really is what we commonly refer to *login*.

Authentication is the process by which you verify that someone is who they claim they are.

Authorization is the process of establishing if the user (who is already authenticated) is allowed to have access to resources.

A machine can authenticate a human with any of the following methods:

- * **Something you know:** password, PIN, etc. Passwords are popular because of their cost and convenience. However, they are not the most secure because they can be stolen or guessed, without the use of a password generator.
- * **Something you have:** smartcard, otp, etc. A *smartcard* is a credit card-sized plastic card with an embedded microprocessor that can store cryptographic keys and do some computation. *OTP* is a one-time password, which is a password that is valid for only one login session or transaction.
- * **Something you are:** biometrics like fingerprint, voice, retina, etc. Biometrics allows us to be our key. Biometrics measures features that are unique, and that cannot be guessed, stolen or shared.
- * **Two-factor authentication:** a combination of the above, like something you know and something you have. Any authentication that requires two out of the three factors above is known as two-factor authentication.

- * **Authentication using cryptography:** Nowadays more and more passwordless authentication protocols are rising, for example, FIDO2, leveraging public and private key cryptography.

2.2 Authentication protocols

User authentication is a common problem in Computer Science and over time many protocols were born to solve it.

In this section, we will see two of these protocols: SAML2 and OpenID Connect, chosen because they are used in the SPID federation.

2.2.1 SAML

The Security Assertion Markup Language 2.0 (**SAML**) was released in March 2005 by the standards organization OASIS (the Organization for the Advancement of Structured Information Standards). [7]

SAML is a standard that defines a framework for exchanging security information between online business partners, used both for authentication and authorization.

SAML2 Terminology

- * **User agent** the user who wants to access a SP application;
- * **Identity Provider (IdP)** the entity that holds the user's identity information and enables the user to login and authorize transactions;
- * **Service Provider (SP)** the entity that wants authenticated users and needs authorized transactions;
- * **SAML Assertion** a SAML statement that contains information about a subject.

How Trust is established

SAML defines how to transfer user identity information between **trusted** parties, from an IdP, which allows the user to authenticate, to a SP, a web application for example.

A SP defines a set of **attributes** that wants to receive from an IdP (like the user's name, surname, email, etc.), and share those requirements with the IdP, along with the SP certificates, in a signed Metadata XML file. The IdP can also share its Metadata XML file.

This **metadata exchange** is what establishes the **trust** between the SP and the IdP.

SAML2 Web SSO Authentication Flow

There are two ways that the SAML authentication flow can be initiated:

- * **IdP-initiated flow:** The user agent acts as the transport mechanism for the SAML assertion.

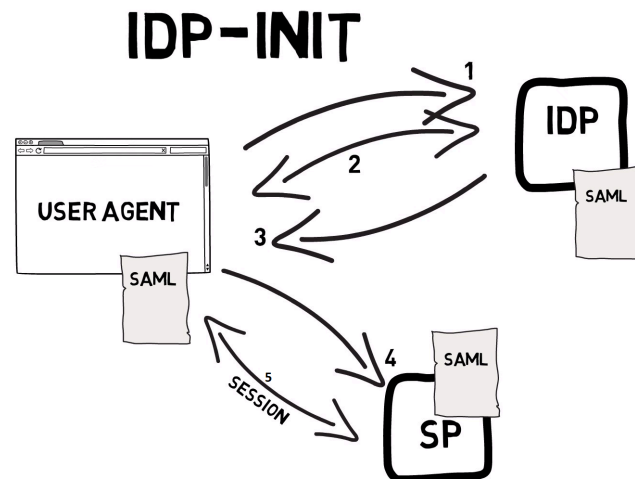


Figure 2.1: SAML IdP Initiated Authentication Flow, credits [8]

1. the user visits and authenticates with the IdP,
2. the user can request access to a service from the IdP,
3. the IdP then generates a SAML assertion and redirects the user to the SP, with the assertion in an HTTP POST message.
4. The SP map the IdP user information to its local database and starts the session with the user.

* SP-initiated flow:

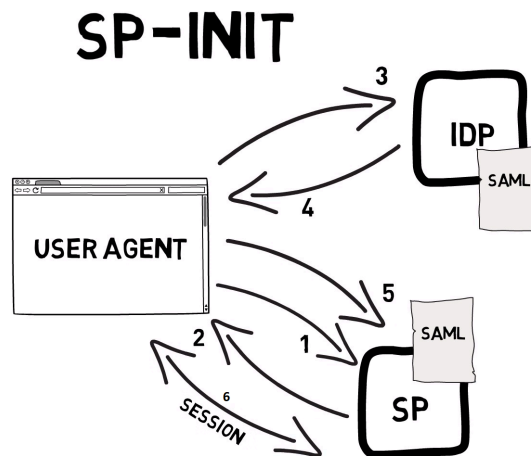


Figure 2.2: SAML SP Initiated Authentication Flow, credits [8]

1. the user visits the SP;
2. the user is not authenticated so the SP redirects the user to the IdP, with a SAML AuthNRequest, a request for authentication message;
3. the user authenticates with the IdP;

4. the IdP then generates a SAML assertion and redirects the user to the SP, with the assertion in an HTTP POST message.
5. The SP maps the IdP user information to its local database and starts the session with the user.

If the IdP is used for multiple SPs, the IdP is a Single Sign-On (SSO) service, and users can access multiple SPs, all sharing the same IdP, because the user is already authenticated with the IdP.

2.2.2 OpenID Connect

OpenID Connect "is a simple identity layer on top of the OAuth 2.0 protocol." [9] released in February 2014.

In order to fully understand OpenID Connect, we must first see what OAuth 2.0 is.

OAuth 2.0 is a protocol published in 2012 (RFC 6749), specifically designed to provide delegated authority. **Delegated authority** is the ability to grant authority to an application, so the application can act on your behalf;

OAuth 2.0 Terminology

- * **Resource Owner (RO)** the user who owns the data;
- * **Client** to whom you delegate authority;
- * **Authorization Server (AS)** generating the access token;
- * **Resource Server (RS)** where the user's data resides;
- * **OAuth 2.0 grant** the authorization given (or granted) by the user to the client;
- * **Access Token** a token issued by an Authorization Server in exchange for the OAuth 2.0 grant;
- * **Refresh Token** if the access token expires, the client can use the optional Refresh Token to obtain a new Access Token from the Authorization Server.

OAuth 2.0 benefits

- * Verify and change the level of access the user grants to the web application.
- * The user can easily update the password without breaking integrations
- * The user can revoke access by simply removing the web application access token.

OAuth 2.0 implementation

There are two ways that the OAuth 2.0 authentication flow can be defined:

1. Client to Server
2. Server to Server

Client to Server Let's say a user (Resource Owner) wants to use a web application (Client), which has a frontend and in the backend (Server) there are both the Resource Server (RS) and Authorization Server (AS).

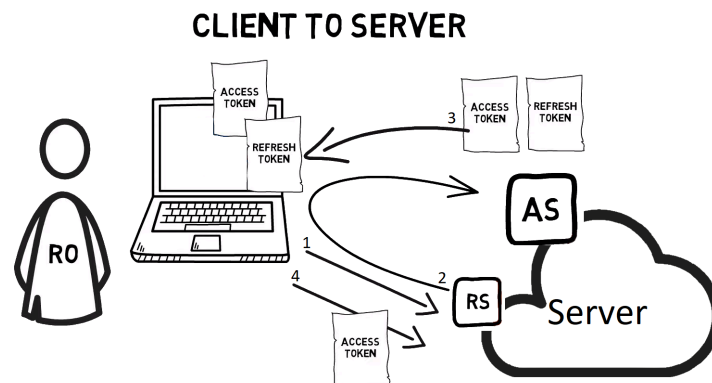


Figure 2.3: OAuth 2.0 Client to Server Authentication Flow, credits [10]

1. The user visits the web application for the first time, the frontend connects to the backend requesting a resource in the RS, but since the frontend doesn't present an access token, the backend redirects the user to the AS.
2. The user authenticates in the Authorization Server;
3. The Authorization Server generates two tokens: an access token and a refresh token and the backend sends those to the Front-end.
4. The Front-end now attaches the access token to the request to the backend, and the user has obtained access to the resources.
5. The user has delegated authority to the Front-end, which can now act on the user's behalf and access the resources without prompting for authentication.

Server to Server We have a situation with two applications: App 1 and App 2. App 1 is a Client in the OAuth 2.0 flow, while App 2 is both an Authentication and Resource Server.

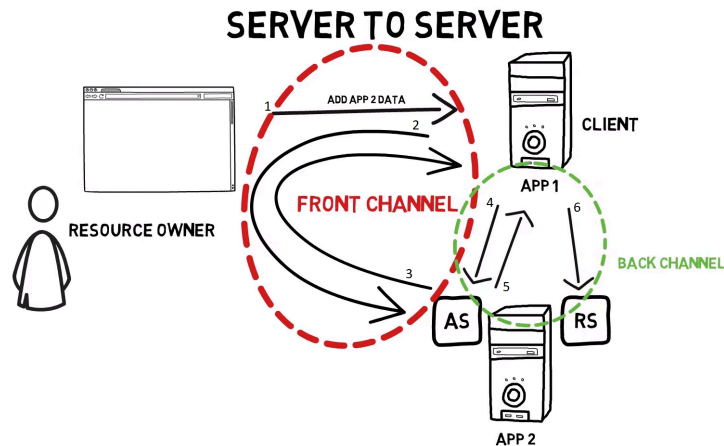


Figure 2.4: OAuth 2.0 Server to Server Authentication Flow, credits [10]

1. Front-Channel communication: the browser of the client is involved, so a malicious actor could potentially pick up the information, so the access token is sent in the next part, between App1 and App2 in a Server-to-Server communication.
 - * Before a normal user can use App 2 authentication in App 1, the admin of App 1 must register the application in App 2, specifying for example Application Name (an identifier) and Callback URLs (redirect URI).
 - * App 2 will generate a Client ID and a Client Secret, which will be used by App 1 in communications with App 2.
 - * Now a user can authenticate in App 1 using App 2 account, the user will visit App 1 and will be redirected to App 2 with the previously shared Client ID, Redirect URI, Response Type and a Scope for the authentication.
 - * The user is presented with a Scope and can change the level of access granted for App 1.
2. Back-Channel communication (Server to Server):
 - * When a user gives consent, an authorization Code is generated and sent from App 2 to App 1.
 - * App 1 can use the authorization code, along with the Client ID and Secret, to request an access token from App 2's Authorization Server (AS).
 - * The AS of App 2, upon successful verification, will respond with an access token and a refresh token. Usually, the access token is a Bearer token.
 - * The access token is used to authorize each request to App 2's Resource Server (RS).

The token format of the OAuth 2.0 token is not specified by the standard, but JWT Tokens are becoming the de-facto standard.

OpenID Connect

OAuth 2.0 is for delegated authentication, so the protocol does not specify rules for user authentication. To provide standard procedures for user authentication using OAuth 2.0 protocol, **OpenID Connect** was created.

OIDC requires OAuth 2.0 and it adds a standardized user authentication, adding an ID Token and a Userinfo endpoint to the OAuth 2.0 flow. ID Token which must be a JWT Token, and a Userinfo endpoint is as useful as the metadata in SAML, making it easier to establish integration and trust with *OpenID Connect Discovery*, helping with the configuration of the OAuth 2.0 integration.

OpenID Connect Authentication flow

Let's see an example of a OIDC Authentication flow with two applications: App 1 and App 2. As the example previously made in OAuth 2.0, App 1 is the Client, but since App 1 is relying on a claim for user authentication, App 1 is also a **Relying Party (RP)**.

App 2 is a **OpenID Provider (OP)**, with the Userinfo Endpoint and both the Authorization Server (AS) and Resource Server (RS).

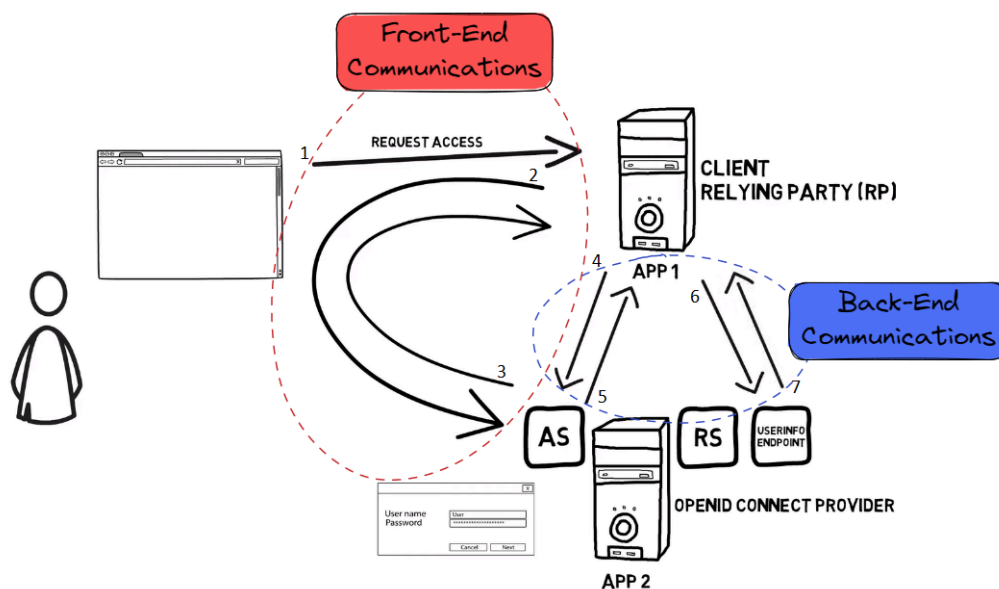


Figure 2.5: OpenID Connect Authentication Flow, credits [10]

1. Front-Channel:

- * The user visits App 1, the Relying Party (RP), which redirects the user to App 2, the Openid connect Provider (OP), with a similar message exchange as in OAuth 2 (Client ID, Redirect URI, Response Type and a Scope) but with Scope: OpenID Profile.
- * The user authenticates in the provider and an authorization code is sent back to App 1 (RP).

2. Back-Channel communication

- * App 1 sends to App 2 the authorization code, along with the Client ID and Secret, to request an access token and an ID token from App 2's Authorization Server (AS).

The ID Token contains some claims about the user, such as a Subject identifier for the user. The ID Token should be small, and if App 1 needs more information about the user, it can request it from the Userinfo Endpoint with the given access token.

2.3 Differences between SAML and OIDC

OIDC is the de-facto authentication standard in web applications, used at Google, Microsoft, Github and pretty much every private company.

Let's compare OIDC to SAML:

- * They have both high standards in **security**, but SAML had major security issues
- * Increased **ease of integration** in OIDC in different systems (web, mobile applications, IoT) thanks to a REST-like API interface;

2.3.1 Shared Features

Most security problems arise from a faulty implementation of the protocols, obviously, the simpler the protocol is, the easier is to identify and correct security issues.

SAML and OIDC use cryptographically signed tokens that support optional encryption, SAML with XML Encryption (XMLEnc). This step prevents disclosure of sensitive attributes after transportation.

2.3.2 Threat modeling

We have seen how these protocols work but when designing a protocol, it is important to consider the potential threats to the system. *Threat modeling* adopts the perspective of a malicious actor to see how it would be possible to damage the system.

SAML Threat model

SAML is mostly used in *Web Browser SAML/SSO Profile with Redirect/POST bindings*, relying on HTTPS channel with TLS to guarantee integrity and message confidentiality at the transport layer. As explained before, SAML is XML based, so it is exposed to the XML threat model and most of the security issues are related to the XML signature and encryption and XML file in general, which are generally complicated. Major security risks are:

- * **XML Signature Wrapping**

In 2012, in the article "On Breaking SAML: Be Whoever You Want to Be" [11] where signature verification algorithms can be circumvented by applying different Xml Signature Wrapping (XSW) attacks.

The XSW attack is possible when XML documents containing XML Signatures are processed by a weakly configured XML parser, where the signature validation and the business logic modules have different views on the document. In particular

a malicious actor crafts a document where the message structure is modified by injecting forged elements that don't invalidate the Signature, so the signature validation module validates the signature successfully, but this alteration is parsed by the application logic module, and results in an injection of arbitrary data.

A proper way to prevent this attack is to:

- Always perform **schema validation** prior to using it;
- Always **validate** the digital signature;
- Always use absolute **XPath** to get elements.

* **Malformed XML Documents**

"The W3C XML specification defines a set of principles that XML documents must follow to be considered well formed. When a document violates any of these principles, it must be considered a fatal error and the data it contains is considered malformed." [12]

* **Invalid XML Documents**

"Attackers may introduce unexpected values in documents to take advantage of an application that does not verify whether the document contains a valid set of values. Schemas specify restrictions that help identify whether documents are valid. A valid document is well formed and complies with the restrictions of a schema, and more than one schema can be used to validate a document." [12]

* **XML eXternal Entity injection (XXE)**

"XXE occurs when untrusted XML input containing reference to an external entity is processed by a weakly configured XML parser." [13]. An entity is a term that refers to multiple types of storage units, one of which is an external general/parameter parsed entity (external entity) that can access local or remote resources via a declared system identifier.

One method to prevent XXE is to disable Document Type Definition (DTD), which also prevents Denial of Service (DoS) attacks.

OIDC Threat model

OIDC is JSON based, and it is exposed to the OAuth 2.0 threat model. The article "OAuth 2.0 Threat Model and Security Considerations." [14] dedicates an entire chapter for Threat modeling for the OAuth 2.0 protocol, since this is not the scope of the thesis, we will just see the most relevant threats.

* **Cross-Site Request Forgery**

Cross-Site Request Forgery (CSRF) attack misuses the trust relationship between the client and the application. A victim is tricked into following a malicious URL, crafted by an attacker, which will redirect the victim to an Authorization Server. The victim is then tricked into granting access to the attacker's application, thus delegating authentication and allowing the access to victim's personal information to the attacker.

OAuth 2.0 recommends the use of a State parameter, a random string generated by the Client, in the Authorization Request to mitigate CSRF attacks. The client should store the State parameter and compare it to the State parameter returned in the Authorization Response. If the two values do not match, the client should reject the response.

- * **Clickjacking**

In a Clickjacking attack, an attacker constructs a malicious website loading the Authorization Server request for granting access as a transparent iframe layer, on top of the malicious page. The resource owner is then tricked into clicking carefully placed buttons on the webpage, thus granting access to the attacker.

To prevent this attack native applications should use external browsers instead of embedded ones, and web applications should use the `X-Frame-Options` header to prevent the page from being loaded in an iframe.

- * **Password phishing by Counterfeit Authorization Server**

When an attacker can intercept Client's requests, via DNS or ARP spoofing, it can redirect the Client to a malicious Authorization Server. The attacker can steal the user's credentials if the user is not careful to verify the authenticity of the website.

This attack can be prevented when the Authorization Server requires the use of TLS security for any requests.

2.3.3 SAML and OIDC Conclusions

Let's see the main differences between the two

- * As we have said before, most security issues arise from faulty implementations, so it is important to choose a protocol that is easy to implement and use correctly.
- * OIDC libraries need to validate more input values than SAML but OIDC, when properly implemented and used, can be as secure as SAML.
- * SAML is more complex than OIDC, major complexity is due to the XML file format and XML signature and encryption, instead of the JSON Web Signature and Encryption of the JWT Tokens in OIDC.
- * OIDC is considered more modern thanks to how it's well suited for newer use cases on heterogenous systems, like web or native mobile applications and IoT.

2.4 SPID, the italian Public Digital Identity System

In this paragraph, we will talk about what SPID is and which protocols it uses. We dive into the technical requirements of both SPID with SAML and with OpenID Connect

2.4.1 What is SPID

SPID is a pair of credentials (username and password) that "represents the digital and personal identity of each citizen"[15], with which he is recognized by the Public Administration and many Private SPID Service Providers.

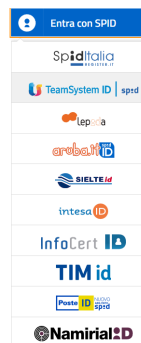


Figure 2.6: Example of SPID login page in a Service Provider

Under the hood, SPID was designed using the SAML protocol and it is currently under revision in order to also provide support for the OpenID Connect protocol. SPID restricts the SAML protocol with more rigid specifications, for example requesting a Signature, in HTTP Post messages, using RSA algorithm with RSA keys of at least 2048 bits and as digest algorithm at least SHA-256. The Response **must** be sent through HTTP Post bindings.

Here an example of the authentication flow in SPID (using SAML):

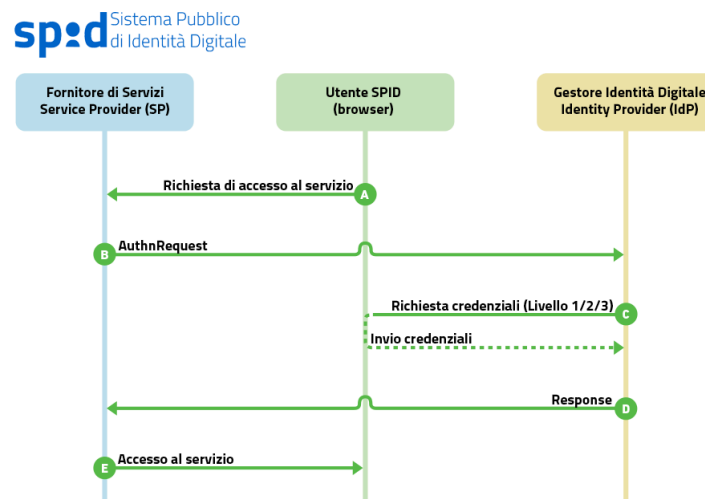


Figure 2.7: SPID Authentication Flow

2.4.2 History of SPID

As we can see from italian open data about SPID[16], the first SPID Service Provider was born in 2015, and the first SPID Identity Provider in 2016. In April 2016 there were 38700 SPID credentials, compared to the over 33 million SPID credentials in January 2023.

SPID was born in 2013 from the proposal of Stefano Quintarelli and colleagues Francesco Caio, Andrea Rigoni. Quintarelli was the head of head of AgID (the Agency for Digital Italy) at that time.

In 2014 the European Union, introduced the eIDAS regulation, which was the first European regulation on electronic identification and trust services for electronic transactions in the internal european market, allowing SPID to be used in any european state. eIDAS also uses the SAML protocol and all official SPID implementations also generates an eIDAS compatible metadata.

SPID was possible because of the ideas that were shared among the italian community since the year 2001, when the first federated identity system was born called "PEOPLE" project. Also in 2005 prof. Pianciamore and Osnaghi wrote the specifications for the "SIRIAC" system (Registration Authentication and Communication Infrastructure Services) based on SAML 1.1.

In Lombardia Region, project SIRIAC was reused to create the "IdPC" system, a federated identity management system for National Services Card (CNS), which went live in April 2006.

Following, the "inter-regional" project ICAR went live, where there was an IdP based on SAML 1.1. From this project, *FedERa* was born, an authentication system to access online Federated services of Emilia-Romagna Region.

In 2012, there was an experience of many years in the management of federated identity systems based on SAML in the italian regions.

One of the great ideas of the SPID project was to delegate to private companies the management of the SPID identity credentials of the citizens, thus reducing the costs of the Public Administration. This information were retrieved from a comment of Daniele Crespi in Stefano Quintarelli Blog[17]

SPID Cost for a Service Provider

Spid prices are available on the official website [18]. SPID follows a *pay per user* model, free of charge to the users who wants to login with SPID. The Service Providers are responsible to pay for the service.

The SP cost is calculated based on the number of users and the type of service requested in a yearly billing period, cost invoiced from each identity provider.

SPID distinguishes login types based on:

1. The level of security used in the authentication:

- * SPID 1st level: Authentication using username and password.

- * SPID 2nd level: Authentication using a password and a One-Time Password (OTP) sent via SMS or an Authentication App given by the SPID Identity Provider.
- * SPID 3rd level: Authentication using a password and an external device to store the private key like a smartcard or a remote digital signature device (using Hardware Security Module HSM).

2. The data that a Service Provider requests:

- * Authentication mode: when an SP requests only registry attributes of the user: SPID ID, Fiscal Number, Name, Surname, Gender, Date of Birth and Place of Birth.
- * Registration mode: when an SP requests additional extra-authentication attributes, like mobile phone, email, address, etc.

SPID in "Authentication" mode is free of charge for logins, with every security level, until 1000 Unique Users have logged in a single Identity Provider in a year. After that threshold, the prices are reported in the following table, according to [18]. SPID in "Registration" mode costs the same regardless of the number of users.

SPID credentials	SPID Authentication	SPID Registration
1st, 2nd level	0,40 EUR	3,50 EUR
3rd level	7,00 EUR	7,00 EUR

Table 2.1: SPID Cost for a Service Provider

Next, AgID shares this formula to calculate the **Cost of SPID for a Service Provider**.

$$SP \text{ Cost} = \sum_{i=1}^N \left[(P^{\text{authentication}} * UU_i^{\text{authentication}}) + (P^{\text{registration}} * UU_i^{\text{registration}}) \right] \quad (2.1)$$

Where:

- * N = number of Identity Providers;
- * $UU_i^{\text{authentication}}$ = number of unique users that have logged using **only** SPID in "Authentication" mode, in the i-th IdP, over a single billing period;
- * $UU_i^{\text{registration}}$ = number of unique users that have logged in **at least once** using SPID "Registration" mode, in the i-th IdP, over a single a billing period;
- * $P^{\text{authentication}}$ = price per login with SPID "Authentication" mode;
- * $P^{\text{registration}}$ = price per login with SPID "Registration" mode;

2.4.3 Future of SPID

In Europe, there are discussions regarding how to enrich the mechanism of Digital Identity and authentication with the mechanism of Self Sovereign Identity (SSI) along with the European Blockchain Services Infrastructure (EBSI). Self-sovereign identities (SSI) are digital identities that are managed in a decentralized manner using the blockchain.

Problems of this approach can be:

1. **Privacy** which can be assured by Zero-Knowledge Proof blockchains.
2. **General public** that is not familiar with the blockchain technology, in order to ensure privacy we can use Social recovery wallets, which introduce the concept of "guardians". "If a user loses their signing key, that is when the social recovery functionality would kick in. The user can simply reach out to their guardians and ask them to sign a special transaction to change the signing pubkey registered in the wallet contract to a new one." [19]

SPID can also improve the delegation methods, allowing the parents of a disabled person to be the delegators, and not only the legal guardian. This problem can also lead to improvements in other scenarios like the delegation to a lawyer from a citizen or from a company to an assistant.

A SPID login is time-consuming (a study from Politecnico di Milano reported an average access in 60 seconds with SPID) and has many steps, increasing the technological gap needed to be capable of using digital services with it. So in the future SPID could improve, lowering those barriers.

One of SPID's main adjectives is to be Open Source, but all SPID Identity Provider require the use of One-Time-Passwords but we are locked in the use of IdP's applications to generate those OTP, because they use a modified version of the TOTP algorithm for the generation of One-Time-Passwords. This article [20] explains how to generate OTPs for SPID unofficially using Google Authenticator, in a rooted Android device.

Another problem for SPID is that from a developer's point of view, the documentation is not always on point and not always internationalized, creating language barriers for not Italian speaking integrators.

2.5 Java Spring web application

"Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications." [21]
Java Spring allows developer to create web applications pretty easily, using Java methods.

Spring Framework uses the *Inversion of Control (IoC)* pattern, which means that the control of the objects is inverted, and the responsibility of managing the objects is delegated to a container.

Spring Framework uses the *Dependency Injection (DI)* pattern, which means that the dependencies between objects are injected into the objects, instead of being created by the objects themselves.

Spring Framework uses the *Aspect Oriented Programming (AOP)* pattern, which means that the cross-cutting concerns are separated from the business logic, and are injected into the business logic.

2.5.1 Java Spring overview

Spring Framework separates into modules all the features, with the Core Container being the base module, accessing Data through Data Access/Integration module, leaving the handling of the networking to the Web module.

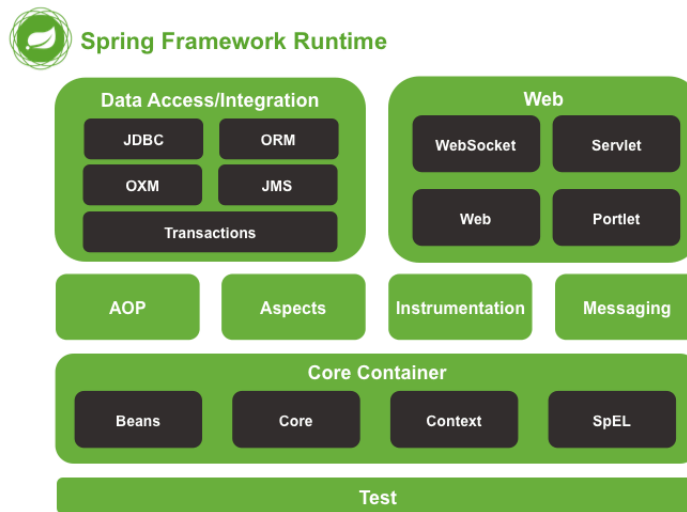


Figure 2.8: Java Spring Overview

One practical example of an application using Java Spring for a fullstack web application, is the one in the next figure. Where an Apache Tomcat Servlet Container is used to host the web application, the Frontend of the webapp could be made in Java Server Page (JSP). The custom domain logic is implemented using the Core container module which connects to the Data Access module to store data.

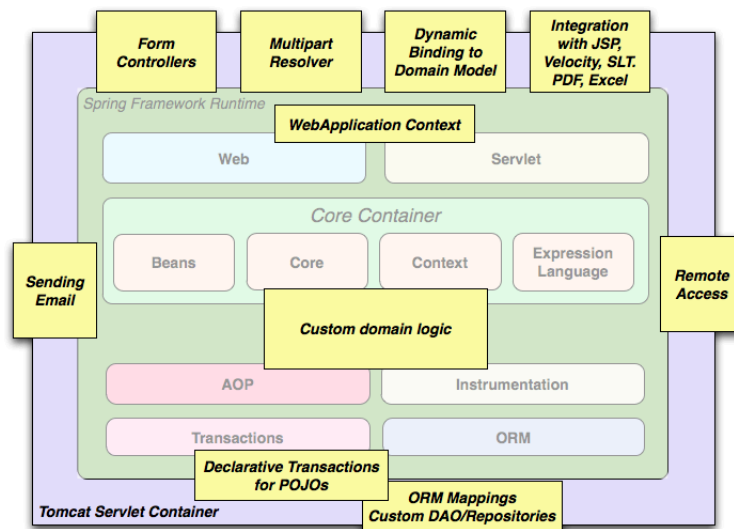


Figure 2.9: Java Spring Fullstack Example

Security with Java Spring Security

"Spring Security provides comprehensive security services for Java EE-based enterprise software applications." [22]

Spring Security is a framework that provides authentication, authorization, and protection against common attacks in a Java Spring application.

For the scope of this internship, I used the *Spring Security SAML* [23] extension, which provides support for the SAML protocol, protecting against exploits.

2.6 Satosa authentication proxy

SATOSA is an authentication proxy, that translates between different authentication protocols, like SAML2 and OpenID Connect. The project is open source[2] and it is a project of "The Identity Python" organization.

SaToSa is born to be a SAML to SAML proxy, but it has been extended to support other protocols, like OpenID Connect to Saml proxy.

Advantages of SATOSA over competitors are:

- * **Open Source:** the project is open source, and it is available on GitHub with a strong community already working on it and optimizing it for SPID;
- * **Metadata handling:** SaToSa automatically creates and signs metadata, and it also exposes endpoints for metadata retrieval;
- * **Standalone:** SaToSa doesn't need Jetty or Apache or Shibboleth to work, it is a standalone application developed in Python, composable with a web server like Nginx;

- * **Logging:** SaToSa handles logging in a very specific way, and it is possible to use custom logging, choosing where to store the logs;

SaToSa for SAML can be configured to work in two ways:

- * **One-to-many:** there is one Service Provider talking to SaToSa, validating against multiple SAML Identity Providers;

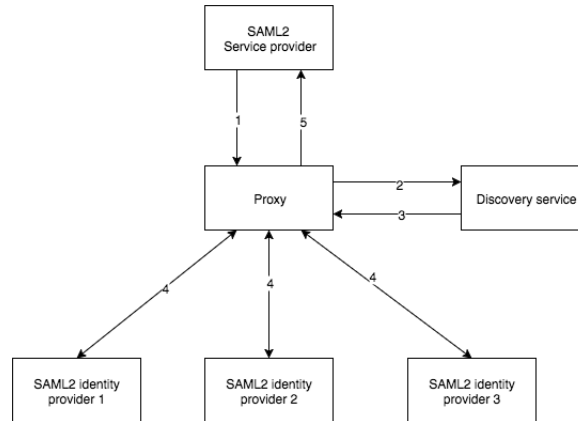


Figure 2.10: SaToSa One To Many Example

- * **Many-to-one:** there are multiple SPs configured to use SaToSa, and the proxy is talking to only one SAML IdP.

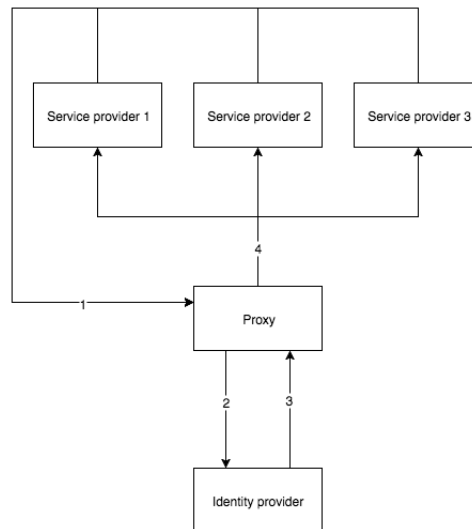


Figure 2.11: SaToSa Many-to-One Example

For the scope of this thesis, we will use SaToSa in the **One-to-many** configuration, to allow the Java Spring SP to use SPID as an authentication method.

Chapter 3

Solution development

Now we discuss the path we decided to take, how we developed the software, and the technologies we leveraged. Then, we outline the final achievements and what can be done to enhance the Proof of Concept.

Over time it has been clear that SPID is currently using the SAML protocol, while the OpenID Connect protocol is under development by the Identity Providers. So while testing different solutions, we decided to use the SaToSa authentication proxy which can be updated in the future to support the SPID protocol with OpenID Connect Providers.

3.1 Architectural overview

The proposed solution is composed of two main components: the Java Spring web application and SaToSa authentication proxy. Then, SaToSa faces the external SPID Identity Providers.

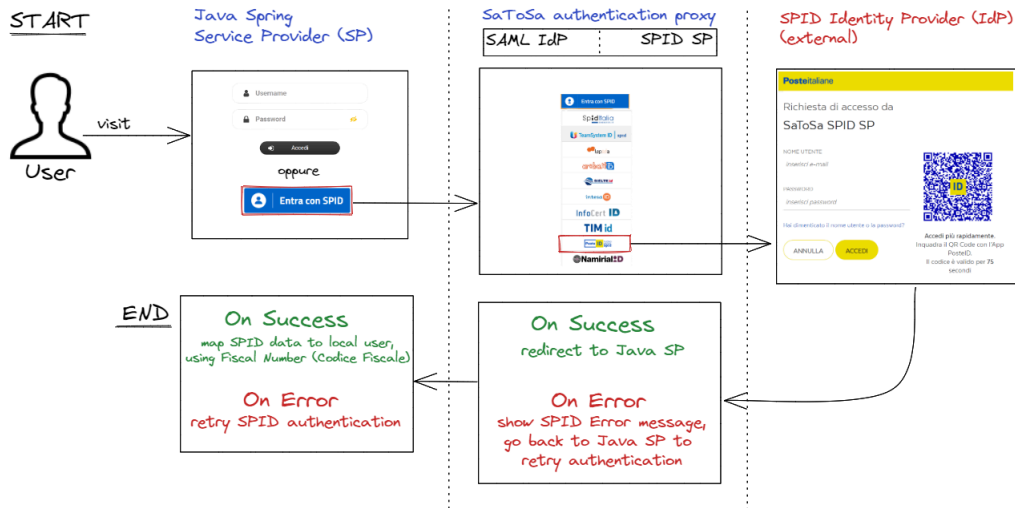


Figure 3.1: Architecture overview

In Figure 3.1 we can see the overall architecture and an example of the authentication flow.

The authentication flow is the following:

1. : A user visits the Java Spring Service Provider and clicks on the "Entra con SPID" button, thus visiting the SAML Entry Point, generating a first SAML AuthN Request towards the SaToSa authentication proxy.
2. SaToSa receives the AuthNRequest and shows (in Nginx) the list of SPID Identity Providers to the user.
3. The user selects one of the official SPID Identity Providers, for example PosteID, and the user is redirected to the selected SPID IdP. SaToSa creates a second SAML AuthNRequest and sends it to the selected IdP.
4. In the IdP the user authenticates and upon successful login or in case of an error, the IdP redirects the user to SaToSa.
5. If the authentication is successful SaToSa automatically redirects the user to the original Java Service Provider. Otherwise, if an error occurred, SaToSa displays the SPID error message and allow the user to go back to the Java Service Provider to retry authentication.
6. In the Java Service Provider, if the authentication is successful, the SAML Processor maps SPID data to a local user in the application, using the Fiscal Number as a common key. If an error occurred, it allows to retry the SPID authentication.

3.2 Solution development

As we can have said in the architecture overview, the solution is composed of two main components: the Java Spring web application and SaToSa authentication proxy. But let's see dive into those components.

3.2.1 Java Spring web application

The Java Spring application will be the SAML Service Provider. In a normal use case scenario, the application can already have other methods of authentication, like a username and password.

In this case, we will add the SAML Service Provider to the existing authentication methods, using a Custom Authentication Provider.

Setup a SAML Service Provider in Java Spring

To setup a SAML Service Provider in Java Spring we must add the dependency *Spring Security SAML extension* and the official Shibboleth repository to download the latest opensaml jar, in the pom.xml file at the root of our Java Spring project.

Now , we can move on to configuring the SAML Service Provider:

1. Create a SAML Entry Point;
2. Handle Login and Logout;
3. Handle Metadata Generation;
4. Handle Metadata Manager to load the IdP metadata;

5. Parser for XML file;
6. SAML HTTP POST binding and SAML Processor;
7. SAML Custom Authentication Provider;
8. Edit the SecurityConfig file;
9. Add the SPID SP Access Button to the frontend;

In the next figure we can see a sample of the `SamlSecurityConfig` java file, with an example similar to what it's used in the final code, including the SAML Entry Point, the SAML Processor, the SAML Custom Authentication Provider, and the SAML Metadata Manager.

The SAML Entry Point is the first step in the authentication flow, it is the URL that the user will visit to start the authentication process. In this case, the user will visit the URL `https://.../saml/login` to start the authentication process.

The SAML Processor is the class that will handle the incoming SAML messages from `HttpRequest` stream. Here we can add the supported bindings (SPID only allows HTTP POST and HTTP Redirect bindings).

The SAML Custom Authentication Provider is the class that will handle the authentication process.

The SAML Metadata Manager is the class that will handle the IdP metadata. In this case, we will use the metadata from the official SPID repository, [24], saved in a local XML file, loaded by the Extended Metadata Delegate along with a Caching Metadata Manager.

The Success Redirect and Authentication Failure handles are responsible to redirect the user accordingly to the authentication result.

The SAML Logout Handler is responsible to invalidate the HTTP session and clearing the authentication information in the application context, following in a redirect to the website homepage.

In the following example, there are variables prefixed with the Java Spring token `@Value`, this shows us that those variables are loaded from the `application.properties` file, typically stored in the `Resources` folder.

This allows easy customization and modularity for key values in the SAML setup, like the default IdP, the keystore information (location, password and alias) where the cryptographic data will be stored.

```

@Configuration
public class SamlSecurityConfig {

    @Value("${saml.keystore.location}")
    private String samlKeystoreLocation;

    @Value("${saml.keystore.password}")
    private String samlKeystorePassword;

    @Value("${saml.keystore.alias}")
    private String samlKeystoreAlias;

    @Value("${saml.idp}")
    private String defaultIdp;

    @Bean(initMethod = "initialize")
    public StaticBasicParserPool parserPool() {
        return new StaticBasicParserPool();
    }

    @Bean
    public ParserPoolHolder parserPoolHolder() {
        return new ParserPoolHolder();
    }

    @Bean
    public SAMLAuthenticationProvider samlAuthenticationProvider() {
        return new com.web.saml.CustomSAMLAuthenticationProvider();
    }

    @Bean
    public SAMLContextProvider[] samlContextProviders() {
        SAMLContextProvider[] samlContextProviders = new SAMLContextProvider[]();
        samlContextProviders.add(new SAMLContextProviderLS());
        samlContextProviders.add(new SAMLContextProviderLS());
        samlContextProviders.add(new SAMLContextProviderLS());
        samlContextProviders.add(new SAMLContextProviderLS());
        return samlContextProviders;
    }

    @Bean
    public MetadataDisplayFilter metadataDisplayFilter() {
        return new MetadataDisplayFilter();
    }

    // Initialization of OpenSAML library
    @Bean
    public static SAMLBootstrap samlBootstrap() {
        return new SAMLBootstrap();
    }

    @Bean
    public SAMLDefaultLogger samlLogger() {
        return new SAMLDefaultLogger();
    }

    // SAML 2.0 WebSSO Assertion Consumer
    @Bean
    public WebSSOProfileConsumer webSSOProfileConsumer() {
        return new WebSSOProfileConsumerImpl();
    }

    // SAML 2.0 Holder-of-Key WebSSO Assertion Consumer
    @Bean
    @Qualifier("hokWebSSOProfileConsumer")
    public WebSSOProfileConsumerHokImpl hokWebSSOProfileConsumer() {
        return new WebSSOProfileConsumerHokImpl();
    }

    // SAML 2.0 WebSSO profile
    @Bean
    public WebSSOProfile webSSOProfile() {
        return new WebSSOProfileImpl();
    }

    // SAML 2.0 Holder-of-Key Web SSO profile
    @Bean
    public WebSSOProfileConsumerHokImpl hokWebSSOProfile() {
        return new WebSSOProfileConsumerHokImpl();
    }

    // SAML 2.0 Enhanced-Client-Proxy Profile
    @Bean
    public WebSSOProfileECPImpl ecpProfile() {
        return new WebSSOProfileECPImpl();
    }

    // SAML 2.0 Logout profile
    @Bean
    public SingleLogoutProfile logoutProfile() {
        return new SingleLogoutProfileImpl();
    }

    @Bean
    public KeyManager keyManager() {
        DefaultResourceLoader loader = new DefaultResourceLoader();
        Resource storeFile = loader.getResource(samlKeystoreLocation);
        Map<String, String> passwords = new HashMap<>();
        passwords.put(samlKeystoreAlias, samlKeystorePassword);
        return new JKSKeyManager(storeFile, samlKeystorePassword, passwords,
            samlKeystoreAlias);
    }

    @Bean
    public WebSSOProfileOptions defaultWebSSOProfileOptions() {
        WebSSOProfileOptions webSSOProfileOptions = new WebSSOProfileOptions();
        webSSOProfileOptions.setIncludeScoping(false);
        webSSOProfileOptions.setBinding("urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST");
        return webSSOProfileOptions;
    }

    @Bean
    public SAMLEntryPoint samlEntryPoint() {
        SAMLEntryPoint samlEntryPoint = new SAMLEntryPoint();
        samlEntryPoint.setDefaultProfileOptions(defaultWebSSOProfileOptions());
        return samlEntryPoint;
    }

    @Bean
    public ExtendedMetadata extendedMetadata() {
        ExtendedMetadata extendedMetadata = new ExtendedMetadata();
        extendedMetadata.setDiscoveryEnabled(false);
        // External metadata
        return extendedMetadata;
    }
}

```

```

@Configuration
public class SamlSecurityConfig {
    // ...

    @Bean
    @Qualifier("spidauthIdp")
    public ExtendedMetadataDelegate spidauthIdpExtendedMetadataProvider() throws
    MetadataProviderException {
        // Use the Spring Security SAML resource mechanism to load metadata from the Java class path.
        org.opensaml.util.resource.Resource resource = null;

        try {
            resource = new ClasspathResource("/saml/spidauthenticationproxy-idp.xml");
        } catch (ResourceException e) {
            e.printStackTrace();
        }

        Timer timer = new Timer("saml-metadata");
        ResourceBackedMetadataProvider provider = new ResourceBackedMetadataProvider(timer, resource);
        provider.setParserPool(parserPool());
        return new ExtendedMetadataDelegate(provider, extendedMetadata());
    }

    @Bean
    @Qualifier("metadata")
    public CachingMetadataManager metadata() throws MetadataProviderException, ResourceException {
        List<MetadataProvider> providers = new ArrayList<>();
        providers.add(spidauthIdpExtendedMetadataProvider());
        CachingMetadataManager metadataManager = new CachingMetadataManager(providers);
        metadataManager.setDefaultIdP(defaultIdp);
        return metadataManager;
    }

    @Bean
    @Qualifier("saml")
    public SavedRequestAwareAuthenticationSuccessHandler successRedirectHandler() {
        SavedRequestAwareAuthenticationSuccessHandler successRedirectHandler = new
        SavedRequestAwareAuthenticationSuccessHandler();
        successRedirectHandler.setDefaultTargetUrl("/");
        return successRedirectHandler;
    }

    @Bean
    @Qualifier("saml")
    public SimpleUrlAuthenticationFailureHandler authenticationFailureHandler() {
        SimpleUrlAuthenticationFailureHandler failureHandler = new
        SimpleUrlAuthenticationFailureHandler();
        failureHandler.setUseForward(true);
        failureHandler.setDefaultFailureUrl("/error");
        return failureHandler;
    }

    @Bean
    public SimpleUrlLogoutSuccessHandler successLogoutHandler() {
        SimpleUrlLogoutSuccessHandler successLogoutHandler = new
        SimpleUrlLogoutSuccessHandler();
        successLogoutHandler.setDefaultTargetUrl("/");
        return successLogoutHandler;
    }

    // Logout handler terminating local session
    @Bean
    public SecurityContextLogoutHandler logoutHandler() {
        SecurityContextLogoutHandler logoutHandler = new SecurityContextLogoutHandler();
        logoutHandler.setInvalidateHttpSession(true);
        logoutHandler.setClearAuthentication(true);
        return logoutHandler;
    }

    // Override default logout processing filter with the one processing SAML messages
    @Bean
    public SAMLLogoutFilter samlLogoutFilter() {
        return new SAMLLogoutFilter(successLogoutHandler(),
            new LogoutHandler[] { logoutHandler() },
            new LogoutHandler[] { logoutHandler() }
        );
    }

    // Filter processing incoming logout messages
    @Bean
    public SAMLLogoutProcessingFilter samlLogoutProcessingFilter() {
        return new SAMLLogoutProcessingFilter(successLogoutHandler(), logoutHandler());
    }

    // Initialization of the velocity engine for http POST binding
    @Bean
    public HTTPPostBinding httpPostBinding() {
        return new HTTPPostBinding(parserPool(), VelocityFactory.getEngine());
    }

    @Bean
    public HTTPRedirectDeflateBinding httpRedirectDeflateBinding() {
        return new HTTPRedirectDeflateBinding(parserPool());
    }

    // Class loading incoming SAML messages from httpRequest stream
    @Bean
    public SAMLProcessorImpl processor() {
        ArrayList<SAMLBinding> bindings = new ArrayList<>();
        bindings.add(HTTPRedirectDeflateBinding());
        bindings.add(HTTPPostBinding());
        return new SAMLProcessorImpl(bindings);
    }
}

```

Figure 3.2: SecurityConfig file

SPID SP Access Button

The SPID federation requires the use of a pre-made button to standardize the login process. A sample component can be found on the official repository, *italia/spid-sp-access-button*[25]. It provides a list of all official Identity Providers with their logos and URLs.

The button can be a simple HTML list with all the official IdPs loaded with javascript into the DOM, see Figure 2.6 to see the final result. Here are HTML and Javascript codes for the SPID Access button:

```

1 <!-- From https://github.com/italia/spid-sp-access-button/ -->
2
3 <!-- AGID - SPID IDP BUTTON MEDIUM 'ENTRA CON SPID' * begin * -->
4 <a class="italia-it-button button-sp-id" spid-idp-button="spid-idp-button-medium-get">
5   <span class="italia-it-button-lcon">
6     </span>
7     <span class="italia-it-button-text">Entra con SPID</span>
8 </a>
9 <div id="spid-idp-button-medium-get" class="spid-idp-button-relative">
10  <ul class="spid-idp-button-menu" data-sp-id-remote aria-labelledby="spid-idp">
11    <li><a class="dropdown-item" href="https://www.spid.gov.it/Maggiori_informazioni"></a></li>
12    <li><a class="dropdown-item" href="https://www.spid.gov.it/riciedi-spid">Non hai SPID?</a></li>
13    <li><a class="dropdown-item" href="https://www.spid.gov.it/serve-atuto">Serve aiuto?</a></li>
14  </ul>
15 </div>
16 <!-- AGID - SPID IDP BUTTON MEDIUM 'ENTRA CON SPID' * end * -->

```

```

1 // From https://github.com/italia/spid-sp-access-button/
2 function spid_addIdpEntry(data, element) {
3   // ...
4   li.innerHTML = `
5     <a href="#">
6       <span class="spid-sr-only">${data['organization_name']}</span>
7       
8     </a>`;
9   }
10  // ...
11 }
12

```

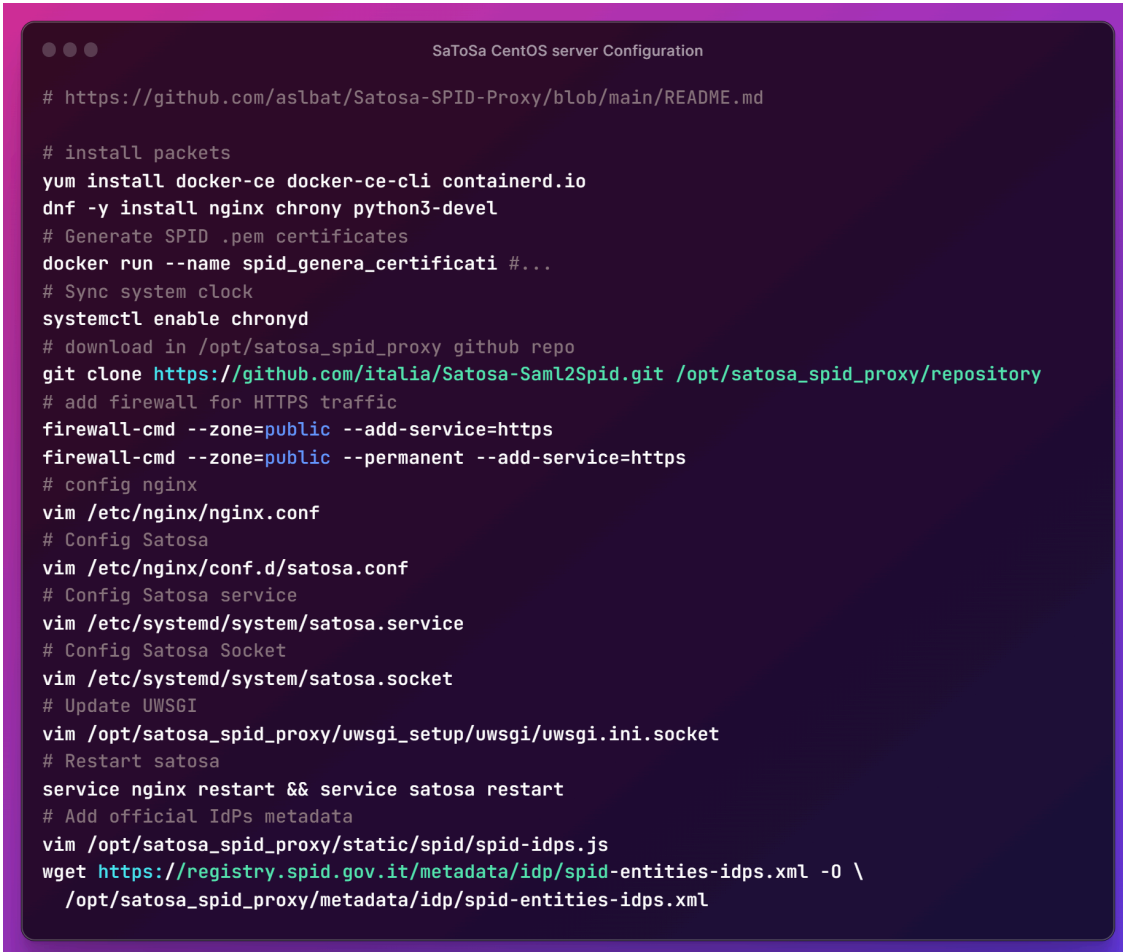
Figure 3.3: SPID SP Access Button Example

3.2.2 SaToSa authentication proxy

The SaToSa authentication proxy must be published with security in mind. For this internship, SaToSa has been deployed on a CentOS 8 instance on a private network, accessible only in a testing environment. The setup has been done using the official documentation[26]

1. Install required modules and libraries, like Docker, Python, Nginx, Chrony, etc.
2. Generate spid .pem certificates using the official Docker image.
3. Update the system clock, in SAML the timing of the requests is important, so errors can be introduced when the system clock is not synchronized.
4. Download using git the SaToSa repository.
5. Set the firewall to allow only the required ports, port 80 for HTTP local testing and port 443 for HTTPS traffic.
6. Configure Nginx frontend.
7. Configure SaToSa.
8. Configure SaToSa network access
9. Configure UWSGI, Universal Web Server Gateway Interface, used to run the SaToSa proxy in a secure environment.
10. Update the changes by restarting the SaToSa service.

11. Add official SPID IdPs in SaToSa and configure the metadata.



```

SaToSa CentOS server Configuration

# https://github.com/aslbat/Satosia-SPID-Proxy/blob/main/README.md

# install packets
yum install docker-ce docker-ce-cli containerd.io
dnf -y install nginx chrony python3-devel
# Generate SPID .pem certificates
docker run --name spid_genera_certificati #...
# Sync system clock
systemctl enable chronyd
# download in /opt/satosia_spid_proxy github repo
git clone https://github.com/italia/Satosia-Saml2Spid.git /opt/satosia_spid_proxy/repository
# add firewall for HTTPS traffic
firewall-cmd --zone=public --add-service=https
firewall-cmd --zone=public --permanent --add-service=https
# config nginx
vim /etc/nginx/nginx.conf
# Config Satosa
vim /etc/nginx/conf.d/satosia.conf
# Config Satosa service
vim /etc/systemd/system/satosia.service
# Config Satosa Socket
vim /etc/systemd/system/satosia.socket
# Update UWSGI
vim /opt/satosia_spid_proxy/uwsgi_setup/uwsgi/uwsgi.ini.socket
# Restart satosa
service nginx restart && service satosa restart
# Add official IdPs metadata
vim /opt/satosia_spid_proxy/static/spid/spid-idps.js
wget https://registry.spid.gov.it/metadata/idp/spid-entities-idps.xml -O \
/opt/satosia_spid_proxy/metadata/idp/spid-entities-idps.xml

```

Figure 3.4: SaToSa server configuration

3.2.3 Testing

Now that we are using the official SaToSa SPID implementation, we can focus only on the Java Spring Service Provider.

We could assume that what is published in the official *italia/Satosia-Saml2Spid*^[27] repository is a working solution, but assumptions are not safe in a production environment.

So we just need to be able to test the authentication flow after changes on the *italia/Satosia-Saml2Spid* repository, this is possible because this version of Satosa creates the metadata and AuthNRequest and handles all the error codes, accordingly to all the SPID requirements, even the not mandatory ones.

To test the authentication flow we can launch the official *italia/spid-saml-check*^[6] docker container inside the SaToSa server instance.

Spid-saml-check contains a *demo* web application that can be used to emulate a SPID Identity Provider and also a *spid-validator* web application that provides a test

suite to check if our solution passes the strict SPID requirements.

In our case, we can add the `spid-saml-check` and launch the authentication flow as usual selecting the local `spid-saml-check` as Identity Provider in Satsosa, then we can select the *spid-validator* application and launch the test suite and we can see that every test passes.

3.3 Findings

In this last section, we discuss the last considerations about the final result, what can be done to improve the solution and personal thoughts.

3.3.1 Achievements

The Java Spring Service Provider is now able to authenticate users using the SPID credentials, delegating authentication to the SaToSa authentication proxy.

This configuration works best when multiple Service Providers need to use the SPID credentials, if we have a new working SAML Service Provider that we want to configure, we just need to exchange metadata information with the SaToSa proxy and the new Service Provider is able to authenticate users with the SPID credentials.

3.3.2 Future improvements

During the final part of the internship and the writing of this thesis, many aspects came up that could be improved in the future.

Adding the support for the OpenID Connect protocol in the SaToSa proxy from both parts of the proxy, allowing the Service Provider to retrieve SPID Credentials using a more modern protocol is a future improvement that SPID will state as mandatory in the following year.

Also, future improvements to SPID will allow the spread in the private sector. As of now, there are 12674 Public Administration services using SPID, but only 182 private service providers using it[28].

I have tested all private service providers and the majority of those are just testing environments or they are not working because they have removed the SPID service or they do not provide access because of SAML libraries errors.

Given the fact that SPID is a mandatory authentication method for Public Administration, many companies are offering to integrate SPID authentication, but many private businesses could argue where the benefits could reside of using this complicated system, still keeping in consideration they also have to pay a 0,40€fee on every login after 1000 unique users per year.

For the ones that may say that the SPID fee is not a problem because a lot of services send SMS or emails to the user which also costs the Service Provider, following there's an example of the cost to send a single SMS using two of the most used *SMS-as-a-service* platforms: Amazon Simple Notification Service (SNS)[29] and Twilio SMS[30].

	Amazon SNS	Twilio SMS
Cost to send one SMS	0.075 \$	0.0927 \$

Comparing these costs of Amazon SNS and Twilio (spending less than 0,10€per SMS) with the cost of SPID for a private Service Provider (reported in the table 2.1, with a cost of 0,40€or 3,50€or 7,00€), SPID's per-user fee is disproportionate compared to similar "*pay-per-use*" services.

The SPID login could be useful for a general web application in the registration phase, where the user could be tasked to insert manually a lot of personal informations

and even though SPID is time-consuming, it will take less time and energy than a user being asked to insert all of its data manually.

3.3.3 Personal evaluation

This internship has been a great experience for me because I had the opportunity to learn a lot about authentication protocols along with fullstack development, in particular how SPID and the SAML protocol works. I also been able to receive value from the company and from some of the amazing members of the open source italian community, that works in SPID.

Acronyms

DoS Denial of Service. [12](#)

DTD Document Type Definition. [12](#)

IdP Identity Provider. [1](#), [2](#), [5](#)

OIDC OpenID Connect. [2](#), [10](#), [11](#)

OP OpenID Provider. [10](#)

PoC Proof Of Concept. [2](#)

RP Relying Party. [10](#)

SAML Security Assertion Markup Language. [2](#), [11](#)

SP Service Provider. [1](#), [2](#), [5](#)

XMLEnc XML Encryption. [11](#)

XSW Xml Signature Wrapping. [11](#)

XXE XML eXternal Entity injection. [12](#)

Bibliography

Bibliographical references

- [11] A. M. e. a. Juraĵ Somorovsky, “On Breaking SAML: Be Whoever You Want to Be,” 2012. [Online]. Available: <https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final91-8-23-12.pdf> (cit. on p. 11).
- [14] e. a. T. Lodderstedt M. McGloin, “OAuth 2.0 Threat Model and Security Considerations,” [Online]. Available: <https://www.rfc-editor.org/rfc/rfc6819> (cit. on p. 12).

Websites consulted

- [1] “SPID private service provider registry,” [Online]. Available: <https://registry.spid.gov.it/private-service-providers> (cit. on p. 1).
- [2] “SATOSA authentication proxy,” [Online]. Available: <https://github.com/IdentityPython/SATOSA> (cit. on pp. 2, 19).
- [3] “SPID Keycloak implementation,” [Online]. Available: <https://github.com/italia/spid-keycloak-provider> (cit. on p. 2).
- [4] “Aperio CAS official website,” [Online]. Available: <https://www.apereo.org/projects/cas> (cit. on p. 2).
- [5] “SPID Shibboleth implementation,” [Online]. Available: <https://github.com/italia/spid-sp-shibboleth> (cit. on p. 2).
- [6] “SPID saml check testing Identity Provider,” [Online]. Available: <https://github.com/italia/spid-saml-check> (cit. on pp. 2, 26).
- [7] “SAML v2.0 Technical Overview,” [Online]. Available: <http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html> (cit. on p. 5).
- [8] “SAML Authentication Flow,” [Online]. Available: <https://youtu.be/SvppXbpv-5k> (cit. on p. 6).
- [9] “OpenID Connect,” [Online]. Available: <https://openid.net/connect/> (cit. on p. 7).
- [10] “OpenID Connect Authentication Flow,” [Online]. Available: <https://youtu.be/rTz1F-U9Y6Y> (cit. on pp. 8–10).

- [12] “OWASP XML Security Cheat Sheet,” [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html (cit. on p. 12).
- [13] “OWASP Xml eXternal Entity (XXE) Prevention Cheat Sheet,” [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html (cit. on p. 12).
- [15] “What is SPID,” [Online]. Available: <https://www.spid.gov.it/en/what-is-spid/> (cit. on p. 14).
- [16] “SPID Open Data,” [Online]. Available: <https://avanzamentodigitale.italia.it/it/progetto/spid> (cit. on p. 15).
- [17] “Blog Stefano Quintarelli,” [Online]. Available: <https://blog.quintarelli.it/spid/> (cit. on p. 15).
- [18] “SPID prices,” [Online]. Available: https://www.agid.gov.it/sites/default/files/repository_files/allegato_4_dt_166_corrispettivi_spid_idp_2019_0.pdf (cit. on pp. 15, 16).
- [19] “Vitalik blog about social recovery wallet,” [Online]. Available: <https://vitalik.ca/general/2021/01/11/recovery.html> (cit. on p. 17).
- [20] “SPID OTP with Google Authenticator” (cit. on p. 17).
- [21] “Introduction to the Spring Framework,” [Online]. Available: <https://docs.spring.io/spring-framework/docs/4.3.29.RELEASE/spring-framework-reference/htmlsingle/#overview> (cit. on p. 17).
- [22] “What is Spring Security,” [Online]. Available: <https://docs.spring.io/spring-security/site/docs/4.2.x/reference/htmlsingle/#what-is-acegi-security> (cit. on p. 19).
- [23] “Spring Security SAML,” [Online]. Available: <https://docs.spring.io/spring-security-saml/docs/1.0.x/reference/htmlsingle/#chapter-quick-start> (cit. on p. 19).
- [24] “SPID Identity Provider Metadata,” [Online]. Available: <https://registry.spid.gov.it/metadata/idp/spid-entities-idps.xml> (cit. on p. 23).
- [25] “SPID Service Provider Access Button,” [Online]. Available: <https://github.com/italia/spid-sp-access-button> (cit. on p. 25).
- [26] “SaToSa authentication proxy SPID guide,” [Online]. Available: <https://github.com/aslbat/Satosha-SPID-Proxy/> (cit. on p. 25).
- [27] “SPID SATOSA implementation,” [Online]. Available: <https://github.com/italia/Satosha-Saml2Spid> (cit. on p. 26).
- [28] “SPID Service Provider Registry,” [Online]. Available: <https://registry.spid.gov.it/private-service-providers> (cit. on p. 28).
- [29] “Amazon Simple Notification Service,” [Online]. Available: <https://aws.amazon.com/it/sns/sms-pricing/> (cit. on p. 28).
- [30] “Twilio SMS pricing,” [Online]. Available: <https://www.twilio.com/sms/pricing/it> (cit. on p. 28).