The present work was submitted to
the Research Group
Software Construction

of the Faculty of Mathematics,
Computer Science, and
Natural Sciences

Master Thesis

Conception of a Security Design Pattern Catalog for Constraint-based Recommender Systems

presented by

Dominik Lammers

Aachen, December 12, 2024

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Alex Sabau, M.Sc.

Acknowledgment

First and foremost, I would like to thank Prof. Dr. rer. nat. Horst Lichter for providing me with the opportunity to write my master's thesis at his chair. I am also grateful to him and Prof. Dr. rer. nat. Bernhard Rumpe for reviewing my thesis.

I extend my sincere thanks to my supervisor, M.Sc. Alex Sabau, for his guidance and constant support. It has always been a pleasure to discuss ideas with him, and I deeply appreciate his helpful suggestions and constructive feedback throughout the course of this thesis project.

I am profoundly grateful to my family for the tremendous support and trust they have given me throughout my studies and in completing this thesis project.

Last but not least, I would like to thank my friends for the wonderful journey we have shared. Whether collaborating on assignments or enjoying our free time together, your companionship has made this experience truly memorable.

Thank you to everyone who has been part of this journey.

Dominik Lammers

Abstract

In the rapidly evolving digital landscape, the security of software systems has become paramount. However, a critical shortage of security experts makes it challenging to adequately protect these systems. Security patterns provide proven solutions to recurring security problems, helping architects design secure systems. Despite their potential, their practical use remains limited due to the lack of security-relevant information necessary for secure implementation and the limited guidance in selecting appropriate patterns.
This thesis addresses these limitations by introducing the Security Design Pattern Description Metamodel, which enables the creation of Security Design Patterns (SDPs) that incorporate essential security information and explicit pattern relationships. To assist architects in selecting suitable SDPs, we introduce the SDP Knowledge Bases Metamodel, which enables Constraint-based Recommender Systems (CBRSs) to recommend appropriate SDPs. Our methodology involves analyzing security solutions in open source software to identify essential elements that can contribute to the practical use of SDPs. Based on these findings, we iteratively developed concrete SDPs and knowledge bases, capturing their elements and relationships within co-evolving metamodels.
The metamodels are validated through application examples, namely OpenID Connect Authentication and Password-based Authentication. These examples show how pattern relationships and important security-relevant information, such as data requirements, can be effectively represented in SDPs to facilitate the implementation of secure systems. In addition, a synthetic recommendation example illustrates the effective use of knowledge bases in a CBRS. By decoupling the pattern description and selection process, this thesis makes security patterns accessible to a broader audience and provides a foundation for advancing research in secure software design.

Contents

List of Tables

List of Figures

1. Introduction

In today's digital age, software systems have become indispensable in many fields, including finance, healthcare, and entertainment. Consequently, these systems have become attractive targets for attackers who exploit vulnerabilities to access confidential information or disrupt operations. The cost of such attacks is expected to exceed 10.5 trillion USD annually by 2025 [eSe23]. In addition, the World Economic Forum's Global Risks Report 2023 ranks cybercrime and cyber insecurity among the top ten global risks in both the short and the long term [Wor23]. These trends underscore the critical importance of security for organizations and their software systems.

To address these threats, architects can employ the *Secure by Design* approach during software development. Secure by Design integrates security early in the *Software Development Lifecycle (SDLC)*, especially during the analysis and design phases. By applying Secure by Design, vulnerabilities can be mitigated proactively, increasing resilience to emerging threats [DJS19]. However, applying Secure by Design requires security experts, and there is a critical shortage of such professionals, with the cybersecurity workforce needing to nearly double to meet current demand [ISC23]. Furthermore, the inclusion of *Insecure Design* in the OWASP Top Ten, a widely recognized list of the most critical security risks to web applications, highlights the prevalence of insecure design in practice [21], indicating limited application of Secure by Design.

One potential solution to mitigate the shortage of security experts and enable architects to apply Secure by Design is the use of security patterns. In general, patterns can support the design and implementation of robust systems by providing abstract, proven solutions to recurring problems [LL13]. Despite the success of patterns in fields such as architecture or object-oriented design, and the existence of security patterns, their adoption in the security domain remains limited [vYJ18; Fer13].

One reason for this limited adoption is that existing security patterns are often too abstract and do not provide the essential information architects need to implement them securely [Hey+07; vYJ18]. For example, the Authenticator pattern [Sch+06] specifies that a user should be authenticated by verifying authentication information. However, the pattern does not specify what the authentication information consists of, such as passwords, nor how it should be securely stored and verified [vYJ18].

Another issue is that existing security patterns often present a single, rigid solution and fail to accommodate changing situational needs. They do not consider that some aspects may be unnecessary in certain contexts, that alternatives may exist, or that solutions not generally considered best practices may still be required in specific situations. For example, the Password-based Authentication pattern [vYJ22c] discourages periodic password changes based on general recommendations [Gra+20]. However, the *Payment Card Industry Data Security Standard* requires periodic password changes in certain sce-

narios [PCI24]. Therefore, a security pattern that addresses periodic password changes is needed so that architects can implement it securely when necessary.

Finally, a qualitative study by Langstrof and Sabau [LS24] found that the complexity of the security domain and the time required to address security are significant challenges for integrating security into the SDLC. This highlights the need for security patterns that provide the information relevant to security. Moreover, selecting appropriate security patterns should be straightforward, thereby making them accessible to architects. However, current approaches are often limited to grouping related patterns and providing informal references between them [vYJ22a]. This requires architects to be familiar with numerous patterns to use them effectively. In addition, informal references can make it difficult to understand how patterns integrate with each other.

1.1. Contribution

To address the challenges with current security patterns, this thesis proposes *Security Design Patterns (SDPs)* comprising a usage aspect and a knowledge aspect.

The usage aspect, defined by a metamodel based on Heyman et al.'s security pattern structure [Hey+07], provides architects with the information needed to understand and implement the pattern. To manage complexity and ensure consistency, we introduce different views that represent specific aspects of an SDP. For example, the Conceptual View outlines the roles involved, their responsibilities, and their relationships, while the Behavioral View models the interactions between these roles. This metamodel enables the creation of consistent patterns that guide secure design implementations.

Furthermore, rather than offering a single SDP for a solution such as *Password-based Authentication (PBA)*, we propose multiple SDPs to accommodate different design options. For instance, one PBA SDP that addresses periodic password changes, while another does not.

To assist architects in selecting appropriate SDPs, we consider using a *Constraint-based Recommender System (CBRS)*. While implementing the CBRS is beyond the scope of this thesis project, we define a metamodel for the *Knowledge Bases (KBs)* required by a CBRS to recommend appropriate SDPs. This includes the knowledge aspect of SDPs, which provides information needed to reason about them, such as quality attributes describing their impact on a system.

To illustrate the applicability of our concepts, we provide examples of both the usage and knowledge aspects of SDPs. Evaluating their practical use by architects is beyond the scope of this thesis project. However, we lay the groundwork for future research to assess their utility through methods such as case studies.

1.2. Structure of this Thesis

The structure of this thesis is as follows:

- In Chapter 2, we introduce relevant definitions and terminology within software engineering, security, and the recommender domain. We also introduce the concept of *Security Solution Patterns* as proposed by Abazi [Aba24], which is related to SDPs.
- In Chapter 3, we present the research questions addressed in this thesis, along with their motivation and the problems they address. In addition, we present the methodology used to create the metamodels and the practical examples.
- In Chapter 4, we present related work on security pattern descriptions and support for selecting appropriate security patterns.
- In Chapter 5, we introduce the metamodel for the usage aspect of SDPs. This metamodel aims to enable the creation of practical SDPs that assist architects in designing secure systems.
- In Chapter 6, we evaluate the metamodel by presenting three application examples. The first example provides a comprehensive overview of an SDP. The second example considers the same security solution and compares it to the first, illustrating the need for multiple SDPs. The third example focuses on a different security solution, demonstrating that the metamodel can be used to create SDPs beyond a single security solution.
- In Chapter 7, we introduce the metamodel that enables the creation of KBs for use by a CBRS in recommending appropriate SDPs. This metamodel also integrates the knowledge aspect of SDPs required for reasoning about appropriate patterns.
- In Chapter 8, we evaluate the metamodel by presenting two KBs for different security solutions, illustrating the need for different KBs. We also present a recommendation example that simulates a CBRS and shows that the KBs can be used effectively.
- In Chapter 9, we discuss our findings from developing the metamodels and application examples, and highlight their limitations.
- In Chapter 10, we conclude the thesis by summarizing the main results and outlining opportunities for future research.

2. Foundation

This chapter introduces the definitions, terminology, and concepts essential to this thesis. First, we outline common definitions and terminology within software engineering and security. Next, we introduce Security Solution Patterns as proposed by Abazi [Aba24], which are related to SDPs. Finally, we introduce the recommender domain, focusing on the KB required by a CBRS to recommend and reason about appropriate SDPs.

2.1. Software Engineering

This section introduces the necessary definitions and terminology of software engineering.

Definition 2.1 (Design Pattern)**.** A *Design Pattern* defines an abstract, proven solution to a recurring design problem that occurs in a particular context. In addition, a Design Pattern has consequences that detail the strengths and weaknesses of applying the pattern [Gam+95].

Design patterns offer abstract solutions that can be implemented in different ways within their context. To demonstrate the general applicability of a design pattern, it should have multiple concrete implementations; otherwise, it is considered a candidate pattern [App97].

Definition 2.2 (Software System)**.** A *Software System* consists of elements designed to fulfill specified functional and non-functional requirements, along with the hardware infrastructure required to operate these elements [RW11].

To understand a system, it is necessary to consider its architecture.

Definition 2.3 (Architecture)**.** The *Architecture* of a system comprises its static and dynamic structures. The static structure includes structural elements and their organization, while the dynamic structure includes runtime elements and their interactions. An Architecture also has externally visible behavior that defines the system's provided functionality and its quality properties [RW11].

An architecture can be represented in various ways, such as focusing on the functionality provided to external entities or on how security is maintained in the system [RW11].

Definition 2.4 (Architectural Pattern)**.** An *Architectural Pattern* specifies an overarching structure and organization of elements in a system [RW11].

The *Microservice* Architectural Pattern specifies the division of a system into multiple independently deployable services, each representing a domain or subdomain [RF20]. For example, a microservice architecture in an e-commerce system might have dedicated services for ordering, payment, and inventory management.

Definition 2.5 (View)**.** A *View* represents specific aspects of the static or dynamic structure of an architecture relevant to one or more stakeholders [RW11].

Representing an architecture in a single model is typically impractical, as it would become unmanageable and provide little value due to its complexity. A view helps define an understandable architecture by limiting the information to what is relevant for explaining a specific aspect.

Definition 2.6 (Viewpoint)**.** A *Viewpoint* defines the elements and modeling rules for constructing a specific type of view [RW11].

A viewpoint is essential for providing consistent and understandable views by clearly defining the structure and guidelines that each view must follow. Stakeholders familiar with a viewpoint can readily understand any view that adheres to it [RW11].

2.2. Security

This section introduces the security principles and terms used in this thesis.

In the context of security, protecting data and systems from unauthorized access, modification, or disruption is essential. These goals are encapsulated in the three fundamental security principles known as the CIA Triad [SC14]:

- *Confidentiality*: Information is only accessible to authorized entities.
- *Integrity*: Information is protected from unauthorized modification during storage, transmission, and processing.
- *Availability*: Authorized entities have reliable access to information and services.

To meet these fundamental security principles, mechanisms such as authentication and authorization are essential.

Definition 2.7 (Authentication)**.** *Authentication* is the process of verifying the identity of an entity, such as a user, to ensure that the entity is who it claims to be [Sho14].

Authentication methods use information that can be classified into three types of authentication factors [Sho14]:

- *Knowledge Factor*: Something you know, such as a password or a PIN.
- *Possession Factor*: Something you have, such as a smart card or a mobile phone.
- *Inherence Factor*: Something you are, such as a fingerprint or facial recognition data.

Authentication methods that use two or more authentication factors of different types are referred to as *Multi-Factor Authentication (MFA)*.

Definition 2.8 (Authorization)**.** *Authorization* is the process of determining whether an authenticated entity is allowed to perform specific actions or access protected resources [Sho14].

There is a strong dependency between authorization and authentication because, to determine whether an entity is allowed to perform an action, the entity's identity must first be authenticated [vYJ22a].

Authentication and authorization are fundamental to ensuring system security. However, if not implemented correctly, attackers can use different techniques to bypass these defenses and gain unauthorized access.

Definition 2.9 (SQL Injection Attack)**.** A *Structured Query Language (SQL) Injection Attack* is an attack where an attacker inserts an SQL statement as input data which does not get properly validated or quoted, leading to the statement being executed on the database. This can result in an attacker being able to read, modify, or delete information, threatening the confidentiality and integrity of a system [Cla12].

2.2.1. Authentication Methods

The authentication methods considered in this thesis are introduced below.

Definition 2.10 (Password-based Authentication)**.** PBA is a knowledge-based authentication method that uses a secret password associated with a unique user identifier, such as a username or email address, to verify a user's identity. Both user identifiers and passwords are stored by the system to which a user authenticates, with passwords stored securely using cryptographic hashing functions [CDW04]. To increase security, PBA is often combined with a different authentication factor to provide MFA.

One of the major problems with PBA is its reliance on the human ability to create and remember strong passwords, particularly given the large number of applications used today [CDW04]. The following authentication method can help reduce the number of passwords a user needs to manage [GN14].

Definition 2.11 (OpenID Connect)**.** *OpenID Connect (OIDC)* is an authentication protocol built on top of OAuth 2.0 that enables a *Relying Party (RP)* to verify the identity of an entity based on authentication performed by an *Identity Provider (IdP)*. The IdP provides an identity token in the form of a *JSON Web Token (JWT)*. This token contains claims that provide details about the authentication event, such as the authentication time, and information about the entity, such as a unique identifier. The IdP digitally signs the JWT using a private key known only to the IdP. This enables the RP to validate the token's integrity and authenticity using the corresponding public key, which the IdP makes publicly available [FKS17].

By delegating authentication to an IdP, the RP does not have to manage credentials. In addition, users can often use the same IdP to authenticate to multiple RPs, reducing the number of passwords to remember. For example, consumer-facing applications often allow users to authenticate using social providers such as Google or GitHub [GN14].

Typically, OIDC and PBA are used only for initial authentication, and tokens are used for subsequent requests. The two Token-based Authentication methods considered in this thesis are introduced below:

Definition 2.12 (Verifiable Token-based Authentication)**.** *Verifiable Token-based Authentication* is a stateless authentication method in which tokens directly contain user information, such as a user identifier, and include a digital signature. The system does not need to store issued tokens, but must verify the authenticity and integrity of received tokens [vYJ22e]. This includes verifying that the token has not expired and that its digital signature is valid.

Verifiable Token-based Authentication provides good performance because all necessary user information is sent with each request, but it increases the amount of data exchanged.

Definition 2.13 (Session-based Authentication)**.** *Session-based Authentication* is a stateful authentication method in which a session identifier, the token, is associated with a session object that contains user information and is stored by the system. The system must ensure that session identifiers are unguessable to prevent attackers from hijacking sessions [vYJ22d].

Session-based Authentication has minimal communication overhead and no sensitive information is exchanged because only the session identifier is sent with requests. However, accessing the session object for each request can degrade performance.

2.3. Security Solution Pattern

In the existing security pattern landscape, most patterns provide a single, rigid solution to a problem, without addressing varying situational needs. The SDPs introduced in this thesis address this limitation by defining a set of SDPs, each focusing on different ways to design a security solution. For example, in PBA, some systems require periodic password changes, while others do not.

However, these design details are not relevant when choosing between different security solution, such as PBA or OIDC authentication. In addition, specifying pattern relationships at the SDP level is impractical because SDPs that belong to the same security solution often share relationships. For example, all PBA SDPs require a Session-based Authentication or Verifiable Token-based Authentication SDP to issue tokens upon successful authentication.

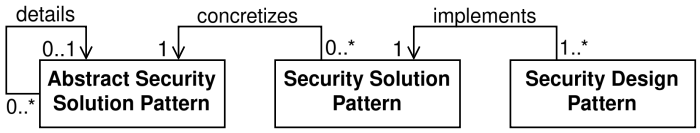


Figure 2.1.: Hierarchy of Security Patterns extended by SDPs [Aba24].

2.3.1. Security Pattern Hierarchy

These issues are addressed by Abazi [Aba24], who introduces a concept to define pattern relationships and provide information that assists in selecting an appropriate security solution without considering design details. To achieve this, Abazi defines two types of security patterns.

Definition 2.14 (Abstract Security Solution Pattern)**.** An *Abstract Security Solution Pattern (ASSP)* specifies a general security concept at an abstract level that can be implemented by different concrete security solutions [Aba24].

The Authenticator pattern [Sch+06] and the Authentication pattern [vYJ22b] can be considered ASSPs. Both patterns address the context of a system that needs to restrict access to protected resources to known subjects, the problem of malicious subjects potentially masquerading as legitimate subjects, and the solution where a subject must provide authentication information to access protected resources. These patterns define the general concept of authentication but lack the specific information needed to implement an authentication solution [vYJ18].

Definition 2.15 (Security Solution Pattern)**.** A *Security Solution Pattern (SSP)* specifies a security solution at a conceptual level, providing the general context, the problem, and the high-level solution to the problem.

In contrast to Definition 2.15, Abazi defines an SSP as a concrete and applicable pattern that provides detailed information for implementing a security solution in an architecture [Aba24]. In this thesis, the SSP definition is adapted to emphasize that SSPs guide architects in selecting appropriate security solutions and provide a high-level understanding without requiring knowledge of the design details. In contrast, SDPs focus on the design details and define how to implement them within an architecture.

ASSPs, SSPs, and SDPs form a hierarchical structure, as shown in Figure 2.1. At the top of the hierarchy is at least one ASSP, and each ASSP can be further detailed by other ASSPs. An SSP then concretizes exactly one ASSP by defining a concrete security solution for it. Finally, each SSP is associated with at least one SDP, which specifies how to securely implement the SSP in an architecture.

PBA is an example of an SSP that concretizes the Authentication ASSP. The pattern may specify that the subject is identified by an email address, that the authentication information is a password, and that a password hash is used to verify identity.

2.3.2. Inter-Pattern Relationships

In this thesis, the security pattern relationships introduced by Schumacher [Sch03] are adapted.

Definition 2.16 (Specializes)**.** Security pattern P_1 *specializes* security pattern P_2 if and only if the context of P_1 is more specific or remains the same as P_2 and the problem of P_1 is more specific or remains the same as P_2 [Sch03].

The Specializes relationship is implicitly included in the security pattern hierarchy in Figure 2.1 such that the *details*, *concretizes*, and *implements* relationships are specializations.

Definition 2.17 (Requires)**.** Security pattern P_1 *requires* security pattern P_2 if and only if there exist a problem in P_1 to which the pattern does not provide a solution but P_2 does [Sch03].

The Requires relationship is critical for a recommender, as a recommender should address any problems that arise from recommending a particular security pattern. For example, the PBA SSP requires a Token-based Authentication ASSP to issue tokens upon successful authentication. If an architect selects the PBA SSP, the recommender should recognize this relationship and suggest an appropriate SSP that Specializes the Token-based Authentication ASSP.

Abazi also introduces inter-pattern relationships, namely the Depends, Benefits, Alternative, Impairs, and Conflicts relationships [Aba24]. However, the proposed definitions are used because they are sufficient for this thesis.

2.3.3. Authentication Hierarchy

This thesis focuses on authentication due to the wide range of available security solutions and its role as a fundamental component in many security patterns. For instance, most authorization security patterns rely on secure authentication solutions [vYJ22a].

In the following, we present the authentication hierarchy of ASSPs and SSPs used in this thesis, along with their interrelationships (see Figure 2.2). While this hierarchy is not exhaustive, it provides a foundation for introducing and demonstrating SDPs.

- Authentication ASSP: It is applicable in systems that need to restrict access to protected resources to known subjects and addresses the problem of malicious subjects masquerading as legitimate ones.
- Single Access Point SSP: It specifies a solution to restrict access to protected resources to known subjects [Sch+06]. It requires the User Authentication ASSP to verify the identity of subjects.
- User Authentication ASSP: It specifies that subjects authenticate using authentication information that is internally compared against evidence [Sch+06; vYJ22b]. It requires the Single Access Point SSP to restrict access to protected resources.

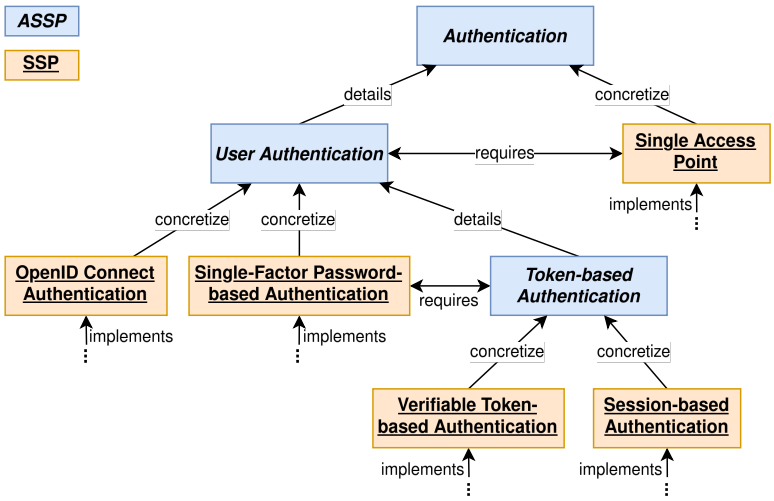


Figure 2.2.: Example of an Authentication Hierarchy of ASSPs and SSPs.

- PBA SSP: It specifies a solution for authenticating subjects based on an identifier and password, as introduced in Definition 2.10. It requires the Token-based Authentication ASSP to issue tokens upon successful authentication, eliminating the need for subjects to provide their password with each request.

- Token-based Authentication ASSP: It specifies that subjects are identified based on a token. It requires the PBA SSP to authenticate a subject before issuing a token.

- Verifiable Token-based Authentication SSP: It specifies a solution for identifying subjects based on a stateless token such as a JWT, as introduced in Definition 2.12.

- Session-based Authentication SSP: It specifies a solution for identifying subjects based on a session identifier, as introduced in Definition 2.13.

- OpenID Connect Authentication SSP: It specifies a solution for identifying subjects based on authentication performed by an IdP. The authentication information is the identity token provided by the IdP, as introduced in Definition 2.11. It does not require the Token-based Authentication ASSP because the identity token could be used as a stateless token to authenticate subsequent requests.

When referring to PBA in this thesis, we consider Single-Factor PBA, which does not include an additional authentication factor unless otherwise stated. In addition, for brevity, the terms OIDC Authentication SSP and OIDC SSP, as well as the terms OIDC Authentication SDP and OIDC SDP are used interchangeably.

2.4. Recommender Systems

This section is based on the work of Felfernig and Burke [FB08] and Aggarwal [Agg16].

Recommender systems assist users in selecting appropriate items, such as movies or music, by providing personalized guidance. *Collaborative Recommender Systems* use the collective preferences and interactions of a large community of users to suggest relevant items. For example, a recommender that recommends hotels based on reviews. *Content-based Recommender Systems*, analyze and match key features of items with an individual user's profile and historical preferences [FB08]. For example, a recommender that recommends music based on the genre of previously listened tracks. However, collecting a large number of user opinions and historical data is impractical for security patterns due to the limited user base and the infrequent nature of system design tasks.

Knowledge-based Recommender Systems address these issues by giving users more control, allowing them to explicitly define their requirements, and often providing an interactive process. *Constraint-based Recommender Systems (CBRSs)*, a subtype of Knowledge-based Systems, are well suited for domains where recommendations are infrequent and items are complex [Agg16]. In such domains, users typically lack complete knowledge of all item details [FB08]. Since both of these characteristics apply to security patterns due to the large variety and complexity of the security landscape, this thesis considers the use of a CBRS for SDPs.

2.4.1. Constraint-based Recommender Systems

This section introduces KBs, which are essential for a CBRS to recommend and reason about appropriate items. In addition, we briefly describe how a CBRS can use a KB for recommendations.

Definition 2.18 (Constraint-based Recommender Knowledge Base)**.** A *Constraint-based Recommender Knowledge Base (KB)* is defined as a tuple $(V_P, V_C, PROD, COMP, FILT)$, where V_P is the set of *Product Variables*, V_C is the set of *Customer Variables*, $PROD$ is the set of *Product Constraints*, $COMP$ is the set of *Compatibility Constraints*, and $FILT$ is the set of *Filter Constraints*.

The following describes each component of a KB and provides examples.

Product and Customer Variables

The Product Variables V_P define the properties that describe the products and their possible values. For example, in a KB for buying real estate, the Product Variables could be defined as follows. Specific product instances are provided in Table 2.1.

$$\begin{aligned} V_P = \{\ & p_1 : \mathit{Identifier} \in \{\mathit{prod}_i \mid i \in \mathbb{N}\}, \\ & p_2 : \mathit{Number\ of\ Bedrooms} \in \mathbb{N}, \\ & p_3 : \mathit{Price} \in \mathbb{N}, \\ & p_4 : \mathit{Type} \in \{\mathit{house}, \mathit{apartment}\}\ \} \end{aligned}$$

Identifier	***Number of Bedrooms***	***Price***	***Type***
prod_1	5	500000	house
prod_2	3	400000	house
prod_3	3	350000	house
prod_4	2	200000	apartment

Table 2.1.: Example products in a CBRS KB for buying real estate.

The Customer Variables V_C define the user requirements that can be specified by the users. For example, in a KB for buying real estate, the Customer Variables could be the following:

$$\begin{aligned} V_C = \{\ & c_1 : \textit{Property Type} \in \{\textit{house}, \textit{apartment}, \textit{either}\}, \\ & c_2 : \textit{Minimum Number of Bedrooms} \in \mathbb{N}, \\ & c_3 : \textit{Minimum Number of Bathrooms} \in \mathbb{N}, \\ & c_4 : \textit{Budget} \in \mathbb{N}\ \} \end{aligned}$$

User requirements $c_i \in V_C$ can be categorized into three groups [FB08]:

- *Constraint*: No recommended item is allowed to violate a constraint. For example, the "property type" Customer Variable can be classified as a Constraint, because no apartment should be recommended when searching for a house.
- *Preference*: May be violated for a good reason. For example, the "budget" Customer Variable can be classified as a Preference, because a house that is slightly over budget but has a large garden and a great location may still be selected.
- *Context*: External circumstances not directly specified by the user. For example, the user's location when recommending restaurants.

Constraints

Product Constraints, i.e., *PROD*, define conditions between Product Variables that ensure that products meet certain inherent criteria [FB08]. For example, the following Product Constraint represents that each house should have at least two bedrooms:

$$\textit{Type} = \textit{house} \ \Rightarrow\ \textit{Number of Bedrooms} \geq 2$$

Compatibility Constraints, i.e., *COMP*, define conditions between Customer Variables to ensure consistency and feasibility [FB08]. For example, the following Compatibility Constraint ensures practical consistency in user requirements:

$$\textit{Minimum Number of Bedrooms} \geq 5 \ \Rightarrow\ \textit{Minimum Number of Bathrooms} \geq 2$$

Filter Constraints, i.e., *FILT*, map Customer Variables to Product Variables, defining how user requirements relate to product properties [FB08]. For example, the following

Filter Constraints map the "property type" Customer Variable to the "type" Product Variable:

$$\begin{aligned} \textit{Property Type} = \textit{house} &\Rightarrow \textit{Type} = \textit{house} \\ \textit{Property Type} = \textit{apartment} &\Rightarrow \textit{Type} = \textit{apartment} \\ \textit{Property Type} = \textit{either} &\Rightarrow \textit{Type} \in \{\textit{house}, \textit{apartment}\} \end{aligned}$$

The advantage of a KB is that user requirements, i.e., $c_i \in V_C$, and product properties, i.e., $p_i \in V_P$, can be defined separately. This allows users to express their requirements without needing to understand the technical details of the products [FB08]. For example, consider a "Neighborhood Type" Customer Variable that lets a user specify the type of neighborhood they prefer to live in:

$$c : \textit{Neighborhood Type} \in \{\textit{family friendly}, \textit{college}, \ldots\}$$

This Customer Variable can be used to create the following Filter Constraint:

$$\begin{aligned} \textit{Neighborhood Type} = \textit{family friendly} \Rightarrow\ & \textit{Crime Rate} = \textit{low} \land \\ & \textit{Distance to Nearest School} \leq 1000\text{m} \land \\ & \textit{Distance to Nearest Park} \leq 500\text{m} \end{aligned}$$

Recommendation Process

This thesis primarily considers KBs for SDPs, which define the knowledge required for a CBRS to recommend and reason about SDPs. However, as this knowledge must be usable within the recommendation process, we provide a general overview of the steps a CBRS takes, as described by Felfernig and Burke [FB08]:

1. Collect user requirements $c_i \in V_C$ and ensure they satisfy all Compatibility Constraints.
2. Identify all products that satisfy the Filter Constraints with respect to the user requirements.
3. If no suitable product is found, identify conflicts within the user requirements and guide the user to enter feasible requirements.
4. Calculate a recommendation score for all appropriate products to rank them.

For example, Joe wants to buy a house and connects to a real estate buying CBRS. The CBRS asks a few questions that result in the following user requirements: $\textit{Property Type} = \textit{house}$, $\textit{Budget} = 450000$, and $\textit{Number of Bedrooms} \geq 4$. Based on the requirements, the CBRS is unable to find any houses in Table 2.1 that meet these requirements. In order to make a recommendation, the CBRS determines that Joe must either increase the budget to 475000 or decrease the number of bedrooms required to three. Joe decides that he is comfortable with only three bedrooms, so $\textit{prod}_2$ and $\textit{prod}_3$ can be recommended. The CBRS ranks both houses by price, so $\textit{prod}_3$ is ranked above $\textit{prod}_2$. Joe agrees with the CBRS and decides to buy house $\textit{prod}_3$.

3. Problem Statement and Method

In their foundational work, Alexander et al. [Ale+77] define patterns for buildings and towns as core solutions to recurring problems, enabling the creation of multiple concrete structures. Building upon this concept, patterns in software engineering facilitate the creation and implementation of robust software systems [Gam+95]. Although patterns are widely used in various areas such as architecture and object-oriented design, their application in the security domain remains limited [vYJ18]. This limited adoption is not due to a shortage of recurring problems or known solutions. Even when focusing exclusively on authentication, numerous problems and solutions exist [Bar+18]. Furthermore, many security patterns and catalogs are available, indicating that the availability of patterns is not an issue [Hey+07].

The limited adoption of security patterns suggests that other factors impede their practical use. One issue is that existing security patterns often provide maintainable solutions while overlooking security-relevant information [vYJ18]. For example, the Security Session pattern [Sch+06] specifies how to implement Session-based Authentication introduced in Definition 2.13. However, the pattern does not explain how to ensure that session identifiers are unguessable, and even suggests using the user identifier in some situations, which is insecure [vYJ18]. Similarly, the Authenticator pattern [Sch+06] accurately represents a high-level authentication process but omits details about the nature of authentication information and how it should be securely stored and verified [vYJ18]. These limitations lead to the first research question:

> **RQ1**: *What information is essential in a security design pattern, and how should a pattern description be structured to make it practical for architects?*

To address RQ1, we developed a metamodel that defines the elements and relationships constituting the usage aspect of SDPs. This metamodel ensures that SDPs follow a consistent structure and include the necessary security-relevant information, thereby improving their usefulness for architects.

Another issue is that existing security patterns often provide a single, rigid solution. They overlook that some elements may not be needed in some contexts, that alternatives may exist, or that solutions not considered best practices may still be required in specific situations. For example, the Password-based Authentication pattern [vYJ22c] requires using a static pepper, a secret value added to passwords during hashing, in addition to a unique salt for each password. While using a pepper can improve security, it introduces complexity, as the pepper must be stored securely and periodically rotated, which can be challenging [Meg24]. In addition, the pattern does not specify how side channels such as email or SMS are involved in password recovery, which is a critical security consideration.

We address this issue by providing multiple SDPs, each of which considers different design options for a security solution. To help architects avoid manually examining every available SDP, we consider using a *Constraint-based Recommender System (CBRS)* to recommend an appropriate SDP. As introduced in Section 2.4.1, the first step in a CBRS recommendation process is to collect requirements to determine all feasible products that can be recommended. Therefore, it is necessary to identify the information that influences which SDPs can be recommended. We refer to this information as recommendation factors, which leads to the second research question:

> **RQ2**: *What recommendation factors can influence the selection of an appropriate security design pattern?*

To further support architects in selecting an appropriate SDP, the CBRS calculates a recommendation score for each feasible SDP, ranking them according to their suitability. However, this requires quantifiable knowledge about SDPs that can be used in the calculation. This need leads to the third and fourth research questions:

> **RQ3**: *What knowledge about a security design pattern must be captured in order to reason about an appropriate pattern and explain the recommendation?*

> **RQ4**: *How should the knowledge be encoded so that a knowledge-based recommender system can reason about appropriate security design patterns?*

To address RQ2, RQ3, and RQ4, we developed a metamodel that specifies the elements and relationships of a *Knowledge Base (KB)* for SDPs, including how the knowledge aspect of SDPs is represented. This metamodel includes both the recommendation factors and the properties of SDPs that influence the recommendation results.

3.1. Research Method

This section presents the method used to develop the metamodels.

3.1.1. Acquisition and Conceptualization

At a high level, we applied the approach introduced by Andrade et al. [And+04]. Their approach emphasizes that to solve a non-trivial problem, such as creating metamodels that describe the usage and knowledge aspects of SDPs, one must acquire information that is then organized into a conceptual model.

This approach is illustrated in their iterative process shown in Figure 3.1, which we adapted for our research. It consists of the *Acquisition* step, where information is gathered, and the *Conceptualization* step, where the acquired information is used to create the conceptual model. The iterations are necessary because the acquisition is usually incomplete or incorrect, which is discovered during the conceptualization [And+04].

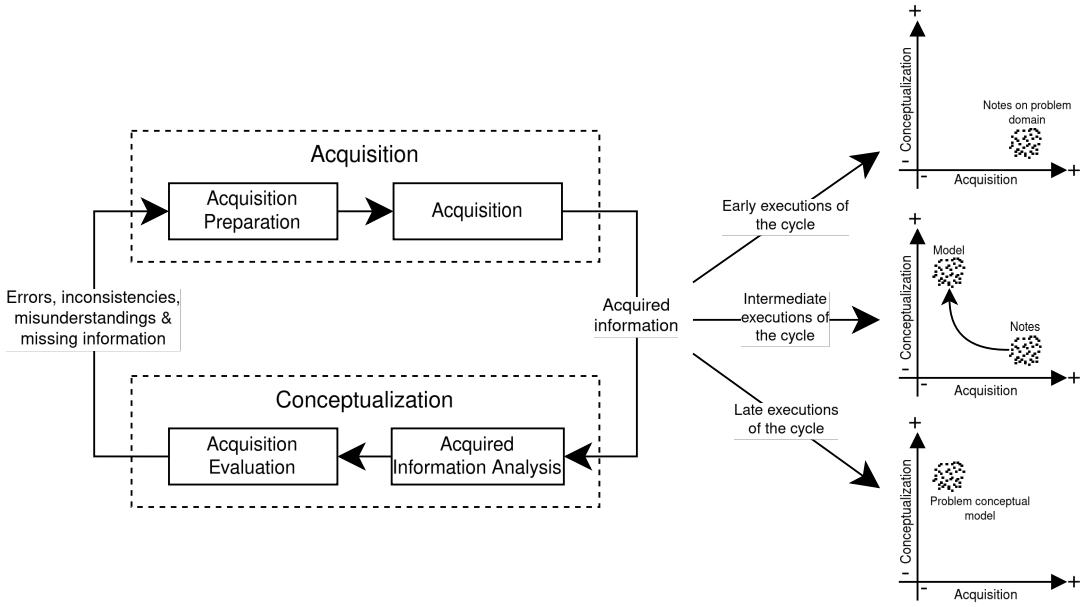


Figure 3.1.: Iterative evolutionary process for developing a conceptual model for a non-trivial problem. The process consists of two steps: information acquisition and conceptualization [And+04].

Acquisition is divided into two steps: acquisition preparation, such as identifying appropriate sources; and acquisition itself, such as collecting information from the sources. Similarly, conceptualization is divided into two steps: analysis of the acquired information, such as extracting concepts from the collected information; and evaluation of the concepts, such as verifying their general applicability [And+04].

Typically, the focus during the initial iterations is on acquisition, whereas later iterations primarily use acquisition to validate and fine-tune the conceptual model as shown in Figure 3.1. By following this iterative process, we were able to systematically develop the metamodels that describe the usage and knowledge aspects of SDPs.

In the following sections, we describe how we conducted acquisition and conceptualization in an initial bottom-up approach followed by a top-down approach.

3.1.2. Bottom-Up Approach

In the bottom-up approach, the acquisition consisted of extracting security solutions from *Open Source Software (OSS)*. In the conceptualization, these solutions were analyzed to identify common elements and concepts that could be used in a metamodel.

To achieve this, we identified suitable OSS during the acquisition preparation. The primary requirement for the OSS under consideration was that they contain explicitly defined architectural documentation that includes security-related information. While

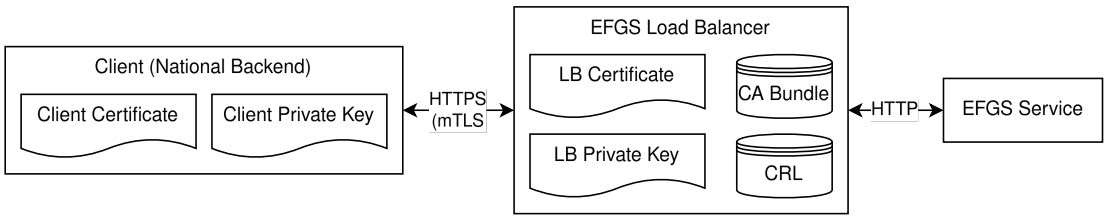


Figure 3.2.: Security solution example for the authentication between the EFGS and National Backends using mTLS.

such documentation could also be created by analyzing the OSS, this is beyond the scope of this thesis project.

Sabau [Sab24] identified the *Corona-Warn-App (CWA)*[1], the official COVID-19 exposure notification app for Germany, as suitable for a bottom-up analysis. This is because data security and privacy were considered from the beginning and were important factors for its acceptance by the population [Dix20; Las21]. In addition, the CWA contains well-documented architectural descriptions that explicitly include security solutions.

We identified the *European Federation Gateway Service (EFGS)*[2] as another suitable OSS. The EFGS allows different national contact tracing applications, such as the CWA, to exchange information, enabling interoperability between solutions. Similar to the CWA, security was an important concern during the design and development of the EFGS, and the architecture descriptions include security requirements and security solutions [Dir20].

Other promising OSS include the *openDesk Project*[3], the *EUdi-Wallet Project*[4], and the *National Once-Only-Technical-System*[5], because they provide architectural descriptions, and security is an important concern for each project. However, these projects were too early in the development process to be useful in the bottom-up approach. Nevertheless, the openDesk and EUdi-Wallet projects can be useful for future work because they use OIDC, for which we provide SDPs. Thus, analysis of these projects may reveal concrete implementations of, or problems with, our OIDC SDPs.

Security Solution and Concept Example

The security solution in Figure 3.2 is an example of the information we gathered, analyzed, and transformed into concepts.

The example shows the static structure of the authentication between the *National Backends (NBs)* and the EFGS. In this process, both parties exchange their certificates and provide a digital signature based on their private keys. The other party then verifies the signature using the certificate provided, thereby proving the sender's identity and

[1] `https://github.com/corona-warn-app`
[2] `https://github.com/eu-federation-gateway-service`
[3] `https://gitlab.opencode.de/bmi/opendesk`
[4] `https://gitlab.opencode.de/bmi/eudi-wallet`
[5] `https://gitlab.opencode.de/bmi/noots`

ensuring the integrity of the communication. The *Certificate Revocation List (CRL)* contains potentially compromised certificates that should not be accepted.

This security solution led to a concept to classify data based on whether it can be publicly known, such as the certificates, or must remain secret, such as the private keys. In addition, we identified that data often needs to meet specific requirements to ensure the security of a solution. For example, the private keys in Figure 3.2 must have a minimum length to prevent attackers from breaking them [Fed24]. Including such requirements in the SDPs is critical because architects cannot be expected to be aware of these details due to the complexity and rapid evolution of the security landscape [LS24].

Furthermore, we identified that security solutions typically involve interactions between entities and are designed to protect specific entities. In this case, the interaction is between the NB and EFGS, and the security solution protects both the EFGS from receiving data from a malicious actor and the NB from sending data to one.

Finally, security solutions often include enforcement mechanisms based on decisions and information. Here, the EFGS enforces that only authenticated NBs are allowed to communicate based on the decision of whether the signature provided corresponds to the NB. In addition, the certificate revocation list provides information about compromised certificates that should not be accepted.

These concepts were iteratively refined by considering various security solutions identified during the acquisition. During conceptualization, we assessed whether the existing concepts could be applied to these security solutions, and if not, we evaluated why not and adapted the concepts accordingly.

3.1.3. Top-down Approach

The goal of the top-down approach was to develop the usage and knowledge aspects of a concrete SDP and to capture the elements and relationships used in the metamodels. The bottom-up approach can be viewed as the initial acquisition preparation, as we used the identified concepts during the creation of the SDPs.

As part of this initial acquisition preparation, we decided to focus on OIDC authentication. We selected OIDC authentication because it is not included in any of the identified security pattern catalogs, helping to reduce bias. In addition, OIDC is widely used in practice, as many applications support login via social providers such as Google or GitHub [GN14], making the pattern relevant to industry

Furthermore, OIDC is a versatile authentication protocol that allows for a variety of design and implementation approaches. For example, there are different methods to authenticate the system to the identity provider, and a system may use one or more identity providers, both of which influence design and implementation. This allowed us to examine how these different aspects can be integrated into SDPs and what information a recommender needs to recommend an appropriate SDP.

In the top-down approach, the development of the SDP was the acquisition that, along with the insights from the identified literature, was used to create the metamodels in the conceptualization. In addition, we divided the top-down approach into two phases.

In the first phase, we developed the usage aspect of an OIDC SDP until only minor refinements were required for both the SDP and the metamodel. In the second phase, we focused on the knowledge aspect of the SDP and explored how to incorporate it into an OIDC SDP KB. This included identifying the different OIDC SDPs that might exist, as this information is essential to the KB.

Next, we selected a second OIDC SDP from those identified and developed its usage aspect to demonstrate the need for multiple SDPs implementing a single SSP. We also used this step to further refine the usage aspect metamodel following the iterative process. It was not necessary to develop the knowledge aspect of this second SDP, as it had already been incorporated into the OIDC SDP KB when we developed the first OIDC SDP.

Finally, we developed a PBA SDP to demonstrate that our metamodels can be applied beyond a single security solution. We followed the same two-phase approach: first focusing on the usage aspect, then addressing the knowledge aspect, which involved identifying different PBA SDPs and integrating them into a PBA KB. These steps allowed us to further refine both metamodels.

4. Related Work

This chapter reviews existing research on *Security Patterns (SPs)*, with a particular focus on SP catalogs. This focus aligns with the goal of this thesis to facilitate the creation of diverse and generally applicable SPs that form an SP catalog.

We examine the catalogs by Schumacher et al. [Sch+06] and Fernandez [Fer13] together, as they share the same structure and contain overlapping SPs, such as the Authenticator pattern. In addition, we examine a more recent catalog by Van den Berghe et al. [vYJ22a].

4.1. Security Pattern Description

In this section, we examine the structure and relationships of SPs in these catalogs. We begin by considering how the catalogs represent and distinguish between patterns at different levels of abstraction, and how pattern relationships are depicted. We then consider the pattern descriptions used within the catalogs. Finally, we examine how these catalogs address different implementation options for a security solution.

4.1.1. Abstraction Levels and Pattern Relationships

The SPs in the catalogs by Schumacher et al. [Sch+06] and Fernandez [Fer13] vary in their level of abstraction and the information they provide. For example, the Authenticator pattern [Sch+06] describes the general problem and solution of authentication at a high level. In contrast, the Credential pattern [Fer13] describes how to securely handle and exchange authentication information between systems, serving as a specific variant of the Authenticator pattern.

This is consistent with our approach, where we consider different types of SPs, including *Abstract Security Solution Patterns (ASSPs)*, which provide information about general security concepts such as authentication, and *Security Design Patterns (SDPs)*, which provide the information needed to implement a concrete security solution.

However, the SPs by Schumacher et al. [Sch+06] and Fernandez [Fer13] do not indicate whether a pattern is a concrete pattern intended for implementation or an abstract pattern providing general information. In addition, the relationships between the abstract and more concrete patterns are not properly defined, making it difficult to understand which pattern is based on another. Finally, even the more concrete patterns, such as the Credential pattern [Fer13], do not provide all the necessary information for secure implementation [vYJ18].

To address this issue, the pattern hierarchy defined by Abazi [Aba24] and extended by our SDPs clearly specifies each pattern's level of abstraction and explicitly illustrates the

pattern relationships. Thus, architects immediately know what information details are expected based on the type of SP, i.e., ASSP, SSP, and SDP, and where more detailed information can be found.

The SP catalog of Van den Berghe et al. [vYJ22a] explicitly defines a pattern hierarchy, where each pattern specifies its parent and child SPs. In contrast to our approach, Van den Berghe et al. [vYJ22a] include explicit references between the solution of an SP and its parent. For example, their Authentication pattern [vYJ22b] defines that an entity authenticates using a credential that is internally compared to evidence. In their PBA pattern [vYJ22c], the combination of an identifier and a password refers to the credential, and a password hash refers to the evidence.

These explicit relationships between SPs help in understanding the patterns because an architect can first become familiar with the abstract pattern and, through the references, understand how the concrete pattern implements it. This approach is not adopted in this thesis because the ASSPs and SSPs defined by Abazi [Aba24] currently only provide the knowledge aspect required by a CBRS to recommend patterns. Future work defining the usage aspects of ASSPs and SSPs should consider whether the inclusion of explicit relationships between pattern elements can and should be applied.

4.1.2. Pattern Description

The SP catalogs by Schumacher et al. [Sch+06] and Fernandez [Fer13] follow a pattern template that defines the high-level parts of the SPs, such as context and problem, but do not provide much detail about the elements that make up each part. While this allows flexibility in structuring the SPs, adopting a more standardized description enhances consistency and improves understandability. For example, by clearly defining how inter-pattern relationships are represented within a pattern or how important data considerations are provided, an architect familiar with the structure can easily identify these elements.

In addition, explicitly defining how security-relevant information is integrated into SPs helps architects implement an SP correctly and allows security experts to provide better feedback. For example, the Security Session pattern [Sch+06] states that a session identifier should be unguessable. By explicitly specifying methods for generating an unguessable session identifier, architects can implement this effectively, and security experts can review the methods to ensure their security or recommend necessary changes.

The catalog by Van den Berghe et al. [vYJ22a], similar to our approach, uses a metamodel that defines the elements and relationships that make up an SP and integrates security-relevant information. This ensures that SPs follow a consistent structure, so that architects familiar with it immediately know where and how information is defined.

However, their metamodel is tailored to use a security-oriented modeling language to represent the behavior of a pattern, while the remaining parts are based on textual descriptions. In contrast, our metamodel does not restrict how behavior can be expressed, allowing the use of common diagrams, such as sequence diagrams, that we expect architects to be familiar with.

Furthermore, their metamodel focuses only on the security aspects and provides no guidance on how to integrate the patterns into an architecture. In contrast, SDPs explicitly include practical examples that can help implement the pattern in an architecture.

Moreover, since there can typically be a number of SDPs for an SSP, relying on a textual description (e.g., for data considerations) can make it difficult to compare two SDPs. Therefore, our metamodel emphasizes diagrams as central elements that contain all relevant information or references for clarity. Nevertheless, the catalog and metamodel of Van den Berghe et al. [vYJ22a] provide important aspects that are considered in our metamodel.

4.1.3. Pattern Details

The catalogs of Schumacher et al. [Sch+06], Fernandez [Fer13], and Van den Berghe et al. [vYJ22a] provide only a single, rigid solution for a security mechanism. This approach simplifies pattern selection and reduces the number of patterns that need to be created. However, it can lead to patterns that ignore aspects required in specific situations. For example, the PBA pattern [vYJ22c] discourages the use of periodic password changes because they are generally not recommended, while some standards and organizations still require their use [PCI24].

We address this issue by providing multiple patterns that consider specific design options, such as one PBA SDP that incorporates periodic password changes and another that does not. A CBRS for SDPs then guides architects in selecting an appropriate SDP.

4.2. Security Pattern Selection

In this section, we consider how to identify an appropriate SP within the catalogs.

Yskout et al. [YSJ12] consider the use of pattern annotations to guide the selection process. These annotations include key security objectives, such as confidentiality or availability, and quality tradeoffs that consider quality attributes, such as integrity or maintainability. In the selection process, architects first consider the security objective, and if multiple patterns exist, use the quality tradeoffs to decide among them.

While this approach may be effective for SSPs, SDPs typically share the same security objectives defined by the SSP they implement but provide different variants for that SSP. In addition, there may be a variety of SDPs for an SSP that address different design options, such as different password reset mechanisms and whether periodic password changes are required for PBA. Since SDPs can be quite detailed, an architect cannot be expected to examine every possible SDP to select the appropriate one.

The SP catalogs of Schumacher et al. [Sch+06] and Fernandez [Fer13] are based on grouping patterns by their security objective, such as patterns relevant for identification and authentication or access control methods. Thus, in contrast to the approach of Yskout et al. [YSJ12], where additional information such as quality tradeoffs supports selection within a group, this method requires the reader to be familiar with all patterns in a group. Moreover, due to the relationships between patterns, it is likely that a reader

will need to be familiar with most of the catalog to use it effectively, even if only a few specific patterns are needed.

The SP catalog by Van den Berghe et al. [vYJ22a] uses a more structured approach based on a hierarchy that separates the context and problem from the patterns themselves. This means that an architect first identifies a context that is applicable to their system, such as the system interacting with users. Each context then has at least one associated problem; for example, in this context, a user could pretend to be someone else. Each problem then references at least one SP that provides a solution for the problem, such as PBA [vYJ22a]. This approach allows an architect to identify a set of SPs needed to design a secure system by identifying all relevant contexts and problems and then selecting a pattern for each.

However, while some problems reference a concrete pattern, other problems only reference a high-level pattern such as authentication. Although these high-level patterns specify which patterns implement them there is no further guidance on how to choose among these child patterns.

This is not an issue in their current catalog, as it only contains a limited number of patterns. However, considering that there are numerous concrete patterns for authentication, such as PBA, OIDC Authentication, Biometric-based Authentication, and more, additional guidance is needed on how to choose between them. The decision becomes even more complex when considering that each of these patterns may have multiple SDPs.

Nevertheless, we agree that separating the pattern descriptions, especially the pattern solution, from the selection process allows architects to identify appropriate patterns more easily. In this thesis this is achieved by an SDP comprising a usage aspect used by an architect to implement the pattern, and a knowledge aspect integrated in a KB allowing a CBRS to recommend an appropriate SDP.

5. Security Design Pattern Description Metamodel

This chapter introduces the *Security Design Pattern Description Metamodel (SDPDM)*, which defines the usage aspect of SDPs. The SDPDM addresses the limitations of existing security patterns that are often too abstract and lack the detailed information required for secure implementation. By clearly specifying the elements and relationships that make up SDPs, the SDPDM ensures that SDPs follow a consistent structure, making them easier to understand and apply.

5.1. SDPDM Structure

The SDPDM adapts the security pattern structure proposed by Heyman et al. [Hey+07]. According to Heyman et al., a good security pattern description must include *Context* and *Problem*, which define the situations in which the pattern is applicable and the problems it addresses.

Next, a security pattern must include a *Solution* to the Problem within the given Context. This Solution should specify both the structure and behavior, preferably presented graphically [Hey+07]. To facilitate this, the following sections introduces the SDPDM viewpoints that enable the creation of practical and understandable SDP Solutions. Furthermore, a security pattern should provide an *Example* demonstrating how the Solution can be implemented in practice [Hey+07]; this is included as part of the SDPDM viewpoints. Lastly, a security pattern should define its *Consequences*, detailing both the strengths and weaknesses of the pattern [Hey+07].

Building upon the structure proposed by Heyman et al. [Hey+07], SDPs require a short non-technical *Summary* at the beginning. This practice can also be found in the pattern catalog by Van den Berghe et al. [vYJ22a]. The Summary should include high-level information covering the Context, Problem, and Solution, and may also highlight the most important Consequences. This enhances comprehensibility, especially for non-security experts.

5.2. SDPDM Viewpoint Structure

For the description of the Solution and the Examples within the SDPDM, we adapt the viewpoint and view concept introduced by Rozanski and Woods [RW11]. This concept is commonly used to model architectures to capture different aspects where using a single model is impractical. In the SDPDM, the viewpoint and view concept help manage

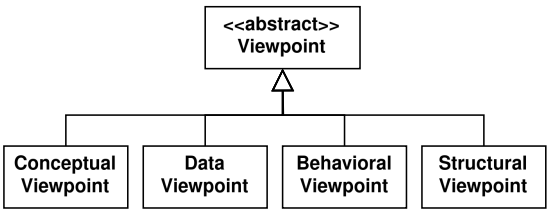


Figure 5.1.: The SDPDM's Viewpoints that define how to represent the Solution and Example of an SDP.

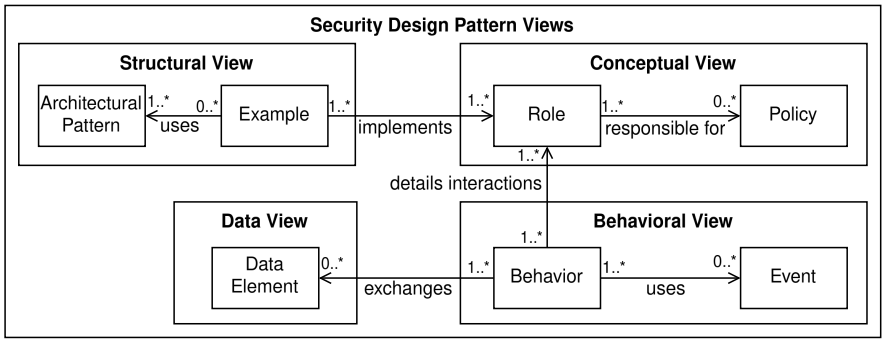


Figure 5.2.: Overview of an SDP that conforms to the SDPDM Viewpoints. Each view highlights its main elements and their relationships.

the complexity of the security domain by focusing on specific aspects at different levels of abstraction. To achieve this, we have identified four viewpoints, which are shown in Figure 5.1.

Before introducing the different Viewpoints, it is essential to note that a viewpoint outlines the elements, relationships, and modeling conventions used to construct a view. In contrast, a view describes a specific aspect that adheres to a particular viewpoint [RW11]. The following provides an overview of each Viewpoint and its main elements, as shown in Figure 5.2. The remaining sections introduce these elements and their relationships in more detail. It is important to note that elements labeled as abstract in a viewpoint do not have concrete manifestations within a view.

Definition 5.1 (Conceptual Viewpoint)**.** The *Conceptual Viewpoint* specifies how *Policies* and *Roles* are represented. Policies define the rules and conditions required to resolve the Problems of an SDP. Roles define the responsibilities required to implement the Policies.

The Conceptual View of an SDP clearly defines the scope of an SDP Solution by specifying the Policies to be considered and the specific Roles with their responsibilities and relationships. It serves as the foundation for other views and provides a high-level understanding of the SDP Solution.

Definition 5.2 (Data Viewpoint)**.** The *Data Viewpoint* defines how to model the *Data Elements* involved in an SDP, including their structure, properties, and security considerations.

The Data View of an SDP specifies which Data Elements are involved in the SDP Solution, how these Data Elements relate to each other, and assigns security-relevant information to the Data Elements. This information includes, for example, the impact that disclosure of a Data Element would have on the security of a user or system. The Data View is critical to the secure implementation of an SDP because improper data design can result in an insecure system [DJS19].

Definition 5.3 (Behavioral Viewpoint)**.** The *Behavioral Viewpoint* specifies how to model the detailed interactions between Roles and how to represent *Events*. An Event represents a specific situation that occurs during these interactions and allows important security considerations to be defined.

The Behavioral View of an SDP models the behaviors required to securely implement the Policies. The modeled behavior typically involves sending and receiving data using the Data Elements modeled in the Data View. Events can highlight and describe important situations within the modeled behavior, representing both exceptional (e.g., "Password Incorrect") and informational (e.g., "Successful Authentication") situations. By using Data Elements and Events, behavior can be modeled in an understandable way while still containing the security-relevant information defined within these elements.

Definition 5.4 (Structural Viewpoint)**.** The *Structural Viewpoint* specifies how to represent the Roles within an architecture. It takes into account both the static structure, i.e., the specific elements and their arrangement, and the dynamic structure, i.e., the runtime elements and their interactions [RW11].

The Structural View of an SDP provides practical examples that facilitate the implementation of the SDP by giving architects a well-documented starting point.

The following sections describe in detail the elements and relationships within each Viewpoint, as well as their relationships to other Viewpoints. We begin with the Conceptual Viewpoint because it is the foundation of the SDPDM Solution. This is followed by the Data Viewpoint and the Behavioral Viewpoint due to their interrelationships and to complete the SDPDM Solution. Finally, the Structural Viewpoint is introduced.

5.3. Conceptual Viewpoint

This section introduces the Conceptual Viewpoint, which outlines the essential elements for understanding an SDP Solution and serves as the foundation for the other Viewpoints.

Definition 5.5 (Entity)**.** An *Entity* is any system, component, individual, organization, or group that interacts with or is part of a system.

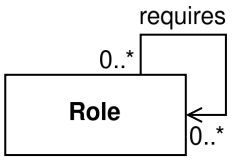


Figure 5.3.: A Role can require other Roles to fulfill its responsibilities.

This definition is adapted from Rozanski and Woods' definition of an external entity [RW11]. Examples of Entities include users, user groups such as administrators, machines or devices, external systems, internal services, and more.

Definition 5.6 (Policy)**.** A *Policy* defines rules and conditions that apply to Entities when an action is performed. A Policy is considered satisfied when all of its rules and conditions have been successfully verified.

For example, in a *Password-based Authentication (PBA)* SDP, a "Registration" Policy defines that a user must provide a valid email address and a strong password, and must complete email verification to register a user account. Satisfying the valid email requirement involves deciding whether the email format is correct and whether the email already exists in the system.

Policies form the foundation of an SDP by clearly describing its scope and how security is maintained.

Definition 5.7 (Role)**.** A *Role* encapsulates a set of responsibilities necessary for the secure application of Policies. A Role is characterized by a unique name and a description of its responsibilities.

For example, in a PBA SDP, the "Password Policy Verifier" Role is responsible for verifying that a given password is strong, such as having a minimum length.

Adapted from Van den Berghe et al. [vYJ22a], Roles highlight what is required to securely implement the Policies. In addition, a Role does not need to correspond to a particular element within an architecture. In practice, multiple Roles may be implemented by the same element, or multiple elements may implement a single Role [vYJ22a].

Definition 5.8 (Requires Relationship)**.** The *Requires* relationship between Roles defines that one Role depends on another Role to fulfill its responsibilities.

For example, in a PBA SDP, a "Registrar" Role is responsible for facilitating the registration, and a "Password Resetter" Role is responsible for facilitating the password reset. Both Roles Require the "Password Policy Verifier" Role to ensure that users use strong passwords.

The Requires relationship, shown in Figure 5.3, is critical to defining an understandable Conceptual View because it supports the definition of fine-grained Roles and indicates the high-level Role interactions.

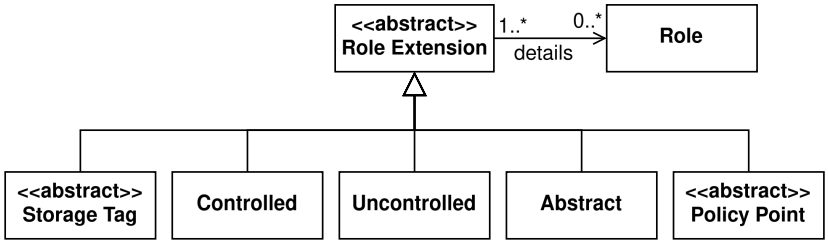


Figure 5.4.: The types of Role Extensions and their relationship to Roles.

Definition 5.9 (Role Extension)**.** A *Role Extension* defines common responsibilities of Roles or characteristics that describe Roles.

Although the responsibilities of Roles vary between different SDPs, there are common responsibilities and characteristics that are generally applicable. Introducing Role Extensions allows for assigning these properties to Roles and enables extensibility by defining additional Role Extensions. The following details the Role Extensions introduced in this thesis, as shown in Figure 5.4.

Definition 5.10 (Storage Tag)**.** A *Storage Tag* specifies that a Role requires some type of storage to fulfill its responsibilities. It enables the definition of different types of storage and the considerations that come with each type, especially regarding security.

The following Storage Tags are considered in this thesis:

- *Persistent Storage* is a long-term storage. It requires regular backups to prevent data loss, and these backups must be stored securely to prevent attackers from exploiting them.
- *Cache* is a short-term storage that stores frequently accessed or large data to increase performance and availability. It requires proper handling of cache misses and cache invalidation processes.
- *Session Storage* is a short-term storage that stores data accessible via an identifier. It requires proper session expiration mechanisms and the identifier must be unguessable.
- *Secret Storage* is a secure client-side storage. It requires different considerations depending on the client types. For example a web application should store secret values using secure cookies and avoid storing them in vulnerable web storage. For further details, see Appendix A.6.

Data storage typically requires special considerations to ensure its security. Storage Tags allow these considerations to be defined in a single location. This ensures that considerations are consistent across SDPs and improves the ability to develop new SDPs.

We provide detailed considerations only for the “Secret Storage” Storage Tag in Appendix A.6 due to the time constraints of this thesis project.

Definition 5.11 (Controlled)**.** A *Controlled* Role is one whose operations are entirely under the control of the architects. This means that every operation the Role performs, as well as the effects of any operations it receives from other Roles, are fully specified [vYJ22a].

Definition 5.12 (Uncontrolled)**.** An *Uncontrolled* Role is one whose operations are not entirely under the control of the architects. This means that the operations the Role performs, and the effects of operations it receives from other Roles, could be influenced or manipulated by an attacker [vYJ22a].

For example, the "Password Policy Verifier" Role is considered Controlled. This means that when it verifies that a password is strong, no additional validation is required. A "User" Role can be considered Uncontrolled because, for example, user input such as an email address provided during login must be validated to protect against SQL injection attacks [DJS19].

The distinction between Controlled and Uncontrolled Roles highlights where potential threats and vulnerabilities may originate. Furthermore, following Van den Berghe et al. [vYJ22a] using Controlled and Uncontrolled emphasizes that Roles are not elements in an architecture, but only describe responsibilities. This distinguishes these concepts from concrete elements where the terms *Trusted* and *Untrusted* are typically used. For example, in a zero-trust microservice architecture, two Controlled Roles can be designed as separate services. Although both Roles are Controlled, they do not necessarily trust each other [Ros+20].

Definition 5.13 (Abstract)**.** An *Abstract* Role is essential to the secure application of Policies, but its design is not relevant or within the scope of the SDP itself. An Abstract Role may reference another security pattern, i.e., ASSPs, SSPs as well as SDPs, if that pattern addresses the Role's responsibilities.

For example, in a PBA SDP, an Abstract "Token Manager" Role is responsible for issuing, verifying, and invalidating tokens. This Role is a black box that references the Token-based Authentication ASSP introduced in Section 2.3.3. Including this Role indicates which Roles require these responsibilities, while leaving the design details to another security pattern.

Abstract Roles address a common problem with existing security patterns, where pattern relationships are often mentioned only informally, despite being critical information [Hey+07].

Another example, in a PBA SDP, is an Abstract "Email System" Role that is responsible for delivering verification and reset emails. The design of the email system is outside the scope of the SDP and is not defined in a separate security pattern. However, the Role is important to indicate which Roles need to send email.

Definition 5.14 (Policy Point)**.** A *Policy Point* describes specific responsibilities required to ensure an action defined by a Policy is only allowed when all of the Policy's rules and conditions have been verified. A Policy Point is classified by one of the following concrete types, as shown in Figure 5.5:

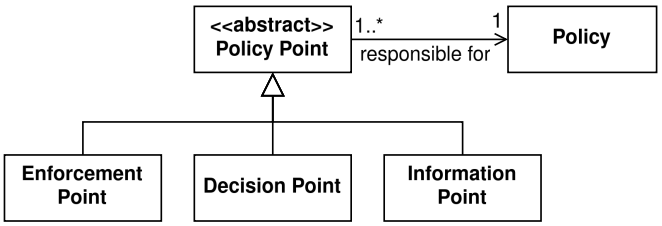


Figure 5.5.: The types of Policy Points and their relationship to Policies.

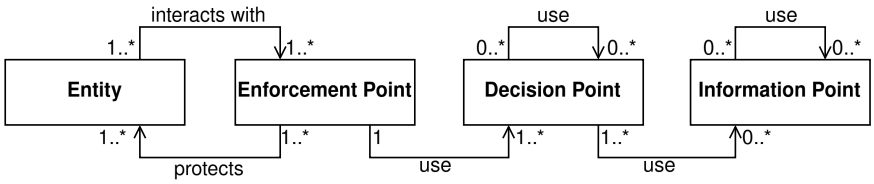


Figure 5.6.: Policy Point hierarchy showing the relationships between Entities, EPs, DPs, and IPs.

- An *Enforcement Point (EP)* is responsible for ensuring that an action, such as registration, is only allowed if all rules and conditions of a Policy are verified.
- A *Decision Point (DP)* is responsible for deciding whether a rule or condition of a Policy is satisfied. Examples include deciding whether a password is strong.
- An *Information Point (IP)* is responsible for providing additional information required to verify a rule or condition of a Policy. Examples include providing the password hashes of users needed to verify credentials.

For example, in a PBA SDP, the "Registrar" Role is the EP for the "Registration" Policy because it is responsible for ensuring that registration is allowed only if all rules and conditions are met. The "Password Policy Verifier" Role is a DP for the "Registration" Policy because it decides whether a provided password is strong. Finally, a "User Manager" Role, which is responsible for managing users, is an IP for a "Login" Policy because the Role provides the password hashes used to verify passwords during login.

By assigning Policies to Roles through Policy Points, as shown in Figure 5.5, it becomes clear which Roles are responsible for handling a Policy and what each Role's responsibilities are regarding a Policy. While we identified the Policy Point concept during the bottom-up analysis, it is also commonly used in the context of authorization [YT05], and Van den Berghe et al. [vYJ22a] also use this concept in their pattern catalog, highlighting its practical application.

The Policy Point hierarchy in Figure 5.6 shows how Policy Points relate to each other. There must always be a single EP that interacts with and protects Entities. Since every Policy must contain at least one rule or condition, otherwise it would always be satisfied,

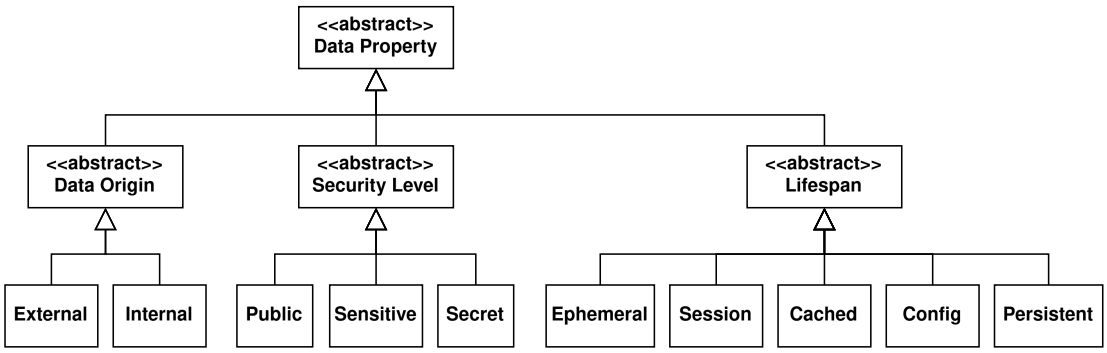


Figure 5.7.: Classification of data based on Data Origin, Security Level, and Lifespan, referred to as Data Properties.

the EP must use at least one DP. To allow for cases where a DP groups multiple decisions made by different DPs, DPs are allowed to use each other. If necessary, a DP can use an IP to request additional information, and IPs may also use each other.

5.4. Data Viewpoint

This section introduces the Data Viewpoint, which outlines how to model the data involved in an SDP and defines the security-relevant information that describes data.

5.4.1. Data Properties

Proper handling of data is critical for maintaining security, as attacks often involve the manipulation or misuse of data. To effectively identify and understand the protection needs of data, ISO/IEC 27002:2022 [22] recommends classifying data with respect to its security needs. While ISO/IEC 27002:2022 [22] suggests data classifications within an organization, we consider general data properties that define characteristics to guide secure implementation within an architecture. The data properties we consider are shown in Figure 5.7 and defined below.

Definition 5.15 (Data Origin)**.** A *Data Origin* classifies data based on the source of the data within a system.

We consider the following Data Origins:

- *External* data originates from Uncontrolled Roles, such as users or external services. Examples include email addresses or passwords provided by users.
- *Internal* data originates from Controlled Roles, such as internal services. Examples include internally stored email addresses and password hashes.

Classifying data by its origin is essential for identifying potential threats and vulnerabilities because attackers typically interact with a system through External data.

However, Internal data must also be handled with care, as insider attackers may have access to internal system components, presenting different security challenges.

Definition 5.16 (Security Level)**.** A *Security Level* classifies data based on the potential impact of its disclosure on the security of the system or its users.

We consider the following Security Levels:

- *Secret* data, if disclosed, would significantly reduce the security of the system or its users. Examples include database passwords, private keys, or tokens.
- *Sensitive* data, if disclosed, can reveal internal operations or user information, potentially aiding attackers. Examples include internal service endpoints or user information such as email addresses.
- *Public* data can be publicly known without compromising the security of the system. Examples include public keys or public API documentation.

Understanding the Security Level of data helps prioritize and implement appropriate measures to protect critical data.

Definition 5.17 (Lifespan)**.** A *Lifespan* classifies data based on the duration for which data is retained and remains accessible within a system.

We consider the following Lifespans:

- *Ephemeral* data is short-lived, often lasting only for the duration of a single request. Examples include *One-Time Passwords* or user credentials submitted during a login attempt.
- *Session* data persists for the duration of an active interaction and is used to maintain state or context. Examples include temporary password reset tokens.
- *Cached* data is temporarily stored to improve system performance and reduce latency by avoiding repeated retrieval or computation. Examples include configuration retrieved from an Identity Provider in OIDC.
- *Configuration* data consists of system settings that change infrequently. Examples include private keys or database passwords.
- *Persistent* data is long-lived and stored for extended periods. Examples include user information and transaction histories.

The Data Properties are primarily for informational purposes and can be useful in software security analysis approaches such as threat modeling. In addition, they can be used to define appropriate countermeasures or security practices for handling data based on their assigned Data Properties [22]. For example, data classified as Internal, Secret, and Configuration should be stored using dedicated secret management systems or encrypted [Bas+22]. Similarly, data classified as Sensitive or Secret and Persistent can be encrypted to mitigate threats if the data storage is compromised.

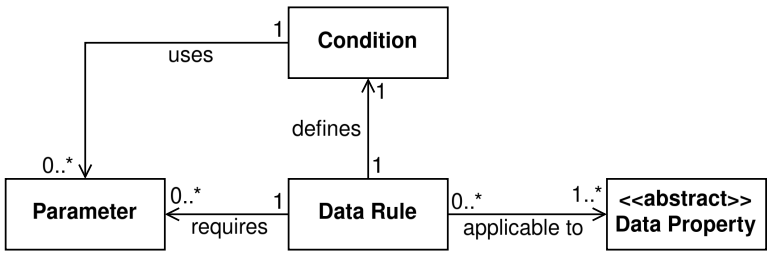


Figure 5.8.: The components and relationships of Data Rules.

5.4.2. Data Rules

To ensure system security, data must adhere to specific rules that address both syntax (data format) and semantics (data meaning) [DJS19]. These rules are typically enforced during input validation to mitigate threats such as SQL injection attacks [DJS19]. This section introduces Data Rules, which allow syntactic and semantic requirements to be integrated into the Data View of an SDP.

Definition 5.18 (Data Rule)**.** A *Data Rule* specifies a syntactic or semantic condition that a value must satisfy. It consists of a unique and descriptive name and the following components, as shown in Figure 5.8:

- *Parameters*: A set of parameters, each with a unique name and a description.
- *Condition*: A Condition defined using a regular expression, a comparison, or a simple description. In the Condition, the data to which the Data Rule applies is denoted by "val", and the Parameters are referred to by their names.
- *Supported Data Properties*: The Data Properties that the data must have for the Data Rule to be applicable.

For example, the "Unique" Data Rule ensures data consistency by requiring that each data entry is uniquely identifiable within a data store. This Data Rule is applicable to all Data Properties except for Ephemeral data, due to its short-lived nature. It has no Parameters, and the Condition states that no two data entries with the same "val" are allowed in the same data store.

By explicitly incorporating data requirements into the Data View, Data Rules address a common issue in existing security patterns, which often define conditions informally or incompletely [vYJ18]. This can cause them to be overlooked and lead to security vulnerabilities.

5.4.3. Data Groups and Data Fields

This section defines the elements used to model data and explains how these elements relate to Data Properties and Data Rules.

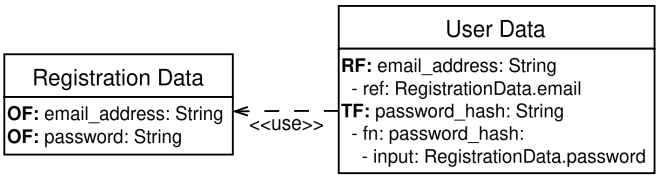


Figure 5.9.: Example Data Groups required during registration, illustrating Origin Fields (OFs), a Reference Field (RF), and a Transformation Field (TF).

Definition 5.19 (Data Group)**.** A *Data Group* defines a general context for related information. It acts as a container for specific data relevant to that context. Each Data Group must have a unique and descriptive name.

For example, the "Registration Data" Data Group shown in Figure 5.9 represents data provided by a user during registration, while the "User Data" Data Group represents the internally stored user information.

Definition 5.20 (Data Field)**.** A *Data Field* represents a specific piece of data within a Data Group. Each Data Field must have a unique and descriptive name within its Data Group and must be classified by a concrete type. These types are the following:

- *Origin Field*: Represents the initial occurrence of data. This means the data is provided by a Role, or defined as a constant value.
- *Reference Field*: Represents data that originates in other Data Groups but is relevant in the context of the current Data Group.
- *Transformation Field*: Represents data derived by applying a Function to one or more Data Fields. The Function must have a unique name and should include a description, pseudocode, or practical examples to illustrate the intended transformation.

For example, in Figure 5.9, the "`email_address`" and "`password`" Origin Fields (labeled as OF) in the "Registration Data" Data Group indicate that the user provides both email address and password as input during the registration process.

The "`email_address`" Reference Field (labeled as RF) in the "User Data" Data Group shows that the email address is stored internally and originates from the "Registration Data" Data Group.

The "`password_hash`" Transformation Field (labeled as TF) indicates that a password hash function must be applied to the password from the "Registration Data" Data Group. The function can provide practical examples and guidance on how to compute a password hash, as shown in Appendix A.5.1.

Data Groups and Data Fields follow the structure of *Unified Modeling Language (UML)* class diagrams, where Data Groups correspond to classes and Data Fields correspond to attributes. To effectively model relationships between Data Fields, Reference

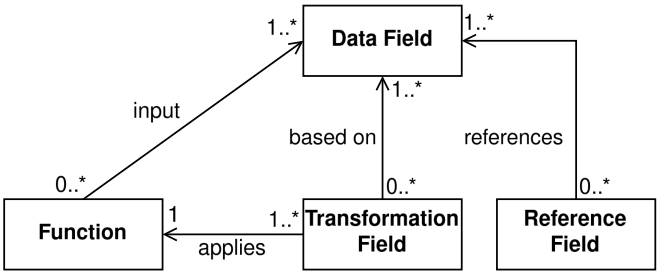


Figure 5.10.: Relationship between the different types of Data Fields.

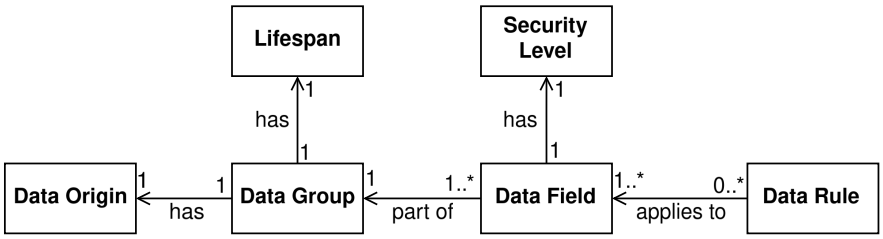


Figure 5.11.: Relationship between Data Groups, Data Fields, Data Properties, and Data Rules.

and Transformation Fields are used to extend modeling capabilities. Both Reference and Transformation Fields allow dependencies between different Data Groups and Data Fields to be clearly shown, as illustrated in Figure 5.10, which is critical for understandability and maintaining security.

For example, in PBA, storing passwords in plaintext is highly insecure but still occurs in practice [21; BLL15]. By introducing a Transformation Field, as shown in Figure 5.9, the relationship between the password and the password hash can be clearly modeled.

Both Data Origin and Lifespan are assigned to Data Groups, as shown in Figure 5.11, and apply to all Data Fields within that Data Group. In contrast, Security Levels and Data Rules are assigned to Data Fields, as they can differ between Data Fields within a Data Group.

The handling of Security Levels and Data Rules differs for Reference and Transformation Fields. In the case of a Reference Field, the Security Level and the Data Rules must be the same as those of the referenced Data Field. In the case of a Transformation Field, both Security Level and Data Rules can change depending on the applied function.

For example, a "`credit_card_number`" Origin Field is classified as Secret. However, a "`masked_credit_card_number`" Transformation Field that takes the "`credit_card_number`" Data Field as input and removes all but the last four digits can be classified as Sensitive.

Data Groups support inheritance, aggregation, and composition. In these relationships, the involved Data Groups must share the same Data Origin and Lifespan to maintain consistency. This restriction aligns with the definition of Data Groups repre-

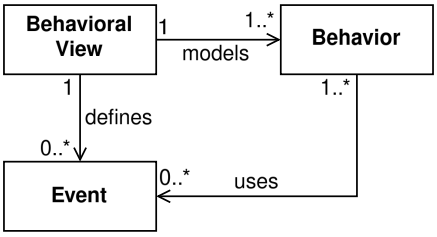


Figure 5.12.: High-level elements and relationships of the Behavioral Viewpoint.

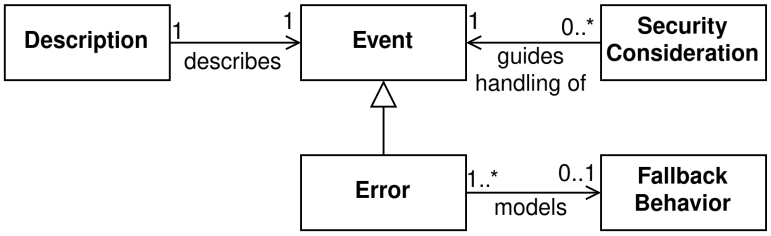


Figure 5.13.: Components and relationships of an Event and Error Event.

senting a particular context, as Data Groups with different Data Origins or Lifespans likely describe different contexts. In addition, this approach imposes no limitations since additional Data Groups can be defined as needed.

5.5. Behavioral Viewpoint

The two main components of the Behavioral Viewpoint, shown in Figure 5.12, are Events and Behavior. Events define specific situations that can occur within the Behavior, while the Behavior details the interactions between Roles. The following section first introduces Events and then explains how to model Behavior.

5.5.1. Events

Definition 5.21 (Event)**.** An *Event* captures an important situation within the modeled Behavior. It is defined by a name and a description that details the situation in which it occurs. An Event contains an optional set of *Security Considerations* that specify potential security issues related to the Event and highlights important aspects relevant to handling the Event.

For example, in a PBA SDP, a “Login Successful” Event occurs when a user has successfully authenticated to the system. The Event has the following Security Consideration: “A login from a new device or location may indicate an unauthorized access attempt and users can be notified”.

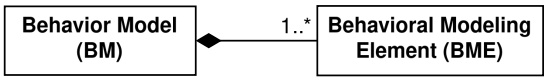


Figure 5.14.: Behavioral Model consisting of multiple Behavioral Modeling Elements.

Definition 5.22 (Error Event)**.** An *Error Event* is a special Event that indicates the occurrence of an unexpected or undesired situation. It contains an optional *Fallback Behavior* that can protect against the situation. A Fallback Behavior must have two possible outcomes: either it successfully mitigates the situation, allowing the Behavior to continue, or it fails, typically resulting in an error message.

For example, a "System Unavailable" Error Event indicates that a system is unresponsive when a Role attempts an action. The "Retry with Exponential Backoff" Fallback Behavior specifies that the action should be retried a number of times, with increasing time in between. Either the action succeeds and the Behavior that caused the Error Event can continue, or it ends.

Events, as shown in Figure 5.13, allow important situations to be defined independently of specific occurrences within the modeled Behavior. This separation enables an Event to be used in multiple locations within an SDP, thereby reducing duplication and synchronization issues. In addition, Events can serve as a checklist for architects, where each Security Consideration must be addressed if relevant in the context of the architect. Finally, Events can guide the implementation of specific log entries within the architects system.

5.5.2. Behavior

The Behavioral View of an SDP models the interactions between the Roles defined in the Conceptual View. We adapt the distinction between a model and its elements as described by Sabau [Sab24], but focus on behavioral aspects as shown in Figure 5.14.

Definition 5.23 (Behavioral Model)**.** A *Behavioral Model (BM)* represents the behavior of a specific aspect of an SDP, detailing how Roles interact to implement Policies.

Definition 5.24 (Behavioral Modeling Element)**.** A *Behavioral Modeling Element (BME)* is a fundamental component in a BM that represents interactions, conditions, or flows that illustrate the dynamic behavior of Roles.

For example, in a PBA SDP, a "Registration" BM models the registration process and includes a "check password strength" BME that models the interaction between a "Registrar" Role and a "Password Policy Verifier" Role.

A BM can be any type of behavioral diagram, such as UML sequence and activity diagrams. This is important because different diagrams are suitable for different scenarios. The BMEs include all possible model elements that represent interactions, conditions, or flows, such as messages in sequence diagrams or activity and decision nodes in activity diagrams.

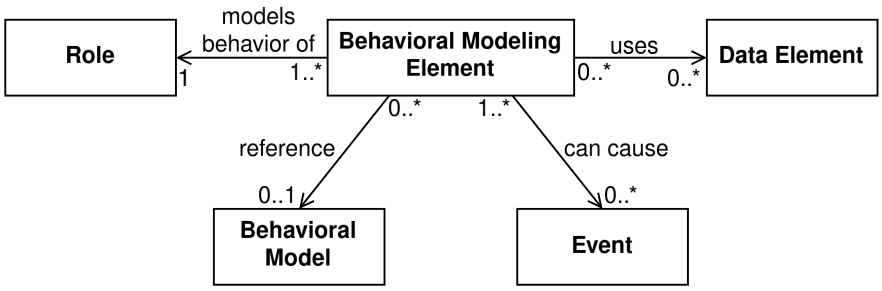


Figure 5.15.: Behavioral Modeling Element models behavior of Roles that can use Data Elements, reference other Behavioral Models, and can cause Events.

In addition to the modeling rules specific to the type of diagram of a BM, we extend BMEs by allowing them to use Events, Data Elements, and references to other BMs, as shown in Figure 5.15.

Event Integration

Explicitly modeling exceptional cases within a BM can reduce readability without adding significant value because exceptional cases often result in an error message and the end of the behavior. Moreover, in the SDPs we developed, BMs frequently contained numerous exceptional cases.

Therefore, a BME can model exceptional cases by referencing Error Events. These Error Events describe the exceptional cases that can occur within a BME, including Security Considerations for handling the Event. In addition, the optional Fallback Behavior allows the BM to continue normally if it is successful. By indicating exceptional cases through references to Error Events, we increase the readability of BMs and ensure that exceptional cases are handled consistently.

For example, in a PBA SDP, a “Verify that password is strong” BME is modeled in both “Registration” and “Password Reset” BMs. In both cases, a “Password Policy Violation” Error Event is referenced, which defines that a provided password does not meet the system’s password requirements and that users should be advised to use stronger passwords. These references show that these BMEs can result in an exceptional case, and the Event itself describes what causes it and what should be done.

However, to not limit what can be modeled, exceptional cases can still be explicitly modeled in a BM if necessary. Note that general Events, such as a “Successful Registration” Event, can also be referenced from a BME to indicate an important situation.

Data Element Integration

Roles often send, receive, or use Data Elements, and since BME’s model the behavior of Roles, this applies to them as well. For example, in a PBA SDP, to login, a “User” Role needs to send an email address and password, and a “Login Manager” Role needs to

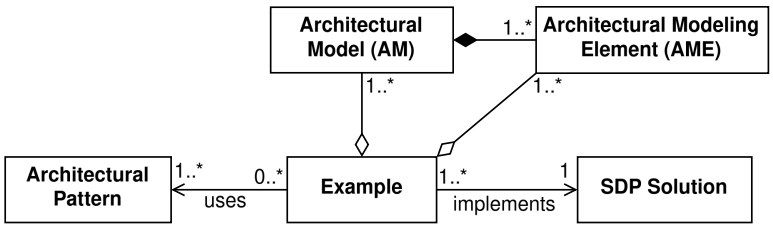


Figure 5.16.: Examples consisting of various Architectural Models and Architectural Modeling Elements that implement an SDP Solution. Each Example uses at least one Architectural Pattern.

retrieve the stored password hash for the email address from the "User Manager" Role.

Since the Data View of an SDP already defines the Data Elements, a BME can reference these Data Elements. These references can be to an entire Data Group or to specific Data Fields within a Data Group. The advantage of this approach is that it clearly shows where Data Elements are used within the BMs, and the Data Properties and Data Rules are included directly in the BMEs.

Behavioral Model Integration

A BME may reference another BM for better understanding and readability. For example, in an OIDC SDP, verifying the token issued by the identity provider requires multiple steps and is needed in multiple BMs. By introducing a separate BM for token verification that is then referenced when needed, we reduce duplication and improve readability.

However, when a BME references a BM, there are considerations. First, the BME can cause any Events that the referenced BM may cause. Second, both BMs should use the same diagram type or compatible types that can be reasonably combined. For example, referencing an activity diagram within a sequence diagram is reasonable, while referencing a state machine is not.

5.6. Structural Viewpoint

The Structural Viewpoint defines how to represent Examples that show the implementation of the SDP Solution within an architecture. These Examples serve as starting points, allowing architects to adopt specific components and discuss the underlying design choices.

Definition 5.25 (Architectural Model)**.** An *Architectural Model (AM)* represents the static or dynamic structure of specific aspects of the SDP Solution.

Definition 5.26 (Architectural Modeling Element)**.** An *Architectural Modeling Element (AME)* is a fundamental component of an AM [Sab24].

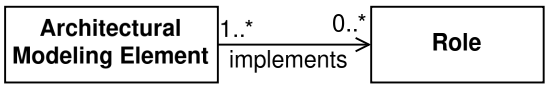


Figure 5.17.: Architectural Modeling Elements can implement the responsibilities defined by Roles.

We adapt the architectural concept model defined by Sabau [Sab24]. In Sabau's concept model, a *design solution* aggregates AMs and AMEs that fulfill an architectural requirement. In our model, a design solution corresponds to an Example that implements an SDP's Solution, as shown in Figure 5.16.

Each Example uses at least one architectural pattern, such as microservice or layered architectural patterns. This enables architects to use Examples that align with their chosen architectural patterns and ensures that the Examples adhere to best practices.

The Structural Viewpoint does not restrict how Examples are represented in terms of AMs and AMEs. An AM can be modeled using UML diagrams, such as component diagrams for representing the static structure and sequence or activity diagrams for representing the dynamic structure. Alternatively, other visualization models can be used, such as the *C4 Model*[1].

5.6.1. Role Realization

While there are no restrictions on how AMs can be modeled, what needs to be modeled is defined by the SDP Solution, i.e., the Role, Data, and Behavioral Views of an SDP. To capture this relationship, AMEs can implement Roles, as shown in Figure 5.17. For example, in a PBA SDP, a "Password Security Component" AME can implement the "Password Policy Verifier" Role, and the "Hasher" Role, which is responsible for computing password hashes.

Implementing Roles is sufficient because the Behavioral View models the interactions of Roles and uses the Data Elements from the Data View. Therefore, if two Roles interact in a Behavioral Model and exchange some Data Elements, this must also be represented in an AM. For example, in a PBA SDP, a "Reset Controller" AME implements the "Password Resetter" Role, responsible for facilitating the password reset. Since the "Password Resetter" Role requires the "Password Policy Verifier" Role to verify that a password is strong, the "Reset Controller" AME and the "Password Security Component" AME must also interact.

The relationship between AMEs and Roles is motivated by Rozanski and Woods [RW11], who specify that an AME should consist of a well-defined set of responsibilities, clearly defined boundaries, and interfaces that describe the functionality an AME provides to other AMEs. Since a Role encapsulates a set of responsibilities, mapping Roles to AMEs specifies the responsibilities that make up AMEs.

[1] `https://c4model.com/`

In addition, the Requires relationship between Roles, which defines that a Role depends on another Role to fulfill its responsibilities, indicates what interfaces an AME must provide. Consider a Role R_1 that is required by a Role R_2. Then the AME that implements Role R_1 must provide at least the functionality required by Role R_2. For example, the "Password Security Component" AME must allow other AMEs to send a password to verify that it is strong.

As specified in Section 5.3, multiple AMEs can implement a Role, or one AME can implement multiple Roles. This is because Roles are architecture independent and simply define the responsibilities and interactions that exist within an SDP. For example, in a PBA SDP, it is not practical to have different Roles for each possible client type, such as mobile or web. However, within an architecture, the distinction between client types is relevant and can be represented by two AMEs that implement the appropriate Role.

6. Application Example of Usage Aspect

This chapter evaluates the SDPDM by presenting practical examples that illustrate the concepts introduced in Chapter 5.

We present the usage aspect of two *OpenID Connect (OIDC)* SDPs and one *Password-based Authentication (PBA)* SDP, focusing on their solutions and examples. The other information, such as context and problem, is mentioned briefly because it is commonly used within patterns and has proven effective [Hey+07]. The first OIDC SDP provides a comprehensive overview of the different components of the SDPDM. The second OIDC SDP illustrates the need to have multiple SDPs for a single SSP, through a comparison with the first pattern. Finally, the PBA SDP demonstrates the applicability of the SDPDM beyond a single security solution. For an introduction to OIDC and PBA, see Section 2.2.1.

To effectively demonstrate the various concepts, only a subset of the usage aspects of the SDPs are presented, while the full versions are available in Appendix A.

6.1. OpenID Connect Security Design Pattern 1

The summary of this SDP is omitted as it closely resembles Definition 2.11. Instead, Figure 6.1 illustrates the high-level flow of the *Authorization Code Flow* used in this SDP, which goes beyond the general OIDC definition. Note that Figure 6.1 does not conform to the SDPDM Viewpoints but is provided for understandability. The following sections introduce the context and problem, followed by the different views.

6.1.1. Context

In this SDP, the Context is defined as follows:

- The system must restrict access to protected resources or actions, making them available only to authenticated users.
- Users interact with the system from clients equipped with browser and input capabilities (e.g., mobile devices) to access protected resources or perform protected actions.
- Communication between users, the system, and the *Identity Provider (IdP)* takes place within a protected environment. A protected environment refers to an environment that is not publicly accessible and that entities must authenticate to access.

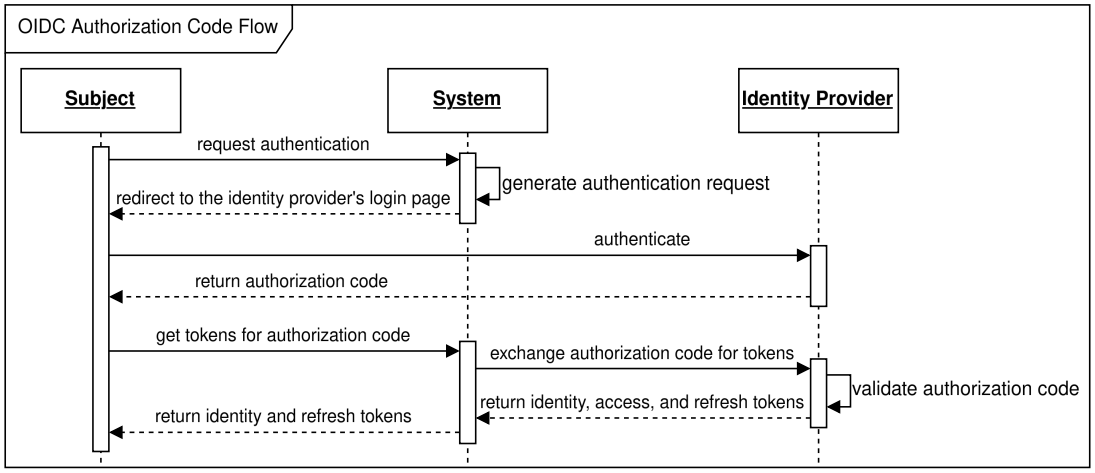


Figure 6.1.: High-level flow of the OIDC Authorization Code Flow used by the SDPs.

This SDP uses the identity token to authenticate subsequent requests, rather than using an internal token such as a session identifier. The restriction to protected environments underscores that using an identity token is less secure because the *Relying Party (RP)* cannot control the contents and lifetime of the token.

6.1.2. Problem

In this SDP, the Problem is defined as follows:

- Malicious attackers may impersonate legitimate users to gain unauthorized access to protected resources or perform actions on their behalf.
- Requiring separate authentication for multiple applications degrades the user experience and increases the likelihood of weak passwords and password reuse.
- Managing separate user accounts for different applications complicates user onboarding and offboarding within organizations.

The last point relates to the expected use of this SDP in organizational environments where centralized user management is required for consistent and efficient handling of user accounts.

6.1.3. Conceptual View

The following introduces the two Policies considered in the SDP and the Roles that ensure their secure application.

Policy (Identity Token Verification)**.** Access to protected resources is granted only to users who present a valid identity token issued and signed by the trusted IdP. The RP

must verify that the identity token received from the users is in the expected format, has a valid signature, and contains valid claims, such as that the token has not expired.

This Policy is required because the identity token authenticates subsequent requests.

Policy (OIDC Authentication)**.** During the authentication process, only the user who initiated the authentication at the RP and successfully authenticated to the expected IdP is allowed to receive and use an identity token. The RP must verify that the identity token received from the IdP is in the expected format, has a valid signature, and contains valid claims, such as that the token has not expired. In addition, *State*, *Nonce*, and *Code Verifier* values are sent in the authentication request and must be validated to protect against attacks.

- The *State* is echoed by the IdP to ensure that the response matches the original authentication request, mitigating *Cross-Site Request Forgery (CSRF)* attacks.
- The *Nonce* is included in the identity token to prevent replay attacks by ensuring the token is unique to the authentication request.
- The *Code Verifier* is sent when exchanging the authorization code for tokens, preventing attackers from using stolen authorization codes.

In the following, the "Identity Token Verification" Policy is referred to as "Token", and the "OIDC Authentication" Policy as "Login".

The following introduces some of the Roles visualized in Figure 6.2 and their Role Extensions. This includes the Policy Points that define the responsibilities a Role has in the context of a Policy. The complete Conceptual View is available in Appendix A.3.4.

The "Subject" Role is responsible for initiating authentication at the RP, authenticating to the IdP, and forwarding the authorization code to the RP upon successful authentication with the IdP. It has the following Role Extensions associated with it:

- *Uncontrolled*: The Role may act maliciously, for example, by attempting to access protected resources using an invalid or modified identity token.
- *Secret Storage Tag*: The Roles stores a session identifier associated with the internally stored state, code verifier, and nonce. In addition, after successful authentication, the identity token is stored because it is used for subsequent requests.

The "Access Manager" Role is responsible for forwarding authentication requests to the "Authentication Manager" Role and ensuring that identity tokens are valid when users access protected resources. It requires the "Token Validator" Role to validate identity tokens. It has the following Role Extensions associated with it:

- *Abstract*: The Role references the Single Access Point SSP, which defines how to protect resources based on authentication information such as an identity token.
- *Enforcement Point (EP)* of the "Token" Policy: The Role enforces that only authenticated users can access protected resources.

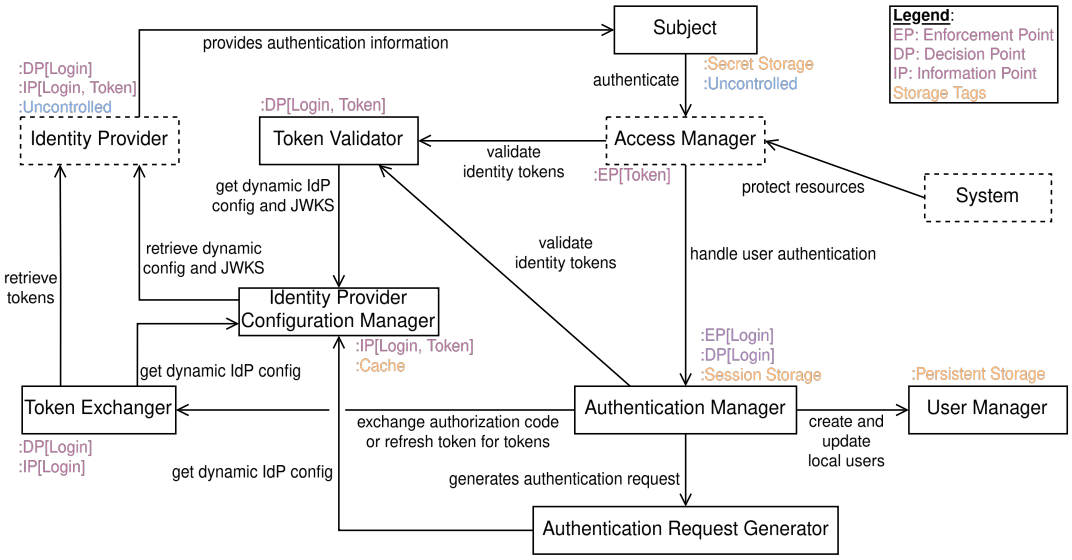


Figure 6.2.: Conceptual View of the OIDC SDP. Each Role is represented by a box, with Abstract Roles shown using dotted lines. Roles not explicitly marked as Uncontrolled are considered Controlled. The "Identity Token Verification" Policy is referred to as "Token", and the "OIDC Authentication" Policy as "Login".

The "Authentication Manager" Role is responsible for facilitating the authentication process and validating the state and nonce values. It requires the "Authentication Request Generator" to generate the authentication request that forwards a user to the IdP and includes the state, code verifier, and nonce. It has the following Role Extensions associated with it:

- *Session Storage Tag*: The Role stores the state, code verifier, and nonce in a session object, as they are needed for validation after successful authentication at the IdP.
- EP of the "Login" Policy: The Role only returns an identity token if authentication is successful.
- *Decision Point (DP)* of the "Login" Policy: The Role validates that the stored state and nonce values match their counterparts during the token exchange.

The "Identity Provider Configuration Manager" Role is responsible for retrieving the configuration required for all interactions with the IdP and the *JSON Web Key Set (JWKS)*, which contains the public keys needed to verify the identity tokens. This Role is critical in reducing configuration issues, as the system operator only needs to provide minimal configuration; the rest is dynamically retrieved from the IdP. In addition, dynamic retrieval of public key information enhances security, as the IdP can frequently rotate its private key without requiring changes to the RP. It has the following Role Extensions associated with it:

- *Cache Storage Tag*: Both the IdP configuration and the JWKS need to be cached to prevent performance degradation caused by requesting them for every operation.
- *Information Point (IP)* for the "Token" and "Login" Policy: The Role provides the configuration required for each request to the IdP and the JWKS required to validate identity tokens.

The "Identity Provider" Role is responsible for core OIDC functionalities, such as exchanging authorization codes for identity tokens. In addition, it provides its configuration and JWKS. It has the following Role Extensions associated with it:

- *Abstract*: The design of the IdP is irrelevant as long as it provides the required functionality.
- *Uncontrolled*: Responses must be validated to ensure authenticity and integrity. However, the IdP must be trusted because it can issue valid tokens for any user.
- DP of the "Login" Policy: The Role verifies that the code verifier sent during the token exchange matches the code challenge sent in the initial authentication request, thereby protecting against code interception attacks.
- IP of the "Login" Policy: The Role provides the configuration and JWKS, and issues identity tokens that confirm successful authentication.

6.1.4. Data View

This section presents some Data Elements with their Data Properties, Data Rules, and relationships; the complete Data View is provided in Appendix A.3.5.

The Data Elements shown in Figure 6.3 model the data required to request authentication from the IdP. The "OIDC Session Data" Data Group is assigned the Internal Data Origin because it is provided by the Controlled "Authentication Request Generator" Role. In addition, the state, nonce, and code verifier required for validation are stored within a session object, so the Data Group is assigned the Session Lifespan.

The "Cryptographically Secure Pseudorandom Number Generator" (CSPRNG) Data Rule specifies that generated values must be unguessable by using a CSPRNG to generate them. The Data Rule requires an "Entropy" Parameter, where entropy is a measure of the randomness or unpredictability of an output [VH14]. For example, an entropy of 128 bits means that there are 2^{128} possible unique outcomes, making it practically impossible to guess. Practical examples can simplify the implementation, such as the following Python example: `csprn = secrets.token_urlsafe(nbytes=16)`

The "OIDC Authentication Data" Data Group models the data required by the subject to authenticate at the IdP. The Data Group is classified as Ephemeral because it is only used to initiate the authentication, and as Internal because it is generated by the Controlled "Authentication Request Generator" Role.

The "`authorization_endpoint`" Reference Field defines the authentication endpoint of the IdP, and the "`redirect_uri`" Reference Field defines where the subject is redirected after successfully authenticating to the IdP.

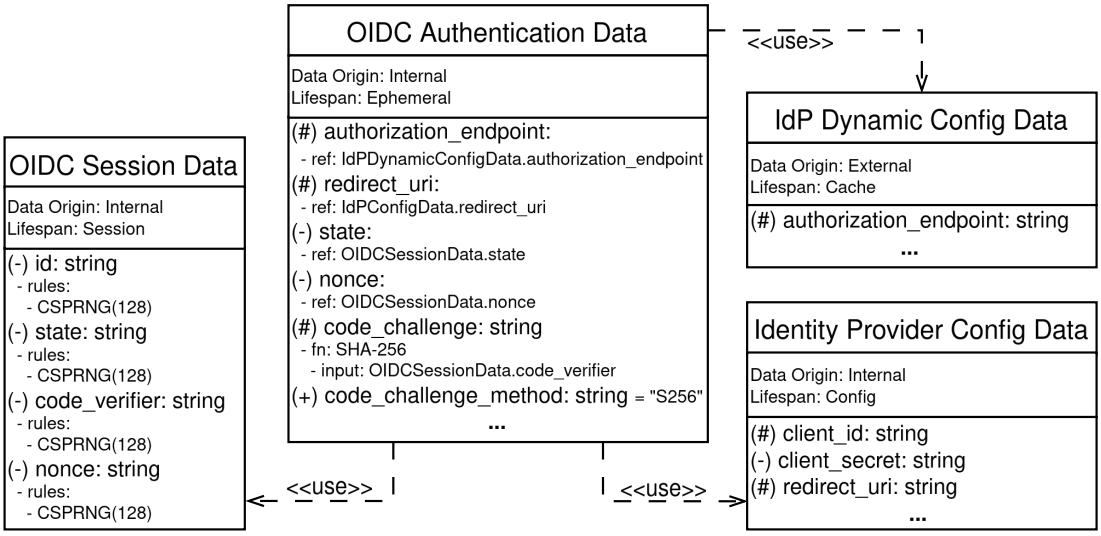


Figure 6.3.: Subset of the OIDC Data View that illustrates the data required to request authentication from the IdP. Security Levels are encoded as follows: Secret (-), Sensitive (#), and Public (+).

Both Data Fields are Reference Fields because they are required in the context of the "OIDC Authentication Data" Data Group, but are defined in a different context. The "`authorization_endpoint`" Reference Field is dynamically retrieved from the IdP, as indicated by its reference to the "Identity Provider Dynamic Config Data" Data Group with its External Data Origin. The "`redirect_uri`" Reference Field belongs to the internal configuration, as indicated by its reference to the "Identity Provider Config Data" Data Group with its Internal Data Origin and Config Lifespan.

In OIDC, a code challenge based on the code verifier can be sent to the IdP during the initial authentication request. Then, when an authorization code is exchanged for tokens, the code verifier is sent along. The IdP verifies that the code verifier matches the code challenge sent during authentication. This ensures that if an attacker gains access to the authorization code, they cannot exchange it for tokens without the code verifier. The code challenge can be the same as the code verifier which is less secure because an attacker who intercepts it can use it directly. In contrast, using a hash computed from the code verifier prevents this, since it is not feasible to determine the code verifier from the hash in reasonable time [SBA15].

This information is critical for security and is provided in the Data View by the "`code_challenge`" Transformation Field, which uses the "`SHA-256`" Function with the "`code_verifier`" Origin Field as input. The "`SHA-256`" Function is defined as follows: `BASE64URL-ENCODE(SHA256(code_verifier))` [SBA15].

The "`code_challenge`" Transformation Field demonstrates that the Security Level can change based on the applied Function. The "`code_verifier`" Origin Field is classified as Secret because an attacker can use it directly, along with the authorization code,

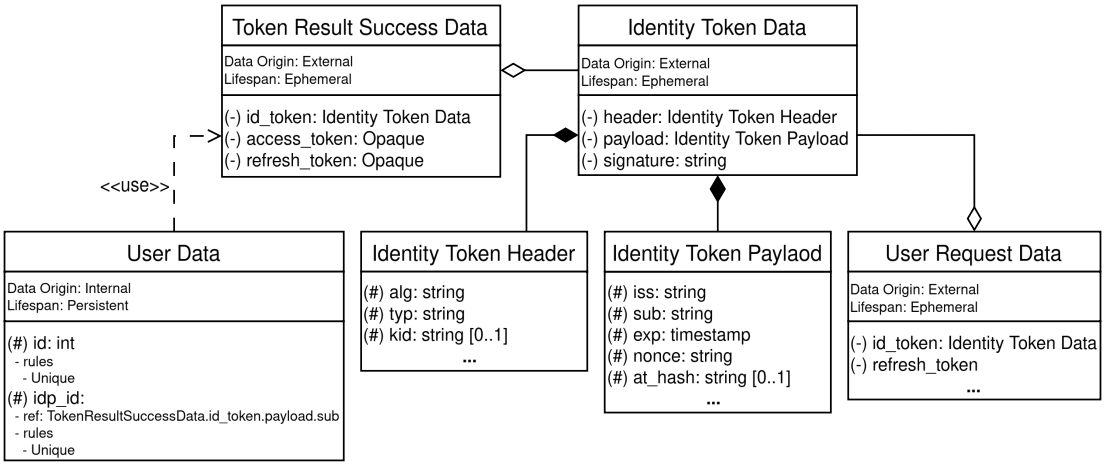


Figure 6.4.: Subset of the OIDC Data View that illustrates the response after a successful token exchange at the IdP. Security Levels are encoded as follows: Secret (`-`), Sensitive (`#`), and Public (`+`).

to obtain tokens. In contrast, the "`code_challenge`" Transformation Field is classified as Sensitive because it is not feasible to determine the corresponding code verifier in reasonable time.

The Data Elements shown in Figure 6.4 model the data that is relevant after a successful exchange of an authorization code for tokens at the IdP.

The "Token Result Success Data" Data Group models the data returned by the IdP, so it is assigned the External Data Origin. In addition to the "`identity_token`" Origin Field, it includes a "`refresh_token`" Origin Field, which allows requesting a new identity token when the current one has expired, and an "`access_token`" Origin Field, which may allow requesting additional user information at the IdP. The "`Opaque`" data type indicates that the structure of both tokens is unknown, as it depends on the IdP.

The "User Data" Data Group models the data of the local user that is stored persistently, as indicated by the Persistent Lifespan. It consists of an "`id`" Origin Field, the local user identifier, and the "`idp_id`" Reference Field, the user identifier within the IdP. This Reference Field clearly shows that the local user is associated with the IdP via the "`sub`" Origin Field of the identity token provided by the IdP. The "Unique" Data Rule, applied to both identifiers, specifies that no two data entries with the same value are allowed within a data store.

The "Identity Token Data" Data Group models the structure of the identity token and is associated with both the "Token Result Success Data" and "User Request Data" Data Groups. This aggregation is necessary because the identity token is provided by the IdP after a successful token exchange and sent by the user to request protected resources.

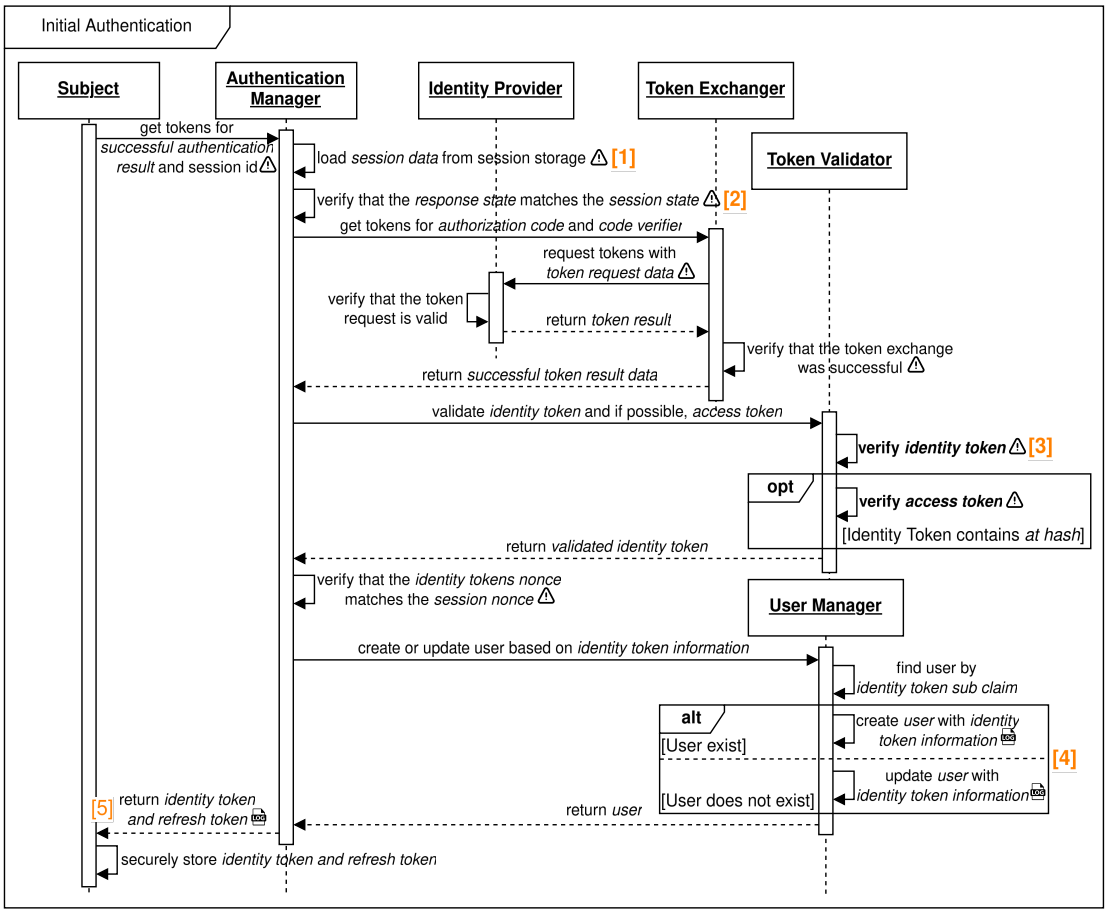


Figure 6.5.: Subset of "Initial Authentication" Behavioral Model starting after the subject authenticates to the IdP. Log Symbols indicate Events, while Exclamation Points represent Error Events. Indexes are provided for reference in the text.

6.1.5. Behavioral View

This section presents a part of the "Initial Authentication" *Behavioral Model (BM)*, shown in Figure 6.5, which begins after the "Subject" Role has successfully authenticated to the IdP and must now exchange the authorization code for tokens. The complete Behavioral View is provided in Appendix A.3.6.

An exclamation point next to a label indicates an Error Event, which means that an exceptional case can occur in that step. Each of these exceptional cases could be modeled with a UML break fragment that represents the behavior to be executed instead of the normal flow [OMG23].

For example, in step [1] of Figure 6.5, an exceptional case can occur if no session exists for a given session identifier. This can be modeled explicitly as shown in Fig-

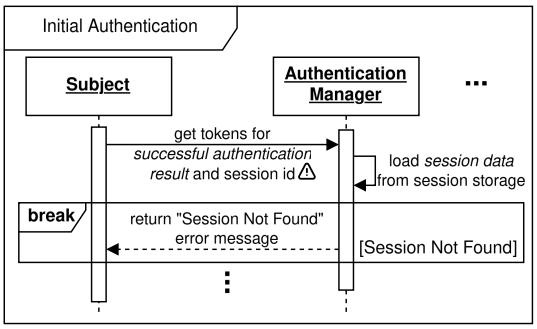


Figure 6.6.: Example of modeling an exceptional case explicitly instead of using an Error Event.

ure 6.6. However, most exceptional cases, including this one, simply propagate back to the "Subject" Role, offering limited value in explicit modeling.

Error Event (Response State Mismatch)**.** The state parameter in the authentication response does not match the stored state. Security Consideration: A mismatched state parameter can indicate a CSRF attack, where an attacker attempts to trick a user into exchanging the attacker's authorization code for tokens.

Step [2] of Figure 6.5 ensures that the state value received from the "Subject" Role matches the state value stored internally. The "Response State Mismatch" Error Event indicates that this step can result in an exceptional case and describes the scenarios in which it can occur.

In addition, step [2] references two Data Fields that show which values are being compared. The "response state" refers to the "`state`" Origin Field of the "Success Response Data" Data Group, which models the response from the IdP after a successful authentication. The "session state" refers to the "`state`" Origin Field of the "OIDC Session Data" Data Group, shown in Figure 6.3, which is the internally stored state.

Error Event (Invalid Token Signature)**.** The signature of the identity token does not match the computed signature. Security Consideration: An invalid token signature can indicate that an attacker has tampered with the identity token. Fallback Behavior: If the token header does not include a public key identifier, and the JWKS contains only one public key, refresh the JWKS cache and retry the operation once.

Step [3] of Figure 6.5 references a separate BM defined in Appendix A.3.6. Modeling identity token validation in a separate BM is advantageous because it involves several steps that would distract from the focus of initial authentication. In addition, identity token validation is required to authenticate subsequent requests, so a separate BM must be provided anyway. Finally, identity token validation is also required in another BM, so using a separate BM reduces duplication.

The exclamation point for step [3] indicates that the referenced BM can cause exceptional cases, such as an "Invalid Token Signature" Error Event. The Fallback Behavior

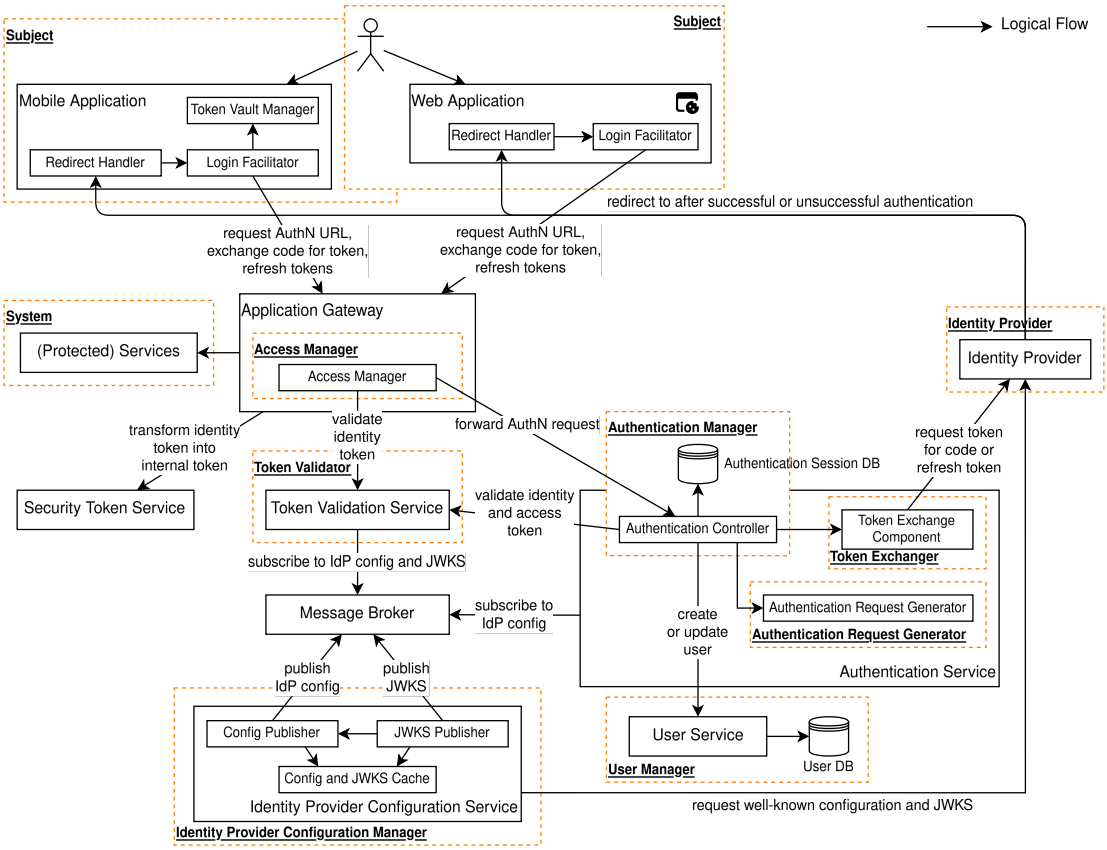


Figure 6.7.: Static structure of a microservice architecture implementing the OIDC Roles. Roles are represented by dashed boxes. Each Architectural Modeling Element within a dotted box implements a part of a Role.

for this Event allows token validation to succeed even if the initial signature verification fails. This can happen if the IdP has rotated its private key, but the JWKS cache still contains the old public key. In this case, refreshing the JWKS cache by requesting the latest public key from the IdP may resolve the problem.

The log file icons indicate Events that highlight important non-exceptional cases within the BM. These Events specify which log entries may be used in an architect's system and provide possible security issues related to the Events.

For example, in step [4] of Figure 6.5, the "User Created" and "User Information Updated" Events are referenced. These Events can be used to detect deviations from typical behavior in a system, such as unusually frequent registrations. Another example is step [5] of Figure 6.5, which refers to a "Successful User Authentication" Event, where Security Considerations suggest notifying users of authentication by an unknown device.

6.1.6. Structural View

This section presents an *Architectural Model (AM)*, illustrated in Figure 6.7, that models the static structure of an Example using the *Microservice* and *Publish-Subscribe* architectural patterns. The Microservice architectural pattern specifies the division of a system into multiple independently deployable services, each representing a domain or subdomain [RF20]. The Publish-Subscribe architectural pattern specifies that services can asynchronously publish messages to a channel, while other services receive those messages by subscribing to the channel [Ric18].

Roles within the AM are visualized by dashed boxes, and each *Architectural Modeling Elements (AMEs)* within a dashed box implements a part of the Role.

The "IdP Configuration Service" demonstrates that a Role can be implemented by multiple AMEs. The "IdP Config Publisher" AME retrieves the configuration from the IdP and publishes it. The "JWKS Publisher" AME retrieves the JWKS from the IdP and publishes it. The "Config and JWKS Cache" AME stores the configuration and JWKS locally and ensures that they are only used from the cache if they are valid. Together, these AMEs implement all of the responsibilities of the "IdP Configuration Manager" Role.

A message broker is used because both the "Token Validation Service" and the "Authentication Service" require the IdP configuration, eliminating the need for each to individually request and cache the configuration. The "IdP Configuration Service" provides functionality for the other services to trigger the republication of the configuration and JWKS, or to request that they be fetched again from the IdP. This is required if the identity token validation in the "Token Validation Service" fails due to an invalid signature, which may be caused by an outdated public key.

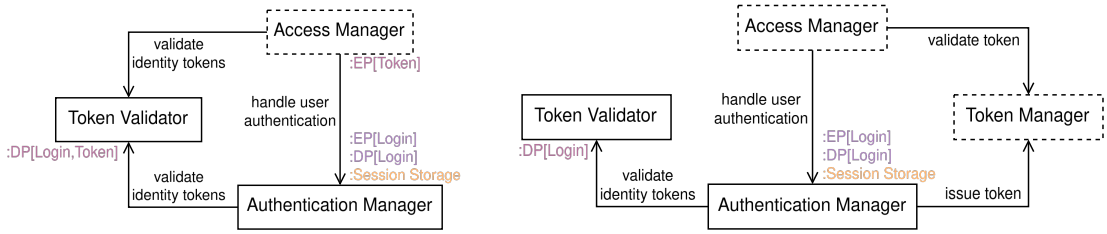
The "Authentication Controller" AME implements the "Authentication Manager" Role, so it is responsible for facilitating the authentication process, including storing the authentication session in a dedicated data store. This illustrates that the Session Storage Tag assigned to the Role translates to a data store in the AM.

6.1.7. Consequences

This SDP has the following Consequences:

- Strength (Maintainability): There is no need for dedicated token management, including issuance, storage, and invalidation. In addition, using a single IdP reduces the complexity of configuration and user management.
- Weakness (Security, Flexibility): The identity tokens are exposed directly to the client, allowing an attacker to gain access to the token. This is problematic because an identity token can contain sensitive information. In addition, the system has no control over the lifetime of an identity token, so a compromised token may be valid for an extended period of time.

Similar to the context introduced in Section 6.1.1, which restricts the use of the SDP to



(a) Subset of Conceptual View of SDP 1. (b) Subset of Conceptual View of SDP 2.

Figure 6.8.: Comparison between the Conceptual View of OIDC SDP 1 and SDP 2.

protected environments, the Consequences highlight the drawbacks of using the identity tokens instead of an internal token.

6.2. OpenID Connect Security Design Pattern 2

This section introduces the second OIDC SDP, focusing on the differences from the previously discussed SDP. To do this, each section introduces a change from the previous SDP and then explains the impact of that change on each part of the SDP, including the Context, Problem, and Views. This demonstrates the need for and benefits of considering multiple SDPs for a single SSP.

Internal Token Usage

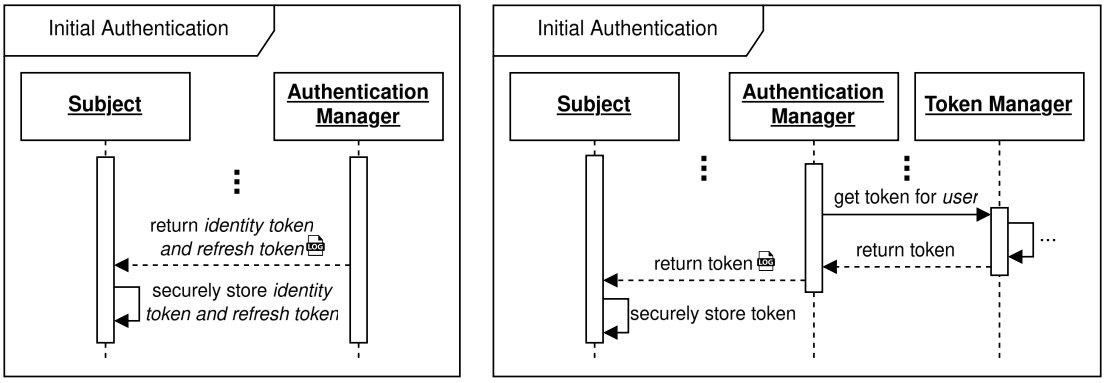
This SDP uses an internal token instead of using the identity token directly to authenticate subsequent requests, resulting in the following changes.

The Context no longer includes the restriction that communication between users, the system, and the IdP must occur within a protected environment. This is because the system has full control over the contents of the internal tokens, allowing sensitive data to be reduced to the minimum required, such as a user identifier. In addition, the system can use shorter token lifetimes, thereby mitigating the impact of a compromised token. The rest of the Context and Problem remains the same.

The “Identity Token Verification” Policy is excluded because token management, including token validation for subsequent requests, is handled by a separate SDP. The “OIDC Authentication” Policy is extended to specify that an internal token is issued upon successful authentication.

An additional Abstract “Token Manager” Role is introduced that references the Token-based Authentication ASSP introduced in Section 2.3.3, as illustrated in Figure 6.8b. The “Token Manager” Role is responsible for issuing tokens, which can be stateful (e.g., a session identifier) or stateless (e.g., a JWT).

The “Authentication Manager” Role requires the new “Token Manager” Role to issue a token upon successful authentication. In addition, the “Access Manager” Role now



(a) Final steps of "Initial Authentication" Behavioral Model of SDP 1.

(b) Final steps of "Initial Authentication" Behavioral Model of SDP 2.

Figure 6.9.: Comparison of final steps in the "Initial Authentication" Behavioral Model of OIDC SDP 1 and SDP 2.

requires the "Token Manager" Role to handle token validation, replacing the previously required "Token Validator" Role.

In the Data View, the "User Request Data" Data Group, shown in Figure 6.4, and the "Token Refresh Data" Data Group are removed. This is because both Data Groups are used in the previous SDP to request a new identity token when it expires, but since this SDP only uses the identity token for initial authentication, they are no longer needed.

In the Behavioral View, the "Refresh Identity Token" BM is excluded because it was only needed to obtain a new identity token when it expired. In addition, the "Token Manager" Role is included in the "Initial Authentication" BM. It is needed by the "Authentication Manager" Role to request a token, which is then returned to the "Subject" Role and stored securely, as shown in Figure 6.9b.

In the Structural View, shown in Figure A.10, the separate "Token Validation Service" used in Figure 6.7 is integrated into the "Authentication Service" because it is no longer needed by the "Access Manager". In addition, a "Token Management Service" that implements the new Abstract "Token Manager" Role is included. It is used by the "Authentication Controller" to request tokens for users and by the "Access Manager" to validate tokens.

Private Key JWT

This SDP uses the *Private Key JWT* authentication method instead of the *Client Secret Basic* method. In the Client Secret Basic method, a shared secret is used to authenticate the RP to the IdP and is sent with the token requests. In the Private Key JWT method, a JWT signed by the RP using a private key is sent with the token request, allowing the IdP to validate authenticity using the RP's public key. This approach enhances security by eliminating the need for a shared secret and ensuring the authenticity and integrity of the token request [Fet21]. This results in the following changes.

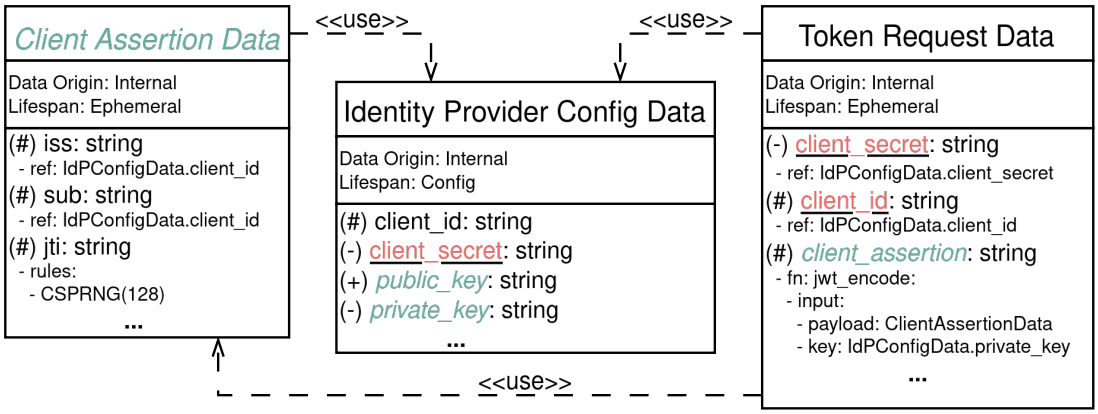


Figure 6.10.: Part of the OIDC Data View that models the token exchange data using a signed JWT. Newly introduced Data Groups and Data Fields are highlighted in green and italics. Data Fields excluded from the previous SDP are highlighted in red and underlined. Security Levels are encoded as follows: Secret (−), Sensitive (#), and Public (+).

The Context, Problem, and Policies remain the same, but the "Token Exchanger" Role, responsible for exchanging an authorization code for tokens, is now also responsible for generating the signed JWT required for the token exchange.

In the Data View, within the "Identity Provider Config Data" Data Group, the shared secret represented by the "`client_secret`" Origin Field is replaced by the "`private_key`" and "`public_key`" Origin Fields, as illustrated in Figure 6.10.

Furthermore, the "Token Request Data" Data Group previously contained a "`client_secret`" Reference Field. In this SDP, this Reference Field has been replaced by the "`client_assertion`" Transformation Field, as shown in Figure 6.10. The input for the "`jwt_encode`" Function is the "Client Assertion Data" Data Group that models the JWT payload, and the "`private_key`" Data Field needed to compute the signature. The Function uses the input to create a signed JWT; Appendix A.4.5 provides a practical example.

In the Behavioral View, the creation of the signed JWT is included as a separate step of the "Token Exchange" Role in the "Initial Authentication" BM shown in Figure 6.5.

In the Structural View, the static structure is not affected by the private key JWT authentication method. However, the dynamic structure that models the token exchange needs to incorporate the generation of the signed JWT.

6.3. Password-based Authentication SDP

This PBA SDP demonstrates that the SDPDM introduced in Chapter 5 can be applied beyond OIDC authentication. It is based on the information from the PBA pattern by

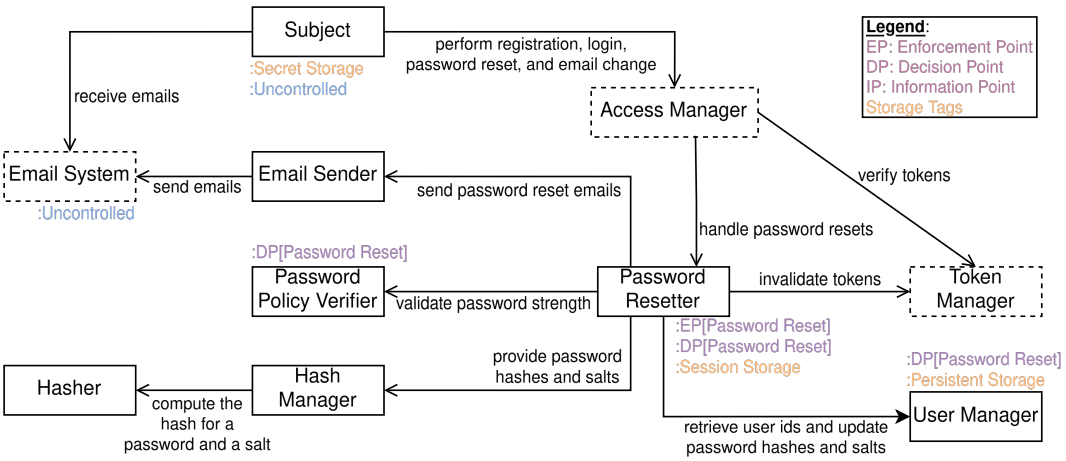


Figure 6.11.: Subset of the Conceptual View that shows the Roles responsible for securely applying the "Password Reset" Policy.

Van den Berghe et al. [vYJ22c], but we have adapted and extended it to fit our concepts and fully cover PBA.

In this section, we limit ourselves to the "Password Reset" Policy because it is sufficient to demonstrate the concepts. We also omit the Summary, Context, Problem, and Consequences, as they are common components in patterns and have been demonstrated in previous examples.

6.3.1. Conceptual View

Policy (Password Reset)**.** The user initiates the password reset by providing the email address associated with the registered account. The user must prove access to the email account by clicking a reset link containing a reset token sent to the email address. If the reset token is valid, the user must enter a strong password. Upon successful password reset, all user tokens must be invalidated.

The specific password requirements are defined separately because both "Registration" and "Password Reset" Policies require them (see Appendix A.5).

The Abstract "Token Manager" Role, shown in Figure 6.11, is responsible for validating, issuing, and invalidating tokens. It references the Token-based Authentication ASSP introduced in Section 2.3.3.

The "Password Resetter" Role is responsible for facilitating the password reset, including generating and verifying the reset tokens. It requires the "Token Manager" Role to invalidate all user tokens after a password change to ensure that no one who gained access using the old password can still access the system. It has the following Role Extensions associated with it:

- EP of the "Password Reset" Policy: The Role only changes the password if all rules and conditions of the Policy are verified.

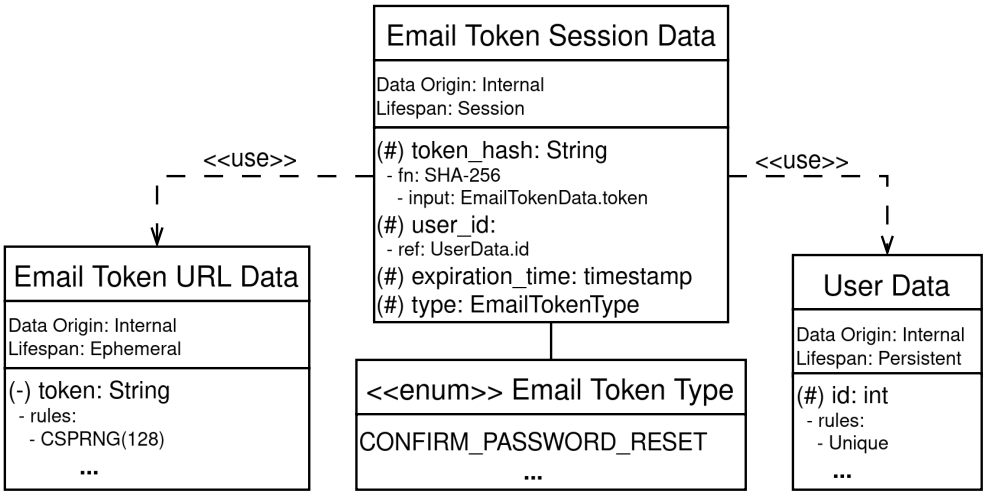


Figure 6.12.: Subset of the PBA Data View that illustrates the data responsible for the email confirmation. Security Levels are encoded as follows: Secret (-), Sensitive (#), and Public (+).

- DP of the "Password Reset" Policy: The Role verifies that the provided email address is in the expected format and validates the reset tokens.

The "User Manager" Role is responsible for managing user accounts including their creation, update, and information retrieval. It has the following Role Extensions associated with it:

- *Persistent Storage Tag*: The Role persistently stores the user information.
- DP of the "Password Reset" Policy: The Role verifies that a user exists for a given email address.

Data View

The "Email Token Session Data" Data Group, shown in Figure 6.12, models the data stored internally to validate that a user has confirmed an action, such as a password reset, by clicking on a token link sent to their email address. In contrast, the "Email Token URL Data" Data Group models the data sent to the user's email address.

The "`token_hash`" Transformation Field in the "Email Token Session Data" Data Group is critical to security because, even if an attacker gains access to the internal data, they cannot misuse it. This is because the "`token`" Origin Field sent to the user's email address is required to confirm an action, but the "`token_hash`" Transformation Field stores only a hash of that token.

The "`type`" Origin Field specifies the action that a token confirms. This simplifies the Data View because the data required to confirm actions is the same in each case, and

only the behavior varies depending on the specific action. For example, in the case of a registration confirmation, the user's account is set to verified; in the case of a password reset confirmation, the user's password is changed.

The "`user_id`" Reference Field shows how the local user, modeled by the "User Data" Data Group, is associated with the email token sessions.

Behavioral View

This section focuses on introducing some of the Events used in BMs, omitting detailed descriptions of the specific BMs. This is because the OIDC SDPs have already demonstrated how Events and Data Elements are used within BMs. Furthermore, the BMs in this PBA SDP do not involve any special cases beyond those previously covered. The complete Behavioral View is available in Appendix A.5.2.

Error Event (User Not Found)**.** The user account associated with the email address does not exist. This can happen when a user tries to log in or reset a password. Security Considerations:

- Detailed error messages can help attackers enumerate users; use generic messages to avoid this (CWE-204: Observable Response Discrepancy[1]). For example, respond with "Login failed; Invalid email address or password" or "If that email address is in our database, we will send you an email to reset your password".
- Timing differences between the successful and unsuccessful cases can help attackers enumerate users; avoid fail-fast behaviors, i.e., ensure that both cases take roughly the same amount of time (CWE-208: Observable Timing Discrepancy[2]).

The Security Considerations in the "User Not Found" Error Event are difficult to integrate explicitly into the BMs because they often need to be addressed at the implementation level. For example, in Python, `a == b` is vulnerable to timing attacks because it terminates early when a difference is found [Hal09]. In contrast, using `hmac.compare_digest(a, b)` provides a constant-time comparison to protect against this.

This highlights the benefits of using Events, where aspects that are difficult to represent explicitly and are important to security can be defined in the Security Considerations.

Event (Login Initiated)**.** A user initiates a login attempt by entering their email address and password. Security Consideration: A high number of login attempts, especially failed attempts, can indicate brute force or credential stuffing attacks.

The "Login Initiated" Event includes valuable information when transformed into a log entry within the architect's system. By capturing the initial login attempt, the successful login, and the exceptional cases that can occur in between, unusual patterns can be identified and addressed accordingly. For example, if ten percent of all login attempts are typically unsuccessful and that number is increasing rapidly, it likely indicates an active attack.

[1] `https://cwe.mitre.org/data/definitions/204.html`
[2] `https://cwe.mitre.org/data/definitions/208.html`

Structural View

This section presents an AM of an Example using the Microservice architectural pattern, visualized in a C4 component diagram (see Figure 6.13).

In the C4 model, a component groups related functionality that is accessible through a well-defined interface. Components cannot be deployed separately; they always belong to exactly one container, in this case a service. A container represents an application or data store that must be running within the system for it to function properly [RF20].

The "Mobile Application" AME consists of the "Password Service" AME that handles interactions with the internal system and the "Token Vault Manager" AME responsible for the secure storage of received tokens. In the case of the "Web Application" AME, tokens are provided as secure cookies, which is the most secure option as specified in Appendix A.6. Web browsers automatically store received cookies and append them to requests, eliminating the need for an additional component to manage token storage.

The "Access Manager" AME must distinguish between the mobile application and the web application when returning and receiving tokens. Figure 6.14 shows part of the dynamic structure during user login to illustrate where the distinction needs to be made. This highlights the need to provide AMs for both the static and dynamic structure.

Similar to the OIDC SDP AM introduced in Section 6.1.6, the AM includes a "Security Token Service" AME. This AME is responsible for converting external tokens into internal tokens, which is critical in a distributed architecture to maintain a consistent token format across the system. In addition, separating internal tokens from external ones ensures that even if an internal token is compromised, it cannot be directly exploited to gain access to the system.

The AME also implements the "Token Manager" Role, making it responsible for issuing, validating, and invalidating external tokens. Centralizing token management in a single service enhances system maintainability and performance by reducing complexity and minimizing communication overhead.

In Figure A.17, we provide an additional AM that represents the static structure for this Example, but uses a C4 container diagram for visualization. This AM offers a high-level view by showing how different containers interact. Unlike the AM in Figure 6.13, most of the Roles are assigned to a single AME, the "User Service". This shows that the assignment of Roles to AMEs can vary depending on the type and purpose of a diagram, even when modeling the same Example.

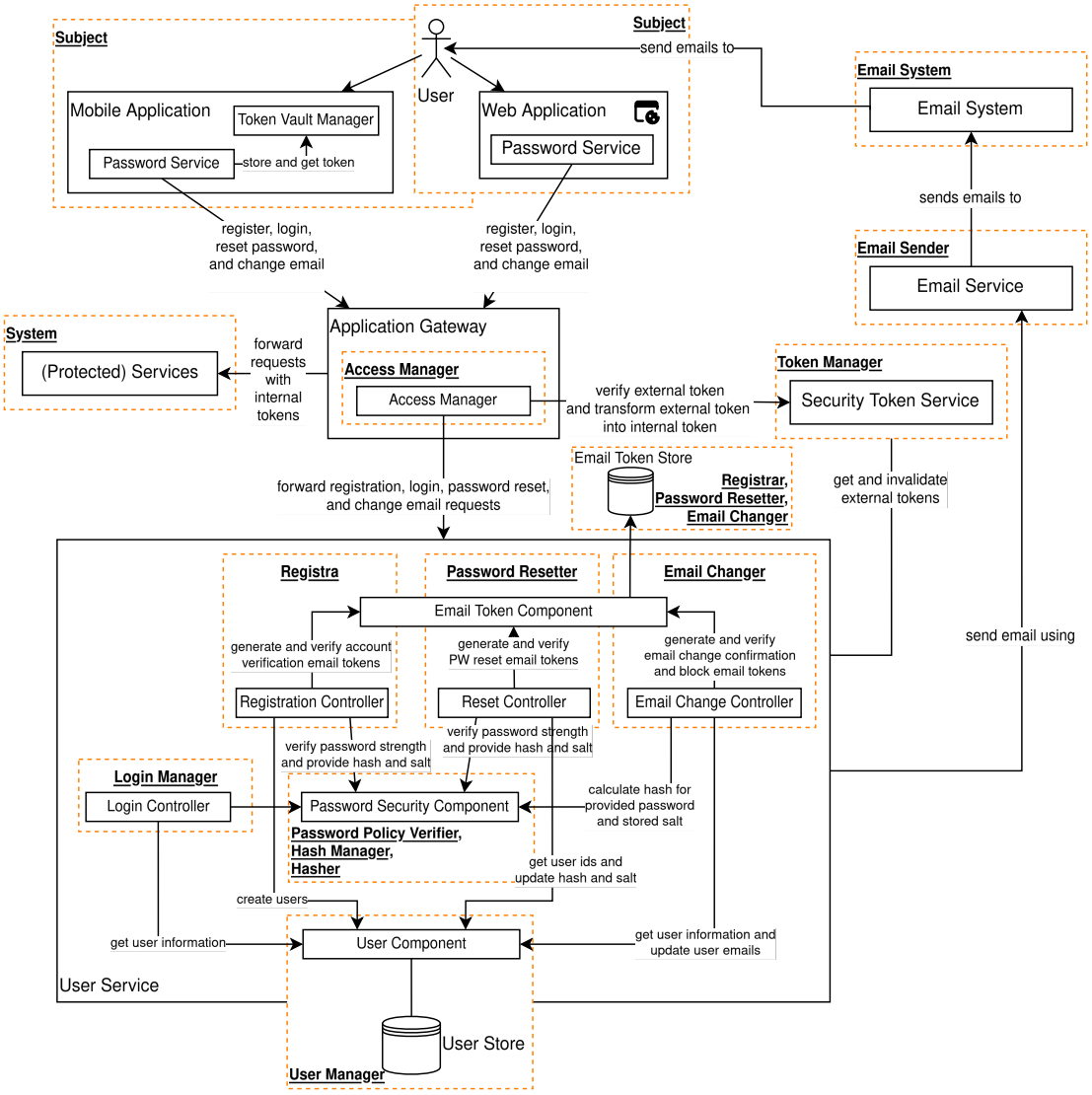


Figure 6.13.: Architectural Model representing the static structure of a microservice architecture that implements the PBA Roles, visualized in a C4 component diagram.

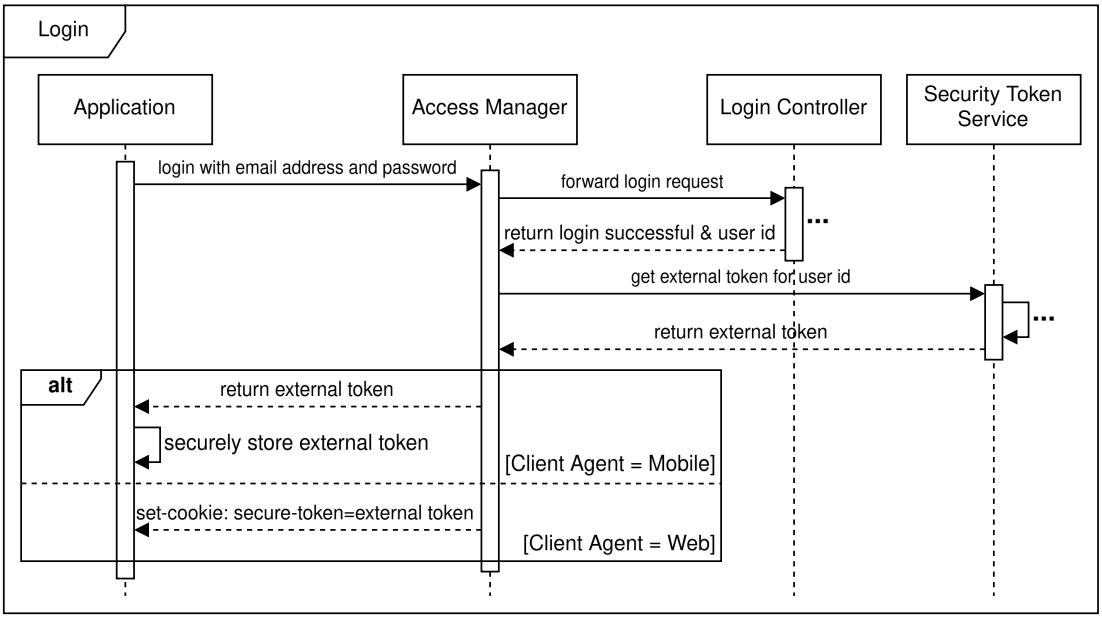


Figure 6.14.: Part of an Architectural Model that represents the dynamic structure for user login.

7. Security Design Pattern Knowledge Base Metamodel

As introduced in Section 2.3, there can be multiple SDPs that provide different implementation options for a *Security Solution Pattern (SSP)*, which defines a security solution at a conceptual level. Therefore, a *Constraint-based Recommender System (CBRS)* can be used by architects who may not be familiar with the different options to choose appropriate SDPs.

To enable a CBRS to recommend and reason about appropriate SDPs, this chapter introduces the knowledge aspect of SDPs. While the usage aspect focuses on the information an architect needs to apply SDPs in practice, the knowledge aspect provides the information that describes and identifies an SDP within a CBRS KB. Returning to the example of buying real estate introduced in Section 2.4.1, the actual property that one lives in represents the usage aspect of a property, while information such as the price, number of bedrooms, or type of property represents the knowledge aspect of a property.

We assume that an architect has chosen an appropriate SSP based on the concepts defined by Abazi [Aba24] and only needs to choose an appropriate SDP for that SSP. In addition, we assume that there are at least two SDPs implementing an SSP; otherwise, the single SDP can be recommended directly. Finally, the use of the KBs within a CBRS is beyond the scope of this thesis project and is addressed in another work.

7.1. Security Design Pattern Knowledge Base Structure

This section introduces the *Security Design Pattern Knowledge Base Metamodel (SDP KB Metamodel)*, which follows the structure of a KB as introduced in Section 2.4.1. This means that the SDPs are described by the product properties $p_i \in V_P$, while the requirements an architect needs to provide to receive appropriate recommendations correspond to the user requirements $c_i \in V_C$.

The SDP KB Metamodel adheres to the hierarchical structure introduced in Section 2.3, which specifies a one-to-many relationship between SSPs and SDPs. Accordingly, for each SSP there is an SDP KB, as shown in Figure 7.1. This KB enables the recommendation of specific design options for that SSP. Multiple SDP KBs are required due to the varying characteristics of SDPs and the different factors that influence their recommendation based on the SSP they implement.

For example, a PBA SDP can be defined by the password reset mechanism used, while an OIDC SDP can be defined based on whether a single *Identity Provider (IdP)* or multiple IdPs are supported. This results in the following sets of Product Variables

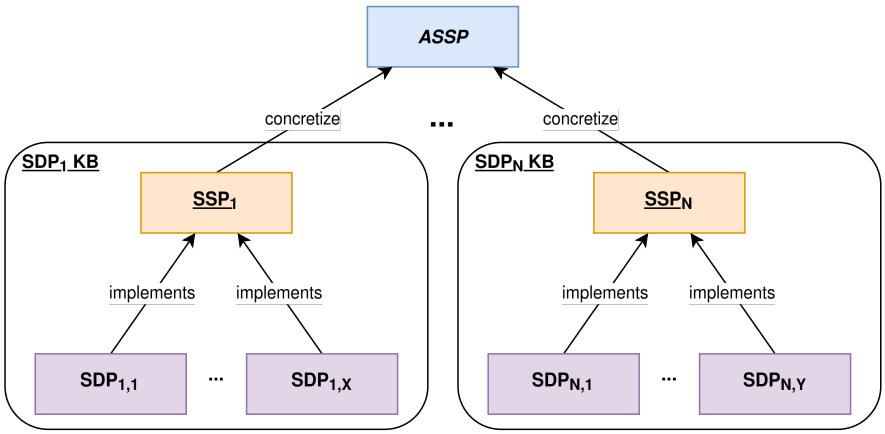


Figure 7.1.: Boundaries of different SDP KBs corresponding to their respective SSPs.

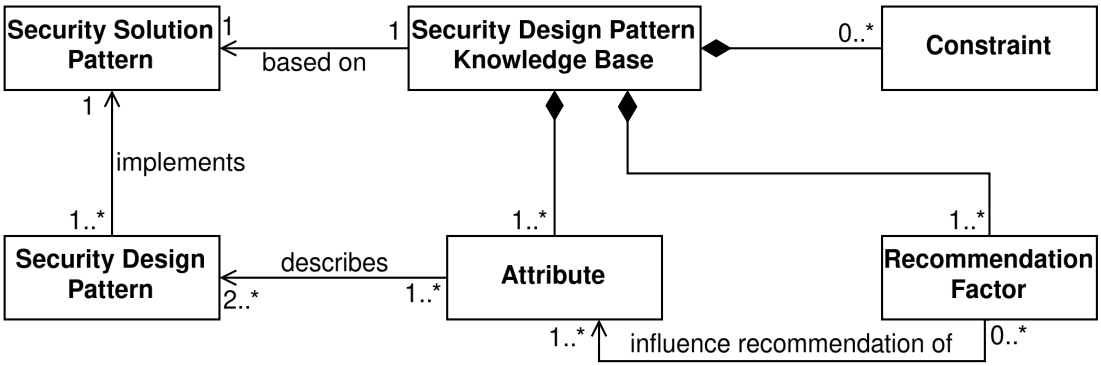


Figure 7.2.: An SDP KB is based on an SSP and contains Attributes (i.e., V_P), Recommendation Factors (i.e., V_C), and Constraints.

V_P, highlighting that the same SDP KB cannot be used to describe both PBA SDPs and OIDC SDPs:

$$V_{P,PBA} = \{p_1 : \textit{Password Reset Mechanism} \in \{\textit{SMS PIN}, \ldots\}\}$$
$$V_{P,OIDC} = \{p_1 : \textit{IdP(s) Used} \in \{\textit{Single IdP}, \textit{Multiple IdPs}\}\}$$

The use of a dedicated SDP KB for each SSP is consistent with the assumption that an architect has selected an SSP and only needs to choose an appropriate SDP for that SSP.

As illustrated in Figure 7.2, an SDP KB contains three primary elements: *Attributes*, *Recommendation Factors*, and *Constraints*. The Attributes correspond to the product properties $p_i \in V_P$, the Recommendation Factors correspond to the user requirements $c_i \in V_C$, and the Constraints include the Product, Compatibility, and Filter Constraints, as introduced in Section 2.4.1. We have adapted the terminology to better fit the pattern

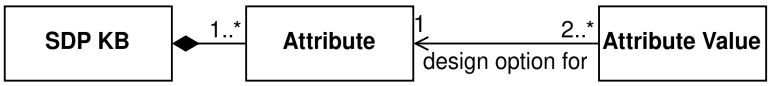


Figure 7.3.: The set V_P of the SDP KB is defined by the Attributes and their possible Attribute Values.

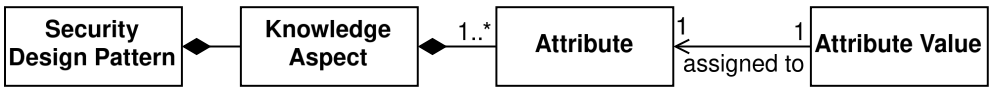


Figure 7.4.: The knowledge aspect of an SDP is defined by the Attribute Values assigned to each Attribute.

context, since the requirements in V_C refer to the system the architect plans to design, not to personal preferences, and SDPs are not products.

The following first introduces how Attributes are represented, including the information they contain. It then introduces the different Recommendation Factors that may be required as input for recommending an SDP, and finally introduces Constraints.

7.2. Attributes

This section details how the V_P set of an SDP KB is represented to enable recommendation and reasoning about appropriate SDPs.

Definition 7.1 (Attribute)**.** An *Attribute* describes a specific aspect of the conceptual solution of the SSP on which the SDP KB is based. An Attribute consists of a finite set of design options and is an element of V_P; that is $a_i \in V_P$. Each Attribute must have a unique and descriptive name.

For example, in a PBA SDP KB, the "Password Reset Mechanism" Attribute describes different methods for authenticating a user during the password reset process.

Definition 7.2 (Attribute Value)**.** An *Attribute Value* is a specific design option for an Attribute; that is $v_j \in a_i$, where $a_i \in V_P$. Each Attribute Value must have a unique and descriptive name within its Attribute.

For example, "Email Token" and "SMS PIN" are Attribute Values for the "Password Reset Mechanism" Attribute.

Attribute Values define the design options that exist for the solution detail described by the corresponding Attribute. Each Attribute must consist of at least two Attribute Values, as shown in Figure 7.3; otherwise, a CBRS would always select the single Attribute Value, rendering the Attribute unnecessary.

The knowledge aspect of an SDP is defined by the Attribute Values assigned for each Attribute, as shown in Figure 7.4. These Attribute Values uniquely identify an SDP.

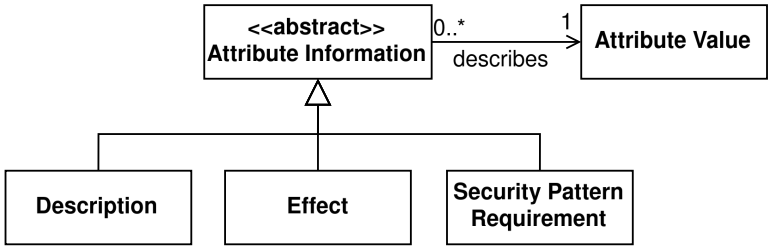


Figure 7.5.: Types of Attribute Information that enable reasoning about Attribute Values and support the recommendation process. Each Attribute Information describes exactly one Attribute Value.

For example, the "Password Reset Mechanism" Attribute with the "Email Token" and "SMS PIN" Attribute Values results in two SDPs:

$$SDP_{PBA,1} = \{a_1 : \textit{Password Reset Mechanism} = \textit{Email Token}\}$$
$$SDP_{PBA,2} = \{a_1 : \textit{Password Reset Mechanism} = \textit{SMS PIN}\}$$

Since the knowledge aspect of an SDP is defined by the assigned Attribute Values, each Attribute Value must have a unique impact on the usage aspect of an SDP. If this were not the case, two different assignments of Attribute Values could correspond to the same SDP. This assumption simplifies the recommendation process because the CBRS does not need to distinguish between two Attribute Values that lead to the same SDP. In addition, it does not limit expressiveness because multiple Attribute Values corresponding to the same SDP can be combined into a single Attribute Value.

For example, consider a "Session Timeout" Attribute, which describes whether users need to reauthenticate after being inactive for a period of time, with "No Timeout", "10 minutes", and "60 minutes" as Attribute Values. Here, we do not expect that the exact timeout duration impacts the usage aspect of an SDP; rather, the usage aspect provides guidance on how to properly choose the timeout duration. Therefore, using "No Timeout" and "Use Timeout" as Attribute Values that indicate whether a session timeout is used is sufficient.

7.2.1. Attribute Information

Definition 7.3 (Attribute Information)**.** *Attribute Information* encompasses any information associated with an Attribute Value.

The following details the types of Attribute Information considered in this thesis, shown in Figure 7.5.

Definition 7.4 (Description)**.** A *Description* provides detailed information about an Attribute Value in an unstructured format.

The Description is intended to be provided by a CBRS to the architect to explain a recommended Attribute Value. However, an advanced CBRS may use *Natural Language*

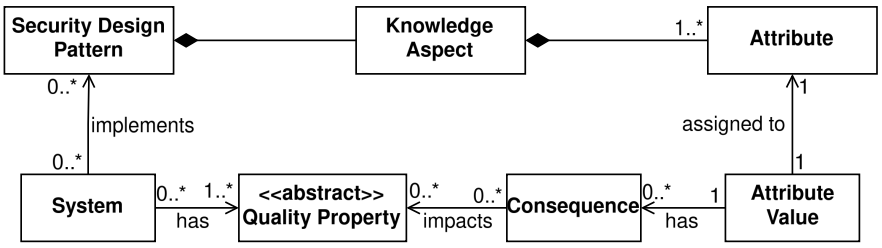


Figure 7.6.: Consequences affect the Quality Properties of systems implementing an SDP, where the Attribute Value is part of the SDP's knowledge aspect.

Processing or require a structured format to extract information from the Description and use it for the recommendation itself.

Definition 7.5 (Quality Attribute)**.** A *Quality Attribute* is a high-level property representing a broad and essential dimension of software quality. Quality Attributes help categorize the overall quality of software systems.

Definition 7.6 (Quality Factor)**.** A *Quality Factor* is a specific, measurable property that breaks down a Quality Attribute into more detailed aspects.

This thesis uses the ISO/IEC 25010:2023 [23] quality model, where Quality Attributes correspond to quality characteristics and Quality Factors correspond to quality sub characteristics. For example, "Confidentiality", which is the degree to which information is accessible only to authorized entities, is a Quality Factor for the Quality Attribute "Security" [23].

This terminology aligns with that used by Abazi [Aba24] and emphasizes that while we use the ISO/IEC 25010:2023 quality model, other quality models may be used or additional Quality Attributes and Quality Factors may be included. For instance, ISO/IEC 25010:2023 [23] does not cover Quality Attributes such as *Privacy* and *Sustainability*, which can be included if needed [Gli+23].

Definition 7.7 (Consequence)**.** A *Consequence* evaluates the impact of using an Attribute Value on the Quality Attributes and Quality Factors of a system. It includes an *Effect* that defines the type of impact, such as positive or negative, on system quality, and a *Reason* that explains the Consequence.

For example, for the "SMS PIN" Attribute Value, a Consequence can evaluate a negative Effect on the "Confidentiality" Quality Factor for the following Reason: "SMS PIN is vulnerable to SIM swapping attacks, where an attacker can gain control of the user's mobile number and obtain the SMS PINs" [Jov20].

A Consequence describes the impact on the Quality Properties of a system implementing an SDP where the Attribute Value is part of its knowledge aspect, as illustrated in Figure 7.6.

The Quality Attribute, Quality Factor, and Effect, as shown in Figure 7.7, can be used by a CBRS to calculate a recommendation score and to show the architect the impact

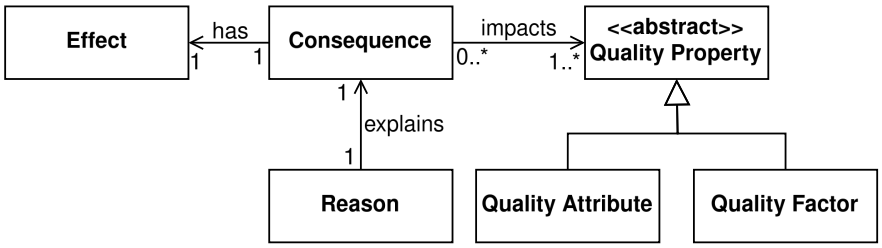


Figure 7.7.: Relationship between Consequence, Quality Property, Effect, and Reason.

of a recommended SDP on system quality. The Reason is intended to be provided by a CBRS to an architect to explain a Consequence. However, a CBRS may use Natural Language Processing or require a structured format to extract information from the Reason itself. For example, for the "SMS PIN" Consequence, a CBRS could identify that the Reason mentions a SIM swapping attack and provide more information about that attack to an architect.

This thesis considers an ordinal scale for the Effect, similar to the approach of Abazi [Aba24]. This scale categorizes the Effect as strongly negative (--), negative (-), neutral (0), positive (+), or strongly positive (++). An ordinal scale simplifies Effect assignment by focusing on whether one Attribute Value is worse, similar, or better than others, eliminating the need to quantify differences. This scale is sufficient for the Attribute Values considered in this thesis but can be extended to include additional ranks if necessary.

Definition 7.8 (Security Pattern Requirement)**.** A *Security Pattern Requirement (SP Requirement)* specifies that an Attribute Value requires the use of another security pattern. Each SP Requirement must reference the required security pattern and may include a description explaining why this specific Attribute Value requires the pattern.

Security Patterns, i.e., ASSPs, SSPs as well as SDPs, may require other Security Patterns to provide a secure solution to a problem as defined in Section 2.3.2. These relationships are incorporated into the usage aspect of an SDP using Abstract Roles. However, this information is also critical for a CBRS to not only recommend a single pattern, but to automatically resolve pattern relationships and provide all the Security Patterns necessary to design a secure system.

If an SSP, or any ASSP in its hierarchy, requires another Security Pattern, this relationship is assumed to be known by the CBRS. However, if only a subset of the SDPs that implement an SSP require a different Security Pattern, this information must be included in the knowledge aspect of an SDP by using the SP Requirements.

For example, in an OIDC KB, a "Token Type for Subsequent Request" Attribute describes whether the identity token received from the IdP or an internal token is used to authenticate subsequent requests, with "Identity Token" and "Internal Token" as Attribute Values. The "Internal Token" Attribute Value has an SP Requirement that references the Token-based Authentication ASSP introduced in Section 2.3.3, indicating that the CBRS must also recommend an SDP for this Security Pattern.

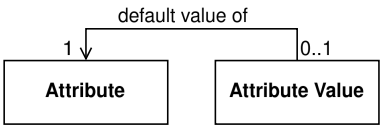


Figure 7.8.: An Attribute can optionally have a Default Value assigned to it.

7.2.2. Attribute Default Value

To simplify the recommendation process, an Attribute can be assigned a *Default Value*, as shown in Figure 7.8. A Default Value is generally considered the preferred choice for an Attribute and must include a rationale for its assignment.

For example, in a PBA SDP KB, the "Password Rotation" Attribute specifies whether users must periodically change their passwords. The "Yes" Attribute Value requires periodic changes, while the "No" Attribute Value does not require periodic changes. The "No" Attribute Value is the Default Value because password rotation is generally not recommended; it can lead to weaker passwords and introduce additional complexity, and there are more effective methods of protecting against the threats that periodic password changes address [Gra+20]. Nonetheless, some standards and organizations still require periodic password changes [PCI24], which are discussed in the next section. Therefore, it is necessary to provide SDPs for such scenarios to allow secure implementation.

Determining when an Attribute Value should be considered a Default Value is challenging, as the assignment of Default Values must be carefully reasoned to avoid introducing bias into the recommendation process [Agg16]. Therefore, Default Values should be used with caution and assigned by security experts.

In the future, potential Default Values may also be identified based on historical data if an Attribute Value is almost always recommended. In addition, a CBRS may not require explicitly defined Default Values, as historical interaction sequences can be used to dynamically predict them [FB08].

While this thesis does not describe in detail how Default Values are used in the recommendation process, one approach is for a CBRS to initially recommend all possible Default Values, focusing on determining appropriate Attribute Values for Attributes without Default Values. The recommendation can then present the Default Values to the architect along with the rationale for their assignment, allowing the architect to accept them or request further recommendations.

7.3. Recommendation Factors

This section describes the requirements $c_i \in V_C$ of an SDP KB that a CBRS may need to collect from an architect in order to recommend an appropriate SDP.

Definition 7.9 (Recommendation Factor)**.** A *Recommendation Factor* defines a requirement or an inherent system property that can impact the recommendation of appropriate

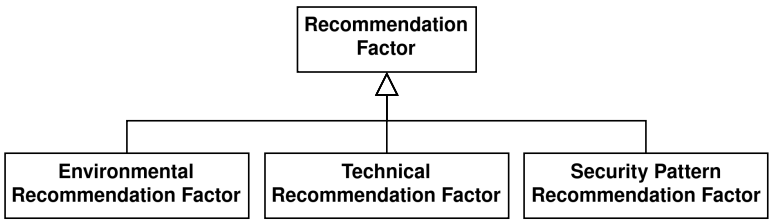


Figure 7.9.: The types of Recommendation Factors that can influence the recommendation of an appropriate SDP.

Attribute Values; that is, $f_i \in V_C$. Each Recommendation Factor consists of a set of possible values and must have a unique and descriptive name.

In the following, the terms "Recommendation Factor" and "Factor" are used interchangeably.

For example, in a PBA SDP KB, a "PCI DSS" Factor indicates whether the *Payment Card Industry Data Security Standard (PCI DSS)* applies to the architect's system, represented by the values "Applies" or "Not Applicable". This Factor is relevant because PCI DSS mandates the use of periodic password changes [PCI24].

Unlike Attributes, which are typically unique to an SDP KB and have a finite set of Attribute Values, Factors can be shared across multiple SDP KBs as long as they influence the recommendation of an SDP within an SDP KB. Moreover, Factors can be continuous. For example, an "Expected Number of Users" Factor allows the user to enter any number. This Factor can help determine the importance of scalability or performance considerations for the system.

7.3.1. Recommendation Factor Types

We define the following types of Factors, as shown in Figure 7.9.

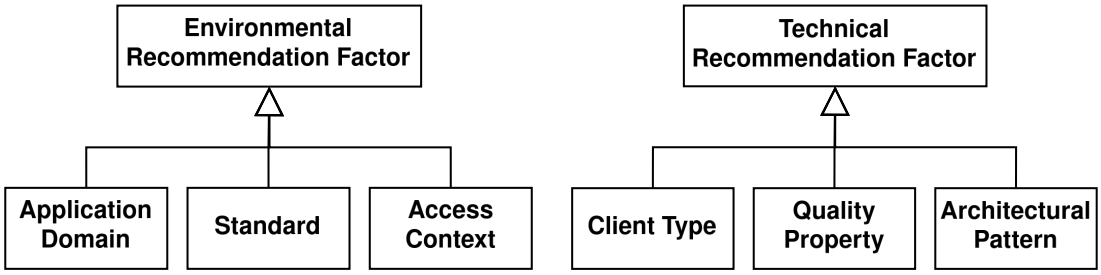
Definition 7.10 (Environmental Recommendation Factor)**.** An *Environmental Recommendation Factor* is a Factor that describes the general environment or context of the architect's system.

In the following, the terms "Environmental Recommendation Factor" and "Environmental Factor" are used interchangeably.

We consider the following Environmental Factors, shown in Figure 7.10a.

The "Application Domain" Environmental Factor specifies the sector in which the system will be used, with values such as "Financial Technology", "Healthcare", or "E-Commerce". The application domain can be relevant in determining required standards or defining Quality Properties that are important in the domain.

The "Standard" Environmental Factor indicates the standards that apply to the system, with values such as "PCI DSS" [PCI24] or "NIST 800-63B" [Gra+20]. Security standards such as PCI DSS may require or prohibit Attribute Values: for example, PCI DSS requires periodic password changes.



(a) Examples of Environmental Recommendation Factors.

(b) Examples of Technical Recommendation Factors.

Figure 7.10.: Example Recommendation Factors considered in this thesis.

The "Access Context" Environmental Factor specifies who accesses the system and how they interact with it, with the following values:

- *Internal Access*: Indicates that the system is accessible only to the internal organization, such as intranet portals.
- *Business-to-Business (B2B) Access*: Indicates that the system is accessible by other organizations, such as partner portals.
- *Business-to-Customer (B2C) Access*: Indicates that the system is accessible by customers, such as social media applications.

Definition 7.11 (Technical Recommendation Factor)**.** A *Technical Recommendation Factor* is a Factor that describes technical aspects of the architect's system.

In the following, the terms "Technical Recommendation Factor" and "Technical Factor" are used interchangeably.

We consider the following Technical Factors, shown in Figure 7.10b.

The "Client Type" Technical Factor specifies the type of clients that interact with the system, with values such as "Mobile", "Web", or "Smart TV". The client type can affect, for example, the authentication process in OIDC.

The "Quality Property" Technical Factor allows the architect to indicate which Quality Properties are important to the system, thereby favoring the Attribute Values that have a positive Effect on those Quality Properties. The values consist of the Quality Properties themselves and may allow the architect to specify how important each Quality Property is to the system.

The "Architectural Pattern" Technical Factor allows the architect to indicate which architectural pattern is used, with values such as "Microservice" or "Layered". The architectural pattern has an impact on the quality of the system, which can guide the recommendation [RF20]. For example, a microservice architecture offers strong scalability due to separately deployable and scalable services, while performance is negatively

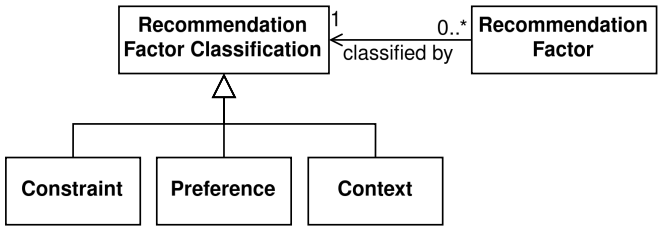


Figure 7.11.: Recommendation Factor Classification types and relationship to Recommendation Factor.

impacted due to communication overhead [RF20]. In this case, a CBRS can favor Attribute Values that positively affect scalability while avoiding those that negatively affect performance to avoid increasing performance issues.

Both Environmental and Technical Factors are designed to be understandable without requiring detailed knowledge of the Attributes and their Attribute Values. The fundamental difference between these two types of Factors is that Technical Factors have an explicit impact on the architecture of the system, whereas Environmental Factors do not directly impose architectural constraints.

Definition 7.12 (Security Pattern Recommendation Factor)**.** A *Security Pattern Recommendation Factor* is a Recommendation Factor that describes a characteristic or property of a security pattern.

In the following, the terms "Security Pattern Recommendation Factor" and "Security Pattern Factor" are used interchangeably.

For example, in an OIDC SDP KB, a "Identity Provider(s) Used" Security Pattern Factor specifies whether a single IdP or multiple IdPs are supported, indicated by "Single IdP" and "Multiple IdPs" as values.

The fundamental difference between Security Pattern Factors and Attributes is that Factors are in the set V_C, while an Attribute is in the set V_P of an SDP KB. This means that a CBRS can directly ask the architect a question to identify the specific value of a Security Pattern Factor that applies to the architect's system.

Security Pattern Factors are necessary because, in some cases, Environmental or Technical Factors cannot identify an appropriate Attribute Value for an Attribute. In such instances, a CBRS could recommend multiple SDPs with varying Attribute Values. However, this recommendation is only feasible if all these Attribute Values can be used in the architect's system. For example, in an organizational environment with a central IdP, an SDP that considers multiple IdPs should not be recommended.

7.3.2. Recommendation Factor Classification

Similar to the user requirements introduced in Section 2.4.1, i.e. $c_i \in V_C$, Factors can be categorized into *Constraint Factors*, *Preference Factors*, and *Context Factors*, as shown in Figure 7.11.

Constraint Factors are those where no SDP should be recommended that violates the Factor [FB08]. For example, the "Client Type" Technical Factor is a Constraint Factor because if an SDP is not compatible with a particular client type, a CBRS should never recommend it.

Preference Factors are those where an SDP that violates the Factor may be acceptable if it is very good, or if there is no other alternative [FB08]. For example, the "Quality Property" Technical Factor can be considered a Preference Factor. Suppose performance and maintainability are preferred, and an SDP offers a highly maintainable and secure solution with a neutral impact on performance. In such a case, an architect may select that SDP.

Context Factors are any information not directly provided as input but known to a CBRS [FB08]. For example, an "Token-based Authentication ASSP Recommended?" Security Pattern Factor that indicates whether the CBRS has already recommended a Token-based Authentication ASSP, is a Context Factor.

7.4. Constraints

This section introduces the constraints that are considered in an SDP KB, similar to the constraints introduced in Section 2.4.1.

Attribute Constraints (*PROD*) define conditions between Attributes. For example, in an OIDC SDP KB, an "Allow Fallback IdPs for Users" Attribute describes whether users can associate their user accounts with multiple IdPs and use any of them to authenticate to the system. The following Attribute Constraint defines that users can only associate multiple IdPs to their account if the system supports multiple IdPs:

$$\textit{Allow Fallback IdPs for Users} = \textit{Yes} \;\Rightarrow\; \textit{IdP(s) Used} = \textit{Multiple IdPs}$$

Compatibility Constraints (*COMP*) define conditions between Factors. For example, suppose the "Quality Property" (QP) Technical Factor allows specifying a list of Quality Properties and their importance on a scale from one to six, where one indicates that the Quality Property is not important and six indicates that it is very important. The following Compatibility Constraint specifies that in a microservice architecture, scalability is important to leverage its strengths, and performance is important to mitigate issues caused by communication overhead [RF20]:

$$\textit{Architecture Type} = \textit{Microservice} \;\Rightarrow\; \textit{QP} = \{(\textit{Scalability}, 5), (\textit{Performance}, 4)\}$$

Filter Constraints (*FILT*) define the mapping between Factors and Attributes. For example, consider the "Quality Property" (QP) Technical Factor as defined above and the "Password Reset Mechanism" Attribute with "SMS PIN" as a possible Attribute Value. The following Filter Constraint specifies that SMS PIN should be avoided if the

system requires high security:

$$\bigvee_{x \in \{4,5,6\}} (Security, x) \in QP \Rightarrow Password\ Reset\ Mechanism \neq SMS\ PIN$$

7.5. Recommendation Process

The recommendation process of a CBRS for SDPs is beyond the scope of this thesis project, but initial concepts and important considerations are provided.

To recommend appropriate SDPs, a CBRS asks the architect questions during an interactive process (e.g., "What client types does your system support?"). Each question aims to identify the specific values of a Factor that apply to the architect's system. The questions and their sequence can vary depending on the architect's answers. In addition, because we expect that the CBRS first recommends an appropriate SSP, it may already know the answer to a question from a previous recommendation. When all relevant questions are answered, the CBRS calculates a recommendation score, for example based on the Consequences assigned to the Attribute Values, and shows the recommended SDPs.

To reduce the number of questions that need to be asked of an architect, the CBRS can use default values; an initial concept of how a CBRS can use them is provided in Section 7.2.2. In addition, it may be possible that no suitable SDP can be identified based on the architect's answers. In such cases, a CBRS should provide guidance to the architect on which Preference Factor values can be adjusted to find an SDP [FB08].

8. Application Examples of SDP KBs

This chapter evaluates the SDP KB Metamodel by presenting practical examples that illustrate the concepts introduced in Chapter 7. In particular, we provide examples of an OIDC SDP KB and a PBA SDP KB. For an introduction to OIDC and PBA, see Section 2.2.1.

The OIDC SDP KB is used to demonstrate the Attributes (i.e., $a_i \in V_P$) that describe and identify the SDPs, the Factors (i.e., $f_j \in V_C$) required to recommend appropriate SDPs, and the Constraints within the KB. This includes a practical recommendation example where we demonstrate how a CBRS for SDPs can recommend an appropriate OIDC SDP. The PBA SDP KB is used to demonstrate that different SDP KBs are required due to the inherently different Attributes.

To effectively demonstrate the various concepts, only a subset of the SDP KBs is presented here; the full versions are available in Appendix A.

8.1. OpenID Connect SDP KB Attributes

This section introduces some of the Attributes and their Attribute Values from the OIDC SDP KB, as shown in Table 8.1. These examples demonstrates how the concepts from Section 7.2 are effectively applied within the KB. This includes Attribute Information, which describes Attribute Values, Attribute Default Values, which a recommender system can use to simplify the recommendation process, and the Attribute Constraints (*PROD*) introduced in Section 7.4. Each of the following sections presents a different Attribute.

8.1.1. OpenID Connect Flow

The "OpenID Connect Flow" Attribute considers the different methods for retrieving tokens from the *Identity Provider (IdP)*. In this example, the Attribute has two different Attribute Values with their Descriptions:

- *Authorization Code Flow with Proof Key for Code Exchange (PKCE)*: This flow requires the user to log in to the IdP using the same device that requires authentication at the *Relying Party (RP)*. The user is redirected to the IdP, and upon successful authentication, receives an authorization code. This authorization code is then automatically exchanged for an identity token that confirms authentication at the IdP [Har12]. PKCE enhances the security of the authorization code flow and is expected to become mandatory in OAuth 2.1 [HPL24]. This flow is suitable for devices with adequate input capabilities, such as mobile phones or web clients.

Attribute	Attribute Values		
a_1: *OIDC Flow*	Authorization Code Flow with PKCE	Device Flow	
a_2: *Identity Provider(s) Used*	Single IdP	Multiple IdPs per User Supported	Multiple IdPs but Only One per User
a_3: *Allow Fallback IdP per User*	Yes	No	
a_4: *Token Type for Subsequent Requests*	Identity Token	Internal Token	
a_5: *Client Authentication Method*	Client Secret Basic	Private Key JWT	

Table 8.1.: The OIDC SDP KB Attributes (i.e., $a_i \in V_P$) and their corresponding Attribute Values.

- *Device Flow*: This flow allows the user to log in to the IdP using a different device from the one that requires authentication at the RP. After initiating authentication on the primary device, the user must visit the IdP on a secondary device using a provided URL, enter a token, and authenticate. The primary device polls the IdP and, upon successful authentication on the secondary device, receives an identity token. The device flow is appropriate for input-restricted devices, such as smart TVs or devices without browser support [Den+19].

The only Attribute Information associated with each Attribute Value is the Description. This is because the choice between the two Attribute Values depends primarily on the type of client device, not on Quality Properties. In addition, neither Attribute Value requires the use of an additional security pattern, so there are no SP Requirements.

It can be argued that the difference between the Attribute Values is conceptual rather than design related. Therefore, it may be handled at the *Security Solution Pattern (SSP)* level by creating separate SSPs for each Attribute Value. However, it is important to consider the usage and knowledge aspects of an SSP to determine whether it addresses OIDC on a general level or focuses on a specific flow. In addition, moving this Attribute to the SSP level can be easily accomplished by splitting this KB into two KBs and removing this Attribute.

8.1.2. Identity Providers Used

The "Identity Provider(s) Used" Attribute considers the relationship between the RP and the IdP(s). In this example, the Attribute has three different Attribute Values with their Descriptions:

- *Single IdP*: The RP uses a single IdP, which can be either a general-purpose or an application-specific IdP. A general-purpose IdP provides application-independent

		Single IdP	Multiple IdPs per User Supported	Multiple IdPs but Only One per User
Maintainability		(+) Using a single IdP greatly reduces the complexity of configuration and user management.	(-) Increased complexity because multiple IdPs can be used. However, users explicitly choose which IdP to use.	(--) Increased complexity due to the need to manage multiple IdPs and determine the correct IdP for each user based on information such as their email domain or IP address.
Reliability	*Availability*	(-) Single point of failure, so if the IdP is unavailable, no users can authenticate.	(0) If an IdP is unavailable, some users can still authenticate.	(0) If an IdP is unavailable, some users can still authenticate.

Table 8.2.: Consequences associated with the "Identity Provider(s) Used" Attribute Values. Each column represents an Attribute Value. Each row corresponds to a Quality Attribute (in bold) and its optionally associated Quality Factors (in italics). Each cell specifies the Effect with its Reason.

information and is common in organizational environments where user data is maintained centrally and the same IdP serves multiple RPs. An application-specific IdP offers application-specific information and often includes advanced functionalities. For example, such an IdP may act as an identity broker integrating other IdPs, enforce PBA with MFA, or manage user roles and permissions.

- *Multiple IdPs per User Supported*: The RP directly supports multiple IdPs, allowing users to choose which IdP to use for login. This approach is common in consumer-oriented applications as it provides users with convenient login options, thereby increasing user conversion rates [GN14].

- *Multiple IdPs but Only One per User*: The RP directly supports multiple IdPs, but each user is associated with a specific IdP, often determined by the user's email domain. This setup is typical in systems that allow external organizations to use their own IdPs, eliminating the need for manual user creation or synchronization.

The Consequences associated with the Attribute Values are shown in Table 8.2. In addition, none of the Attribute Values require additional security patterns, so there are no SP Requirements.

This Attribute is complex because it requires a deeper understanding of OIDC. This is because a single application-specific IdP can provide functionality similar to the other Attribute Values. For example, Keycloak[1], an open source identity and access management system, can enable and manage authentication through multiple social providers such as Google and GitHub, while the RP only interacts with Keycloak.

[1] `https://www.keycloak.org/`

8.1.3. Allow Fallback Identity Providers for Users

The "Allow Fallback IdPs for Users" Attribute considers whether users can log in to their account using multiple IdPs. In this example, the Attribute has two different Attribute Values with their Descriptions:

- *Yes*: A user can associate multiple IdPs with their account and authenticate through any of these IdPs.
- *No*: A user is associated with a single IdP that must be used for authentication.

The Consequences associated with the Attribute Values are omitted as they do not offer additional insights and are not relevant to the remainder of the section. In addition, none of the Attribute Values require additional security patterns, so there are no SP Requirements.

The "Yes" Attribute Value requires that the RP integrates multiple IdPs, and users can choose which IdP to use, which is defined by the following Attribute Constraint:

$$\textit{Allow Fallback IdPs for Users} = \textit{Yes} \Rightarrow \textit{IdP(s) Used} = \textit{Mult. IdPs per User Supported}$$

The Attribute Constraint also suggests a sequence in which a CBRS should first identify an appropriate Attribute Value for the "Identity Provider(s) Used" Attribute, and only consider this Attribute if the "Multiple IdPs per User Supported" Attribute Value is appropriate.

8.1.4. Token Type for Subsequent Requests

The "Token Type for Subsequent Requests" (TTSR) Attribute considers the token used to authenticate subsequent requests. In this example, the Attribute has two different Attribute Values with their Descriptions:

- *Identity Token*: The identity token issued by the IdP is used directly to authenticate subsequent requests.
- *Internal Token*: The identity token is used only for the initial authentication of a user, and an internal token is issued for subsequent requests.

The Consequences associated with the Attribute Values are shown in Table 8.3. While these Consequences may not be exhaustive, it clearly shows that there are limited benefits to using the identity token directly. Consequently, the "Internal Token" Attribute Value is the Default Value for this Attribute, emphasizing that the identity token should be used with caution.

The "Internal Token" Attribute Value has an SP Requirement that references the Token-based Authentication ASSP introduced in Section 2.3.3. This relationship cannot be represented at the SSP level because no additional security pattern is required when the identity token is used directly. Moreover, referencing an ASSP ensures that a CBRS can recommend an appropriate SSP for this pattern.

		Identity Token	Internal Token
Maintainability		(+) No additional internal token management is required, including generation, expiration, verification, and storage.	(-) Additional token management required, including generation, expiration, verification, and storage.
Security	*Confidentiality*	(-) Identity tokens are exposed to the client, potentially allowing an attacker to access them. This is problematic because identity tokens may contain sensitive information and the RP has no control over the lifetime of the token, so a compromised token may be valid for an extended period of time.	(+) The RP has full control over the token, so the sensitive information it contains can be minimized and appropriate security mechanisms, such as short lifetime or token revocation, can be implemented.
	Integrity		
Reliability	*Availability*	(-) When the identity token expires, a new token must be issued by the IdP. If the IdP is unavailable during this process, the user loses access to the RP even though the RP itself is operational.	(+) After successful authentication, subsequent requests are independent of the IdP. So if the IdP is unavailable and a token expires, it can be refreshed without having to communicate with the IdP.

Table 8.3.: Consequences associated with the "Token Type for Subsequent Requests" Attribute Values. Each column represents an Attribute Value. Each row corresponds to a Quality Attribute (in bold) and its optionally associated Quality Factor (in italics). Each cell specifies the Effect with its Reason.

The "Identity Token" Attribute Value should not be used when the RP integrates multiple IdPs. This is because identity tokens may not have a consistent format, which increases complexity and reduces the positive maintainability Consequence of using the identity token directly. This is addressed by the following Attribute Constraints:

$$TTSR = \textit{Identity Token} \Rightarrow \textit{IdP}(s)\ \textit{Used} \notin \{\textit{Multiple IdPs per User Supported}, \textit{Multiple IdPs but Only One per User}\}$$

This Attribute is motivated by the fact that using the identity token has certain advantages, such as containing user information, allowing its integrity to be verified, and not requiring additional components. However, this Attribute also highlights the significant drawbacks of using the identity token, and discourages architects from using it. Nevertheless, it is required to provide SDPs that offer secure solutions to ensure that if an architect chooses to use the identity token, it is implemented securely.

Attribute	SDP A	SDP B
OIDC Flow	Authorization Code Flow with PKCE	Device Flow
Identity Provider(s) Used	Single IdP	Multiple IdPs per User Supported
Allow Fallback IdP per User	No	Yes
Token Type for Subsequent Requests	Internal Token	Internal Token
Client Authentication Method	Private Key JWT	Client Secret Basic

Table 8.4.: Two example SDP knowledge aspects that exist in the OIDC SDP KB.

8.2. OpenID Connect SDP KB Recommendation Example

In this section, we present a practical recommendation example in which we simulate the role of a CBRS to identify appropriate SDPs for a synthetic scenario. Two example OIDC SDPs that can be recommended, defined by their associated Attribute Values, are shown in Table 8.4.

In the recommendation process, the CBRS must first identify any Attribute Values that must be used in the architect's system because they depend on Constraint or Context Factors. For example, if a system must use an organization's IdP, the "Single IdP" Attribute Value must be used for the "Identity Provider(s) Used" Attribute. The CBRS then calculates a recommendation score for all feasible SDPs and provides the architect with these SDPs ranked by their recommendation score.

8.2.1. Scenario

Joe has worked at several companies where time management systems were too difficult to use and only accessible through legacy web applications. He decides to create a startup to develop an easy-to-use time management system for organizations. He plans to sell the system as a *Software-as-a-Service*, where he is responsible for running the system, and organization members can access it through modern web and mobile applications. Having no prior experience in designing secure systems, Joe connects to a recommender system for secure system design.

As a first step, the recommender asks Joe some questions to identify an appropriate SSP. Based on the requirements that different organizations should be able to use the system and that usability is an important Quality Property, the recommender suggests using the "OpenID Connect Authentication" SSP. Joe selects the recommended pattern, and since he does not know how to design a secure system using OIDC authentication, he asks the recommender for an appropriate SDP.

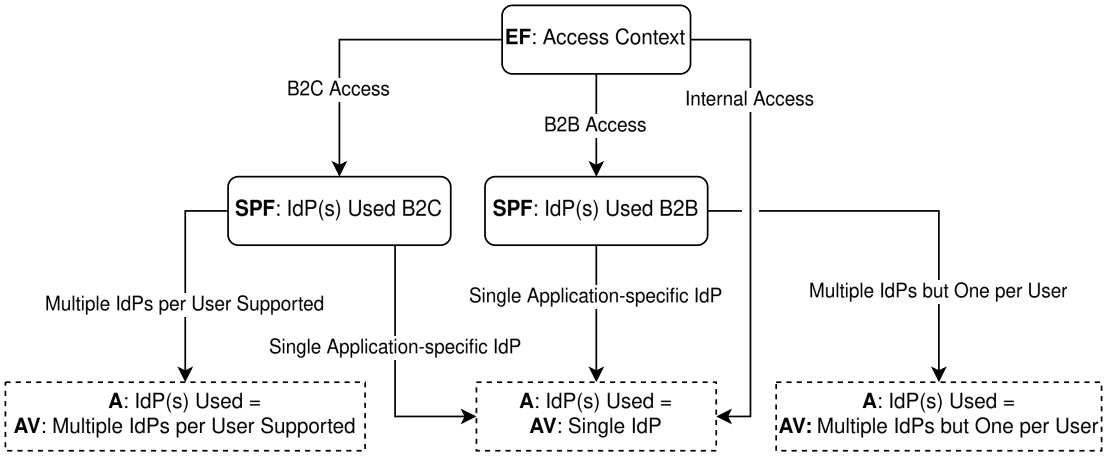


Figure 8.1.: Sequence to identify the appropriate Attribute Value for the "Identity Provider(s) Used" Attribute. Rounded rectangles represent Environmental Factors (EF) and Security Pattern Factors (SPF). Dotted rectangles represent the assignment of an Attribute Value (AV) to an Attribute (A).

8.2.2. Recommendation Process

The following sections present the recommendation process of a CBRS that guides Joe in selecting the most appropriate OIDC SDP. Each section aims to identify an Attribute Value for a specific Attribute and may use information provided in previous subsections.

OpenID Connect Flow

To determine the appropriate Attribute Value for the "OIDC Flow" Attribute introduced in Section 8.1.1, the "Client Type" (CT) Technical Factor is used. The relevant values considered by the KB are "Mobile", "Web", "Desktop", and "Embedded System with User Interface" (ES with UI), such as smart TVs or home automation panels.

In this scenario, Joe has already provided the CBRS with the requirement that the system must support both web and mobile clients during the initial recommendation of the SSP. Therefore, the CBRS does not explicitly ask Joe and identifies the "Authorization Flow with PKCE" as the appropriate Attribute Value. This is based on the following Filter Constraint:

$$\bigvee_{x \in \{Mobile, Web, Desktop\}} x \in CT \Rightarrow OIDC\ Flow = Authorization\ Flow\ with\ PKCE$$

Identity Providers Used

To determine the appropriate Attribute Value for the "Identity Provider(s) Used" Attribute introduced in Section 8.1.2, the "Access Context" Environmental Factor is used,

as shown in Figure 8.1. The CBRS asks Joe the following question "Who should be able to access the system?" with the following response options:

(A) "Internal organization members." (Internal Access)

(B) "Members of other organizations." (B2B Access)

(C) "Individual customers." (B2C Access)

Joe chooses option (B) because his Software-as-a-Service solution will be used by members of several external organizations.

Joe's choice means that it must be possible to integrate the external organizations' IdPs into the system. This can be achieved by having a single application-specific IdP, such as Keycloak, manage the organizations' IdPs, and the system only interacts with that single IdP. Alternatively, the system can directly integrate the organizations' IdPs without the need for an additional component.

To determine this, the CBRS uses the "IdP(s) Used B2B" Security Pattern Factor, which has possible values of "Single Application-specific IdP" and "Multiple IdPs but One per User", as shown in Figure 8.1. Joe chooses the "Single Application-specific IdP" value because using an additional component is acceptable to him, as it reduces system complexity. This identifies the "Single IdP" as the appropriate Attribute Value for the "Identity Provider(s) Used" Attribute.

The sequence in Figure 8.1 has the advantage that an architect first only needs to consider the "Access Context" Environmental Factor. This Factor does not require knowledge of OIDC and can be easily specified by an architect. The specific Security Pattern Factors then limit the possible values based on the Access Context. For example, in this scenario, Joe does not need to know about the "Multiple IdPs per User Supported" value. This highlights both the benefit of using a CBRS to reduce the technical details an architect needs to know, and the need for Security Pattern Factors.

Possible Security Design Patterns to Recommend

Based on Joe's responses, the following Attribute Values must be used:

$$
\begin{aligned}
&a_1\text{: } \textit{OIDC Flow} && = \textit{Authorization Flow with PKCE}\\
&a_2\text{: } \textit{Identity Provider(s) Used} && = \textit{Single IdP}\\
&a_3\text{: } \textit{Allow Fallback IdP per User} && = \textit{No}
\end{aligned}
$$

The Attribute Values for the "OIDC Flow" and "Identity Provider(s) Used" Attributes are explicitly determined based on the questions asked. In contrast, the Attribute Value for the "Allow Fallback IdP per User" Attribute is implicitly required because it can only be "Yes" if multiple IdPs per user are supported, as defined by the Attribute Constraint in Section 8.1.3.

This means that the "Token Type for Subsequent Requests" and "Client Authentication Method" Attributes remain to be determined. This leads to the following possible

Attribute	Attribute Value	Security	Availability	Maintainability
Token Type for Subsequent Requests	Identity Token	- (1)	- (1)	+ (3)
	Internal Token	+ (3)	+ (3)	- (1)
Client Authentication Method	Client Secret Basic	0 (2)	0 (2)	0 (2)
	Private Key JWT	++ (4)	0 (2)	- (1)

Table 8.5.: The remaining two Attributes to be selected and their Effect on the Security, Maintainability, and Availability Quality Properties. The numeric value specifies the value used to calculate the recommendation scores.

SDPs that the CBRS could recommend to Joe:

$$SDP_{OIDC,1} = \{Common\ Attributes,\ a_4 = Identity\ Token,\ a_5 = Client\ Secret\ Basic\}$$
$$SDP_{OIDC,2} = \{Common\ Attributes,\ a_4 = Internal\ Token,\ a_5 = Client\ Secret\ Basic\}$$
$$SDP_{OIDC,3} = \{Common\ Attributes,\ a_4 = Internal\ Token,\ a_5 = Private\ Key\ JWT\}$$

Here, the "Common Attributes" represent the Attribute Values shared among the SDPs, omitted for brevity.

The combination of "Identity Token" and "Private Key JWT" Attribute Values is prohibited by an Attribute Constraint because it is impractical to use a highly secure client authentication method with a less secure token type.

Recommendation Scores

The following formula, adapted from Felfernig and Burke [FB08], is used to calculate the recommendation score of the possible SDPs:

$$score(x) = \sum_{q \in QP} e_q s_q(x) \tag{8.1}$$

$$s_q(x) = \sum_{i=1}^{n} effect_q(a_{i,x}) \tag{8.2}$$

In Equation (8.1), QP represents the set of Quality Properties an architect is interested in, e_q is the importance of a Quality Property q to the system, and $s_q(x)$ denotes the rating of an SDP for a Quality Property q. In Equation (8.2), $a_{i,x}$ is the Attribute Value of Attribute a_i for SDP x, and $effect_q(a_{i,x})$ is the rating of the Attribute Value for Quality Property q. The rating is based on the Consequences of an Attribute Value, where Effects are mapped from zero to four, with zero representing a strongly negative Effect and four representing a strongly positive Effect.

During the initial SSP recommendation, Joe specified that security is paramount and that availability and maintainability are equally important to the system, i.e., $e_{Security} = 0.5$, $e_{Availability} = 0.25$, and $e_{Maintainability} = 0.25$. Based on the Effects for the Attribute

Attribute	Attribute Values		
a_1: *Password Reset Mechanism*	SMS PIN	Email Token	Offline Backup Codes
a_2: *Password Peppering*	HMAC	Symmetric Encryption	No
a_3: *Password Rotation with History*	Yes	No	
a_4: *User Account Lockout*	Yes	No	

Table 8.6.: The PBA SDP KB Attributes (i.e., $a_i \in V_P$) and their corresponding Attribute Values.

Values shown in Table 8.5, we have the following recommendation scores. We only consider Attribute Values that differ between the SDPs in the calculation of $s_q(x)$.

$$\begin{aligned}
score(SDP_{OIDC,1}) &= e_{Sec} \cdot s_{Sec}(SDP_{OIDC,1}) + \dots \\
&= 0.5 \cdot (effect_{Sec}(\textit{Id. Token}) + effect_{Sec}(\textit{Client Secret Basic})) + \dots \\
&= 0.5 \cdot (1+2) + 0.25 \cdot (1+2) + 0.25 \cdot (3+2) = 3.5 \\
score(SDP_{OIDC,2}) &= 0.5 \cdot (3+2) + 0.25 \cdot (3+2) + 0.25 \cdot (1+2) = 4.5 \\
score(SDP_{OIDC,3}) &= 0.5 \cdot (3+4) + 0.25 \cdot (3+2) + 0.25 \cdot (1+1) = 5.25
\end{aligned}$$

Final Recommendation

Based on the recommendation scores, the CBRS recommends the SDPs in the following order: $SDP_{OIDC,3}$, $SDP_{OIDC,2}$, and $SDP_{OIDC,1}$. Joe trusts the CBRS and chooses $SDP_{OIDC,3}$, so the CBRS returns the usage aspect of the SDP to Joe.

Since the SDP's "Internal Token" Attribute Value has an SP Requirement for the Token-based Authentication ASSP, the CBRS asks Joe if he wants to continue the recommendation process. Joe agrees, and the recommendation process repeats. In the end, Joe has a set of SDPs that he can use to design a secure time management system.

8.3. Password-based Authentication SDP KB

This section introduces the PBA SDP KB to demonstrate that different SSPs require the use of a dedicated SDP KB. For this, selected Attributes are explained, where Table 8.6 shows all the Attributes and Attribute Values that make up this KB. These Attributes show that it is impractical to consider them together with the OIDC SDP KB. A recommendation example is omitted because the example in the previous section already covered all cases.

		SMS PIN	Email Token	Offline Backup Codes
Interaction Capability		(+) Users are used to receiving SMS, so the process is intuitive and easy to follow.	(+) Users are used to receiving email, so the process is intuitive and easy to follow.	(-) Requires detailed instructions to explain their use and ensure that users keep backup codes safe.
Reliability	*Availability*	(0) While cellular service can be unavailable, it is quite reliable and often available in areas with limited internet access.	(-) If the email service is unavailable, the email is delayed, or identified as spam, a user cannot reset their password.	(+) Backup codes do not rely on any additional services.
Security		(--) SMS PIN is vulnerable to SIM swapping attacks, where an attacker gains control over the user's mobile number and can receive the SMS PINs [Jov20].	(0) If the user's email account is compromised, an attacker can intercept reset links. However, using a secure password and MFA for the email account mitigates this risk.	(+) Stealing backup codes requires physical access or significant effort, as such codes are often stored locally by the user.

Table 8.7.: Consequences associated with the "Password Reset Mechanism" Attribute Values. Each column represents an Attribute Value. Each row corresponds to a Quality Attribute (in bold) and its optionally associated Quality Factor (in italics). Each cell specifies the Effect with its Reason.

8.3.1. Password Reset Mechanism

The "Password Reset Mechanism" Attribute specifies different methods of authenticating a user during the password reset process. In this example, the Attribute has three different Attribute Values with their Descriptions:

- *SMS PIN*: This mechanism sends a one-time PIN to the user's registered mobile phone number. The user must enter this PIN, which the system validates to allow the password reset.
- *Email Token*: This mechanism sends a password reset link containing a secure token to the user's registered email address. The token associates the user with the reset link, enabling the password reset upon verification.
- *Offline Backup Codes*: This mechanism relies on pre-generated one-time backup codes that are provided to the user after registration. The user can use these codes to reset their password without relying on external communication channels.

The Consequences associated with the Attribute Values are shown in Table 8.7. In addition, none of the Attribute Values require additional security patterns, so there are no SP Requirements.

	Maintainability	Interaction Capability	Security
Yes	(-) The system must store password histories, track password age, and notify users when a password change is needed.	(-) Periodic password changes frustrate users and may require password reset if users forget their new password.	(--) Periodic password changes can lead to weaker passwords, as users may find it difficult to remember new strong passwords every few months.
No	(+) No additional password management required.	(+) No interruptions during authentication and a simplified user experience.	(0) Users can use stronger passwords, but compromised passwords may be valid longer.

Table 8.8.: Consequences associated with the "Password Rotation with History" Attribute Values. Each column corresponds to a Quality Attribute. Each row represents an Attribute Value. Each cell specifies the Effect with its Reason.

8.3.2. Password Rotation with History

The "Password Rotation with History" Attribute specifies whether users are required to update their passwords periodically and cannot reuse some previous passwords. In this example, the Attribute hast two Attribute Values with their Descriptions:

- *Yes*: Users must update their passwords periodically, and the system maintains a history of previously used passwords to ensure that new passwords are different from recent ones.
- *No*: Users do not have to update passwords periodically.

The Consequences associated with the Attribute Value are shown in Table 8.8. The negative Effect of the "Yes" Attribute Value on security may seem counterintuitive, since changing passwords periodically reduces the time an attacker has to exploit a compromised password [PCI24]. However, more effective mechanisms, such as slow hashing algorithms or rejecting common and previously compromised passwords, provide better security [Sto+21].

Due to the significant drawbacks of using password rotation with history, we assign the "No" Attribute Value as the Default Value for this Attribute This indicates that the use of password rotation should generally be avoided. However, certain standards, such as the *Payment Card Industry Data Security Standard*, require password rotation with history in certain situations [PCI24]. Therefore, the Attribute must be part of the KB.

9. Discussion

This chapter discusses the results of this thesis with respect to the research questions defined in Chapter 3. First, we discuss the SDPDM introduced in Chapter 5, which defines the usage aspect of SDPs that allows architects to effectively implement an SDP in their system. Second, we discuss the SDP KB Metamodel introduced in Chapter 7, which defines the elements and relationships of the KBs required by a CBRS to recommend appropriate SDPs, including the knowledge aspects of SDPs.

9.1. Security Design Pattern Usage Aspect

This section discusses the results used to answer our first research question:

> **RQ1**: *What information is essential in a security design pattern, and how should a pattern description be structured to make it practical for architects?*

The SDPDM is based on the security pattern structure of Heyman et al. [Hey+07]. Thus, an SDP consists of context, problem, solution, example, and consequences. These components provide a strong foundation and ensure that SDPs have a consistent format.

Adopting the viewpoint and view concept to describe the solution and example of an SDP has several advantages. First, since views are common in software architecture, architects are likely to be familiar with the concept. Second, because each view focuses on a specific aspect, architects can find the relevant information more easily, improving understandability.

The Conceptual View defines the scope of an SDP, with Policies describing what the solution addresses and Roles specifying what is needed to implement these Policies. Abstract Roles address the problem that existing security patterns often only reference each other informally [Hey+07]. By explicitly modeling how other security patterns are integrated, this approach improves understanding and simplifies combining patterns.

For example, in the PBA SDP introduced in Section 6.3, the "Password Resetter" Role requires the "Token Manager" Abstract Role to invalidate tokens after a successful password reset. In a Session-based Authentication SDP, a "Session Invalidator" Role could be responsible for invalidating user sessions. To combine both SDPs, the relationship from the "Password Resetter" to the "Token Manager" can be replaced by a relationship to the "Session Invalidator", as shown in Figure 9.1. Since the Behavioral View and the Structural View are based on Roles, this can be applied consistently across these views.

The SDPDM addresses the limited inclusion of security-relevant information in existing security patterns in several ways.

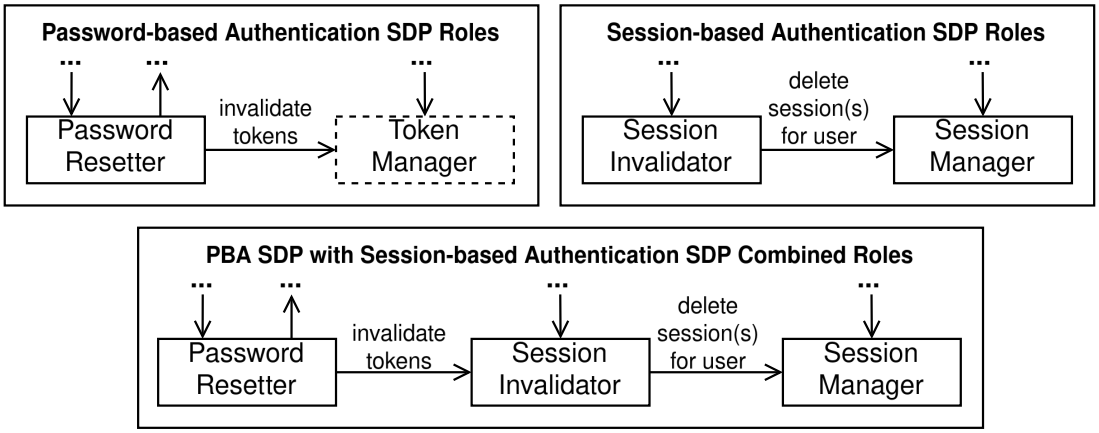


Figure 9.1.: Example for combining the Roles of a PBA SDP with a Session-based Authentication SDP.

The Data View models the Data Elements used in an SDP, incorporating data-related security considerations. Data Properties indicate the protection needs of Data Elements and can assist in identifying appropriate countermeasures. Data Rules specify conditions that Data Elements must fulfill to ensure security, and by explicitly including these rules, we reduce the likelihood of them being overlooked. Practical examples of Data Rules enhance understanding and help guide implementation. Finally, Data Fields can effectively model the dynamic nature of data in security solutions.

The Behavioral View provides understandable *Behavioral Models (BMs)* that shows interactions between Roles using Data Elements and Events. Events, with their Security Considerations, clarify how to handle important situations securely and highlight potential threats. Security Considerations can serve as a checklist for architects, ensuring that critical aspects are addressed. Finally, Error Events can be effectively used to model exceptional cases.

Unlike many existing security patterns that only list known uses, the Structural View provides concrete examples of how to implement an SDP. This gives architects a starting point, enabling them to adapt these examples or discuss their suitability before implementing the pattern in their own systems.

In summary, the SDPDM is essential for creating practical and understandable SDPs. By analyzing existing security solutions and security patterns, we included important security considerations for both data and behavior. The Role concept, adapted from Van den Berghe et al. [vYJ22a], clarifies how SDPs relate to each other, making it easier to combine multiple SDPs. The application examples introduced in Chapter 6 demonstrated the SDPDM in practice. Nevertheless, there are limitations and threats to validity, which we discuss next.

9.1.1. Limitations

First, the SDPs presented in Chapter 6 are candidate or proto-patterns [App97]. This is because to qualify as a full pattern, two properties must be met. A pattern must have concrete implementations that show that it addresses a recurring problem, often referred to as the rule of three, and a pattern must be useful to architects [App97]. The detailed nature of SDPs with multiple SDPs for a security solution can make it harder to find such recurring implementations in practice.

Second, due to the time constraints of this thesis project, the SDPDM has only been demonstrated with OIDC authentication and PBA. Therefore, we cannot assert that it is generally applicable. Future work should develop SDPs for other security objectives, such as authorization or automated attack detection.

Third, while the SDPDM ensures consistency and that security-relevant information is included in the SDPs, architects must become familiar with it before they can use the patterns effectively. In addition, security patterns from other catalogs cannot be directly integrated, as they do not conform to the SDPDM.

Finally, the examples in Chapter 6 were created based on available specifications and best practices, but we cannot guarantee their correctness and completeness. For example, a recent conference talk recommended keeping JWTs within internal systems [Kra]. Thus, providing SDPs using the OIDC identity tokens to authenticate requests should be carefully evaluated. Nevertheless, the Kubernetes API uses OIDC identity tokens for authentication, which shows that it is used in practice [DA21] In addition, we clearly state the negative impact of using the identity token on various quality attributes such as security.

9.2. Security Design Pattern Knowledge Base

This section discusses the research questions related to the SDP KB Metamodel, which specifies the elements necessary for a CBRS to recommend appropriate SDPs. The use of a CBRS is necessary because there can be multiple SDPs for a conceptual security solution defined by a SSP, and architects cannot be expected to be familiar with all of them.

9.2.1. Recommendation Factors

We begin by examining the findings that address the second research question.

> **RQ2**: *What recommendation factors can influence the selection of an appropriate security design pattern?*

Existing security pattern catalogs typically group related patterns and consider security objectives and quality tradeoffs to support the selection process. In contrast, this thesis introduces various Recommendation Factors that can influence the selection of an appropriate SDP.

Among these, Environmental and Technical Factors are essential for simplifying the recommendation process. They do not require detailed knowledge of specific SDPs, thereby enabling architects to provide the necessary input more easily. This is particularly beneficial for non-security experts, who may lack the technical knowledge needed due to the complexity and rapid evolution of the security domain [LS24].

One example is the "Standard" Environmental Factor, which is valuable because standards are frequently used in practice to derive security requirements [LS24]. By incorporating standards into a KB, it can be ensured that recommended SDPs conform to those standards applicable to an architect's system. In addition, standards can significantly narrow down the list of feasible SDPs, thereby simplifying the recommendation process.

The introduced Factor Classifications are essential to make these Recommendation Factors usable in a CBRS. By clearly indicating which Factors can be violated if, for example, no SDP is found and which cannot, the CBRS can make appropriate recommendations. However, the current concept may be too restrictive by requiring each Factor to be classified by exactly one Factor Classification. For instance, a standard could be considered a Constraint Factor, strictly forbidding the recommendation of SDPs that do not conform to it, or a Preference Factor, allowing the recommendation of SDPs that may not fully conform but are still acceptable options.

In summary, the introduced Recommendation Factors provide a solid foundation for identifying the feasible SDPs in an architect's context. Furthermore, by classifying the Factors appropriately, we enable the CBRS to suggest changes to the architect's requirements when they conflict. The example in Section 8.2.2 illustrates how these Factors can be effectively utilized by a CBRS.

9.2.2. Security Design Pattern Reasoning

This section discusses the results used to answer the third and fourth research questions:

> **RQ3**: *What knowledge about a security design pattern must be captured in order to reason about an appropriate pattern and explain the recommendation?*

> **RQ4**: *How should the knowledge be encoded so that a knowledge-based recommender system can reason about appropriate security design patterns?*

To effectively reason about SDPs, we identified the need for a unique SDP KB for each SSP. This is due to the different characteristics of SDPs based on the SSP they implement. Within a KB, we distinguish SDPs using Attributes that address specific aspects of the SSP, and Attribute Values that define possible design options for these Attributes. The knowledge aspect of an SDP is defined by the Attribute Values assigned to each Attribute.

While Recommendation Factors are used to narrow down the list of SDPs to feasible ones, a CBRS requires quantifiable knowledge to reason about SDPs. To provide such quantifiable knowledge, we assign Consequences to each Attribute Value, reflecting its

impact on a system's quality properties. This approach aligns with existing security pattern methods and is suitable for the architecture domain, where quality properties are critical [YSJ12; RF20].

However, this thesis proposes a novel approach in which a CBRS automatically reasons about SDPs, including the architect's specific requirements, thereby enabling customized recommendations. Thus, even if the same set of feasible SDPs is considered, a CBRS can recommend different SDPs based on varying quality property priorities. To integrate this into a CBRS, we introduced a five-point scale to quantify the Consequences of each Attribute Value. This scale is sufficient for the Attribute Values presented in Chapter 8, but can be extended if needed.

Furthermore, inter-pattern relationships are crucial for the CBRS. By incorporating these relationships into the KB, the CBRS can automatically recommend all SDPs required to securely implement a system. We achieve this by introducing SP Requirements that can be assigned to Attribute Values and reference a required security pattern. Future work may consider automatically combining SDPs so that the architect receives a single integrated pattern instead of multiple separate SDPs.

Since architects must make informed decisions, each Consequence has an associated reason, and each Attribute Value has a description. While this information is not required for the reasoning, we expect it to be valuable for architects, allowing the CBRS to provide both a recommendation score and supporting explanations.

Finally, the goal of having multiple SDPs for a single SSP is to address different situational needs, but in some cases one Attribute Value may be generally preferable. For example, periodic password changes are typically not appropriate [Gra+20], but are mandated in PCI DSS [PCI24]. Such considerations can be represented by Default Values that the CBRS may recommend unless a Recommendation Factor explicitly requires an alternative Attribute Value.

In summary, Consequences are essential for a CBRS to reason about appropriate SDPs, and SP Requirements enable the recommendation of a set of SDPs rather than a single isolated SDP. The example in Section 8.2.2 illustrates how Consequences, along with their five-point scale, can be effectively used by a CBRS to recommend appropriate SDPs.

9.2.3. Limitation

First, each possible combination of Attribute Values results in a unique SDP, ensuring that architects only need to consider what is relevant in their context. However, this can lead to a large number of SDPs, which may be difficult to develop and maintain manually. Future work should consider whether automated tools can assist. For example, it may be possible to use SDP templates that generate concrete SDPs based on the recommended Attribute Values.

Second, architects should not be restricted to choosing a single appropriate value for a Recommendation Factor. For instance, an architect may need authentication for both mobile devices and smart TVs. This is possible with OIDC authentication, but requires using both the "Authorization Flow with PKCE" and the "Device Flow" Attribute

Values described in Section 8.1.1. Since an OIDC SDP can currently only have one of these Attribute Values, the CBRS cannot satisfy this requirement as is. One solution would be to introduce a third Attribute Value covering both flows, but this would further increase the number of SDPs. Another solution is for the CBRS to recommend two SDPs, one for each flow, but then the KB must somehow indicate that this combination is possible. Alternatively, if multiple SDPs can be automatically combined, a single integrated pattern could be returned.

Third, Default Values essentially have a weight of one in the recommendation score, while other Attribute Values have a weight of zero, unless explicitly required by a Recommendation Factor. However, there can be cases where one Attribute Value is more suitable than others, but the others are still acceptable. To account for this, a special type of Filter Constraint could be introduced to map Recommendation Factors to Attribute Values, influencing their weight in the recommendation score. For example, in a PBA SDP KB, if mobile is the only client type, such a Filter Constraint could increase the weight for all SDPs containing the "SMS PIN" Attribute Value. Conversely, if mobile is not used, a Filter Constraint could lower the weight for those SDPs.

Finally, similar to the usage aspect of SDPs, the SDP KB Metamodel has only been demonstrated with OIDC authentication and PBA. Therefore, we cannot assert that the SDP KB Metamodel is generally applicable.

10. Conclusion

Security patterns can provide valuable information for implementing secure systems, especially for non-security experts who often lack the necessary knowledge due to the complexity and rapid evolution of the security domain. The goal of this thesis is to present a first concept of practical security patterns. This chapter summarizes the research methodology and results, and briefly mentions opportunities for future research.

10.1. Summary

This thesis introduces SDPs designed to help architects securely implement a given security solution and that can be recommended by a CBRS. Using a CBRS is essential, as multiple SDPs can exist for the same security solution, each addressing different design options.

To achieve this, a bottom-up approach was first applied to iteratively collect and analyze security solutions in OSS, identifying common elements that improve the practical application of SDPs. In parallel, relevant literature was examined to understand existing security pattern approaches and to identify why their adoption in practice is limited. Based on these findings, a top-down approach was then used to iteratively develop concrete SDPs and capture the involved elements and relationships in two metamodels.

The SDPDM, which defines the usage aspect of SDPs, addresses the lack of security-relevant information and the often informal inter-pattern relationships found in existing patterns. It includes critical security considerations for both data and behavior, explicitly models inter-pattern relationships, and enables the integration of practical examples. By adapting the viewpoint and view concept, the SDPDM establishes clear boundaries between different aspects of an SDP.

The SDP KB Metamodel enables the use of a CBRS to recommend appropriate SDPs. By decoupling pattern descriptions from the selection process, architects do not need prior knowledge of SDPs to find an appropriate SDP. In addition, this approach allows integrating various kinds of information that can be easily provided by architects, such as applicable security standards, to identify appropriate SDPs.

To evaluate the SDPDM, two OIDC SDPs and one PBA SDP were developed. The first OIDC SDP evaluated the ability of the SDPDM to be used to create practical SDPs. The second OIDC SDP demonstrated the need for multiple SDPs for a security solution by comparing it to the first pattern. The PBA SDP demonstrated that the SDPDM can be applied beyond a single security solution.

To evaluate the SDP KB Metamodel, an OIDC KB and a PBA KB were developed. Both KBs illustrate the attributes defining the knowledge aspect of SDPs, demonstrating

that multiple KBs are required due to the varying characteristics of different security solutions. In addition, the OIDC KB provides a recommendation example that simulates a CBRS that recommends an appropriate OIDC SDP for a synthetic scenario. This demonstrates that the KBs can be effectively used by a CBRS.

These application examples show that the SDPDM can be used to create practical SDPs, highlight the need for multiple SDPs for a single security solution, and demonstrate that the SDP KB Metamodel can effectively guide a CBRS in recommending appropriate SDPs.

In conclusion, the SDPDM appears promising in providing architects with the information required for secure implementation, and the SDP KB Metamodel can enhance how architects select appropriate SDPs. Further research is needed to explore their practical use for architects and to determine whether these concepts are generally applicable beyond authentication.

10.2. Future Work

In this section, we present possible future research directions to extend and further evaluate the results of this thesis.

10.2.1. Further Evaluation

The SDPs and KBs proposed in this thesis have demonstrated the applicability of the SDPDM and SDP KB Metamodel, but further evaluation is needed to assess their practical usefulness and soundness. In addition, to be recognized as a full SDP, an SDP must have concrete implementations and clearly support architects in implementing secure systems.

Future work could analyze OSS to identify concrete implementations of the SDPs, thus meeting the first requirement for them to be considered full patterns. Such analysis could also reveal issues in the SDPs, identify vulnerabilities in the analyzed OSS, or indicate more fundamental problems in the SDPDM. Potential OSS for evaluating the OIDC SDPs include the *openDesk Project*[1], the *EUdi-Wallet Project*[2], and two experimental prototypes[3,4]. For the PBA SDP, we have not yet identified any suitable OSS.

Another area of future work is to identify relevant security threats for the considered security solutions and evaluate how effectively the SDPs mitigate these threats. This may reveal security flaws in the SDPs and, if so, how well they can be addressed with the SDPDM. For the OIDC SDPs, the studies of Fett et al. [FKS17], Navas and Beltran [NB19], and Mainka et al. [Mai+17] are valuable, as they analyze the security of OIDC and identify potential security threats.

[1] `https://gitlab.opencode.de/bmi/opendesk`
[2] `https://gitlab.opencode.de/bmi/eudi-wallet`
[3] `https://github.com/michaelvl/oidc-oauth2-bff`
[4] `https://github.com/michaelvl/oidc-bff-apigw-workshop`

Future work could conduct case studies to determine whether the SDPs support the implementation of secure systems, thus meeting the second requirement for them to be considered full patterns. For this, the evaluation method by Yskout et al. [YSJ15] could be adapted. They conducted a study with student groups required to implement security requirements in a system, with only some groups having access to security patterns. This approach could be extended by giving certain groups access to a CBRS that uses the proposed SDP KBs, allowing for an assessment of how useful a CBRS is in selecting appropriate SDPs. However, implementing the CBRS and developing additional SDPs would be necessary to ensure meaningful evaluation results.

In addition, future work can explore the inclusion of additional OIDC SDPs and PBA SDPs to evaluate whether the identified SDPs in the KBs are both necessary and distinct. Given the potential for a large number of SDPs, strategies to minimize the manual effort required for their development and maintenance can be investigated. One possible approach is the introduction of SDP templates, which could enable the automatic generation of specific SDPs based on the recommended attribute values.

Finally, since the SDPs in this thesis focus on authentication, future work may explore SDPs for other security objectives to evaluate the general applicability of the concepts. Authorization appears particularly promising, as it involves multiple security solutions and typically depends on authentication. This means that sufficient information should be available and the integration with the presented SDPs can be explored.

10.2.2. Security Design Pattern Tool Support

Future work could implement a tool that utilizes interactive elements within Behavioral Model allowing an architect to click on an element to immediately access the necessary information. This addresses a current limitation where architects must manually look up the Data Elements and Events referenced in a Behavioral Model.

Furthermore, a tool could use the mapping between Roles and *Architectural Modeling Elements (AMEs)* to improve the visualization of an architecture, building on Sabau's concepts [Sab24]. For example, architects could select a Role to see the AMEs that implement it. Similarly, selecting a relationship between two Roles could highlight their interactions in an *Architectural Model (AM)*.

Sabau's concepts focus on filtering an traceability of AMs and AMEs based on the security requirements they fulfill [Sab24]. When an architect provides security requirements to a recommender system, the Roles of the selected SDP could automatically be assigned these requirements. This would allow features such as filtering or highlighting AMEs based on the security requirement they fulfill.

Finally, a tool could facilitate the automatic combination of multiple SDPs, which is especially useful if multiple SDPs are recommended. By providing a single pattern that integrates all these SDPs, architects do not need to switch between them, greatly enhancing understandability. However, this requires further research, as automatically combining SDPs is more challenging than manual approaches. Potential challenges include merging common Roles, such as all Roles responsible for managing users, to a single Role.

A. Appendix

A.1. OpenID Connect Knowledge Base

For the "OIDC Flow" Attribute, see Section 8.1.1.
For the "Identity Provider(s) Used" Attribute, see Section 8.1.2.
For the "Token Type for Subsequent Requests" Attribute, see Section 8.1.4.

A.1.1. Allow Fallback IdPs for Users

Yes: A user can associate multiple IdPs with their account and authenticate through any of these IdPs.

- (+) *Reliability - Availability*: Users can authenticate to the application as long as at least one of their associated IdPs is available.
- (-) *Maintainability*: A one-to-many relationship between a user and the IdPs complicates user management. For example, synchronizing user information can be challenging when IdPs provide different user information or format it differently.

No: If the IdP associated with a user is unavailable, the user cannot authenticate to the application.

- (+) *Maintainability*: A one-to-one relationship between a user and IdP simplifies user management.
- (-) *Reliability - Availability*: If the IdP associated with a user is unavailable, the user cannot authenticate to the application.

Attribute Constraints

Allow Fallback IdPs for Users = *Yes* $\Rightarrow$ *IdP(s) Used* = *Mult. IdPs per User Supported*

A.1.2. Client Authentication Method

Client Secret Basic: A shared secret is used to authenticate the RP to the IdP and is sent with the token requests.

- (0) *Security - Confidentiality*: Provides easy and secure authentication when using HTTPS. However, may be vulnerable to eavesdropping if not properly secured.
- (0) *Maintainability*: Minimal overhead as it simply adds an authorization header.
- (+) *Performance*: Minimal overhead as it simply adds an authorization header.

Private Key JWT: A JWT signed by the RP using a private key is sent with the token request, allowing the IdP to validate authenticity using the RP's public key.

- (++) *Security - Confidentiality, Integrity*: Uses asymmetric encryption for JWT signing, eliminating the need for shared secrets.
- (0) *Performance*: Asymmetric encryption adds some processing overhead compared to symmetric methods.
- (-) *Maintainability*: It requires the creation and signing of JWT tokens. It also requires proper handling of private and public keys.

Attribute Constraints

$$\textit{Client Authentication Method} = \textit{Private Key JWT} \Rightarrow \ \textit{TTSR} = \textit{Internal Token}$$

A.2. PBA Knowledge Base

For the "Password Reset Mechanism" Attribute, see Section 8.3.1.
For the "Password Rotation with History" Attribute, see Section 8.3.2.

A.2.1. Password Peppering

HMAC: Compute and store a *Hash-based Message Authentication Code (HMAC)* from the hash value of the salted password. The resulting hash is not reversible, as HMAC is a one-way function.

- (++) *Security*: Adds a robust layer of security to hashed passwords, making it more difficult for attackers to reverse-engineer passwords, even if they have access to the hashed password and salt.
- (-) *Performance*: HMAC adds additional processing, which may impact system performance, especially under high load.
- (--) *Maintainability*: Require secure HMAC key generation, storage, and rotation to maintain security. In particular, key rotation is challenging because the HMAC is not reversible, i.e., it is a one-way function.

Symmetric Encryption: Compute and store a symmetric ciphertext of the hash value of the salted password.

- (+) *Security*: Adds a robust layer of security to hashed passwords, making it difficult for attackers to reverse-engineer passwords even if they have access to the hashed password and salt. However, if the key is compromised, an attacker can simply decrypt the passwords.
- (-) *Maintainability*: Requires secure management of encryption keys, including secure storage and rotation. However, key rotation is quite simple, as the encrypted hashed passwords can simply be decrypted with the old key and re-encrypted with the new key.

No: Does not implement pepper, i.e. not adding a secret value to the password hash before storing it.

- (+) *Maintainability*: No additional components are required, making it easier to apply changes and test the system.

- (+) *Performance*: Faster due to fewer cryptographic operations required during password hashing.
- (-) *Security*: If an attacker has access to hashed user passwords and salts, they can attempt to brute force the password. However, with a proper password policy in place and the use of slow, memory-intensive hashing algorithms, this will also be difficult.

Default Value - No: Implementing password peppering using HMAC or symmetric encryption significantly increases system complexity by requiring management of a secret key and proper implementation of key rotation. In addition, with a proper password policy and the use of proven password hash functions, even if password hashes are leaked, it is quite difficult for an attacker to brute force passwords. Therefore, password peppering should only be used in systems where security is critical and sufficient resources can be devoted to properly implementing password peppering.

A.2.2. User Account Lockout

Yes: Locks the user account after a predefined number of consecutive failed login attempts to prevent unauthorized access and mitigate brute force attacks.

- (++) *Security*: Limits the number of login attempts, reducing the feasibility of brute force or credential stuffing attacks and ensuring that user accounts remain secure.
- (-) *Reliability - Availability*: Attackers can deliberately trigger account lockouts, effectively denying access to legitimate users.

No: Does not implement account lockout policies, allowing users unlimited login attempts.

- (+) *Reliability - Attackers* cannot deliberately trigger account lockouts, effectively denying access to legitimate users.
- (-) *Security*: Accounts are more vulnerable to brute force or credential stuffing attacks.

A.3. OpenID Connect SDP 1

A.3.1. Summary

This OpenID Connect pattern allows applications, especially within organizations, to rely on an Identity Provider (IdP) to securely handle user logins. Instead of managing usernames, passwords, and other authentication details internally, the application delegates this responsibility to the identity provider.

When a user needs to log in, the application redirects the user to the trusted IdP. The IdP authenticates the user, often using credentials the user already has within the organization. Upon successful authentication, the IdP returns an authorization code, which the system then exchanges for an identity token. This identity token contains user claims, such as the user's unique identifier and email, and is signed by the IdP to ensure its integrity and authenticity.

The identity token is also used to authenticate subsequent user requests, simplifying the overall design and allowing for faster development. However, this approach should only be used when both the application and the IdP operate within a protected environment (e.g., accessible only from within an organization's VPN) to maintain security. For applications that operate in a broader environment, we recommend using internal tokens.

For example, an enterprise application on a corporate network can use the organization's existing IdP to authenticate employees, enabling easy onboarding, offboarding, and access to applications.

A.3.2. Context

- The system must restrict access to protected resources or actions, making them available only to authenticated users.
- Users interact with the system from clients equipped with browser and input capabilities (e.g., mobile devices) to access protected resources or perform protected actions.
- Communication between users, the system, and the IdP takes place within a protected environment. A protected environment refers to an environment that is not publicly accessible and that entities must authenticate to access.
- The system uses a single IdP that implements the *OpenID Connect Core Specification* to provide basic OIDC functionality, and the *OpenID Connect Discovery Specification* to allow dynamic retrieval of the IdP's configuration.

A.3.3. Problem

- Malicious attackers may impersonate legitimate users to gain unauthorized access to protected resources or perform actions on their behalf.
- Requiring separate authentication for multiple applications degrades the user experience and increases the likelihood of weak passwords and password reuse.
- Managing separate user accounts for different applications complicates user onboarding and offboarding within organizations.

A.3.4. Conceptual View

Diagram: Figure 6.2

Policy (OIDC Authentication)**.** During the authentication process, only the user who initiated the authentication at the RP and successfully authenticated to the expected IdP is allowed to receive and use an identity token. The RP must verify that the identity token received from the IdP is in the expected format, has a valid signature, and contains valid claims, such as that the token has not expired. In addition, *State*, *Nonce*, and *Code Verifier* values are sent in the authentication request and must be validated to protect against attacks.

- The *State* is echoed by the IdP to ensure that the response matches the original authentication request, mitigating CSRF attacks.
- The *Nonce* is included in the identity token to prevent replay attacks by ensuring the token is unique to the authentication request.
- The *Code Verifier* is sent when exchanging the authorization code for tokens, preventing attackers from using stolen authorization codes.

Policy (OIDC Authentication (Login))**.** During the authentication process, only the user who initiated the authentication at the RP and successfully authenticated to the expected IdP is allowed to receive and use an identity token. The RP must verify that the identity token received from the IdP is in the expected format, has a valid signature, and contains valid claims, such as that the token has not expired.

Role (Subject)**.** An Uncontrolled Role responsible for authenticating to the system to access protected resources. During the authentication, the subject provides credentials to the IdP.
Secret Storage: The session identifier associated with the internally stored state, nonce, and code verifier must be stored. In addition, after successful authentication at the RP, the identity token and refresh tokens must be stored.

Role (System)**.** A Controlled, Abstract Role that must contain some resources that require authentication to access.

Role (Access Manager)**.** A Controlled, Abstract Role responsible for forwarding authentication requests to the appropriate roles and protecting the system from unauthenticated access.
Reference: Single Access Point SSP.

Role (Authentication Manager)**.** A Controlled Role responsible for facilitating the authentication process between the different roles. In addition, it performs validations using the state and nonce.
Session Storage: It stores the state, nonce, and code verifier required for validation after successful authentication at the IdP.
EP[Login]: Only returns the identity and refresh tokens after a successful authentication process.
DP[Login]: Ensures that the state received by the subject received after successful authentication matches the state stored, protecting against CSRF attacks. It also verifies that the nonce in the identity token matches the nonce stored, protecting against replay attacks.

Role (Authentication Request Generator)**.** A Controlled Role responsible for generating the dynamic data required for the authentication request to the IdP and combining this data with IdP configuration data to form a proper authentication request.

Role (Token Exchanger)**.** A Controlled Role responsible for exchanging either an authorization code during initial authentication or refresh token when the identity token

expires for (fresh) tokens.
DP[Login]: Verifies that returned token result is successful.
IP[Login]: Provides the identity, refresh, and access tokens that must be properly validated.

Role (Token Validator)**.** A Controlled Role responsible for verifying the authenticity and integrity of the identity token and, if possible, the access tokens issued by the IdP.
DP[Login, Token]: Verifies the authenticity and integrity of the identity token and, of possible access token during initial authentication and token refresh, and only the identity token when requesting protected resources.

Role (Identity Provider Configuration Manager)**.** A Controlled Role responsible for retrieving the dynamic configuration and JWKS of the IdP. In doing so, the role reduces potential configuration issues because a system operator only needs to provide minimal configuration and the rest is dynamically retrieved from the IdP. It also facilitates easy key rotation in general and when a key is compromised, thereby increasing security.
Cache: The dynamic configuration and JWKS is often needed by other roles, so the data must be cached to ensure availability and performance.
IP[Login,Token]: Provides the configuration required for all interactions with the IdP and the JWKS required to verify the signature of the identity tokens.

Role (Identity Provider)**.** An Uncontrolled Role responsible for authenticating the subject. It maintains a separate user list and implements the OIDC core and discovery specification. The IdP must be trusted, as it can spoof any identity.
DP[Login]: Ensures that the code challenge sent during user authentication matches the code verifier hash calculated during token exchange.
IP[Login,Token]: Provides its dynamic configuration and JWKS.

Role (User)**.** A Controlled Role responsible for creating new local users if none exist, or updating existing user information. Each user has a locally unique identifier and is associated with the IdP identifier.
Persistent Storage: Must store users persistently.

A.3.5. Data View

Diagram: Figure A.1. For Data Rules see Appendix A.5.1.

A.3.6. Behavioral View

For clarity, the “Access Manager” Role is not included in the BMs (i.e., it would just “forward” every message to and from the “Subject” Role).

Initial Authentication

Behavioral Model: Figure A.2. Data Element and Event references: Table A.1.

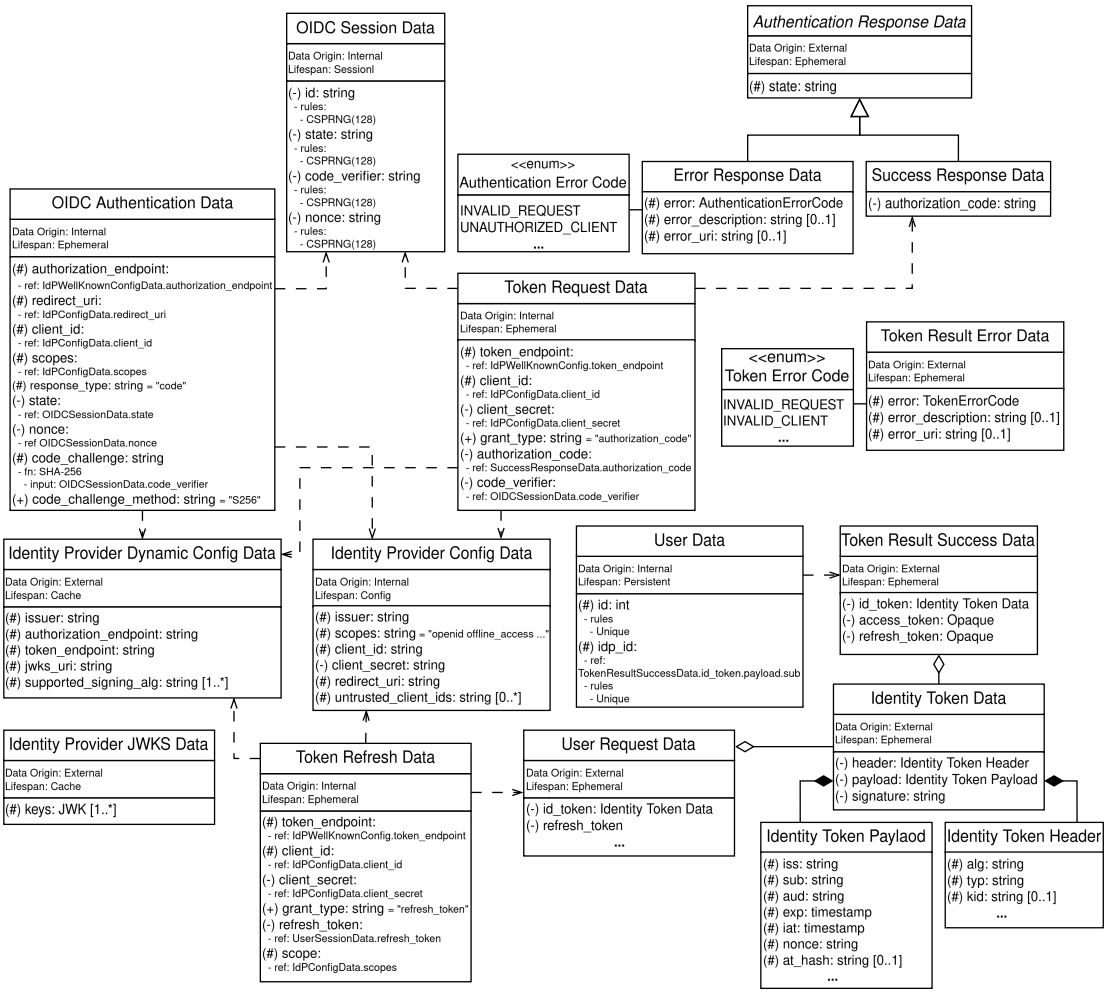


Figure A.1.: OIDC SDP 1 Data View Diagram

Refresh Identity Token

Behavioral Model: Figure A.3. Data Element and Event references: Table A.2.

Verify Identity Token

We recommend not to implement the following validation manually, but to find a suitable library for the programming language used. However, make sure that the library performs all validations; if not, implement the missing validations manually.

Behavioral Model: Figure A.4. Data Element and Event references: Table A.3.

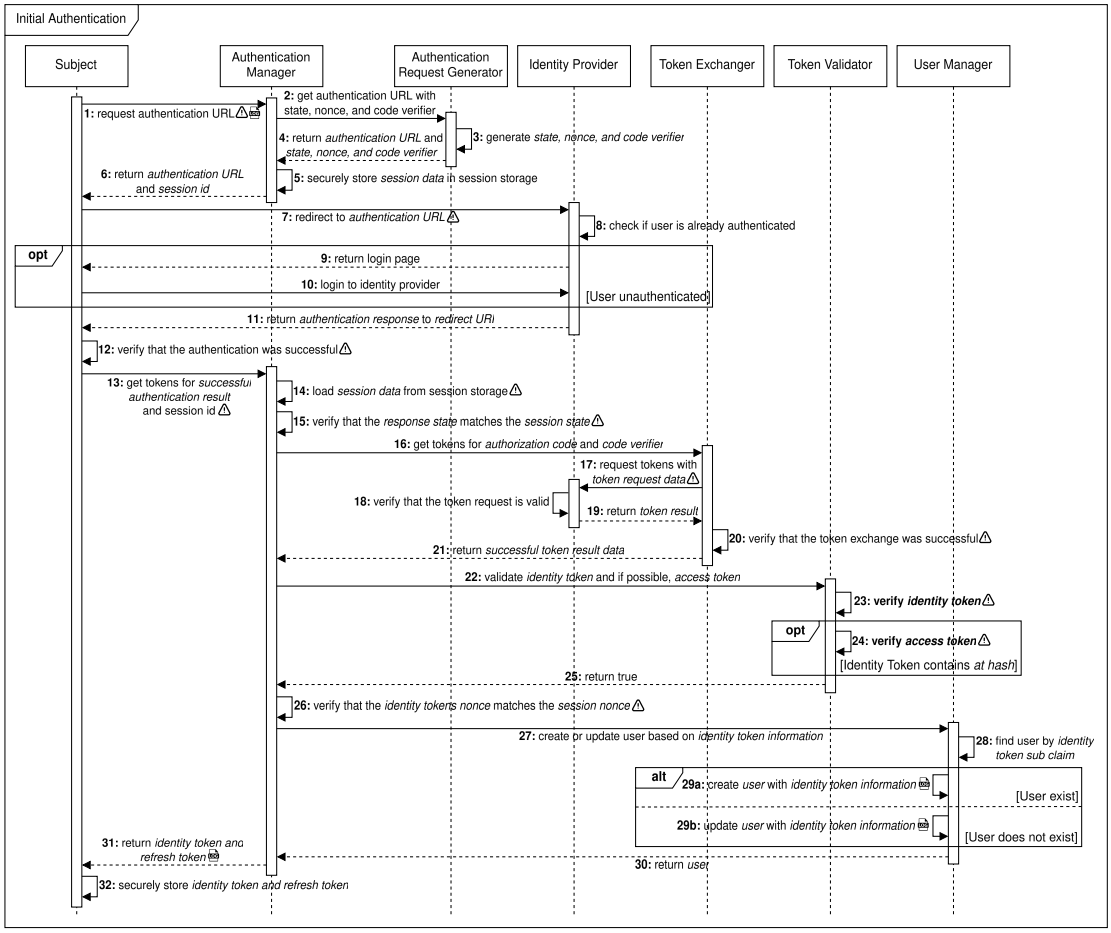


Figure A.2.: OIDC SDP 1 “Initial Authentication Behavior” Behavioral Model

Verify Access Token

We recommend not to implement the following cryptographic operations manually, but to find a suitable library for the programming language used. It is likely that the library chosen to verify the identity tokens can also be used here.

Behavioral Model: Figure A.5. Data Element and Event references: Table A.4.

Get Identity Provider Configuration

Behavioral Model: Figure A.6. Data Element and Event references: Table A.5.

Get Identity Provider JSON Web Key Set

Behavioral Model: Figure A.7. Data Element and Event references: Table A.6.

Step	Data Elements	Events
1	-	System Unavailable, Authentication Initiated
3	OIDCSessionData	-
4	OIDCAuthenticationData, OIDCSessionData	-
5	OIDCSessionData	-
6	OIDCAuthenticationData, OIDCSessionData.id	-
7	OIDCAuthenticationData	Identity Provider Unavailable
11	AuthenticationResponseData, OIDCAuthenticationData.redirect_url	-
12	-	Identity Provider Authentication Failure
13	SuccessResponseData	System Unavailable
14	OIDCSessionData	OIDC Session Missing
15	SuccessResponseData.state, OIDCSessionData.state	Response State Mismatch
16	SuccessResponseData.authorization_code, OIDCSessionData.code_verifier	-
17	TokenRequestData	Identity Provider Unavailable
19	TokenResultData	-
20	-	Token Exchange Failure
21	TokenResultSuccessData	-
22	TokenResultSuccessData.id_token, TokenResultSuccessData.access_token	-
26	TokenResultSuccessData.id_token.payload.nonce, OIDCSessionData.nonce	Nonce Mismatch
27	TokenResultSuccessData.id_token.payload	-
28	TokenResultSuccessData.id_token.payload.sub	-
29a	TokenResultSuccessData.id_token.payload	User Created
29b	TokenResultSuccessData.id_token.payload	User Updated
30	UserData	-
31	TokenResultSuccessData.id_token, TokenResultSuccessData.refresh_token	Successful Authentication
32	TokenResultSuccessData.id_token, TokenResultSuccessData.refresh_token	-

Table A.1.: Data Element and Event references for Figure A.2

Events

Event (Authentication Initiated)**.** A user initiates the authentication process by requesting to log in.
Security Consideration: An authentication request from an unexpected client (e.g., IP address is outside the protected environment) should be blocked.

Event (Successful Authentication)**.** A user successfully authenticated and obtains an identity token.

Event (User Created)**.** The user corresponding to the identity token has been successfully created.

Event (User Information Updated)**.** The user information were updated.

Error Event (System Unavailable)**.** The system is unresponsive or unavailable when a user attempts an action. Possible causes include server downtime, network issues with the protected environment, or infrastructure failures.
Fallback Behavior: Retry with Exponential Backoff with at most three retries and an initial delay of two seconds.

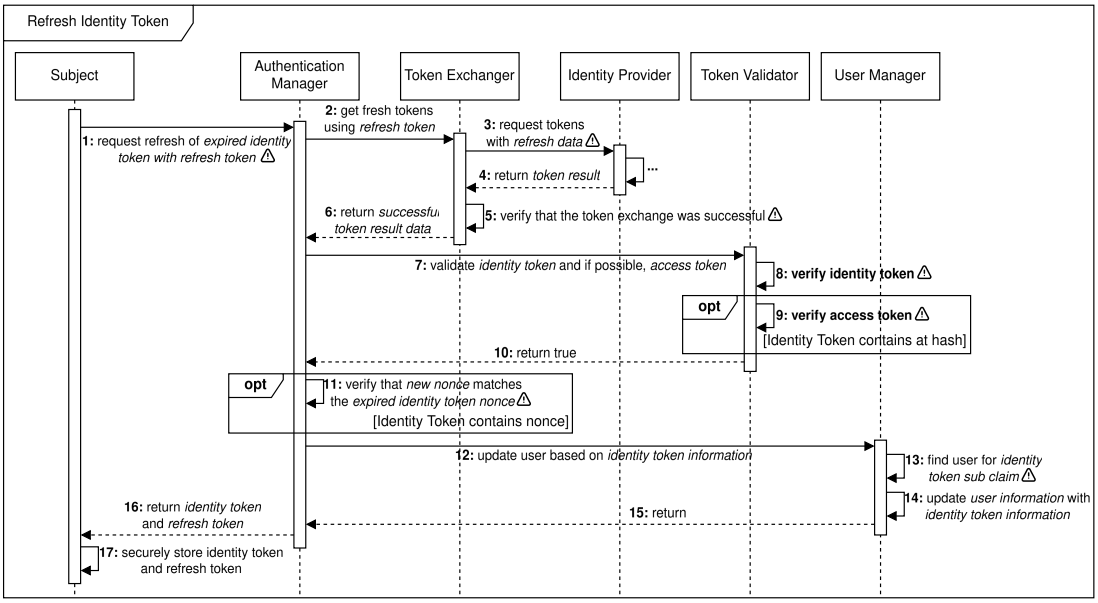


Figure A.3.: OIDC SDP 1 “Refresh Identity Token” Behavioral Model

Step	Data Elements	Events
1	`UserRequestData`	`System Unavailable`
2	`UserRequestData.refresh_token`	-
3	`TokenRefreshData`	`Identity Provider Unavailable`
4	`TokenResultData`	-
5	-	`Token Refresh Failure`
6	*TokenResultSuccessData*	-
7	`TokenResultSuccessData.id_token`, `TokenResultSuccessData.access_token`	-
11	`TokenResultSuccessData.id_token.payload.nonce`, `UserRequestData.id_token.payload.nonce`	`Nonce Mismatch`
12	`TokenResultSuccessData.id_token.payload`	-
13	`TokenResultSuccessData.id_token.payload.sub`	`User Not Found`
14	`UserData`, `TokenResultSuccessData.id_token.payload`	
16	`TokenResultSuccessData.id_token`, `TokenResultSuccessData.refresh_token`	-
17	`TokenResultSuccessData.id_token`, `TokenResultSuccessData.refresh_token`	-

Table A.2.: Data Element and Event references for Figure A.3

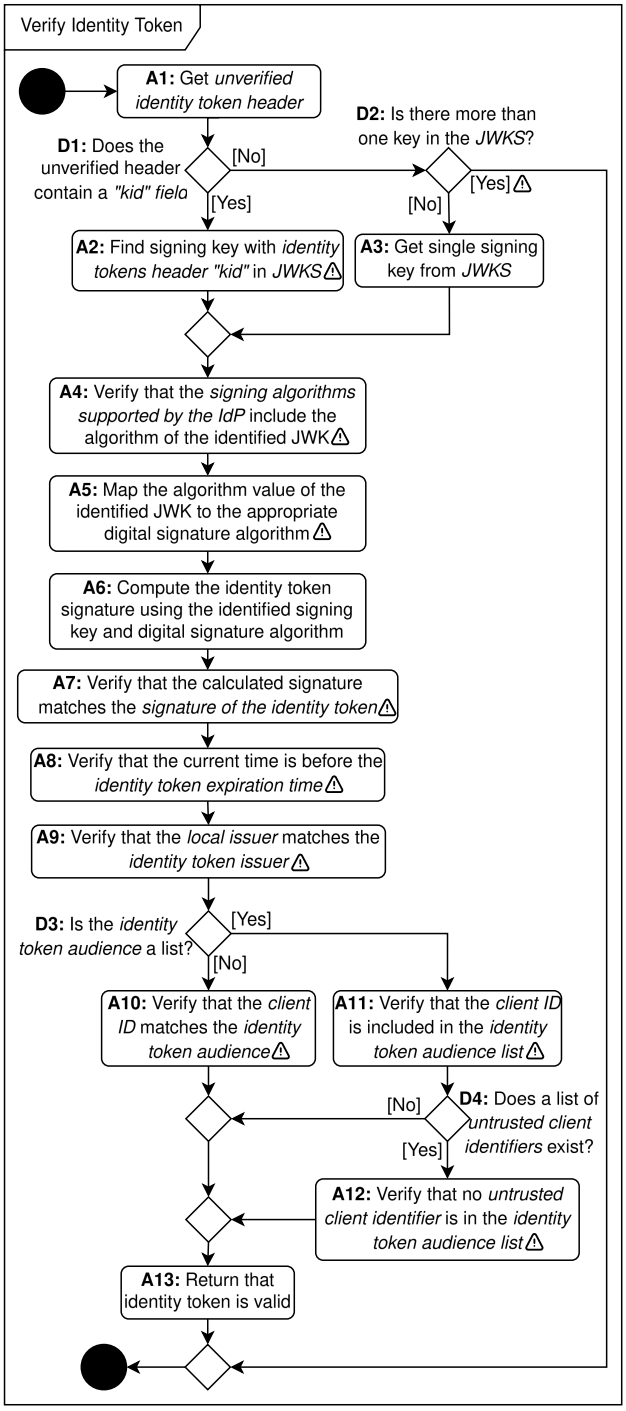


Figure A.4.: OIDC SDP 1 “Verify Identity Token” Behavioral Model

Step	Data Elements	Events
A1	IdentityTokenHeader	-
D1	IdentityTokenHeader.kid	-
D2	IdentityProviderJWKSData.keys	Ambiguous Signing Key
A2	IdentityTokenHeader.kid, IdentityProviderJWKSData.keys	Key ID Not Found in JWKS
A3	IdentityProviderJWKSData.keys	-
A4	IdentityProviderDynamicConfigData.supported_signing_algs	Unsupported Signing Algorithm
A5	-	Unknown Cryptographic Algorithm
A7	IdentityTokenData.signature	Invalid Token Signature
A8	IdentityTokenPayload.exp	Identity Token Expired
A9	IdentityTokenPayload.iss, IdentityProviderConfig.client_id	Invalid Issuer
D3	IdentityTokenPayload.aud	-
A10	IdentityProviderConfigData.client_id, IdentityTokenPayload.aud	Invalid Audience
A11	IdentityProviderConfigData.client_id, IdentityTokenPayload.aud	Invalid Audience
D4	IdentityProviderConfigData.untrusted_client_ids	-
A12	IdentityProviderConfigData.untrusted_client_ids, IdentityTokenPayload.aud	Untrusted Client Audience

Table A.3.: Data Element and Event references for Figure A.4

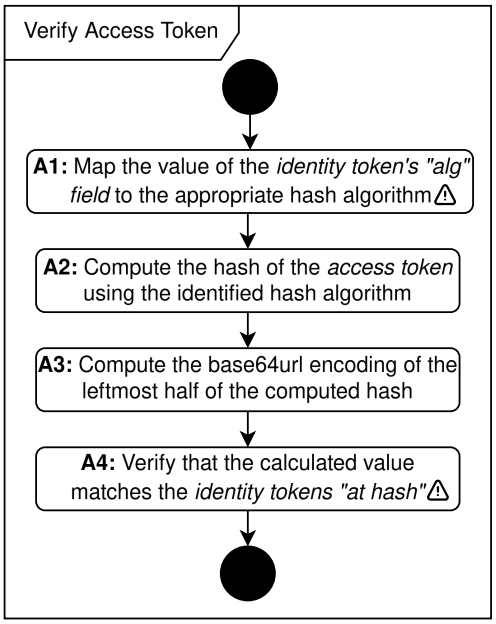


Figure A.5.: OIDC SDP 1 “Verify Access Token” Behavioral Model

Step	Data Elements	Events
A1	TokenResultSuccessData.id_token.header.alg	Unknown Cryptographic Algorithm
A2	TokenResultSuccessData.access_token	-
A4	TokenResultSuccessData.id_token.payload.at_hash	Invalid Access Token Hash

Table A.4.: Data Element and Event references for Figure A.5

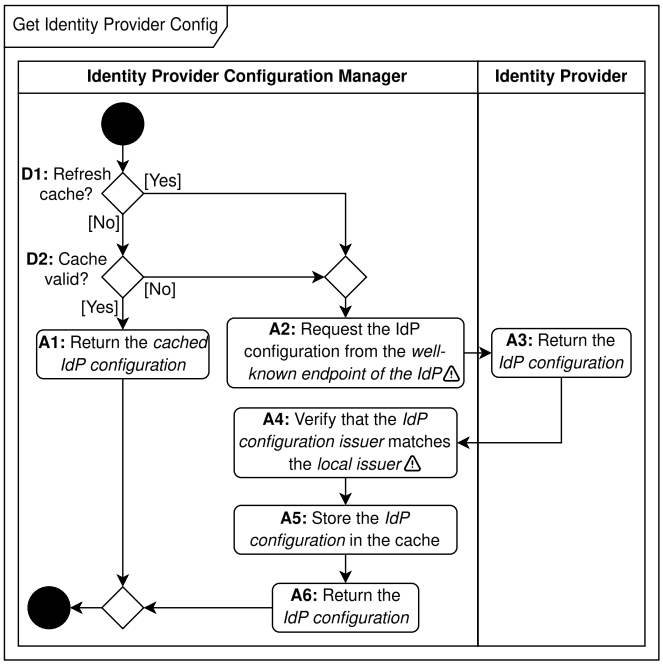


Figure A.6.: OIDC SDP 1 "Get Identity Provider Configuration" Behavioral Model

Step	Data Elements	Events
A1	IdentityProviderDynamicConfigData	-
A2	IdentityProviderConfigData.issuer + '/.well-known/openid-configuration'	Identity Provider Unavailable
A3	IdentityProviderDynamicConfigData	-
A4	IdentityProviderDynamicConfigData.issuer, IdentityProviderConfigData.issuer	Invalid Issuer
A5	IdentityProviderDynamicConfigData	-
A6	IdentityProviderDynamicConfigData	-

Table A.5.: Data Element and Event references for Figure A.6

Step	Data Elements	Events
A1	IdentityProviderJWKSData	-
A2	IdentityProviderDynamicConfig.jwks_uri	Identity Provider Unavailable
A4	IdentityProviderJWKSData	-
A5	IdentityProviderJWKSData	-
A6	IdentityProviderJWKSData	-

Table A.6.: Data Element and Event references for Figure A.7

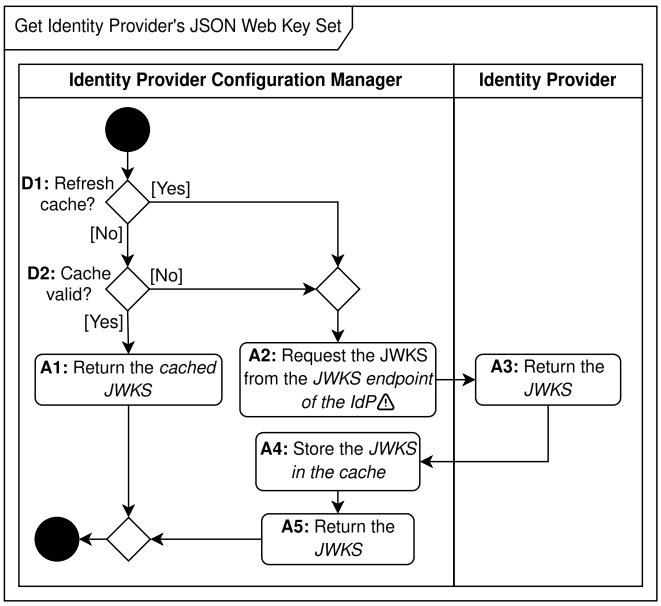


Figure A.7.: OIDC SDP 1 "Get Identity Provider JSON Web Key Set" Behavioral Model

Error Event (Identity Provider Unavailable)**.** The IdP is unresponsive or unavailable when a user attempts an action, which means that users can no longer authentication to the system, and authenticated users may lose access when their identity token expire.
Fallback Behavior: Retry with Exponential Backoff with at most three retries and an initial delay of two seconds.

Error Event (OIDC Session Missing)**.** The system could not find the session data corresponding to the client's request. This occurs when a user sends an authorization code but either fails t o send a session identifier or there is no session for the session identifier.

Error Event (Response State Mismatch)**.** The state parameter in the authentication response does not match the stored state.
Security Consideration: Can indicate a CSRF attack attempt, where an attacker tries to trick a subject into exchanging the attacker's authorization code for tokens.

Error Event (Token Exchange Failure)**.** Token exchange process failed between the system and the IdP. Possible causes include the code verifier hash not matching the code challenge sent in the authentication request (indicated by an `invalid_grant error code`), or the client secret used to authenticate the RP to the IdP is incorrect (indicated by an `invalid_client error code`). For a complete list of possible causes and resulting errors, see the Token Error Response.
Security Consideration: A mismatched code verifier can indicate a code interception attack, where an attacker intercepts an authorization code and attempts to exchange it for an identity token.

Error Event (Token Refresh Failure)**.** Token refresh failed between the system and IdP. Possible causes include the client secret used to authenticate the SP to the IdP being incorrect (indicated by an `invalid_client error code`), or the refresh token associated with the user being invalid, expired, or granted to another client (indicated by an `invalid_grant error code`). For a complete list of possible causes and resulting errors, see the Token Error Response.

Error Event (Nonce Mismatch)**.** The nonce in the identity token does not match the none stored in the session.
Security Considerations:

- A nonce mismatch may indicate a replay attack, where an attacker replays a previously captured token response.
- A nonce mismatch may indicate a token substitution attack, where an attacker was able to replace the identity token in the response with a different token.

Error Event (Ambiguous Signing Key)**.** Identity token lacks a "kid" and multiple keys are present in the JWKS, i.e., it is unclear which signing key should be used to compute the signature of the identity token.
Fallback Behavior: Refresh JWKS cache and retry once.

Error Event (Key ID Not Found In JWKS)**.** The "kid" in the identity token header does not match any key in the JWKS.
Fallback Behavior: Refresh JWKS cache and retry once.

Error Event (Unsupported Signing Algorithm)**.** The algorithm used in the identity token is not one of those supported by the IdP.
Fallback Behavior: Refresh IdP Configuration cache and retry once.

Error Event (Unknown Cryptographic Algorithm)**.** The identity token uses an unknown or unsupported cryptographic algorithm.

Error Event (Invalid Token Signature)**.** The signature of the identity token does not match the computed signature.
Security Consideration: An invalid token signature can indicate that an attacker has tampered with the identity token.
Fallback Behavior: If the token header does not contain a "kid" and the JWKS contains only a single public key, refresh the JWKS cache and retry once.

Error Event (Identity Token Expired)**.** The identity token has expired.
Fallback Behavior: Obtain fresh token(s) using refresh token.

Error Event (Invalid Issuer)**.** The issuer claim in the identity token or dynamic configuration does not match the expected local issuer.

Error Event (Invalid Audience)**.** The audience claim in the identity token does not match or include the system's client identifier.
Security Consideration: The identity token may be issued by the IdP but for another party.

Error Event (Untrusted Client Audience)**.** The audience claim in the identity token contains at least one identifier that the system does not trust.

Error Event (User Not Found)**.** The user corresponding to the identity token's "sub" claim is not found in the local users.

Error Event (Invalid Access Token Hash)**.** The access token hash in the identity token does not match the computed hash of the access token.

A.3.7. Structural View

Structural Model: Figure 6.7.

A.3.8. Other

Attributes

- OIDC Flow: Authorization Code Flow with PKCE
- Identity Provider(s) Used: Single IdP
- Allow Fallback IdP per User: No
- Token Type for Subsequent Requests: Identity Token
- Client Authentication Method: Client Secret Basic

A.4. OpenID Connect SDP 2

This is only a partial description of the OIDC SDP 2 with its changes from the OIDC SDP 1. Please refer to the OIDC SDP 1 pattern description (Appendix A.3) and the discussed changes for more information (Section 6.2).

A.4.1. Summary

This OpenID Connect pattern allows applications, especially within organizations, to rely on an Identity Provider (IdP) to securely handle user logins. Instead of managing usernames, passwords, and other authentication details internally, the application delegates this responsibility to the identity provider.

When a user needs to log in, the application redirects the user to the trusted IdP. The IdP authenticates the user, often using credentials the user already has within the organization. Upon successful authentication, the IdP returns an authorization code, which the system then exchanges for an identity token. This identity token contains user claims, such as the user's unique identifier and email, and is signed by the IdP to ensure its integrity and authenticity. After the identity token has been successfully verified an internal token such as an stateless JWT or stateful session identifier is issued and can be used to authenticate subsequent requests.

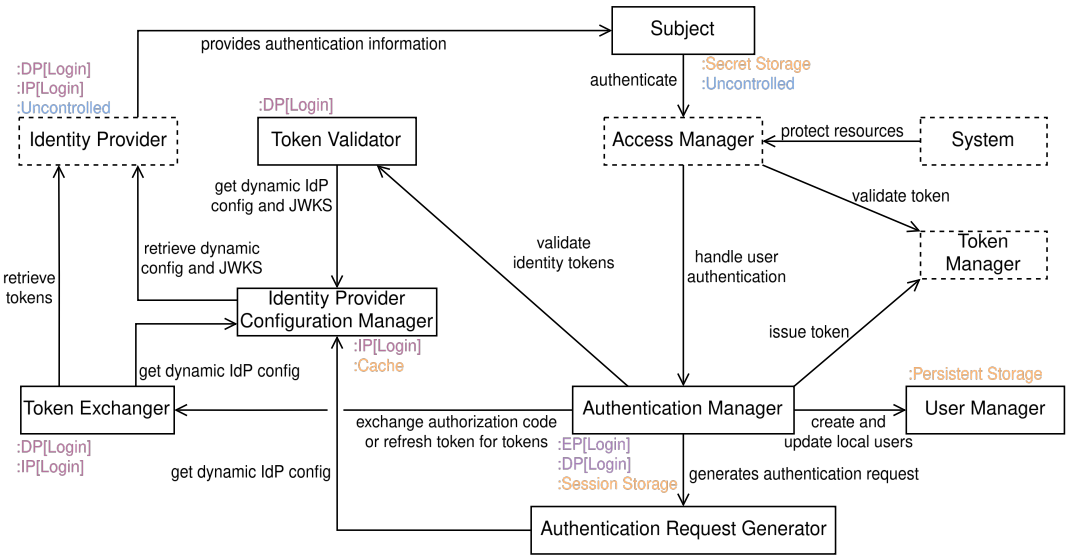


Figure A.8.: OIDC SDP 2 Conceptual View Diagram

A.4.2. Context

- The system must restrict access to protected resources or actions, making them available only to authenticated users.
- Users interact with the system from clients equipped with browser and input capabilities (e.g., mobile devices) to access protected resources or perform protected actions.
- The system uses a single IdP that implements the *OpenID Connect Core Specification* to provide basic OIDC functionality, and the *OpenID Connect Discovery Specification* to allow dynamic retrieval of the IdP's configuration.

A.4.3. Problem

- Malicious attackers may impersonate legitimate users to gain unauthorized access to protected resources or perform actions on their behalf.
- Requiring separate authentication for multiple applications degrades the user experience and increases the likelihood of weak passwords and password reuse.
- Managing separate user accounts for different applications complicates user onboarding and offboarding within organizations.

A.4.4. Conceptual View

Role (Token Manager)**.** A Controlled, Abstract Role that issues, and verifies tokens.
Reference: Token-based Authentication ASSP.

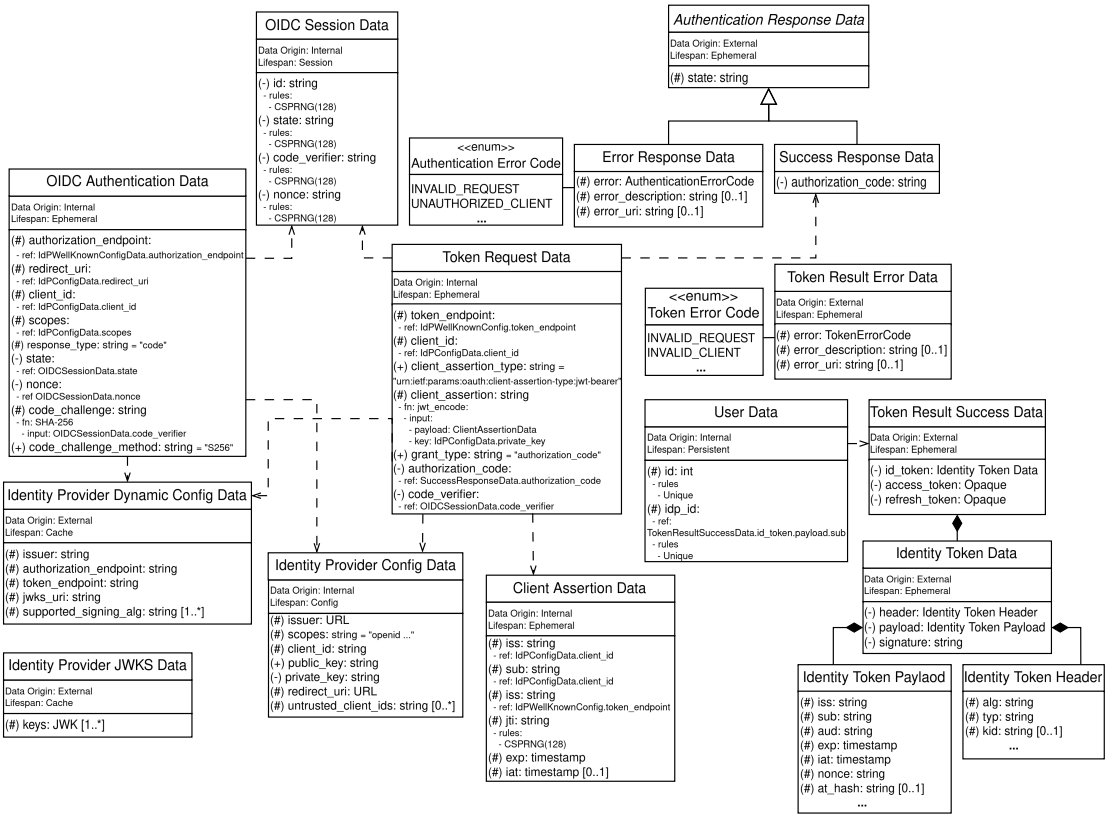


Figure A.9.: OIDC SDP 2 Data View Diagram

A.4.5. Data View

jwt_encode:

```
import jwt
from cryptography.hazmat.primitives import serialization

# b"-----BEGIN PRIVATE KEY-----XXXXX-----END PRIVATE KEY-----"
pkcs8 = load_private_key_pem()

private_key = serialization.load_pem_private_key(pkcs8,
    password=None)

client_assertion = jwt.encode(
    payload, private_key, algorithm="RS256"
)
```

Behavioral View

Behavioral Model: Figure A.2. Changes:

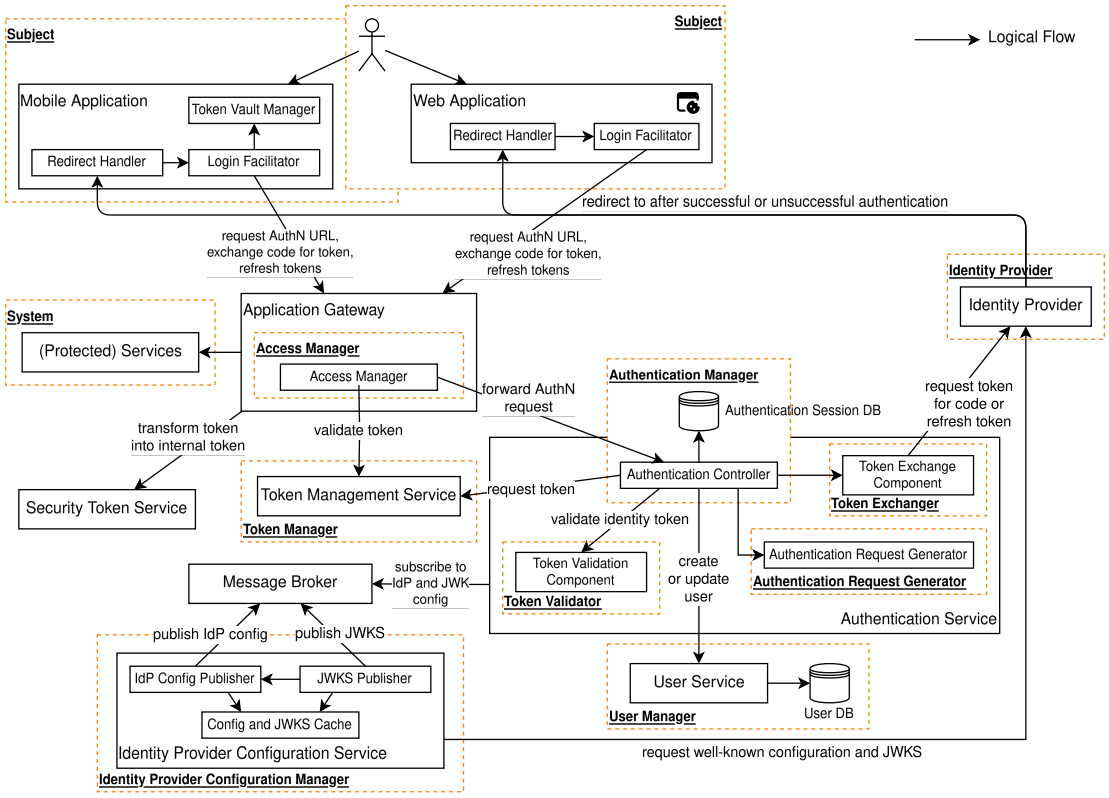


Figure A.10.: OIDC SDP 2 Strucutral View Example Diagram

- Between step 16 and 17: Create and signed client assertion (Data Element: `TokenRequestData.client_assertion`)
- Between step 30 and 31: See Figure 6.9.

Remaining BMs remain the same, except that the “Refresh Identity Token” BM is removed.

Structural View

Architectural Model: Figure A.10.

A.4.6. Consequences

- Strength (Security, Flexibility): The use of an internal tokens allows to limit the information contained to a minimum and enables the use of extended security measures. Furthermore, the RP authenticates to the IdP using asymmetric encryption, so no shared secret is required.

- Weakness (Maintainability): Using an internal token increase system complexity because it is responsible for token management, including issuance, storage, and

invalidation.

A.4.7. Other

Attributes

- OIDC Flow: Authorization Code Flow with PKCE
- Identity Provider(s) Used: Single IdP
- Allow Fallback IdP per User: No
- Token Type for Subsequent Requests: Internal Token
- Client Authentication Method: Private Key JWT

A.5. Single-Factor Password-based Authentication SDP

Summary

This Single-Factor Password-based Authentication pattern verifies the user's identity based on the user's email address and a secret password.

To gain access to the system, the user must first register by providing an email address and password. After confirming the email address, the user logs in by entering their email address and password, which the system checks against the stored information to grant or deny access.

If the user forgets the password, the system can send a password reset link to the registered email address. After confirming access to this email by clicking on the link in the email, the user can change their password.

If the user needs to change the registered email address, they must provide their current password and confirm access to both the old and new email addresses. Confirming the old email address is necessary to ensure that an attacker cannot change the email if the user's password is compromised.

Context

- The system must restrict access to protected resources or actions, making them available only to authenticated users.
- Users interact with the system from clients equipped with an user interface and input capabilities (e.g., mobile devices) to access protected resources or perform protected actions.

Problem

- Malicious attackers may impersonate legitimate users to gain unauthorized access to protected resources or perform actions on their behalf.

Conceptual View

Common password requirements, based on OWASP Authentication Cheat Sheet, NIST Special Publication 800-63B, and Best practices for password hashing and storage:

- *Minimum Length*: At least 8 characters.
- *(Optional) Maximum Length*: Between 64 and 128 characters to prevent DoS attacks, e.g., Long Password Denial of Service.
- All characters should be allowed including ASCII, spaces, and Unicode
- Block common and previously breached passwords. For example, using the API or password list of "HaveIBeenPwned".

Policy (Registration)**.** Users can register by providing a valid email address that is not already in the system, a strong password, and password confirmation. Upon initial registration, a confirmation link containing a confirmation token is sent to the provided email address. After the user clicks on the link, the user account is activated and the user can log in.

Policy (Login)**.** Login is allowed to a registered user account using the email address and correct password. Upon successful login, an authentication token is issued.

Policy (Password Reset)**.** The user initiates the password reset by providing the email address associated with the registered account. The user must prove access to the email account by clicking a reset link containing a reset token sent to the email address. If the reset token is valid, the user must enter a strong password. Upon successful password reset, all user tokens must be invalidated.

Policy (Email Change)**.** To change the email address, a user must be logged in and provide their user password and new email address. The user must verify access to both the new and old email by clicking confirmation links containing different confirmation tokens that are sent to the email addresses. If the change is unexpected, the user can click a "Block Email Change" link included in both emails to cancel the email change. After a successful email change, all user tokens must be invalidated.

Role (Subject)**.** An Uncontrolled Role that wants to authenticate to the system to access protected resources. During both registration and login, the Subject provides an email address and password. To confirm registration, reset the password, and change the email address, the Subject must prove access to the email address by clicking the token link(s) sent to the address.
Secret Storage: Upon successful authentication, the subject receives a token and must securely store it.

Role (Access Manager)**.** A Controlled, Abstract Role that forwards registration, login, password reset, and email change requests to the appropriate roles and protects the system from unauthenticated access.
Reference: Single Access Point SSP.

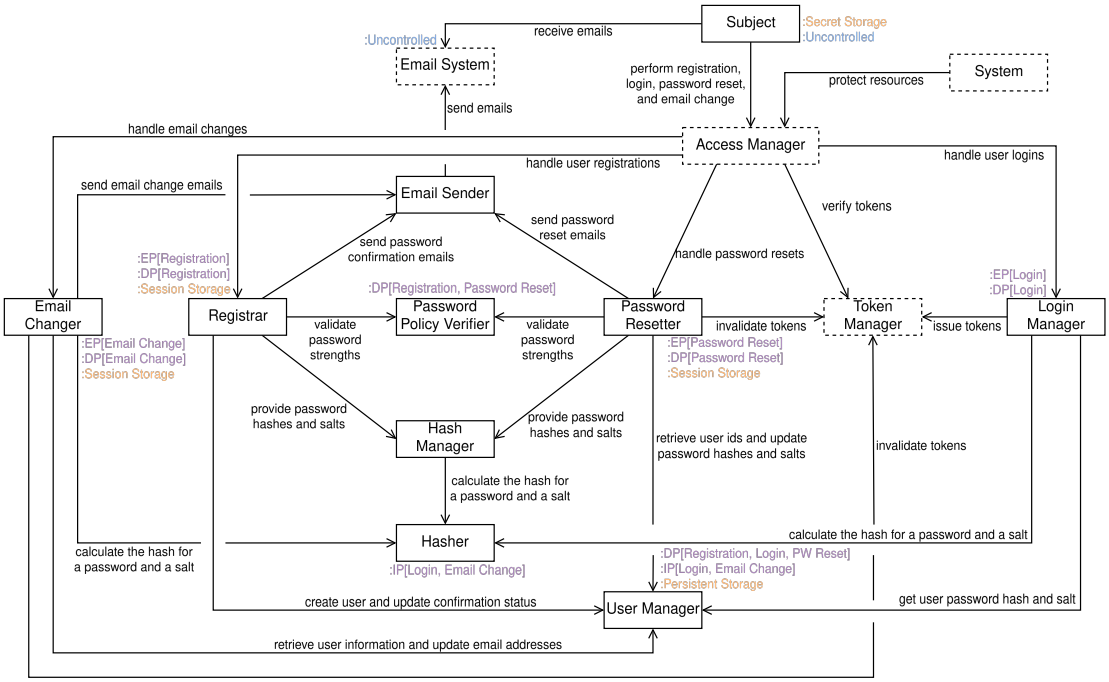


Figure A.11.: PBA SDP Conceptual View Diagram

Role (System)**.** A Controlled, Abstract Role that must contain some resources that require authentication to access.

Role (Registrar)**.** A Controlled Role that handles the registration process. It can optionally issue a token directly after email confirmation to improve the user experience, but can compromise the user if the verification email is intercepted by an attacker.
DP[Registration]: Verifies that the email is in the expected format, that the password and password confirmation values match, and that a received token corresponds to a confirmation session.
Session Storage: Stores the account confirmation tokens until a subject confirms their email address.

Role (Password Policy Verifier)**.** A Controlled Role that verifies that a provided password meets the minimum password requirements.

Role (Hash Manager)**.** A Controlled Role that generates the password salt used by the Hasher to compute the password hash for a new password.

Role (Hasher)**.** A Controlled Role that computes a password hash for a given plaintext password and salt.

Role (User Manager)**.** A Controlled Role that manages the users including their creation, update, and information retrieval.

DP[Registration, Login, PW Reset]: Checks if a user exists for a given email address.
IP[Login, Email Change]: Provides the password hash and salt to verify provided credentials.

Role (Email Sender)**.** A Controlled Role that sends email to users.

Role (Email System)**.** An Uncontrolled, Abstract Role that encapsulates all the components needed to send email to users, including the systems and client email servers.

Role (Password Resetter)**.** A Controlled Role that handles the password reset process.
DP[Password Reset]: Verifies that the email is in the expected format, that the password and password confirmation values match, and that a received token corresponds to a password reset session.
Session Storage: Stores the password reset token until a subject provides the token along with the new password.

Role (Token Manager)**.** A Controlled, Abstract Role that issues, verifies, and invalidates tokens.
Reference: Token-based Authentication ASSP.

Role (Email Changer)**.** A Controlled Role that handles the email change process.
DP[Email Change]: Verifies that the new email is in the expected format, that the password provided is correct, and that the subject has confirmed both the old and new email addresses.
Session Storage: Stores the old and new email confirmation tokens and the email change block token until a subject either confirms both or blocks the change.

Role (Login Manager)**.** A Controlled Role that handles the login process.
DP[Login]: Verifies that the email is in the expected format and that the password provided is correct.

A.5.1. Data View

Diagram: Figure A.12

Data Rule (Email)**.** Ensures data consistency and provides protection against SQL injection attacks.

- Supported Origins: All
- Supported Lifespans: All
- Supported Types: String
- Condition: The “val” follows at least the syntactic rules defined in Email Address Validation - OWASP Cheat Sheet

Data Rule (Unique)**.** Ensures data consistency by ensuring that data entries can be uniquely identified within a data store.

- Supported Origins: All
- Supported Lifespans: Config, Session, Cached, Persistent

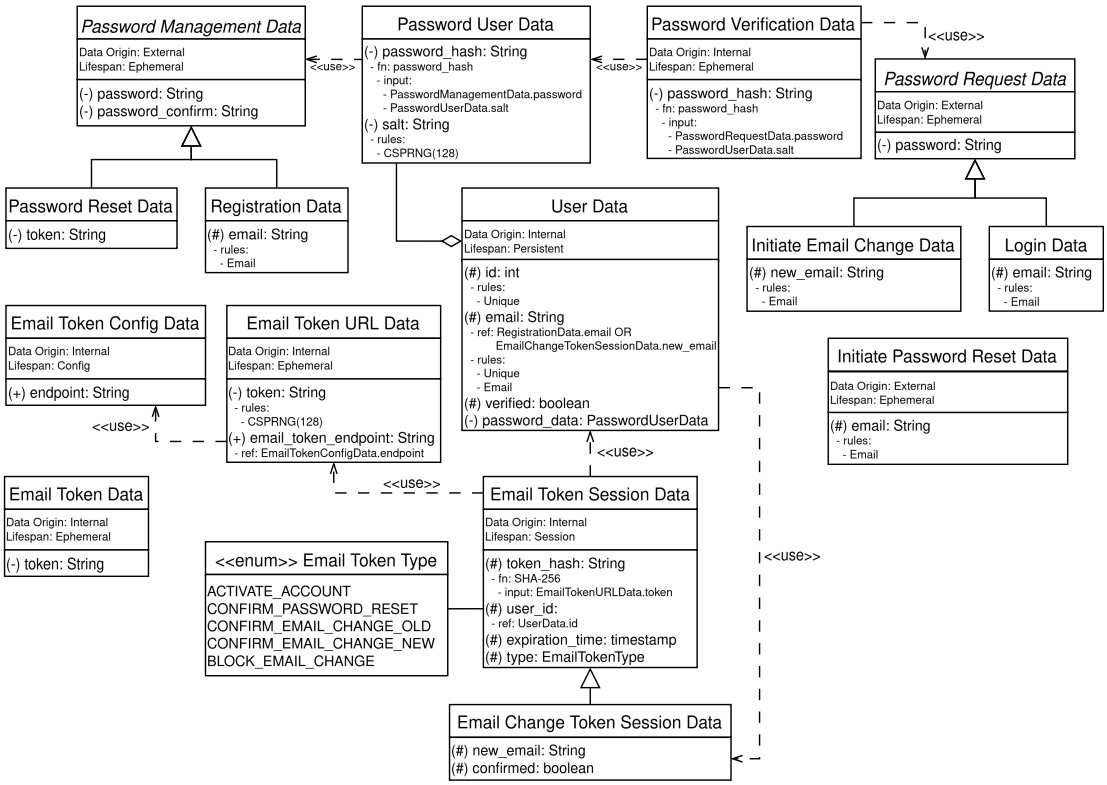


Figure A.12.: PBA SDP Data View Diagram

- Supported Types: String, Number
- Condition: No two data entries with the same "val" are allowed in the same data store.

Data Rule (CSPRNG)**.** The data value must be generated using a *Cryptographically Secure Pseudorandom Number Generator (CSPRNG).*

- Supported Origins: All
- Supported Lifespans: All
- Supported Types: String
- Input Parameters
  - entropy(Integer): Specifies the entropy in bits that the generated value must have. Entropy is a measure of the randomness or unpredictability in an output, indicating how difficult it is to predict the next value.
- Condition: "val" must be generated using a CSPRN that provides "entropy" bits of entropy (e.g., in Python `secrets.token_urlsafe(16)` where 16 bytes corresponds to 128 bits).

SHA-256:

```
import hashlib

input = "..."
sha256_hash = hashlib.sha256(input.encode('utf-8'))
hash_dig = sha256_hash.hexdigest()
```

Password Hash: Generate and verify a password in Python using Argon2. The salt is automatically generated by argon2id and embedded in the output, i.e. the output must be saved exactly as generated in the database.

```
import argon2

ph = argon2.PasswordHasher()

# Generate Password
password_hash = ph.hash("secret password")

# Validate Password
try:
    ph.verify(password_hash, "provided password")
    print("Password is correct")
except:
    print("Password verification failed:")
```

Note: This can be further extended by detailing the various argon2id parameters and adding additional hashing algorithms with a comparison of when to use which algorithm.

A.5.2. Behavioral View

For clarity, the "Access Manager" Role is not included in the BMs (i.e., it would just "forward" every message to and from the "Subject" Role).

Registration

Behavioral Model: Figure A.13. Data Element and Event references: Table A.7.

Login

Behavioral Model: Figure A.14. Data Element and Event references: Table A.8.

Password Reset

Behavioral Model: Figure A.15. Data Element and Event references: Table A.9.

Email Change

Behavioral Model: Figure A.16. Data Element and Event references: Table A.10.

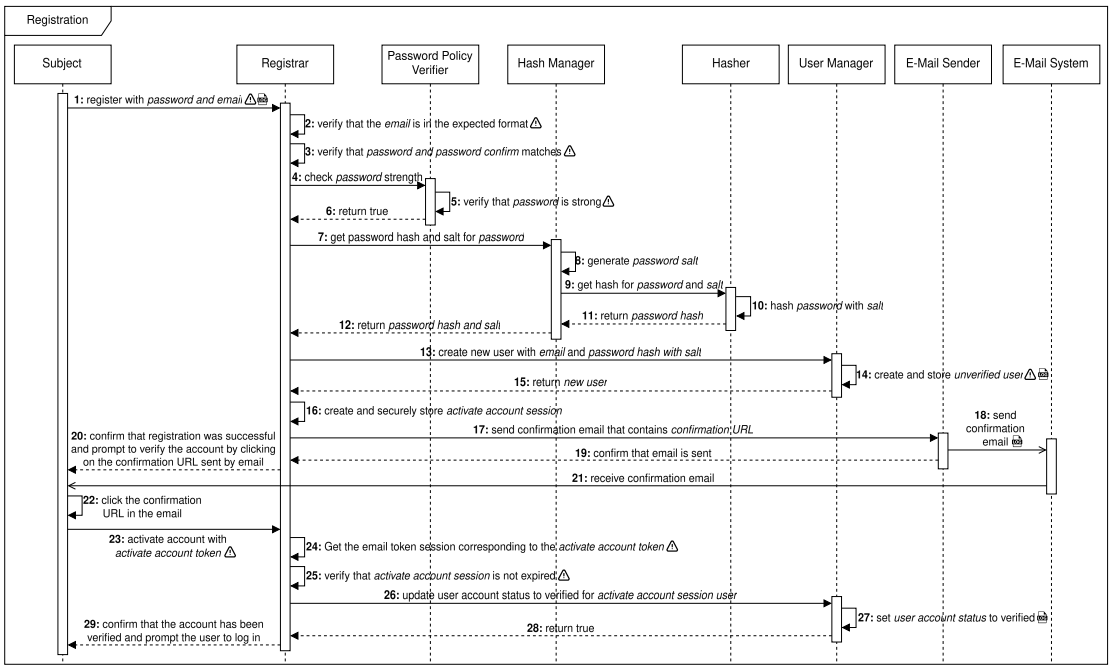


Figure A.13.: PBA SDP "Registration" Behavioral Model

Step	Data Elements	Events
1	RegistrationData	Registration Initiated, System Unavailable
2	RegistrationData.email	Unexpected Email Format
3	RegistrationData.password, RegistrationData.password_confirm	Password Confirm Mismatch
4	RegistrationData.password	-
5	RegistrationData.password	Password Requirement Violation
7	RegistrationData.password	-
8	PasswordUserData.salt	-
9	RegistrationData.password, PasswordUserData.salt	-
10	RegistrationData.password, PasswordUserData.salt	-
11	PasswordUserData.password_hash	-
12	PasswordUserData	-
13	RegistrationData.email, *PasswordUserData*	-
14	UserData	User Already Exist, User Created
15	UserData	-
16	EmailTokenSessionData	-
17	EmailTokenURLData	-
18	-	Email Send
23	EmailTokenData	System Unavailable
24	EmailTokenData.token	Email Token Unknown
25	EmailTokenSessionData.expiration_time	Email Token Expired
26	EmailTokenSessionData.user_id	-
28	UserData.verified	User Email Verified

Table A.7.: Data Element and Event references for Figure A.13

Step	Data Elements	Events
1	LoginData	Login Initiated, System Unavailable
2	LoginData.email	Unexpected Email Format
3	LoginData.email	-
4	LoginData.email	User Not Found
5	UserData	-
6	LoginData.password, PasswordUserData.salt	-
7	LoginData.password, PasswordUserData.salt	-
8	PasswordVerificationData	-
9	PasswordVerificationData.password_hash, PasswordUserData.password_hash	User Password Wrong
12a	-	Login Successful
13a	-	Login Successful

Table A.8.: Data Element and Event references for Figure A.14

Step	Data Elements	Events
1	InitiatePasswordResetData	Password Reset Initiated, System Unavailable
2	InitiatePasswordResetData.email	Unexpected Email Format
3	InitiatePasswordResetData.email	-
4	InitiatePasswordResetData.email	User Not Found
5	UserData	-
6	EmailTokenSessionData	-
7	EmailTokenURLData	-
8	-	Email Send
13	PasswordResetData	System Unavailable
14	PasswordResetData.token	Email Token Unknown
15	EmailTokenSessionData.expiration_time	Email Token Expired
16	PasswordResetData.password, PasswordResetData.password_confirm	Password Confirm Mismatch
17	PasswordResetData.password	-
18	PasswordResetData.password	Password Requirement Violation
20	PasswordResetData.password	-
21	PasswordUserData.salt	-
22	PasswordResetData.password, PasswordUserData.salt	-
23	PasswordResetData.password, PasswordUserData.salt	-
24	PasswordUserData.password_hash	-
25	PasswordUserData	-
26	EmailTokenSessionData.user_id, PasswordUserData	-
27	EmailTokenSessionData.user_id, PasswordUserData, UserData.password_data	-
29	EmailTokenSessionData.user_id	-
31	EmailTokenSessionData	-
33	-	Email Send
35	-	Password Reset Successful

Table A.9.: Data Element and Event references for Figure A.15

Step	Data Elements	Events
1	InitiateEmailChangeData	Email Change Initiated, System Unavailable
3	UserData	-
4	InitiateEmailChangeData, UserData	-
5	InitiateEmailChangeData.new_email	Unexpected Email Format
6	InitiateEmailChangeData.password, UserData.password_data.salt	-
7	InitiateEmailChangeData.password, UserData.password_data.salt	-
8	PasswordVerificationData	-
9	PasswordVerificationData.password_hash, UserData.password_data.password_hash	User Password Wrong
10	EmailChangeTokenSessionData	-
11	EmailChangeTokenSessionData	-
12	EmailChangeTokenSessionData	-
13	EmailTokenURLData, UserData.email, InitiateEmailChangeData.new_email	-
14	UserData.email	Email Send
15	InitiateEmailChangeData.new_email	Email Send
22a	EmailTokenData	System Unavailable
23a	EmailTokenData.token	Email Token Unknown
24a	-	Email Change Blocked
25a	EmailChangeTokenSessionData.user_id	-
27a	EmailChangeTokenSessionData.user_id	-
22b	EmailTokenData	System Unavailable
23b	EmailTokenData.token	Email Token Unknown
24b	EmailChangeTokenSessionData.expiration_time	Email Token Expired
25b	EmailChangeTokenSessionData.confirmed	-
26b	EmailChangeTokenSessionData.confirmed	-
29b	EmailTokenData	System Unavailable
30b	EmailTokenData.token	Email Token Unknown
31b	EmailChangeTokenSessionData.expiration_time	Email Token Expired
32b	EmailChangeTokenSessionData.confirmed	-
33b	EmailChangeTokenSessionData.user_id, EmailChangeTokenSessionData.new_email	-
34b	UserData.email, EmailChangeTokenSessionData.new_email	-
36b	EmailChangeTokenSessionData.user_id	-
38b	EmailChangeTokenSessionData.user_id	-

Table A.10.: Data Element and Event references for Figure A.16

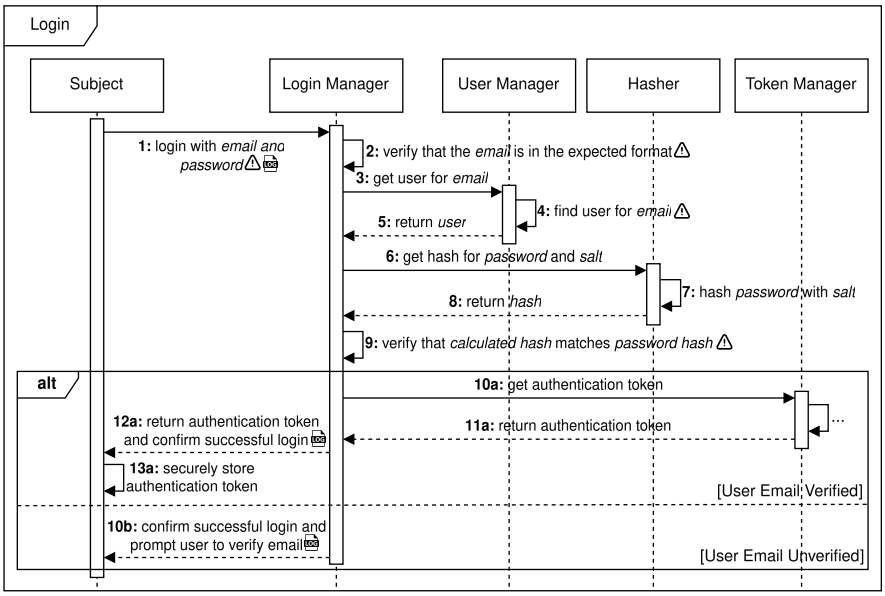


Figure A.14.: PBA SDP "Login" Behavioral Model

Events

Event (Registration Initiated)**.** A user initiates the registration process by providing their email address and password.
Security Consideration: A high frequency of registration attempts from the same IP address or unusual patterns may indicate automated bots or DoS attacks. Implementing a CAPTCHA or limiting the number of registration attempts per IP address can help mitigate this.

Event (User Created)**.** A new user account has been successfully created.
Security Consideration: Monitoring account creation patterns can help detect mass account creation by malicious actors.

Event (User Email Verified)**.** The user has successfully verified their email address

Event (Registration Successful)**.** The user has completed all registration steps, including email verification, and can now access the system.

Event (Login Initiated)**.** A subject initiates a login attempt by entering their email address and password.
Security Consideration: A high number of login attempts, especially failed ones, can indicate brute force or credential stuffing attacks.

Event (Login Successful)**.** A user has successfully authenticated and gained access to the system.
Security Consideration: Logins from new devices, locations, or IP addresses may indicate

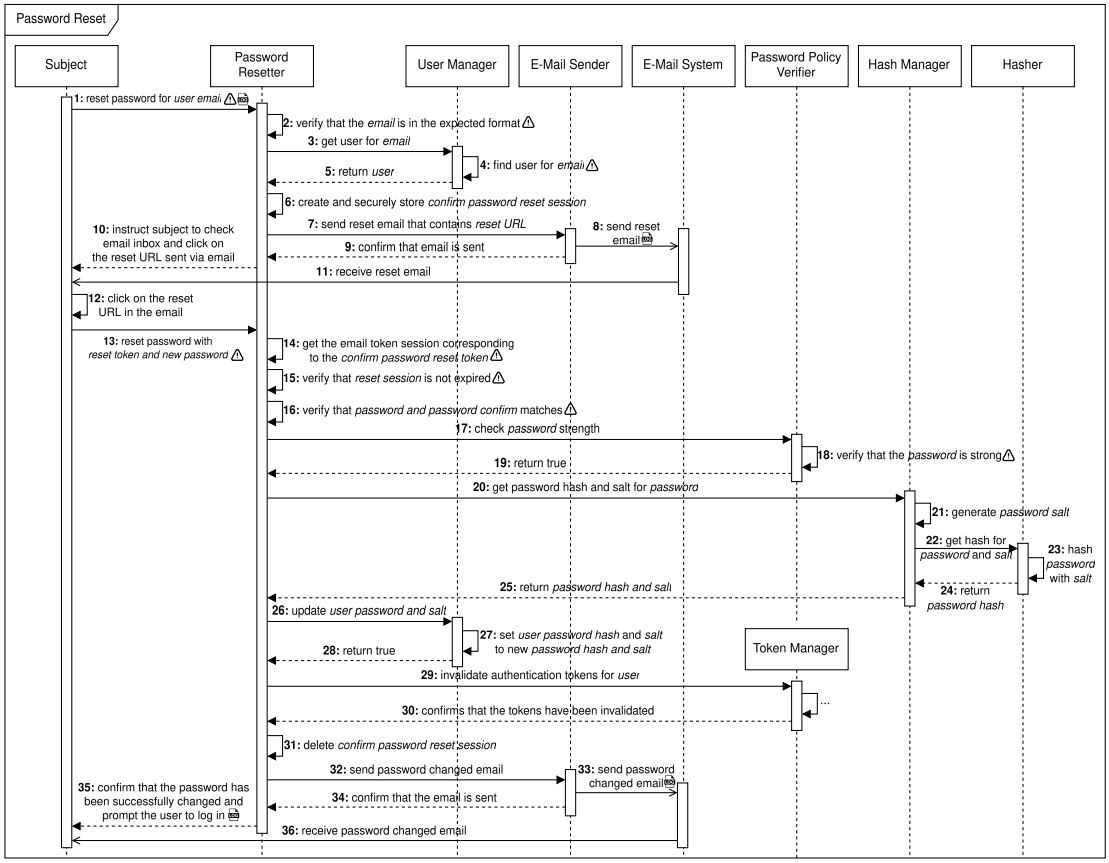


Figure A.15.: PBA SDP “Password Reset” Behavioral Model

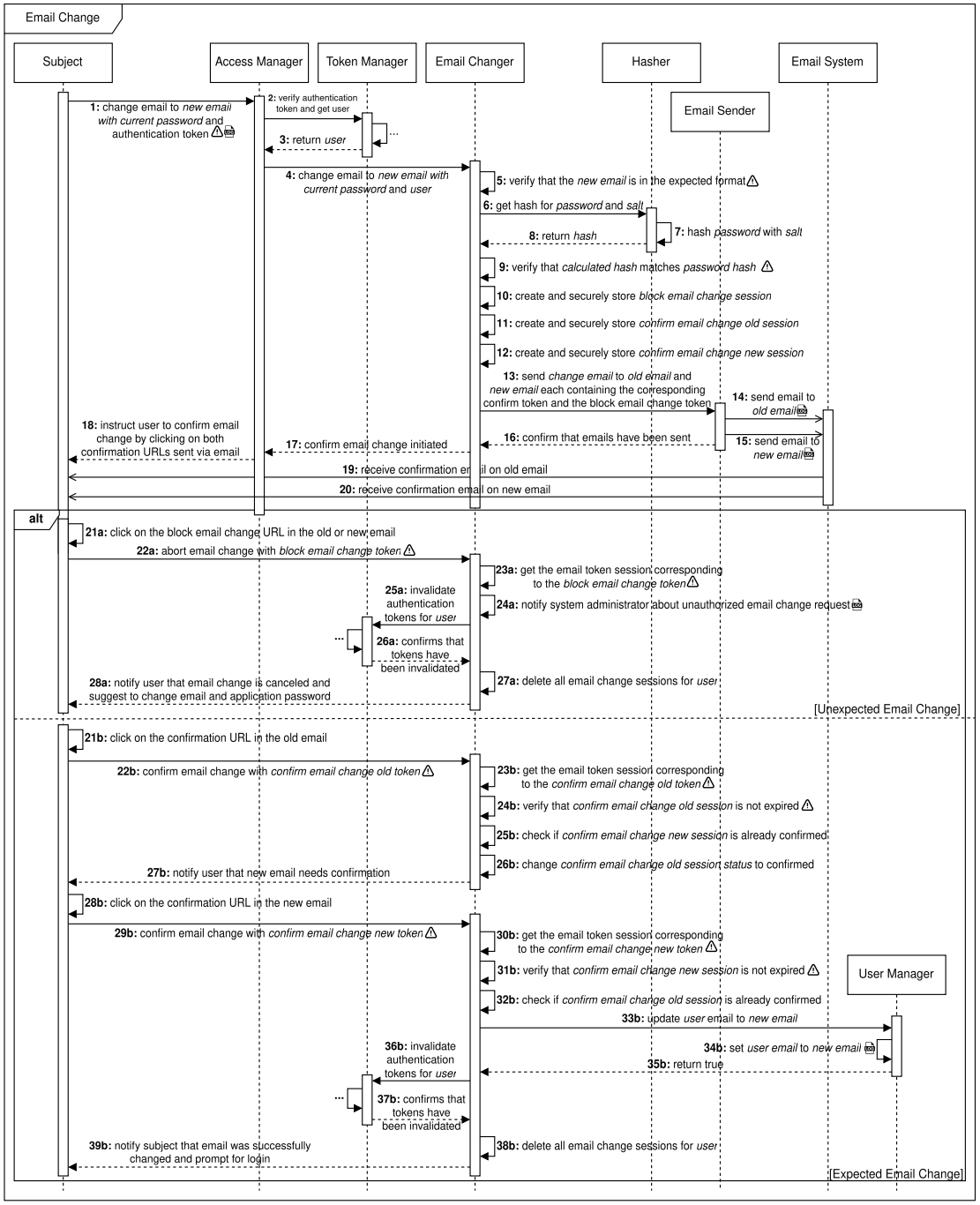


Figure A.16.: PBA SDP “Email Change” Behavioral Model

unauthorized access attempts and should be monitored. To reduce risk, notify users of such logins.

Event (Password Reset Initiated)**.** A user has requested to reset their password, initiating the password reset process.

Event (Password Reset Successful)**.** A user has successfully changed their password.

Event (Email Change Initiated)**.** A user has initiated the process to change their registered email address.

Event (Email Change Blocked)**.** An attempt to change the user's email address has been actively blocked by clicking a Block Email Change link at the old or new email address.
Security Consideration: Indicates malicious activity that attempts to redirect user communications or compromise accounts. User account may be temporarily disabled pending identity verification.

Event (Email Change Successful)**.** The user's email address was successfully updated after both the old and new email addresses were verified.

Event (Email Send)**.** An email has been sent to the user, such as verification links, password reset instructions, or notifications.

Error Event (System Unavailable)**.** The system is unresponsive or unavailable when a user attempts an action. Possible causes include server downtime, network issues, or infrastructure failures.
Fallback Behavior: Retry with Exponential Backoff with at most three retries and an initial delay of two seconds.

Error Event (Password Confirm Mismatch)**.** The password and password confirmation fields do not match during registration or password change attempts.

Error Event (Password Requirement Violation)**.** A provided password does not meet password requirements, such as minimum length.

Error Event (Unexpected Email Format)**.** An email address does not conform the defined email format, for example, it contains invalid characters.
Security Consideration: This may indicate a SQL injection attack or other injection attacks that use crafted input to exploit the system.

Error Event (User Already Exist)**.** A user has attempted to register a new account or change an email to an email address that is already associated with an existing user.
Security Considerations:

- Detailed error messages can help attackers enumerate users; use generic messages to avoid this (CWE-204: Observable Response Discrepancy). For example, respond with "A link to activate your account has been emailed to the address provided".

- Timing differences between the successful and unsuccessful cases can help attackers enumerate users; avoid fail-fast behaviors, i.e., ensure that both cases take roughly the same amount of time (CWE-208: Observable Timing Discrepancy).

Error Event (User Not Found)**.** The user account associated with the email address does not exist. This can happen when a subject tries to log in or reset a password.
Security Considerations:

- Detailed error messages can help attackers enumerate users; use generic messages to avoid this (CWE-204: Observable Response Discrepancy). For example, respond with "Login failed; Invalid email address or password" or "If that email address is in our database, we will send you an email to reset your password".
- Timing differences between the successful and unsuccessful cases can help attackers enumerate users; avoid fail-fast behaviors, i.e., ensure that both cases take roughly the same amount of time (CWE-208: Observable Timing Discrepancy).

Error Event (User Password Wrong)**.** The password provided does not match the stored password for the user account during a login or email change attempt.
Security Considerations:

- Detailed error messages can help attackers enumerate users; use generic messages to avoid this (CWE-204: Observable Response Discrepancy). For example, respond with "Login failed; Invalid email address or password" or "If the password is correct, we will send confirmation emails to both the old and new email addresses".
- Timing differences between the successful and unsuccessful cases can help attackers enumerate users; avoid fail-fast behaviors, i.e., ensure that both cases take roughly the same amount of time (CWE-208: Observable Timing Discrepancy).

Error Event (Email Token Unknown)**.** The email verification, password reset token, or email change token provided does not match any issued tokens.
Security Consideration: May indicate token tampering or replay attacks where an attacker tries to reuse or forge tokens.

Error Event (Email Token Expired)**.** The email verification or password reset token has expired and is no longer valid. Tokens typically have a limited validity period for security reasons.

A.5.3. Structural View

Architectural Model 1: Figure 6.13. Architectural Model 2: Figure A.17.

A.5.4. Consequences

- **Strength (Maintainability):** Single-Factor Authentication, especially when implemented without requiring user lockout after a number of failed login attempts and relying only on password salts for secure storage, requires minimal dependencies and fewer components. This simplicity also reduces potential points of failure, enabling quicker identification and resolution of issues.

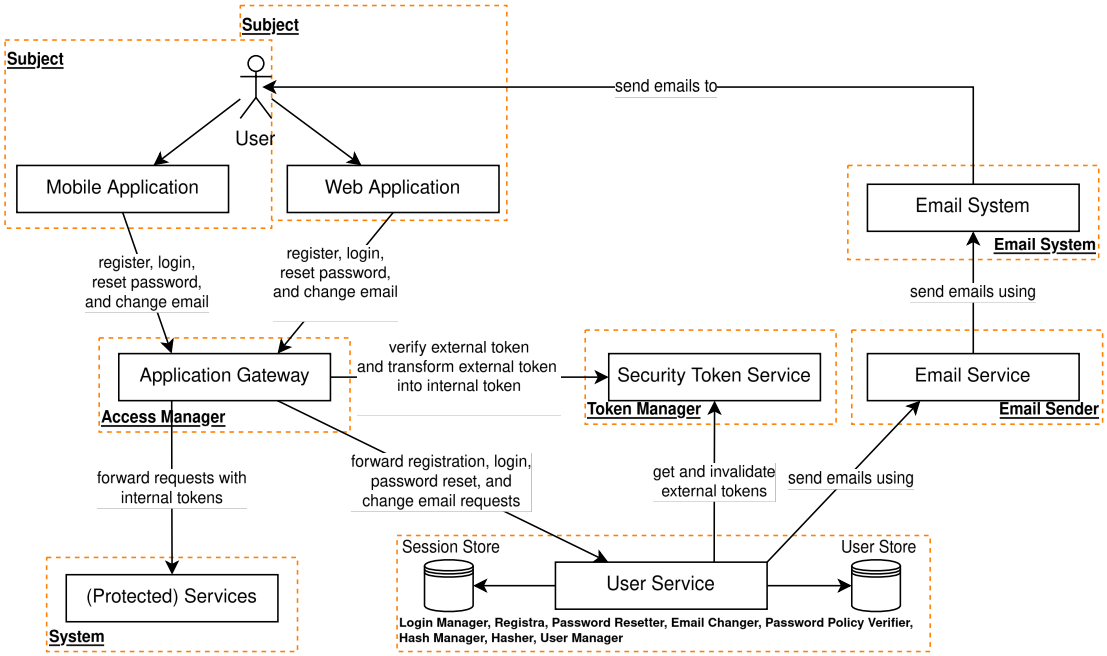


Figure A.17.: Architectural Model representing the static structure of a microservice architecture that implements the PBA Roles, visualized in a C4 container diagram.

- **Weakness (Security):** The reliance on a single password makes the system particularly vulnerable to attacks such as phishing, credential stuffing, or brute force attacks. Furthermore, users often reuse identical passwords across systems, leading to cascading breach scenarios where a compromise in one system exposes multiple others. In addition, many regulatory standards mandate stronger authentication mechanisms, making Single-Factor Authentication insufficient for compliance in sensitive environments.

A.5.5. Other

Attributes

- Password Reset Mechanism: Email Token
- Password Peppering: No
- Password Rotation with History: No
- User Account Lockout: No

Sources

- Changing A User's Registered Email Address
- Authentication Cheat Sheet
- Forgot Password Cheat Sheet
- Password-based Authentication - Security Pattern Catalog

A.6. Secret Storage

Secure client-side storage of sensitive and secret information is critical to maintaining system security and protecting user data. The following are best practices and recommendations for achieving secure storage for different types of clients:

Web Client

Secure Cookies: Use cookies with appropriate attributes to securely store secrets such as JSON Web Tokens or session identifiers:

- `HttpOnly`: Prevents client-side scripts from accessing the cookie, mitigating the risk of *Cross-Site Scripting (XSS)* attacks.
- `Secure`: Ensure that the cookie is only sent over secure HTTPS connections.
- `SameSite=Strict`: The cookie is only sent with requests that originate from the same site. This provides protection against *Cross-Site Request Forgery (CSRF)* attacks, but it may interfere with legitimate cross-site interactions, such as third-party authentication or payment gateways.
- `SameSite=Lax`: The cookie is sent with same-site requests and with top-level navigation to your site (e.g., when a user clicks a link to your site from another site). This provides a balance between security and usability by providing CSRF protection while still allowing certain cross-site requests that are likely initiated by the user.

- `Path`: Specifies the subset of URLs that the cookie applies to (e.g., `Path=/api/v1`), which limits where the cookie is sent.
- *Cookie Prefixes*:
  - `__Secure-`: Requires the cookies to have the `Secure` attribute.
  - `__Host-`: Requires the cookies to have the `Secure` attribute, no `Domain` attribute, and a `Path` attribute set to `/`.

Web Storage: Avoid storing sensitive data in the browser's *localStorage* or *sessionStorage* due to vulnerability to XSS attacks. If necessary, consider encrypting the data on the server side before storing it, but prefer cookies for better security.

Native Mobile Clients

iOS (Keychain): Use the iOS Keychain to store sensitive information. The Keychain encrypts data and restricts access to the application, ensuring secure storage of tokens and passwords.
Android (Keystore): Use the Android Keystore to securely generate and store cryptographic keys. The cryptographic key can be used to store the sensitive data.

Desktop Clients

Windows (Data Protection API (DPAPI)): Use DPAPI to encrypt sensitive data before storing it locally. DPAPI ties the encryption key to the user's credentials, making it secure and accessible only to the user who encrypted the data.
macOS (Keychain): Use the macOS Keychain to store sensitive information. The Keychain encrypt data and restricts access to the application, ensuring secure storage of tokens and passwords.

Bibliography

[21] *A04:2021-Insecure Design*. https://owasp.org/Top10/A04_2021-Insecure_Design/. Accessed: 2024-09-22. OWASP Foundation, 2021 (cit. on pp. 1, 36).

[22] *Information security, cybersecurity and privacy protection - Information security controls*. Standard ISO/IEC 27002:2022. International Organization for Standardization, 2022. URL: https://www.iso.org/standard/75652.html (cit. on pp. 32, 33).

[23] *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model*. Standard ISO/IEC 25010:2023. International Organization for Standardization, 2023. URL: https://www.iso.org/standard/78176.html (cit. on p. 67).

[Aba24] A. Abazi. "Conception of a Security Solution Pattern Catalog for Constraint-based Recommender Systems." Available at: https://swc.rwth-aachen.de/docs/bachelor-master-theses/2024-012-BT_Abazi/2024-11-27_BTArbenAbazi-ThesisReport.pdf. Bachelor's Thesis. Rheinisch Westfälische Technische Hochschule Aachen, Sept. 2024 (cit. on pp. 3, 5, 9, 10, 21, 22, 63, 67, 68).

[Agg16] C. C. Aggarwal. *Recommender Systems*. Cham: Springer International Publishing, 2016. DOI: 10.1007/978-3-319-29659-3 (cit. on pp. 12, 69).

[Ale+77] C. Alexander et al. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977. ISBN: 978-0-19-501919-3 (cit. on p. 15).

[And+04] J. Andrade et al. "A Methodological Framework for Viewpoint-Oriented Conceptual Modeling." In: *IEEE Transactions on Software Engineering* 30.5 (2004), pp. 282–294. DOI: 10.1109/TSE.2004.1 (cit. on pp. 16, 17).

[App97] B. Appleton. *Patterns and software: Essential concepts and terminology*. Available at: https://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/appleton-patterns-intro.pdf. 1997 (cit. on pp. 5, 89).

[Bar+18] M. H. Barkadehi et al. "Authentication Systems: A Literature Review and Classification." In: *Telematics and Informatics* 35.5 (2018), pp. 1491–1511. DOI: 10.1016/j.tele.2018.03.018 (cit. on p. 15).

[Bas+22] S. K. Basak et al. "What Are the Practices for Secret Management in Software Artifacts?" In: *2022 IEEE Secure Development Conference (SecDev)*. 2022, pp. 69–76. DOI: 10.1109/SecDev53368.2022.00026 (cit. on p. 33).

[BLL15] E. Bauman, Y. Lu, and Z. Lin. "Half a Century of Practice: Who Is Still Storing Plaintext Passwords?" In: *Information Security Practice and Experience*. Springer International Publishing, 2015, pp. 253–267. DOI: 10.1007/978-3-319-17533-1_18 (cit. on p. 36).

[CDW04] A. Conklin, G. Dietrich, and D. Walz. "Password-Based Authentication: A System Perspective." In: *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of The*. 2004. DOI: 10.1109/HICSS.2004.1265412 (cit. on p. 7).

[Cla12] J. Clarke-Salt. *SQL Injection Attacks and Defense*. Elsevier, 2012. ISBN: 978-1-59749-963-7 (cit. on p. 7).

[DA21] D. D'Silva and D. D. Ambawade. "Building A Zero Trust Architecture Using Kubernetes." In: *2021 6th International Conference for Convergence in Technology (I2CT)*. 2021. DOI: 10.1109/I2CT51068.2021.9418203 (cit. on p. 89).

[Den+19] W. Denniss et al. *OAuth 2.0 Device Authorization Grant*. RFC 8628. 2019. DOI: 10.17487/RFC8628 (cit. on p. 76).

[Dir20] Directorate General for Health and Food Safety. *European Interoperability Certificate Governance - A Security Architecture for contact tracing and warning apps*. Tech. rep. European Commission, 2020. URL: https://health.ec.europa.eu/publications/european-interoperability-certificate-governance-security-architecture-contact-tracing-and-warning_en (cit. on p. 18).

[Dix20] A. Dix. "Die deutsche Corona Warn-App–ein gelungenes Beispiel für Privacy by Design?" In: *Datenschutz und Datensicherheit - DuD* 44 (2020), pp. 779–785. DOI: 10.1007/s11623-020-1366-1 (cit. on p. 18).

[DJS19] D. Deogun, D. B. Johnsson, and D. Sawano. *Secure By Design*. 1st Edition. Manning Publications, 2019. ISBN: 978-1-61729-435-8 (cit. on pp. 1, 27, 30, 34).

[eSe23] eSentire. *2023 Official Cybercrime Report*. https://www.esentire.com/resources/library/2023-official-cybercrime-report. 2023. (Visited on 09/22/2024) (cit. on p. 1).

[FB08] A. Felfernig and R. Burke. "Constraint-Based Recommender Systems: Technologies and Research Issues." In: *Proceedings of the 10th International Conference on Electronic Commerce*. ICEC '08. 2008, pp. 1–10. DOI: 10.1145/1409540.1409544 (cit. on pp. 12–14, 69, 73, 74, 83).

[Fed24] Federal Office for Information Security (BSI). *BSI TR-02102-1 Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Tech. rep. Available at: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102. Federal Office for Information Security (BSI), 2024 (cit. on p. 19).

[Fer13] E. Fernandez-Buglioni. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. John Wiley & Sons, 2013. ISBN: 978-1-119-97048-4 (cit. on pp. 1, 21–23).

[Fet21] D. Fett. "FAPI 2.0: A High-Security Profile for OAuth and OpenID Connect." In: *Open Identity Summit 2021*. Gesellschaft für Informatik e.V., 2021, pp. 71–82. ISBN: 978-3-88579-706-7 (cit. on p. 55).

[FKS17] D. Fett, R. Küsters, and G. Schmitz. "The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines." In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. 2017, pp. 189–202. DOI: 10.1109/CSF.2017.20 (cit. on pp. 7, 94).

[Gam+95] E. Gamma et al. *Design Patterns*. Addison Wesley Professional Computing Series. 1995. ISBN: 9780201633610 (cit. on pp. 5, 15).

[Gli+23] M. Glinz et al. "Towards a Modern Quality Framework." In: *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. 2023, pp. 357–361. DOI: 10.1109/REW57809.2023.00067 (cit. on p. 67).

[GN14] R. Gafni and D. Nissim. "To Social Login or Not Login? Exploring Factors Affecting the Decision." In: *Issues in Informing Science and Information Technology* 11 (2014), pp. 57–72. DOI: 10.28945/1980 (cit. on pp. 7, 8, 19, 77).

[Gra+20] P. A. Grassi et al. *Digital Identity Guidelines - Authentication and Lifecycle Management*. Tech. rep. NIST Special Publication (SP) 800-63B. National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.SP.800-63b (cit. on pp. 1, 69, 70, 91).

[Hal09] C. Hale. *A Lesson In Timing Attacks*. Accessed: 2024-11-07. 2009. URL: https://codahale.com/a-lesson-in-timing-attacks/ (cit. on p. 59).

[Har12] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. 2012. DOI: 10.17487/RFC6749 (cit. on p. 75).

[Hey+07] T. Heyman et al. "An Analysis of the Security Patterns Landscape." In: *Third International Workshop on Software Engineering for Secure Systems (SESS'07: ICSE Workshops 2007)*. 2007. DOI: 10.1109/SESS.2007.4 (cit. on pp. 1, 2, 15, 25, 30, 43, 87).

[HPL24] D. Hardt, A. Parecki, and T. Lodderstedt. *The OAuth 2.1 Authorization Framework*. Internet-Draft draft-ietf-oauth-v2-1-12. Work in Progress. Internet Engineering Task Force, 2024. URL: https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/12/ (cit. on p. 75).

[ISC23] ISC2. *2023 ISC2 Cybersecurity Workforce Study*. 2023. URL: https://media.isc2.org/-/media/Project/ISC2/Main/Media/documents/research/ISC2_Cybersecurity_Workforce_Study_2023.pdf (cit. on p. 1).

[Jov20] R. P. Jover. "Security Analysis of SMS as a Second Factor of Authentication: The Challenges of Multifactor Authentication Based on SMS, Including Cellular Security Deficiencies, SS7 Exploits, and SIM Swapping." In: 18.4 (2020), pp. 37–60. DOI: 10.1145/3424302.3425909 (cit. on pp. 67, 85).

[Kra] M. Kraus. *Protect Your Frontend: Why JWTs Should Stay Backstage*. Accessed: 2024-12-03 (cit. on p. 89).

[Las21] W. Lasarov. "Im Spannungsfeld zwischen Sicherheit und Freiheit: Eine Analyse zur Akzeptanz der Corona-Warn-App." In: *HMD Praxis der Wirtschaftsinformatik* 58.2 (2021), pp. 377–394. DOI: 10.1365/s40702-020-00646-3 (cit. on p. 18).

[LL13] J. Ludewig and H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. 3. korr. dpunkt. verlag, 2013. ISBN: 978-3-86490-092-1 (cit. on p. 1).

[LS24] T. Langstrof and A. R. Sabau. "The Current State of Security – Insights from the German Software Industry." In: *CoRR* abs/2402.08436 (2024). DOI: 10.48550/arXiv.2402.08436 (cit. on pp. 2, 19, 90).

[Mai+17] C. Mainka et al. "SoK: Single Sign-On Security — An Evaluation of OpenID Connect." In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2017, pp. 251–266. DOI: 10.1109/EuroSP.2017.32 (cit. on p. 94).

[Meg24] N. Meganathan. "What Is the Effectiveness of Salt and Pepper in Preventing Rainbow Table Attacks in Modern Password Hashing Algorithms?" In: *International Journal of Innovative Science and Research Technology (IJISRT)* (2024), pp. 242–248. DOI: 10.38124/ijisrt/IJISRT24SEP406 (cit. on p. 15).

[NB19] J. Navas and M. Beltrán. "Understanding and Mitigating OpenID Connect Threats." In: *Computers & Security* 84 (2019). DOI: 10.1016/j.cose.2019.03.003 (cit. on p. 94).

[OMG23] O. M. G. (OMG). *OMG Unified Modeling Language (UML) Specification*. Tech. rep. Version 2.5.1. Object Management Group, 2023. URL: https://www.omg.org/spec/UML (cit. on p. 50).

[PCI24] PCI Security Standards Council. *PCI DSS v4.0.1: Payment Card Industry Data Security Standard*. Tech. rep. Available at: https://www.pcisecuritystandards.org/document_library. PCI Security Standards Council, 2024 (cit. on pp. 2, 23, 69, 70, 86, 91).

[RF20] M. Richards and N. Ford. *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, 2020. ISBN: 978-1492043454 (cit. on pp. 6, 53, 60, 71–73, 91).

[Ric18] C. Richardson. *Microservice Patterns: With Examples in Java*. Manning, Shelter Island, NY, 2018. ISBN: 978-1-61729-454-9 (cit. on p. 53).

[Ros+20] S. Rose et al. *Zero trust architecture*. Tech. rep. NIST Special Publication (SP) 800-207. National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.SP.800-207 (cit. on p. 30).

[RW11] N. Rozanski and E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2011. ISBN: 978-0-13-290612-8 (cit. on pp. 5, 6, 25–28, 41).

[Sab24] A. R. Sabau. "A Guided Modeling Approach for Secure System Design." In: *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. 2024, pp. 105–110. DOI: 10.1109/ICSA-C63560.2024.00026 (cit. on pp. 18, 38, 40, 41, 95).

[SBA15] N. Sakimura, J. Bradley, and N. Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. RFC 7636. 2015. DOI: 10.17487/RFC7636 (cit. on p. 48).

[SC14] S. Samonas and D. Coss. "The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security." In: *Journal of Information Systems Security* 10.3 (2014), pp. 21–45 (cit. on p. 6).

[Sch+06] M. Schumacher et al. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2006. ISBN: 978-1-118-72593-1 (cit. on pp. 1, 9, 10, 15, 21–23).

[Sch03] M. Schumacher. "Security Engineering with Patterns : Origins, Theoretical Model, and New Applications." In: Springer, 2003. ISBN: 978-3-540-45180-8 (cit. on p. 10).

[Sho14] A. Shostack. *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014. ISBN: 978-1-118-81005-7 (cit. on pp. 6, 7).

[Sto+21] A. van der Stock et al. *OWASP Application Security Verification Standard 4.0.3*. Tech. rep. Version 4.0.3. Open Web Application Security Project (OWASP), 2021. URL: https://owasp.org/www-project-application-security-verification-standard/ (cit. on p. 86).

[VH14] A. Vassilev and T. A. Hall. "The Importance of Entropy to Information Security." In: *Computer* 47.2 (2014), pp. 78–81. DOI: 10.1109/MC.2014.47 (cit. on p. 47).

[vYJ18] A. van den Berghe, K. Yskout, and W. Joosen. "Security Patterns 2.0: Towards Security Patterns Based on Security Building Blocks." In: *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment*. SEAD '18. 2018, pp. 45–48. DOI: 10.1145/3194707.3194715 (cit. on pp. 1, 9, 15, 21, 34).

[vYJ22a] A. van den Berghe, K. Yskout, and W. Joosen. "A Reimagined Catalogue of Software Security Patterns." In: *Proceedings of the 3rd International Workshop on Engineering and Cybersecurity of Critical Systems*. EnCyCriS '22. 2022, pp. 25–32. DOI: 10.1145/3524489.3527301 (cit. on pp. 2, 7, 10, 21–25, 28, 30, 31, 88).

[vYJ22b] A. van den Berghe, K. Yskout, and W. Joosen. *Authentication*. https://gitlab.kuleuven.be/distrinet/research/security-patterns/security-pattern-catalogue/-/blob/main/docs/patterns/01_01_001__authentication.md. Commit Hash: ebfb2eb9cfc800060ea44db5fc9e5d19c6ea7606. 2022 (cit. on pp. 9, 10, 22).

[vYJ22c] A. van den Berghe, K. Yskout, and W. Joosen. *Password-based authentication*. https://gitlab.kuleuven.be/distrinet/research/security-patterns/security-pattern-catalogue/-/blob/main/docs/patterns/01_01_002__authentication_pwd.md. Commit Hash: b0ca944b49c73cccd33f6da8a6c5c2d67f4c2b79. 2022 (cit. on pp. 1, 15, 22, 23, 57).

[vYJ22d] A. van den Berghe, K. Yskout, and W. Joosen. *Session-based Access Control*. https://gitlab.kuleuven.be/distrinet/research/security-patterns/security-pattern-catalogue/-/blob/main/docs/patterns/01_01_006__session_based_access_control.md. Commit Hash: c9b8250d1dd7567d29a343ec491f1713cb8f1cdb. 2022 (cit. on p. 8).

[vYJ22e] A. van den Berghe, K. Yskout, and W. Joosen. *Verifiable token-based authentication*. https://gitlab.kuleuven.be/distrinet/research/security-patterns/security-pattern-catalogue/-/blob/main/docs/patterns/01_01_003__verifiable_token_based_authentication.md. Commit Hash: 801234a6ec461d716123084903b70b782ba1a72a. 2022 (cit. on p. 8).

[Wor23] World Economic Forum. *Global Risks Report 2023*. Accessed: 2024-09-23. 2023. URL: https://www3.weforum.org/docs/WEF_Global_Risks_Report_2023.pdf (cit. on p. 1).

[YSJ12] K. Yskout, R. Scandariato, and W. Joosen. "Does Organizing Security Patterns Focus Architectural Choices?" In: *2012 34th International Conference on Software Engineering (ICSE)*. 2012, pp. 617–627. DOI: 10.1109/ICSE.2012.6227155 (cit. on pp. 23, 91).

[YSJ15] K. Yskout, R. Scandariato, and W. Joosen. "Do Security Patterns Really Help Designers?" In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. 2015, pp. 292–302. DOI: 10.1109/ICSE.2015.49 (cit. on p. 95).

[YT05] E. Yuan and J. Tong. "Attributed based access control (ABAC) for Web services." In: *IEEE International Conference on Web Services (ICWS'05)*. 2005, p. 569. DOI: 10.1109/ICWS.2005.25 (cit. on p. 31).

Glossary

AM Architectural Model

AME Architectural Modeling Element

ASSP Abstract Security Solution Pattern

BM Behavioral Model

BME Behavioral Modeling Element

CBRS Constraint-based Recommender System

CSRF Cross-Site Request Forgery

DP Decision Point

EP Enforcement Point

IdP Identity Provider

IP Information Point

JWKS JSON Web Key Set

JWT JSON Web Token

KB Knowledge Base

MFA Multi-Factor Authentication

OIDC OpenID Connect

OSS Open Source Software

PBA Password-based Authentication

PCI DSS Payment Card Industry Data Security Standard

RP Relying Party

SDP Security Design Pattern

SDPDM Security Design Pattern Description Metamodel

SP Security Pattern

SQL Structured Query Language

SSP Security Solution Pattern

UML Unified Modeling Language