Institute of Information Security

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Designing Privacy-Preserving Architectures for Cloud-Based Services

Nico Haas

Course of Study:	Informatik
Examiner:	Prof. Dr. Ralf Küsters
Supervisor:	Julian Liedtke, M.Sc., Immanuel Kunz, M.Sc.
Commenced:	July 15, 2020
Completed:	January 15, 2021

Abstract

Privacy is becoming increasingly relevant in society. One reason is the growth in digital networking of people, partly resulting from the increased use of end devices. According to Adam Moore, the definition of privacy includes the "right to control access to and uses of places, bodies, and personal information" [Moo08]. This definition assumes the users themselves can govern the flow of their personal data. In many cases, however, user data is still being stored without consent, partly for commercial purposes or misused by other third parties. One specific case is the Facebook-Cambridge Analytica scandal in 2018 [CG18]. In order to protect personal data, the EU introduced a uniform set of rules: The General Data Protection Regulation (GDPR) [EU16]. Implemented in 2018, it aims to ensure privacy at an early stage of software development ("privacy by design"), and data protection as a default setting ("privacy by default").

In software development, architectural patterns are used for designing a software architecture, each representing "a package of design decisions that is found repeatedly in practice, has known properties that permit reuse, and describes a class of architectures" [BCK03]. These patterns do not address privacy, or address it insufficiently.

The following two questions arise: Which architectural patterns exist that implement privacy requirements in a software architecture? How can these patterns be selected in a given application context?

This thesis proposes *privacy-preserving architectural patterns* implementing privacy requirements from early on in the software development process. Furthermore, a methodology is presented that assigns appropriate patterns to the respective use case. A use case demonstrates the applicability of the methodology. Finally, the the presented architectural patterns and methodology are discussed.

Kurzfassung

Die Privatsphäre ist ein Bereich, der aktuell zunehmend an Relevanz in der Gesellschaft gewinnt. Ein Grund hierfür ist die fortschreitende digitale Vernetzung der Menschen, mit bedingt durch die vermehrte Nutzung von Endgeräten. Laut Adam Moore umfasst die Definition von Privatsphäre "ein Recht auf Kontrolle des Zugangs zu und der Nutzung von persönlichen Informationen" [Moo08]. Dem zur Folge wird davon ausgegangen, dass der Nutzer selbst den Datenfluss regulieren kann. In vielen Fällen werden aber bis heute Nutzerdaten ohne Zustimmung gespeichert, die zum Teil für kommerzielle Zwecke oder von anderen Dritten missbraucht werden. Ein konkreter Fall ist der Facebook Cambridge Analytica Skandal aus dem Jahr 2018 [CG18].

Um den Schutz der persönlichen Daten zu wahren, wurde in der EU mit der Datenschutzgrundverordnung (DS-GVO) ein europaweit einheitliches Regelwerk eingeführt, welches seit 2018 Anwendung findet. Ein Bestreben der DS-GVO ist die Sicherstellung der Privatsphäre in einem frühen Stadium der Software-Entwicklung ("privacy by design") und Datenschutz als Grundeinstellung ("privacy by default").

Im Bereich der Softwareentwicklung existieren sogenannte Architekturmuster, die "ein Bündel aus Entwurfsentscheidungen darstellen, die wiederholt in der Praxis vorkommen, bekannte Eigenschaften besitzen, Mehrfachnutzung erlauben und eine Klasse von Architekturen beschreiben" [BCK03]. Jedoch wird die Privatsphäre, mittels dieser Muster, nicht oder in unzureichender Form berücksichtigt.

An dieser Stelle setzt die Thesis an und stellt einen Lösungsversuch für die folgenden zwei Problemstellungen vor: Welche Architekturmuster existieren, die Anforderungen an den Datenschutz in der Architektur umsetzen? Wie können diese Patterns in einem Applikationskontext gewählt werden?

Dazu werden in erster Linie *Privatsphäre unterstützende Architekturmuster* (engl. "privacy-preserving architectural patterns") vorgeschlagen, die Privatsphäre von früh an im Softwareentwicklungsprozess berücksichtigen. Desweiteren wird eine Methodik vorgestellt, die dem jeweiligen Anwendungsfall entsprechende Muster zuordnet. Ein Machbarkeitsnachweis demonstriert die Methodik anhand eines Anwendungsbeispiels. Abschließend wird die Zweckmäßigkeit der hier vorgestellten Architekturmuster und Methodik diskutiert.

Contents

List of Figures

List of Tables

Acronyms

AWS Amazon Web Services. 20

CC Common Criteria. 30

CSP cloud service provider. 20

FIP fair information practice. 28

GDPR General Data Protection Regulation. 3

GoF Gang of Four. 23

HE homomorphic encryption. 19

IaaS Infrastructure as a Service. 20

IDC International Data Corporation. 13

IoT Internet of Things. 13

NIST National Institute of Standards and Technology. 20

OECD Organisation for Economic Co-operation and Development. 28

OSS open-source software. 68

PaaS Platform as a Service. 20

PET Privacy Enhancing Technologies. 14

PIA privacy impact assessment. 34

PII personally identifiable information. 45

PPAP privacy-preserving architectural pattern. 14, 107

RQ research question. 107

SaaS Software as a Service. 20

SDLC Software Development Life Cycle. 14

SSO single sign-on. 79

TEE trusted execution environment. 61

Tor The Onion Routing. 19

TPP third party processor. 42

TTP trusted third party. 29

UI user interface. 36

VPN virtual private network. 65

1 Introduction

The goal of protecting privacy becomes a greater challenge considering the growth of data collecting services. The International Data Corporation (IDC) predicts, as digitization continues, all data that is created, captured, or replicated will increase fivefold by 2025 compared to 2018—in figures to 175 Zetabytes [Ryd18]. According to the IDC, the now almost ubiquitous technology of cloud computing will account for half of it, which allows services, such as the provision of storage or software, to be used over the Internet (the *cloud*). The Internet of Things (IoT), as an example, uses cloud services in almost every application. The concept of IoT is to connect end devices (the *things*) in a network with the goal of supporting people in various areas of life; both in the private sector (be it *smart home* or *smart cities*), as well as in the field of industrial production (*Fourth Industrial Revolution*) [AGM+15; XXL18]. Considering current and supposed future sales figures, the increasing number of IoT devices is apparent: 50 billion are currently in use, another 30 billion devices are predicted to be in use next year. The fact that companies will eventually move their business to the cloud [HH10], emphasizes cloud computing becoming nearly pervasive. The world of ubiquitous computing, envisioned by Mark Weiser, seems to be a stone's throw away [Wei91]. More companies and private individuals using the cloud implies more personal data is moving to the cloud. Known examples are recent incidents where private data had become public[1], or has been collected by companies for targeted advertising[2]. Protecting this data is made difficult by the fact that privacy, and therefore privacy infringements, is not easy to define. Privacy is a complex and multidimensional concept (see Chapter 2). One of various emerging definitions reads as follows: "A right to privacy is a right to control access to and uses of places, bodies, and personal information" [Moo08]. These definitions are not binding and counteracting is not punishable by law. To prevent the violation of privacy, a number of legal regulations have been adopted in various parts of the world. A known example is the EU's General Data Protection Regulation (GDPR), which imposes heavy penalties for non-compliance. The GDPR complicates the way data collecting companies, like Google and Facebook, are operating within the EU [HV18]. It supports Ann Cavoukian's idea of *privacy by design* [Cav+09], intending to enforce privacy at an early stage of software development. Considering the classical software life cycle, privacy has to be enforced directly after the requirements have been established, and before the actual implementation takes place.

[1] https://www.forbes.com/sites/daveywinder/2019/09/05/facebook-security-snafu-exposes-419-million-user-phone-numbers/?sh=632b25cc1ab7

[2] https://policies.google.com/privacy?hl=en-US

1.1 Motivation and Research Questions

There are different approaches to implement privacy by design. The GDPR, for example, proposes the use of so-called Privacy Enhancing Technologiess (PETs). These PETs are already existing and "cover the broader range of technologies that are designed for supporting privacy and data protection"[3]. If these technologies are used, certain privacy risks can be mitigated or completely eliminated. PETs serve their purpose effectively, if the software developer applies them in the right context.

Another way to implement privacy into software systems is to apply *privacy patterns* which "are design solutions to common privacy problems—a way to translate privacy by design into practical advice for software engineering" [DGZ15].

Beyond PETs, a pattern is designed to present the software developer a solution to a privacy problem in a specific context. A pattern is composed of a context, a problem and a solution, and can involve a PET as a part of it.

Hoepman introduced Privacy Design Strategies to serve as a guideline for building privacy-friendly software systems, including regulatory requirements [Hoe14].

In the design process of the Software Development Life Cycle (SDLC), the fundamental structure of the software system is significant, also referred to as software architecture. Traditional architectural patterns are proven reusable solutions to recurring problems in software architecture within a specific context. Definitions of a software architecture are vague and ambiguous (see Chapter 2). What constitutes a software architecture, how it is defined in this thesis, and where the aforementioned privacy by design techniques come into play is explained in Chapter 4.

This thesis addresses the following two research questions: Which architectural patterns exist that implement privacy requirements in a software architecture? How can these patterns be selected in a given application context?

In the first place, a new class of architectural patterns is defined, which is built similar to traditional architectural patterns of software design. Patterns of this kind are then identified—the privacy-preserving architectural patterns. These patterns are designed to address recurring privacy problems, describe elements of a software architecture and their interconnection to fulfill privacy requirements. In a subsequent step, a method is presented to assign certain privacy preserving architectural patterns to a given context. For this purpose, the definition of classical architectural patterns is enhanced by elements that represent properties of the multidimensional construct privacy. In the sense of this thesis, the aforementioned PETs can be specific implementations of the proposed patterns. They correspond to architectural tactics, which are used to augment or implement architectural patterns [BCK03].

In summary, this thesis constitutes the following contributions:

- The definition and identification of *privacy-preserving architectural patterns (PPAPs)*
- A methodology for selecting the suitable patterns for a given context

[3] https://www.enisa.europa.eu/topics/data-protection/privacy-enhancing-technologies

1.2 Outline

The remainder of this thesis is organized as follows. Chapter 2 deals with the setting and basic concepts used in this thesis. It elaborates the following terms: Cloud computing, privacy, software architecture, pattern, and tactic. Existing literature in the scope of this thesis is examined in Chapter 3. In this context, research is discussed that either try to engineer privacy, or are relevant for defining the proposed PPAPs. It points out how this thesis differs from existing research. Chapter 4 reflects the main contribution of this thesis. It provides the definition of a PPAP and identifies various PPAPs, using the introduced quality attribute privacy. Chapter 5 presents a methodology, supporting an architect to select the suitable PPAPs in a specific context. A use case, applying the aforementioned methodology, is presented in Chapter 6. Chapter 7 discusses the methodology, both in general and applied to the use case; and identifies limitations. Finally, this thesis is summarized and concluded in Chapter 8.

2 Background

This thesis identifies PPAPs for cloud-based applications, allowing architects to build privacy-friendly software systems. Basic knowledge of relevant terms and concepts is presented in this chapter, providing a better understanding for the remainder of this thesis.

The first section discusses the complex term privacy and its usage in this thesis. Addressing privacy in the cloud is illustrated in *Computing and Privacy*. The characterization of a software architecture is covered in the last section, *Software Architecture*. In addition, methods that are used for designing a software architecture are described, including architectural patterns.

2.1 Privacy–a Multidimensional Concept

The meaning of privacy has evolved from its classical sense and has become more complex in the digital world. Defining privacy as "the right to be left alone" [BW90], today's interconnection of the Internet is no longer sufficiently covered. The challenge nowadays is to ensure the privacy of individuals operating in the digital world. Bélanger and Crossler review multiple definitions of privacy, concluding that "privacy refers to the desire of individuals to control or have some influence over data about themselves" [BC11].

Security is commonly categorized in confidentiality, integrity and availability of data. While it relates to all information that is used by a software system, privacy, on the other hand, addresses personal information. As of today there is no universal definition of privacy, because there are various views of what privacy is and what the consequences of non-compliance are. Van Rest et al. states that "privacy is a broad, abstract, and subjective concept" [RBE+12]. The viewpoints may consist of cultural privacy concerns or national regulations [BJKL04]. Different perspectives are outlined in the following.

Companies are developing their own privacy strategies. This can have a positive impact on their image, or a negative impact on market shares by non-compliance [AFT06].

In the literature, many definitions and characterizations are proposed defining (information) privacy. Solove claims, most definitions being inaccurate: They define privacy either too broad or too narrow [Sol02]. He proposes a taxonomy for privacy, trying to avoid people's lack of clarity what it is [Sol05]. Figure 2.1 illustrates his classification of privacy into four places (*activities*), where privacy can be violated: At the data subject (corresponds to the activity invasion), on the way from the data subject to the data holder (information collection, e.g. surveillance or interrogation), at the data holder (information processing), and from the data holder to third parties (information dissemination).

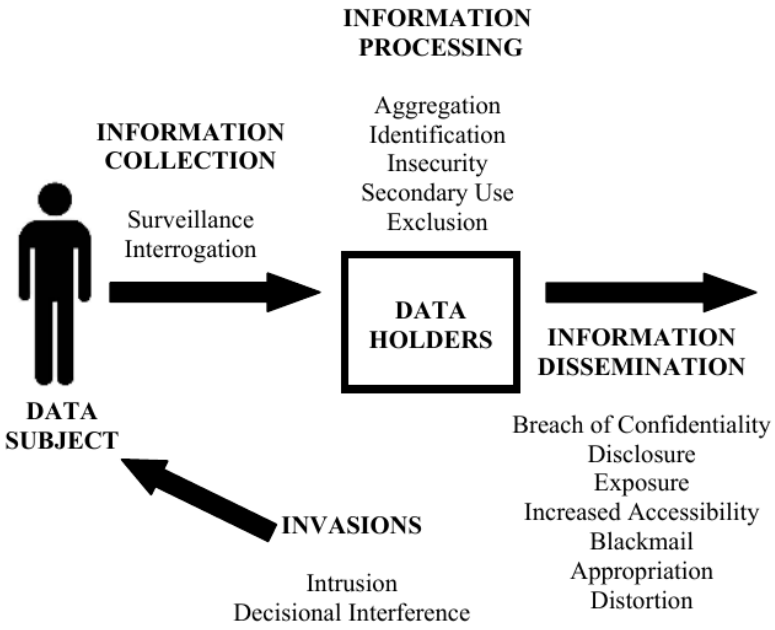

Figure 2.1: Solove classifies privacy into groups where privacy can be violated: *Information collection*, *information processing*, *information dissemination* and *invasion* [Sol05]

Privacy is defined on the legal basis as well, and more privacy stricter regulations emerge. To give their citizens more control over their personal data, the European Union (EU) introduced and implemented the GDPR [EU16]. Personal data is any information relating to an individual, who can be directly or indirectly identified. In GDPR terms (and in the taxonomy of Solove, see Figure 2.1), *data subjects* represent these individuals. A *Data controller* decides on the further processing of personal data. The third entity within the GDPR framework, the *data processor*, processes personal data on behalf of the data controller.

One part of this regulation lists 8 privacy rights every citizen (data subject) has, which every company, dealing with personal data from within the EU, has to ensure. These comprise the right to be informed, - of access, - of rectification, - to erasure, - to restrict processing, - to data portability, - to object and the rights in relation to automated decision making and profiling. In addition, they demand companies to implement the *privacy by design* and *privacy by default* approaches (Article 25). According to Article 5 of the GDPR, companies have to ensure: They only use and collect the amount of data that is needed for the purpose of the business (*data minimization*), the processed data is relevant to the purpose communicated with the data subject (*purpose limitation*), personal data is only stored as long as it is necessary (*storage limitation*), and state of the art security tools are applied (*integrity and confidentiality*).

Ann Cavoukian is considered to be the person who first coined the term privacy by design [Spi12]. In her paper, “Privacy by design: The 7 foundational principles”, she proposes a framework for proactively embedding privacy into the software development process from early on [Cav+09]. The principles are: *Proactive not Reactive*, *Privacy embedded into the system*, *Privacy by default*, *Full Functionality*, *End-to-End Security*, *Visibility and Transparency* and *Respect for User Privacy*.

Privacy by design, according to the GDPR, corresponds to the principle of *Privacy embedded into the system* in the work of Ann Cavoukian. It is the term for enforcing privacy from the beginning of software development. *Privacy by default*, which is the same in both the GDPR and Ann Cavoukian's work, represents the implementation of privacy as a standard, i.e. users (data subject in the GDPR case) do not have to apply their own privacy settings or techniques—the system has to be designed for it.

Privacy Enhancing Technologies

The GDPR requires companies to discover existing tools for privacy protection. To realize the privacy by design concept, the GDPR advises to use PETs. PETs are technologies, specifically designed to increase privacy in the system. Borking et al. define it in this way: "PETs have been defined as a coherent system of ICT [Information and Communications Technology] measures that protects privacy by eliminating or reducing personal data or by preventing unnecessary and/or undesired processing of personal data; all without losing the functionality of the data system" [BR01].

In terms of the Privacy by Design Framework and the GDPR, PETs are the tools of choice for building software systems that support privacy. Like privacy, there is no universally accepted definition for PETs. It is mostly dependent on and defined by legislative requirements. For example, the European Union Agency For Network And Information Security provides a guideline in terms of a *PETs controls matrix*[1]. PETs are categorized in *secure messaging*, *virtual private networks*, *anonymization networks* and *anti-tracking tools for online browsing*.

Common PETs are encryption, use of pseudonyms, differential privacy, data tokenization and onion routing (e.g. using the The Onion Routing (Tor) network) [APT13; BRD+15; Dwo08; GRS99; PK01]. Encryption, for example, is a highly effective tool to protect personal information from unauthorized access. Applying the current standards, it is impossible to decrypt the ciphertext in reasonable time without having the private key.

Privacy Classification

The scope of PETs is as broad as software systems themselves. Consequently, the way they work varies significantly. This is due to reasons of privacy in itself: It is complex and multidimensional. A single PET is designed to address specific parts of privacy and cannot solve privacy as a whole. In terms of the GDPR, privacy can be grouped into the 8 privacy rights listed above, and the *data protection principles* (Article 5): *Lawfulness, fairness and transparency*; *purpose limitation*; *data minimization*; *accuracy*; *storage limitation*; *integrity and confidentiality*; and *accountability* [EU16]. Heurix et al. categorize privacy into 7 different dimensions, where each dimension represents a property that can be used to distinguish PETs from one another [HZNF15]. For example, one dimension addresses the state of data: One type of PET addresses data located on the server (e.g. encryption), other PETs address data in transit (e.g. TOR), and others protect data being processed (e.g. homomorphic encryption (HE), allowing to perform calculations on encrypted data).

[1] https://www.enisa.europa.eu/publications/pets-controls-matrix/pets-controls-matrix-a-systematic-approach-for-assessing-online-and-mobile-privacy-tools

Privacy is not uniquely defined and depends on multiple factors, e.g. the legal situation. The trend is to incorporate privacy from early on in software development. The PPAPs proposed in this thesis realize this idea, using patterns to build a software architecture with privacy in mind from early on (software architectures and patterns are explained in Section 2.3).

This section describes privacy as a complex and multidimensional construct. In Chapter 3, relevant papers are investigated for the categorization of privacy. The definition of a PPAP, presented in Chapter 4, is constructed to reflect this aspect. The PETs presented above are proven solutions for certain specific privacy problems. A PET represents a concrete realization of a PPAPs solution, and hence PETs are incorporated in Chapter 4 as well. The categorization of privacy plays an important role in Chapter 5, where a methodology is developed for building software systems. It takes into account the different properties of privacy.

2.2 Cloud Computing and Privacy

The scope of this thesis covers cloud-based applications. To show the relevance of privacy in the cloud setting, the necessary background knowledge is given here. The idea of cloud computing is to use IT resources *as a service*, and let the cloud service provider (CSP) manage the IT infrastructure behind it. Managing multiple logically linked cloud servers, the CSP can offer flexibility, efficiency, cost savings and security, that a single entity (a person or a company) may not be able to achieve on its own. The common view of cloud computing, corresponds to the definition of the *National Institute of Standards and Technology (NIST)*[2]. It defines the cloud model consisting of the *essential characteristics*, *deployment models*, and *service models* [MG+11]. The former encompasses that users can manage the cloud resources (*on-demand self-service*), access the cloud with any end device (*broad network access*), adjust resources to the demand very fast (*resource pooling* and *rapid elasticity*), and *measure the service*. Deployment models describe how the cloud is used: As a *public cloud* (anyone can use it), *private cloud* (cloud is addressed to one organization), *community cloud* (multiple organizations use it together), or a *hybrid cloud* (private and public clouds are used simultaneously). The NIST distinguishes between three service models: *Software as a Service (SaaS)*, *Platform as a Service (PaaS)* and *Infrastructure as a Service (IaaS)*. In ascending order, they indicate the degree of responsibility the cloud consumer occupies. In SaaS, the user has at most influence on configuration settings. On the other side of the spectrum (IaaS), the cloud user can develop on the operating system level. It is also possible for a customer to use IaaS or PaaS to create an application which is in turn offered as SaaS to other (end) users, making the customer a service provider. An example of today are the Amazon Web Services (AWS)[3]. Plenty of companies outsource their business to AWS. The top four companies with the largest payments to AWS are *Netflix*, *Twitch*, *LinkedIn* and *Facebook*[4]. In terms of privacy, the CSP (AWS in this example) is usually not free of legal requirements. Independent of the chosen model, privacy plays an important role, because end users trust the service provider with their personal data. In the case described above, a service provider itself is a customer of a CSP, and in turn trusts the CSP. The latter case

[2] https://www.nist.gov
[3] https://aws.amazon.com/
[4] https://www.contino.io/insights/whos-using-aws

implies, that an end user has to trust both: The service provider and the CSP. In addition, an end user usually shares the service with other end users, who can thus also gain insight into personal data.

Examples range from using the cloud as a data storage, to the use of a complex application. A complex application can include location determination, payment system, interaction between clients, etc. In addition, the cloud's enormous computing power is exploited to analyze big data, mostly affecting personal data. In the application of smart homes and smart cities the cloud can serve as a control center. Regarding the former, the majority of data is inherently sensitive. All mentioned examples include personal data, which can be abused by a service provider, or even by third parties.

Due to the enormous increase in the number of IoT devices (see Chapter 1), new technologies, extending the cloud, have been developed to reduce the workload on the cloud: *Edge computing* and *fog computing*. Edge computing constitutes the principle of moving data storage and processing toward the end devices, e.g. directly at the end devices. In fog computing, parts of the service are moved to the LAN of the end devices by so-called fog nodes [LGL+15]. They usually have more computing power than end devices, but less than the cloud. The idea is to combine the enormous computing power of few cloud servers with the computing power of many devices (fog nodes and end devices). Moving data toward end devices can have a positive impact on privacy, as these technologies enable to keep personal data more towards the user's side.

This section provides an overview of basic cloud terms and clarifies privacy is not to be neglected in the cloud. These considerations contribute to the understanding necessary for examining the related work in the next chapter. The terminology given in this section is used in Chapter 4, which outlines the scope of the cloud in which the thesis is located.

2.3 Software Architecture and Patterns

The core of this work is the definition and identification of PPAPs. This section shows the location of architectural patterns, which include PPAPs, in the software systems. Definitions and properties of a software architecture and architectural patterns are examined.

As in the traditional sense of architecture, a software architecture reflects the fundamental structure of the system. Accordingly, the construction of a software architecture takes place early in the development process. Various models for software development have emerged over the last decades, e.g. the traditional waterfall model, incremental models, or new agile models (for instance the Scrum Framework) [Rup10]. Despite different ways of execution, they all share the principle of the *Software Development Life Cycle* (SDLC). It divides the development of software into 6 phases: Requirement analysis, design, development, testing, deployment, and maintenance. *Cycle* underlines the on-going character of a software system, each phase being executed multiple times.

The SDLC starts with the analysis of the requirements. Including these requirements, the blueprint for the software system is created in the design phase—the software architecture. Building the architecture involves the participation of the different stakeholders, and is located directly before the development of the system. Figure 2.2 illustrates the 6 phases in the SDLC, highlighting where building an architecture takes place.

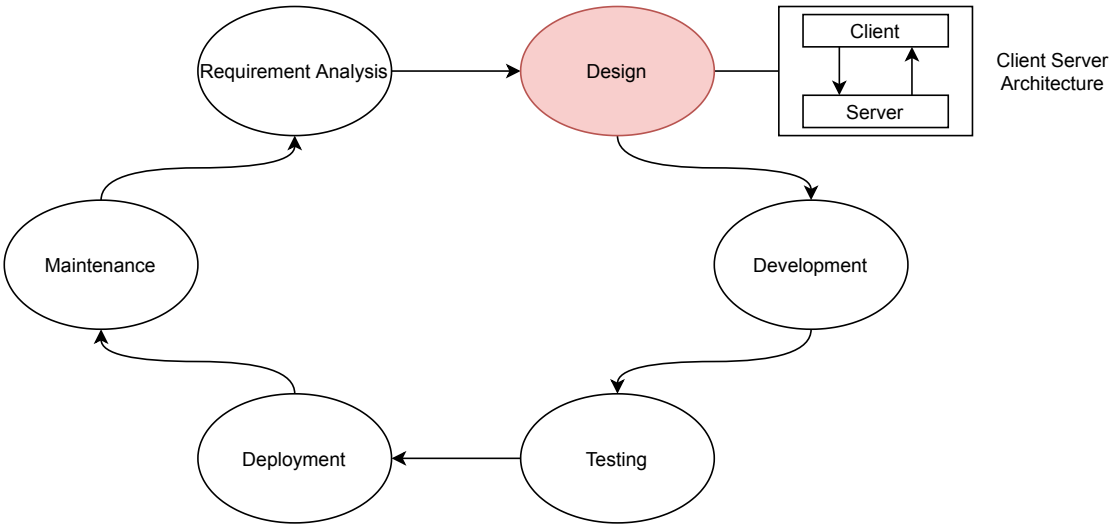

Figure 2.2: Overview of the SDLC: The design phase is highlighted, representing where a software architecture is built.

2.3.1 Software Architecture

There are multiple definitions of a software architecture and views on it's abstraction level vary. The Carnegie Mellon University provides a series of definitions [Uni17].

Bass et al. define software architecture as "the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them" [BCK03]. It implies the architecture being an abstraction of the software system. The definition of Bass et al. builds on another definition that includes "*structures* needed to *reason* about the system"[5].

Another class of definitions focuses on major design decisions: Martin Fowler suggests to incorporate elements representing "important stuff, whatever that is" [Fow03].

This thesis assumes the stakeholders being interested in making their system privacy-friendly, fearing negative effects on their business (see Section 2.1 *Privacy–a Multidimensional Concept*). The stakeholder's perception determines the elements being considered important. Nevertheless, the structures being used to *reason* about the system is important [BCK03].

In the work of Bass et al. the key elements are architecture structures [BCK03]. The authors distinguish between module structure, component-and-connector (C&C) structure, and allocation structure. The module structure is of static nature, whereas an allocation structure deals with linking to the hardware. A C&C structure describes the dynamic behavior of single architectural elements (components) and their relations (expressed by connectors). In this thesis, the C&C view is of relevance, because it represents the flow of (personal) data. *Service C&C structures* is a special form, used to define elements within its structure, providing services that can be consumed by other elements.

[5] No reference in the book is given

With the help of architectural patterns, proven architecture structures can be reapplied, making the development process significantly faster and more error-free. Architectural patterns are discussed in the following subsection.

2.3.2 Architectural Patterns

In the process of creating an architecture, analogies can be drawn from the architecture of building to the software domain. Christopher Alexander introduced architectural patterns with his work "A Pattern Language: Towns, Buildings, Construction" in the year 1977 [Ale77]. An architectural pattern is a template for solving a recurring problem in the same way it has been successfully applied before. The solution is an abstract view on the architectural parts which are fundamental. Applying one pattern many times always results in different concrete realizations [Ale77]. The usage of patterns has found its way into the software world, originated in the work of the so-called *Gang of Four (GoF)* [GHJV93]. They propose *design patterns*, depicting best-practices for programming in object-oriented languages and have been acknowledged by the software development community. In the course of time, patterns have established themselves in the field of software architecture. A software architecture pattern is a reusable solution to a recurring problem, within the context of software architectures. Bass et al. define it as follows: "An architectural pattern is a package of design decisions that is found repeatedly in practice, has known properties that permit reuse, and describes a class of architectures" [BCK03]. In contrast to the aforementioned design patterns, architectural patterns provide a more abstract view of the system. What the degree of abstraction for the resulting architectural structure is, depends on the definition of a software architecture. The simplest structure of a pattern divides it into the following: The *context* of the problem, a *problem* formulation and a *solution*. For more clarity the pattern can be extended by further elements, e.g. variants showing different implementations or sketches guiding the solution graphically. Common architectural patterns are the *Layered Pattern*, *Model View Controller Pattern* and *Client Server Pattern*. The latter is also used in this thesis for identifying PPAPs, because it reflects the fundamental operation of a cloud: Offering a service to multiple clients, while the service provider represents the server.

The client-server architecture includes the components of a server and one or more clients. The server and the respective client are connected through their ports (connectors). One type of connector consists of an HTTP connection in TCP/IP networks, as is common in the World Wide Web. A Server provides a service, a client can make use of. If a client wants to use the service of a server, it sends a request for the service. The server is on standby and responds to the client's request. Figure 2.3 illustrates the simplified form of a cloud-based application in the client-server model, representing a service C&C structure.

Bass et al. introduces *architectural tactics*, being less abstract than a pattern and usually representing a single component of an architecture [BCK03]. They can be used to create and extend architectural patterns, or can act as a concrete implementation. The authors define tactics with respect to achieving quality attributes, mostly representing the building blocks an architectural pattern consists of, and addressing single forces of a pattern. In this thesis, architectural tactics mostly represent the aforementioned PETs. Thus, a PET can act as a concrete implementation of a pattern or augment a pattern.

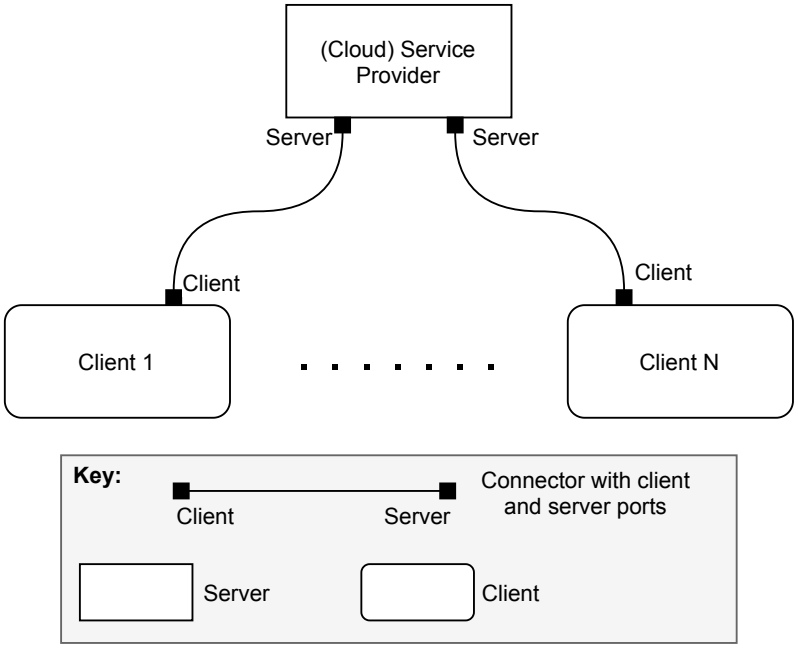

Figure 2.3: An example client-server architecture of a cloud-based application.

It is unusual to speak of inventing patterns. Instead they are discovered or identified. Patterns take up established techniques, i.e. solutions that have already been introduced and used several times, and package them as patterns. These proven solution approaches aim at simplifying similar processes for others in the future. This holds true for all kinds of patterns and tactics discussed so far.

2.3.3 Quality Attributes

To evaluate and design a software architecture, quality attributes can be considered. Quality attributes are characteristics of a software system that are used by relevant stakeholders to evaluate the quality of the system [BCK03]. They investigate 7 quality attributes for the design and analysis of a software architecture: *Availability*, *interoperability*, *modifiability*, *performance*, *security*, *testability*, and *usability*. The quality attributes are divided into categories, classifying architectural tactics, which support the achievement of quality attributes.

The security quality attribute, for example, is divided into the following categories: *Detect attacks*, *resist attacks*, *react to attacks* and *recover from attacks*. It is depicted in Figure 2.4, accompanied with some of the corresponding architectural tactics.

An architectural tactic can potentially fall under multiple categories of different quality attributes, i.e. fulfilling multiple quality attributes to a certain degree. E.g., the architectural tactic corresponding to the PET *encryption* achieves the privacy as well as the security quality attribute (within the *resist attacks* category). This thesis takes the view of Bass et al.'s work, describing tactics as individual parts of patterns. A pattern consists of several tactics, or tactics are used to specify existing patterns more precisely. Tactics are not written on a pure code level, and distinguished from design patterns proposed in the work of the Gang of Four [GHJV93].

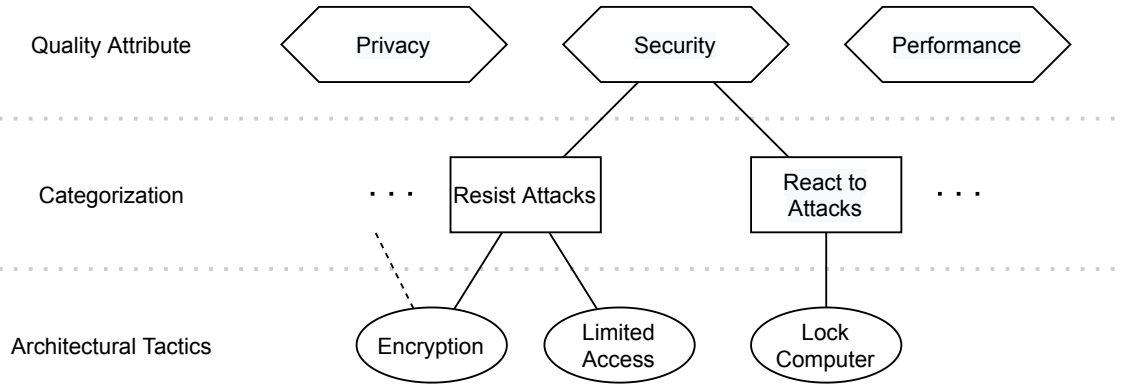

Figure 2.4: The relationship between quality attributes and architectural tactics.

2.3.4 Deriving a New Quality Attribute

Bass et al. provide guidelines to develop a new quality attribute, presented in the following and illustrated in Figure 2.6 [BCK03].

First, the characterization describing the quality attribute should be identified in a dialog with the stakeholders. E.g., the quality attribute security may be characterized by confidentiality, integrity and availability. Based on this, specific scenarios are created, accurately describing the quality attribute. From these scenarios the general scenario is built. A scenario consists of the following parts: The *source of stimuli* indicates where the stimulus originates, the *stimuli* represents a quality attribute violating event that requires and triggers a countermeasure, the *artifact* and the *environment* define which system element is affected and in which process the stimulus arrives, the *response* is the countermeasure to the stimulus, and the *response measure* determines the effectiveness of the response. Figure 2.5 illustrates a specific scenario of the security quality attribute.

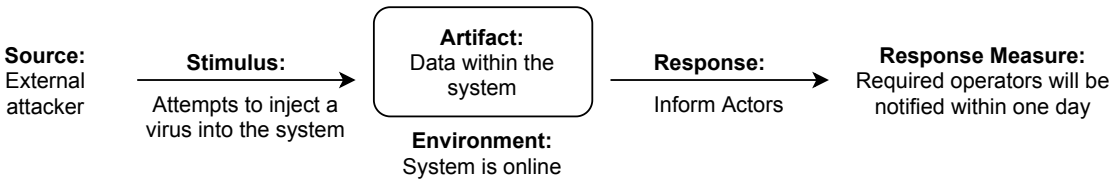

Figure 2.5: Specific scenario for the quality attribute security in the category *react to attacks*.

After the general scenario is captured, individual solution approaches are collected. E.g. existing patterns are examined (design or architectural patterns, dealing with the new quality attribute), experts are consulted, or the general scenario created earlier is used. When these individual solution approaches are found, a model can be created. It describes the parameters, influencing the quality attribute. Architectural tactics and patterns can be identified by using this model, consulting experts, or examining existing systems and literature that are assumed to address this quality attribute. Finally a checklist with 7 categories is created: *Allocation of responsibilities*, *coordination model*, *data model*, *management of resources*, *mapping among architectural elements*, *binding time decisions*, and *choice of technology*.

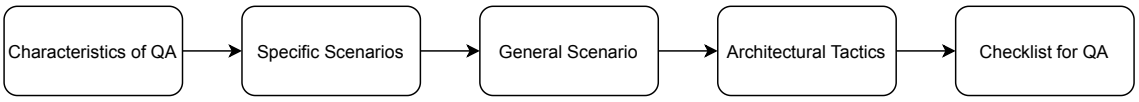

Figure 2.6: Deriving a new quality attribute (QA) according to Bass et al. [BCK03].

This section introduces different views of a software architecture and architectural patterns. In Chapter 4, these existing views are used to form the valid definitions of a software architecture and architectural patterns in this thesis. The usage of different quality attributes for designing a software architecture is depicted. An approach for deriving a new quality attribute is shown. Privacy is neither considered as quality attribute as a whole, nor as part of security. In Chapter 4, the privacy quality attribute is derived to identify relevant architectural tactics and PPAPs. Since this thesis falls within the scope of cloud-based applications, the C&C *Client Server Pattern* serves as a foundation for each PPAP identified in this thesis.

This chapter provided background information and the technical foundation, being important to understand the content of this thesis. Relevant work in research is discussed in the next chapter, including privacy engineering approaches and different kinds of patterns in the privacy and security domain.

3 Related Work

The foundation for the thesis is established in Chapter 2. Relevant results from literature, concerning privacy and patterns, are examined in the following. Their impact on the definition and identification of the PPAPs are discussed.

The first section discusses work presenting approaches to incorporate privacy in the development process of software systems. This thesis leverages patterns to enforce privacy in software architectures. Relevant work that directly addresses privacy patterns, or that can be exploited to define privacy patterns, is identified in the subsequent section. The results from these two sections pave the way for designing a software architecture, using privacy patterns: The privacy-preserving architectural patterns (PPAPs).

3.1 Privacy Engineering

This section discusses related work aiming at translating privacy requirements into a software system. One way to systematically incorporate privacy goals into the software design is to detect privacy threats in risk models. LINDDUN is a framework for threat analysis of a software system to detect privacy violations in data flows, aiming to enforce privacy at an early stage in the software development [WJ15]. It investigates the following 7 properties of privacy, forming the name LINDDUN: Linkability, identifiability, non-repudiation, detectability, disclosure of information, unawareness and non-compliance. Starting with the modeling of the system by a data flow diagram (DFD), the framework identifies privacy threats via LINDDUN threat categories. They also provide a table of PETs from which the most necessary ones are extracted at the end of the analysis procedure.

The system-level abstraction of DFDs can be leveraged to develop scenarios for deriving the privacy quality attribute and to find architectural tactics in form of PETs. Although LINDDUN is used early in the development process, it is not intended to design software architectures. It aims to find privacy vulnerabilities and suggests PETs to fix them. This thesis uses architectural patterns to introduce privacy from defined privacy requirements on the software system. These patterns provide more abstraction on the system flow than applying PETs (architectural tactics).

Attempts to characterize privacy, without involving a risk model and not addressing a specific domain, are discussed in the following.

Venter and Eloff present a taxonomy for information security, distinguishing technologies between proactive and reactive [VE03]. While proactive technologies secure data before a security breach occurs, reactive technologies are deployed after breaches are detected.

The idea of proactive technologies is reflected in this thesis. Patterns are identified, proactively embedding privacy in the architecture, i.e. without the end user having to do anything.

Policy notices inform the data subject about the collection, processing and disclosure of personal data [CRC05]. Milne et al. conducted a study of over 2400 participants, examining their reactions to privacy policy notices on the Internet [MC04; MCG06]. One of the conclusions is that participants are annoyed with long complicated privacy notices, and therefore demand to work further on improving the privacy policy notices.

Despite privacy policy notices providing a solution to enhance privacy, they do not constitute a proactive approach. This thesis solves the problem of collecting, processing and disclosing too much personal data in advance—with proactive architectural patterns, following the privacy by default principle (presented in Chapter 2).

Spiekermann and Cranor propose a framework for allowing the design of privacy enhancing systems [SC08]. To make the system more privacy-friendly, they distinguish between measures of two types: *Privacy-by-policy* and *privacy-by-architecture*. The former comprises measures, giving users a certain amount of control over their data processing. This is usually implemented through the *Notice and Choice* approach, derived from the fair information practices (FIPs). FIPs are privacy principles, recommended by the Organisation for Economic Co-operation and Development (OECD), as a basis for discussing and establishing privacy requirements. However, the highest degree of privacy can be achieved through privacy-by-architecture. This approach enforces privacy directly in the software architecture, using architectural elements that are composed of privacy-preserving technologies. Their framework is divided into privacy stages from zero to three. The higher the privacy stage is, the less identifiable individuals are, when linking their stored personal data to their identity. Measures including privacy-by-policy can only cover the first two privacy stages, 0 and 1. Privacy-by-architecture, on the other hand, can fulfill stages 2 and 3, reflecting the authors' statement that (only) this approach can provide the highest level of privacy. The authors claim that the principle of privacy-by-policy is used too often in practice, because it is easier to implement. But privacy-by-architecture is always preferable if possible. Their basis for choosing appropriate architecture measures consists of two properties: *Network centricity* and *identifiability*. The former indicates whether the personal data is mostly processed on the server side (*network-centric* architectures), or on the client side (*client-centric*).

Even if the authors do not explicitly state it, their framework represents the principles of privacy by design and privacy by default (introduced in the GDPR afterwards), the PPAPs are also built on. Applied to the privacy pattern domain, this thesis adopts the view of differentiating between privacy-by-policy and privacy-by-architecture. The preference of implementing the privacy-by-architecture over the privacy-by-policy principle is followed. Only privacy patterns are identified, directly influencing the software architecture from the outset. Privacy patterns, only providing privacy-by-policy, are excluded. The dimensions of network centricity and identifiability are both considered in this thesis. In the former case, shifting personal data towards the client or the server constitutes a trade-off between privacy and usability. Trade-offs are included in the definition of a PPAP and discussed further in Chapter 6. The "engineering privacy" framework lacks a step-by-step process and automation, that would enable architects to effectively select and implement appropriate measures for designing privacy supporting systems. Chapter 5 presents a methodology, supporting architects to build a privacy-friendly software architecture.

Heurix et al. focus on the categorization of privacy in the context of PETs [HZNF15]. They consider privacy as a construct consisting of several properties. In order to classify and identify common characteristics of PETs, they developed a taxonomy dividing privacy into several dimensions. Each dimension determines a distinct subdomain of privacy a PET can address, and thus specify a different

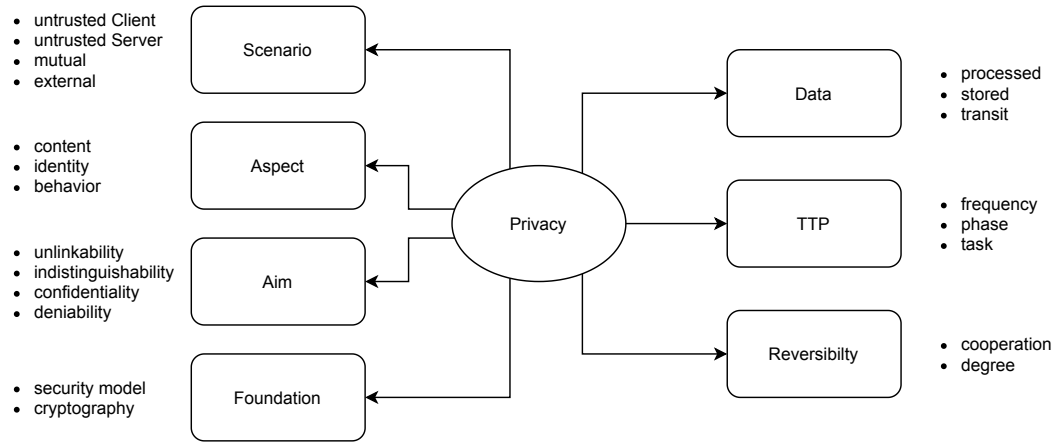

Figure 3.1: Privacy Taxonomy for PETs [HZNF15].

privacy property. Illustrated in Figure 3.1, the taxonomy consists of 7 elements : *Scenario*, *aspect*, *aim*, *foundation*, *data*, *trusted third party (TTP)*, and *reversibility*. These dimensions can span further branches, describing them in more detail. For the sake of clarity, the individual dimensions are briefly described below only in relation to the first child of the respective branch. Relevant dimensions are discussed in more detail in Chapter 4. The *scenario* dimension represents the attacker model, consisting of an untrusted client, an untrusted server, mutual, or an external. The overarching goal is depicted in *aspects*, including the elements of content, identity, and behavior. The *aim* specifies the achievement of the aspects: Unlinkability, indistinguishability, confidentiality, or deniability. *Foundation* specifies the security model, a PET is in, and the applied cryptographic primitives. The *data* dimension specifies the state of data, a PET addresses: Data can be processed, stored or in transit. The second last item indicates if a *TTP* must be involved, and is further distinguished in frequency, phase and task. Finally, *reversibility* states whether the output of a PET can be reversed. It is divided into the degree of reversibility and necessity of the data subjects cooperation.

This thesis follows the assumption of Heurix et al., considering privacy as a multidimensional construct, and implying different PETs as necessary in different application contexts. Since a PET corresponds to an architectural pattern (see Chapter 2), this assumption is applied to architectural patterns in this thesis. To apply the multidimensional aspect of privacy, the taxonomy of Heurix et al. is considered, and its dimensions adapted to the architectural context of this thesis. This chapter introduces the dimensions briefly. Because an architectural tactic provides a different level of abstraction than an architectural pattern [BCK03], the PET-dimensions are not adopted identically. The dimensions *foundation* and *reversibility* are excluded, because this thesis assumes them to be too specific to be mapped in the software architecture. Chapter 4 determines for each adopted dimension and its subdivided properties whether they are applicable, and the extent to which they are incorporated into PPAPs. Various PETs, classified into the taxonomy by Heurix et al., are taken into consideration in form of architectural tactics for PPAPs.

Martinez et al. categorize privacy in the context of the application of PETs in smart cities [MPS13]. They consider the privacy of citizens to be a 5-dimensional construct: *Identity*, *query*, *location*, *footprint*, and *owner privacy*. For each dimension, they identified several privacy issues and associated existing solutions in the form of PETs. The dimensions result from a mapping to one

or both of the following two privacy models. First, they propose a mapping for each dimension to the *3D database model* of Domingo-Ferrer which distinguishes between respondent, user and owner privacy [Dom07]. Second, the W^3 privacy dimensions related to location-based applications of Pérez and Solanas is considered [PS11]. It explores the following three questions, forming the name W^3: Where is the request coming from? Who is the user sending the request? What is the request about?

Chapter 2 presents the guideline for introducing a new quality attribute [BCK03]. The privacy dimensions of Martinez et al. are considered for deriving the privacy quality attribute privacy in the next chapter. The PETs, they found for the different dimensions, are considered in form of architectural tactics, for instance *use of pseudonymizers* and *private information retrieval*.

3.2 Privacy-Related Patterns

Chapter 2 presents software design patterns, representing proven solutions to commonly recurring problems in the software development. They consist at least of the parts *context*, *problem*, and *solution*. Architectural patterns follow the same approach, but are solutions to recurring problems in the context of software architectures. They are characteristically broader in scope than software design patterns [Hoe14]. This section discusses relevant work in the area of patterns that helps define and identify architectural patterns that are consistent with the conclusions made in the previous section.

The section emerges as follows: First, the security pattern domain is examined for patterns that aid in identifying PPAPs. Subsequently, the literature is examined that includes privacy patterns.

3.2.1 Security Patterns

Although in this thesis privacy is not considered as a subset of security, they are related and overlap in certain areas (see Chapter 2). Therefore, security patterns are examined in the following.

Yoder and Barcalow introduced patterns to the information security community in 1997 [YB97]. They proposed 7 security patterns to embed security in the development of applications.

In 2002, Markus Schumacher argued that security is insufficiently considered in the process of designing software (design phase in terms of the SDLC) [Sch02]. His work presents a pattern template, which is designed by taking into account security standards. He advocates the use of security patterns, because they provide solutions being more readable and offer a clearer structure than standards. In this case, the *Common Criteria (CC)* is used, mapping the standards to the pattern elements context, problem, forces and solution. It is an international standard (ISO/IEC 15408) providing best practices for evaluating information systems to determine whether they meet the imposed security standards [MFP07]. Using the proposed security pattern template, Schumacher identifies two security patterns that relate to the privacy domain: *Protection against Cookies* is intended to protect against the persistent storage of cookies, which can contain personal data. Its solution consists of reducing the use of cookies on the web client, e.g. by deleting them periodically. The forces consists of the functional security requirements from the CC: Anonymity, unlinkability and usability. *Restricting cookies* increases the first two (privacy) properties, while

potentially reducing the usability since web sites may not work properly anymore. The second pattern addressing privacy, *Pseudonymous Email*, protects the identity when logging on to websites (in this case an email service) using pseudonyms instead of personal data.

Individual elements of the structure of these security patterns are adopted for the definition of a PPAP in Chapter 4. The approach of deriving patterns from standards is also pursued in this thesis. Current legislation, in form of the legal framework GDPR, is taken into account to define and identify PPAPs. In addition, solutions proven in practice and literature are considered. The first privacy pattern identified by Schumacher is not consistent with the proactive principle of privacy-by-architecture. It represents a solution, the client can implement on its side independently of the system (setting cookies in the web browser). This thesis takes the approach of incorporating privacy requirements as a standard in the system, without the user having to implement or set anything additional. The idea of using pseudonyms, protecting the identity, is included in the pattern catalog presented in Chapter 4.

Schumacher is also involved as a co-author of the book "Security Patterns: Integrating security and systems engineering", following up the idea of his work described above and identifies multiple security patterns [SFH+13]. In addition to including international security standards like the CC, they also consider security information of known software companies, and organizations like the NIST and the SANS (Sys-Admin, Audit, Network, Security) Institute[1]. *System Access Control Architecture Patterns* constitute one group of patterns, addressing the issue of permitting and denying access to different users. The *Limited Access* pattern, e.g., represents a solution, presenting to a user only the currently available functions, while hiding everything for which he has no authorization.

The identified patterns represent architectural patterns, addressing security. As described in Chapter 2, the security and privacy domains overlap when, for example, technologies of the security domain protect personal data. This overlap is found in the book, and certain patterns are incorporated into PPAPs, e.g. *Limited Access*.

3.2.2 Privacy Patterns

Identifying patterns has been incorporated in the privacy domain by various authors. The idea in the security domain to use established standards is adopted. In case of privacy, these are usually given by law.

First, this thesis considers related work, presenting approaches to classify privacy patterns. These considerations help to organize different types of privacy patterns with respect to the level of abstraction and to identify new privacy patterns. Then, related work is reviewed that identifies either general, or domain-specific privacy patterns, e.g., from the IoT domain. They are discussed with respect to the considerations made in Section 3.1 and the level of abstraction being examined in the following.

[1] https://www.sans.org/

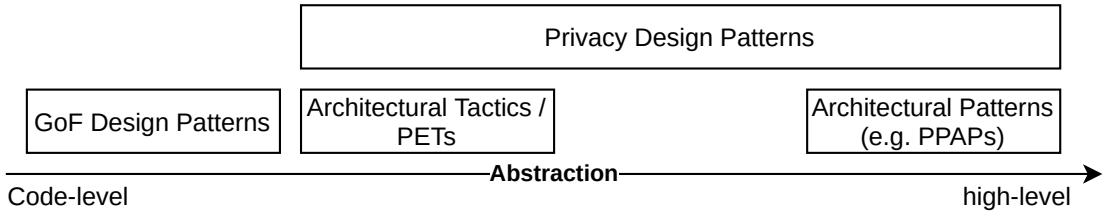

Figure 3.2: Abstraction spectrum: Where PPAPs are located in comparison to GoF's design patterns, architectural tactics, and privacy design patterns.

Classifying Privacy Patterns

Van Rest et al. provide a new privacy by design definition, using the privacy framework of Solove (presented in Chapter 2), and incorporating the EU legislation [RBE+12]. The paper seeks to support the implementation of the general privacy by design scheme into domain-specific applications. Considering patterns from the security and privacy domain, they divide privacy patterns into different categorizations: Sets of patterns related to anonymization and pseudonymization (example patterns are: *Aggregation*, *pseudonymous email*); hiding of personal data (*encryption*); data minimization (no pattern provided); transparency, auditing and accounting patterns (*right of inspection by data subject*); or informed consent (*privacy statement*). For each categorization they present existing privacy patterns, e.g. pseudonymous email for the former. According to the authors, an implementation of a pattern does not have to be existent yet, allowing to identify patterns for which a provable implementation is assumed to exist in the near future. They observe that the difficulty in identifying patterns lies in achieving the right abstraction.

In the spectrum of privacy design patterns, PPAPs try to achieve the highest level of abstraction, allowing the design of privacy-friendly architectures (see Chapter 2). Where PPAPs, privacy design patterns and architectural tactics are located with respect to the abstraction level, is illustrated in Figure 3.2. This thesis follows their *extended privacy by design* definition, involving the legal perspective. The presented privacy patterns by van Rest et al. are covered in other research discussed in this chapter—the authors do not identify new ones. They represent lower-level patterns than architectural ones, mostly representing architectural tactics (e.g. PETs). Of these patterns, those that follow the principle of privacy-by-architecture are considered, e.g. encryption. The claim, pattern implementations do not have to exist yet, is adopted in the form of architectural tactics in the definition of a PPAP.

Stating privacy must be enforced at the core of the system and should not be implemented as an additional feature, Hoepman proposes 8 privacy design strategies applying the privacy by design principle earlier in the SDLC than privacy design patterns [Hoe14]. He considers IT systems as information storage systems, modeled by a database metaphor. The 8 strategies comprise: *Minmise, hide, separate, aggregate, inform, control, enforce,* and *demonstrate*. Because these strategies have more abstract goals, they can serve as a characterization of privacy design patterns. A pattern can be included in multiple strategies, e.g. the *use pseudonyms* pattern [PH10] follows both the *hide* and *minimise* strategy. To develop these strategies, current legislation and frameworks that affect privacy are examined: OECD Privacy Guidelines, the European legislation, and the ISO 29100 Privacy Framework. Although privacy design patterns occur in the design phase, privacy by

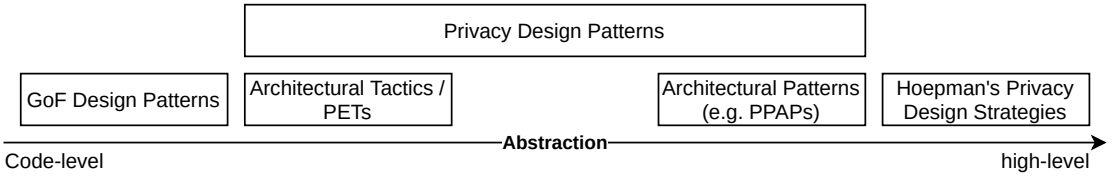

Figure 3.3: Abstraction spectrum: Where PPAPs are located in comparison to GoF's design patterns, architectural tactics, privacy design patterns, and Hoepman's privacy design strategies.

design, according to the author, starts earlier than the use of detailed patterns. Hoepman argues that strategies can be used to classify privacy design patterns and their associated PETs. Several privacy design patterns are identified with the help of these strategies, e.g. *attribute-based credentials*.

As illustrated in Figure 3.2, privacy design patterns vary in their abstractness. This thesis assumes many privacy design patterns to occur late in the design phase, implying less abstraction. Hoepman's strategies are close to a PPAPs degree of abstraction. These strategies represent higher-level goals, not providing architectural elements—and not presenting any kind of patterns. This thesis tries to realize higher level goals of privacy and still retain the structure of architectural patterns, embedding privacy into the software architecture. Figure 3.3 illustrates their location at the end of the privacy patterns abstraction spectrum, and before Hoepman's strategies. Four of the strategies follow the principle of privacy-by-architecture: *Minimise*, *hide*, *separate* and *aggregate*. A pattern is considered a privacy-by-architecture pattern, if it follows at least one of these four strategies. This thesis follows Hoepman's assumption that privacy should be incorporated from the outset, emphasized by the introduction of a new quality attribute privacy, considering his strategies for its derivation. Presented patterns are incorporated into PPAPs, e.g. attribute-based credentials. In this thesis a cloud model is assumed to be more complex than an information storage system. Multiple clients, e.g., use the cloud-based service simultaneously, whereas each client forms a separate part of the software architecture. PPAPs aim at protecting privacy, keeping in mind the multiple dimensions of privacy.

Colesky et al. claim that Hoepman's privacy design strategies are too abstract to incorporate legal prerequisites into the development of software systems [CHH16]. In the work of Colesky et al., privacy tactics are proposed as a further categorization for patterns, being less abstract than Hoepman's strategies. The idea is to make privacy by design more practical than using privacy design strategies. Privacy tactics are located between privacy design patterns and strategies from an abstraction point of view: While strategies reflect the goal of the architecture to achieve privacy by design, tactics give a more concrete way to achieve this goal and provide more abstraction than patterns. Tactics have the purpose to fulfill certain quality attributes (see Chapter 2). In Colesky et al.'s work, the quality attribute of concern is privacy. Referencing the work of Bass et al. [BCK03], they categorize privacy using Hoepman's Strategies. Per strategy, they incorporate up to four tactics extracted by examining over 100 Privacy Design Patterns. Like strategies, tactics are applied to classify privacy design patterns, being "an approach to privacy by design which contributes to the goal of an overarching privacy design strategy". The strategies initially defined by Hoepman are reviewed and their direct relationship to GDPR principles is shown.

Viewing privacy as a quality attribute is considered in this thesis. For identifying PPAPs, the approach of Bass et al. is adopted, including Hoepman's strategies and legislative requirements. In contrast to the work of Colesky et al., Chapter 4 presents the derivation of the privacy quality attribute, including respective scenarios (presented in Chapter 2). While both groups of authors

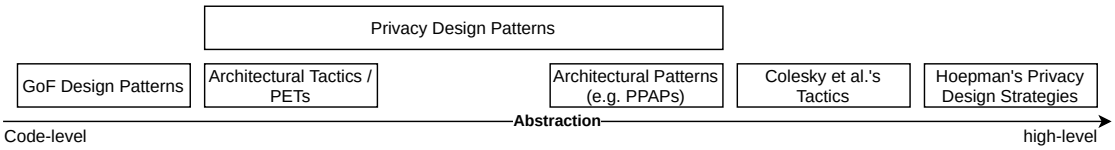

Figure 3.4: Abstraction spectrum: Where PPAPs are located in comparison to GoF's design patterns, architectural tactics, privacy design patterns, Hoepman's privacy design strategies, and Colesky et al.'s tactics.

regard tactics as a tool to fulfill quality attributes, they differ in their application: While Colesky et al. consider tactics as primary goals for patterns, Bass et al. consider *architectural tactics*, constituting elements of an software architecture. Instead of tactics being categorizations for patterns, like in the work of Colesky et al. [CHH16], this thesis uses architectural tactics as atomic building blocks for an architectural pattern [BCK03]. Figure 3.4 highlights the differences between tactics and architectural tactics with respect to the level of abstraction: Tactics according to Colesky et al. are located between Hoepman's strategies and (architectural) patterns. In contrast, architectural tactics, as used in the work of Bass et al. and this thesis, are on the lower end of the privacy design pattern abstraction spectrum.

General and Cloud-based Privacy Patterns

Pearson examines challenges encountered when trying to offer privacy-friendly cloud services [Pea09]. One of these approaches are privacy impact assessments (PIAs), representing a risk estimation through a systematic analysis of data processing in terms of privacy. Privacy design patterns is another way for applying privacy in the cloud domain. Pearson states that they provide guidelines through their template structure. To incorporate privacy design patterns in the cloud computing domain, she argues that more work needs to be done in this area. She claims the usage of PIAs is generally preferable to privacy design patterns, because they consider more of the subtleties of privacy in relation to individual contexts of the system.

By identifying PPAPs for cloud-based applications, this thesis advocates the view of focusing on privacy patterns for the cloud domain. The problem of patterns not taking enough account of the individual characteristics of the system is mitigated in this thesis: Chapter 5 proposes a methodology, constructed to incorporate contextual elements of PPAPs with the help of the architect.

Referring to the work of Schumacher [Sch02], Romanosky et al. identify three new privacy patterns in the context of web-based applications [RAH+06]. They follow the same assumption made in this thesis: Security in the form of security patterns does not solve privacy problems, or at least not directly. The identified patterns are intended to improve the control of users over their personal data, which corresponds to the definition of privacy given in the previous chapter. The problem patterns have in general, Romanosky et al. claim, is reflected in privacy patterns as well. They try to solve all of the different sub-problems given in the problem part (called forces in the pattern world), but usually succeed only in the form of compromises.

This thesis identifies patterns related to privacy as well and has a similar structure. The same view is taken that patterns in the security domain include privacy only to a certain degree. As in the work of Romanosky et al., privacy is seen as the main driver for identifying patterns. The idea

that different forces lead to trade-offs is considered in the PPAPs in the Consequences element as trade-offs between the quality attributes. No patterns are presented that are relevant for the software architectures of a cloud service. Instead, they are solutions on the web client only, for browsing the Internet with privacy in mind. Nevertheless, the underlying techniques such as anonymizing the IP, are embedded in the PPAPs in Chapter 4.

Hafiz identifies four privacy design patterns for software systems, providing anonymity in various domains like online communication [Haf06]. For categorizing the patterns with respect to privacy, he takes into account the proposed anonymity properties and the attacker model of Pfitzmann and Waidner [PK01], as well as the degree of anonymity presented by Reiter and Rubin [RR98]. Relationships are built in between the patterns, guiding the developer in using the appropriate ones. In a subsequent work, Hafiz introduces a pattern language [Haf13]. Consisting of 12 identified privacy patterns (including the four patterns from his previous paper), the pattern language is intended to create new PETs. Two example patterns are: *Oblivious transfer* or *Pseudonymous Identity*. *Oblivious transfer* allows a client the exchange of data without the other participant (server or another client) not knowing what data is received. The latter one is a more general pattern of the *Pseudonymous Email* pattern proposed by Schumacher [Sch02], using pseudonyms to protect the identity of a client.

The identified privacy patterns by Hafiz are architectural and follow the principle of privacy-by-architecture. They are not designed to cover the area of cloud computing. Pearson claims that further work would be needed to develop and assess the efficacy of new privacy design patterns tailored to different types of cloud scenario [Pea09]. In identifying PPAPs that address cloud-based applications, this work considers Hafiz' patterns and adapts them accordingly. The idea of identifying patterns that are related to each other is adopted in this thesis and incorporated into the definition of a PPAP. Instead of identifying patterns for the creation of PETs [Haf06], PPAPs are intended to support the design of privacy-friendly architectures. As presented in Chapter 2, PPAPs can include PETs in the form of architectural tactics, suggesting possible implementations of an architectural pattern.

Taking into account the Microsoft developer privacy guidelines, Pearson and Benameur identifies privacy design patterns and propose a decision support system (DSS) [PB10]. They identify privacy patterns with the goal to improve the software development process by implementing privacy from the outset. The typical pattern structure is extended with the *Classification* element, used for automated analysis of the patterns' relations. The DSS aids developers in implementing these patterns early in the development process. For selecting the appropriate patterns, the developer is asked about the specific applications context. Depending on the user's selection of answers, the DSS outputs the patterns to be implemented. The system provides developers a tool that aims to allow them to implement privacy without much prior knowledge about privacy technologies, and without having to tediously browse through privacy guidelines. All identified patterns follow the principle of privacy-by-policy. The authors argue that privacy design patterns that adhere to the privacy-by-architecture principle can still be introduced into their system later on. In a similar approach, Pearson and Shen propose a rule-based selection system that outputs respective privacy patterns using only contextual information [PS10]. The following elements are added to the typical pattern structure: *Applicable Context* (depicting the contextual environment the pattern can be used in), *Selection Rule Repository* (the location where the rules are stored) and *Selection Rules* (the name of the rules to be applied). Example for the former are: *Sensitivity of data*, *Location of (stored) data*, or *Number of users of system*. The element *classification* is added as well, but not

considered for the proposed system. For allowing the system to provide the developer with potential patterns, a domain expert has to create patterns first. For each pattern, the *applicable context* and corresponding selection rules are determined. The rule names are added to the respective pattern element, and the entire rules are placed to a separate *rules repository*. The system interacts with the developer, e.g. asking about the number of clients using the service, or whether the data contains sensitive information. The *Pattern Selector* determines with the contextual information given by the developer, and the specified rules from the rules repository, the potential patterns that can be implemented. Pearson and Shen identifies two patterns for demonstration purposes: *Obligation Management* and *Sticky Policies*. The item *Sensitivity of data* is included in the *Applicable Context* element of both patterns. One of its corresponding rules states, that the use of the former design pattern (DP1) is preferred, if the developer sets the *sensitivity of information* option to one in the context evaluation phase.

The idea of incorporating privacy properties as pattern elements is adopted in this thesis. Discussions from the first section are applied, e.g. the privacy dimensions of Heurix et al. From the discussions in the first section, it is concluded that this thesis only considers patterns, proactively embedding privacy into the software architecture. The identified privacy patterns of Pearson et al. are of the privacy-by-policy type, and therefore not considered for identifying PPAPs [PB10; PS10]. Pearson et al. provide a decision support system for selecting privacy-by-policy patterns arguing that privacy-by-architecture patterns can be added to their system at a later stage [PB10]. This thesis assumes that this justification does not follow one of the major assertions from the work of Spiekermann and Cranor: First applying privacy-by-architecture and only then privacy-by-policy [SC08]. Pearson et al. provide systems for pattern selection, either considering privacy properties [PB10], or the context of the application [PS10]. Chapter 5 presents a selection procedure for PPAPs, considering the context, without the need to go through guidelines, and considering privacy properties. It includes the expertise of the architect, because this thesis assumes the privacy-by-architecture approach to involve more complex approaches than their privacy-by-policy counterpart. The requirement of an architect's expertise is not assumed to constitute a drawback, since the design process involves the presence of an architect.

Graf et al. present a catalog consisting of 12 identified privacy patterns for PETs [GWGT10]. These constitutes user interface (UI) patterns that can be used in PETs in the areas of interaction, privacy policies, and visualization. The *Dynamic Privacy Policy Display* pattern, e.g., falls under the second area of privacy policies. It displays a privacy disclaimer when the mouse pointer is moved to a specific area of the screen, e.g. near a login button.

The identified privacy patterns are neither architectural, nor do they follow the principle of privacy-by-architecture. Similar to the work of Hafiz [Haf13], the patterns are designed for PETs and not for enhancing the software architecture. Consequently, this thesis does not consider them for identifying patterns.

Inspired by the Alexander's building architectural patterns, the software design patterns from the GoF, and the privacy by design principle, Doty and Gupta have identified privacy design patterns to facilitate the implementation of privacy into software [DG13] . They collaborate with others on a project that serves as a documentation for privacy patterns and provide a standardized privacy pattern language [DGZ15]. The catalog of identified patterns is publicly available at https://privacypatterns.org/, including over 70 patterns at the time of writing. The individual elements of a pattern are: *Summary*, *context*, *problem*, *solution*, *consequences*, and *examples*.

Two example privacy patterns are: *Protection against Tracking*[2] and *Onion Routing*[3]. The former corresponds the aforementioned *Protection against Cookies* pattern proposed by Schumacher [Sch02], recommending the user to delete cookies in the web client at regular intervals to avoid server-side tracking.

Despite the inspiration of architectural patterns, not all patterns proposed by Doty et al. form architectural ones. They correspond to the privacy design patterns in Figure 3.2 allowing them to vary in the level of abstraction. The *Protection against Tracking*, e.g., does not affect the software architecture of the system and actively involves the user in the privacy implementation. These two properties result in the pattern following the principle of privacy-by-policy. There are patterns that follow the privacy-by-architecture principle as well, e.g. *Onion Routing*. Depending on the abstraction level, individual patterns are taken from their catalog to identify a PPAP, e.g. built in as architectural tactics. The structure of the patterns, as inspired from architectural patterns, is adopted for the definition of a PPAP.

Siljee identifies privacy patterns to be a starting point for creating a pattern catalog [Sil15]. All patterns fulfill privacy transparency. Two privacy patterns are identified and an overview of existing privacy transparency patterns is given, including patterns of Doty et al. [DGZ15].

All the identified patterns are following the principle of privacy-by-policy and are therefore not taken into consideration in this thesis. In addition, they offer little abstraction making these patterns to move more towards the low-level approaches in Figure 3.2.

Colesky et al. propose two pattern systems, one containing patterns that follow the Hoepman strategy of *inform* [CC18] and one following the Hoepman strategy of *control* [CCD+18]. The systems are intended to improve existing patterns from the Doty et al. [DGZ15] catalog, for example by allowing relations between them. These pattern systems are intended to assist in picking and applying the right patterns, by adding more relations to the existing patterns and developing them further.

The identified privacy patterns follow the principle of privacy-by-policy. Hence, these patterns are not considered in this thesis. The idea of including relations between patterns, allowing to apply these related patterns simultaneously, is built into the definition of PPAP.

Privacy Patterns from IoT domain/other Domains

Applications in IoT often use cloud computing and process personal data (see Chapter 2). Since privacy plays an important role as well, the following is devoted to research related to the application of privacy patterns in the IoT domain—and consequently cloud-based applications.

Washizaki et al. conducted a search for IoT patterns and categorized those according to the quality attributes they satisfy [WYH+19]. Out of 136 patterns found in 33 papers, one addresses the quality attribute Privacy. In addition to privacy, the found pattern *layered architecture for IoT applications* also addresses performance, reliability, security and scalability [KBL17]. They claim that the patterns found are mostly not IoT specific, but general privacy and architectural patterns that are adapted.

[2] https://privacypatterns.org/patterns/Protection-against-tracking
[3] https://privacypatterns.org/patterns/Onion-routing

That only one pattern is found, addressing the quality attribute privacy is assumed to be an indicator that privacy is not sufficiently taken into account. This thesis identifies patterns for cloud-based applications, and hence can be applied in the IoT Domain as well. The single pattern addressing privacy is considered in the identification of the PPAPs.

While the work of Washizaki et al. surveys all types of design patterns in the IoT domain, Pape and Rannenberg take the approach of examining exclusively privacy patterns pertaining to software in general [PR19]. Instead of identifying new IoT privacy patterns, they demonstrate the direct use of general privacy in patterns in the IoT domain regarding cloud, fog, and edge computing architectures. 7 Privacy Patterns from the collaborative catalog on the *https://www.privacypatterns.org* website are found, that can be mapped to an architecture in the IoT. At the time of writing some of these patterns do not exist in the current catalog (anymore). The authors present the *three-layer service delivery mode*, regarding the different layers of an IoT application: Cloud, fog and edge computing (presented in Chapter 2). They exploit this model, applying the general privacy patterns in the IoT domain. The *personal data storage*, e.g., shifts personal data towards the end devices, protecting it from the cloud computing server and other third parties. Another pattern, *Added noise measurement obfuscation*, obfuscates data on the client side before sending it to the cloud. Focusing their work on mapping privacy patterns to IoT architectures, they employ Spiekermann and Cranor's policy-by-architecture approach without explicitly referencing their work. Each of the patterns used, which are still currently in the catalog, follow one or more principles of Hoepman's privacy design strategies that address privacy-by-architecture: They *separate*, *minimize*, *aggregate*, or *hide* personal data. The previously mentioned trade-offs that have to be made when designing a software system to meet one or more quality attributes are also addressed in their work. Weinberg is cited as discovering a trade-off between privacy and convenience in IoT [WMAH15]. In terms of quality attributes, it corresponds to the trade-off between privacy and usability: The usability quality attribute is sacrificed, implementing technologies that are designed to enhance privacy. An example can be found in the *Added noise measurement obfuscation* pattern, where Pape and Rannenberg discover the trade-off between the degree of protecting the personal data (privacy quality attribute) and the meaningfulness of this data for running the service properly (usability).

Aiming to build privacy-friendly architectures in the IoT domain, Pape and Rannenberg apply patterns, that follow the privacy-by-architecture principle, and corresponding strategies proposed by Hoepman. This thesis identifies patterns that follow the approach of proactively implementing privacy goals in the architecture. Patterns in this thesis aim at applications in the cloud, which is often used in the IoT domain (see Chapter 2 Background). Ideas from the work of Pape and Rannenberg are considered for deriving the PPAPs presented here, e.g. the *personal data storage* pattern, exploiting the *three-layer service delivery mode*. Conflicts of privacy with other quality attributes are incorporated into a PPAP, helping in the selection of appropriate patterns in a specific context.

This chapter first discussed relevant research in the literature that deals with engineering of privacy. Privacy properties are built into the definition of a PPAP, e.g., the dimensions presented by Heurix et al. [HZNF15]. This thesis identifies patterns that are architectural [BCK03], proactive (adapted from the work of Venter and Eloff [VE03]), and follow the principle of privacy-by-architecture [SC08].

Thereafter, this thesis reviewed work that is of relevance in identifying privacy patterns. Existing patterns were examined, including those from the privacy and security domains. They are either not privacy related, not architectural, do not follow the privacy-by-architecture principle, do not

Paper	PP	AP	PBA	PBP	Cloud
Schumacher et al. [SFH+13]	✗	✗	✗	✗	✗
Romanosky et al. [RAH+06]	✓	✗	✗	✓	✓
Schumacher et al. [Sch02]	✓	✗	✗	✗	✓
Hafiz [Haf06; Haf13]	✓	✓	✓	✗	✗
Pearson et al. [PB10; PS10]	✓	✓	✗	✓	✓
Graf et al. [GWGT10]	✓	✗	✗	✓	✓
Doty et al. [DG13; DGZ15]	✓	✓	✓	✓	✓
Siljee [Sil15]	✓	✗	✗	✓	✓
Washizaki et al. [WYH+19]	✗	✓	✓	✓	✓
Pape and Rannenberg [PR19]	✓	✓	✓	✗	✓

Table 3.1: Examining existing research in the domain of privacy-related patterns: ✓ when the respective work contains designated privacy patterns (PP) , presents architectural patterns (AP) , follows the privacy-by-architecture principle (PBA) or the privacy-by-policy principle (PBP), or is designed for cloud-based applications (Cloud). When it does not satisfy an investigated property, the corresponding cell is marked with ✗.

have the desired pattern structure, or are not suitable for the application in the cloud domain. An overview of the examined privacy-related patterns is given in Table 3.1, assigning each paper to the kind of privacy patterns they represent. The patterns of Doty et al.'s catalog encompasses all properties, since they identify multiple types of patterns [DGZ15]. The pattern found by Washizaki et al. satisfies the other properties, but is not in any pattern structure [WYH+19].

4 Privacy-Preserving Architectural Patterns

This chapter introduces the main contributions of this thesis, comprising the definition and the identification of privacy-preserving architectural patterns (PPAPs).

The first section outlines the scope of the cloud, in which this thesis is located. Section 4.2 first determines the definition for a software architecture which is used in this thesis. It takes into account the background knowledge and relevant work in research, provided in Chapters 2 and 3. Building on this notion of a software architecture, the definition and template of a PPAP are presented. Section 4.3 *Quality Attribute Privacy* demonstrates the approach that has been taken to identify the PPAPs presented in this thesis. This includes the introduction of the privacy quality attribute, its categorization and its general scenario. Individual PPAPs are proposed thereafter: Each section consists of the specific scenario corresponding to a PPAP, the identified architectural tactics and the fully described pattern.

4.1 Scope

In order to determine privacy threats in the cloud, the scope of this thesis is described. This thesis considers a cloud, consisting of three participating groups: The cloud service provider (CSP), the clients, and optionally a third party. The relationship of a client and a cloud service provider corresponds to the client-server architecture, presented in Chapter 2. A server offers a service to the client, the cloud service. If multiple clients use the service in an interactive fashion, the communication takes place exclusively via the server as the intermediary.

Chapter 2 presents various models in cloud computing, including the service models (*as a service*) and deployment models (private, community, public, and hybrid cloud). Using virtualization techniques, the CSP can leverage its IT resources to offer different models to different clients. This thesis assumes a cloud-based service, comprising the software as a service model, and the public cloud deployment model (the service being available to the public).

For the sake of simplicity, in this thesis a service offered by a CSP is assumed, directly addressing the clients as end users with its service. For example, a CSP offers a ride-sharing service in which clients, acting as drivers or passengers, exchange data via a mobile application, provided by the CSP. Unless stated otherwise, the term *client* is used to describe the end device, that processes data sent by the server, generates data on its own, and sends data to the server. Despite the fact that several clients share a service, the clients can act isolated from each other in their usage. The definition of a PPAP, presented in the next section, involves the attacker scenario, which specifies the malicious actor in the system the pattern's solution is protecting against. Chapter 5 presents a methodology for selecting PPAPs, considering the privacy properties to be satisfied, and the setting. The setting includes whether a client uses an application in isolation or multiple clients use the service together, for example, through interaction.

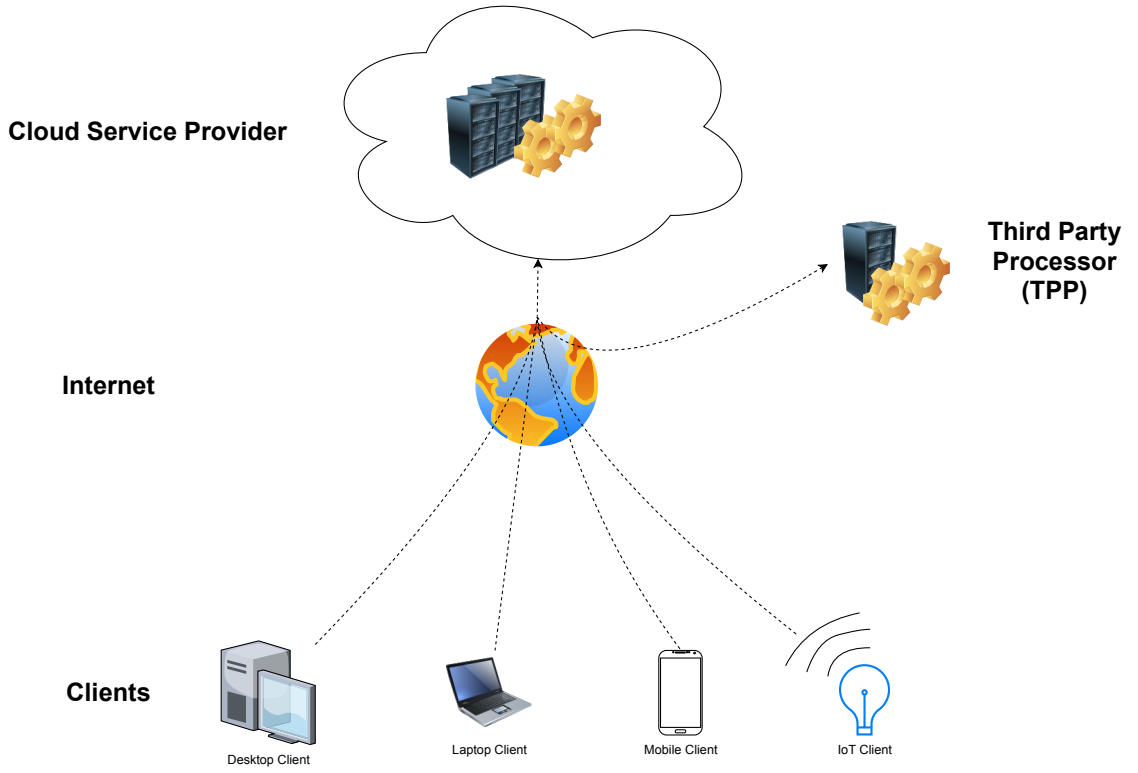

Figure 4.1: Scope of the cloud in this thesis.

Depending on the business model, another actor may be involved, which receives data from the CSP for analysis purposes. In this thesis this party is called the *third party processor (TPP)*.

The concrete solution of a PPAP can rely on a third party, whether it acts as a middleman between the cloud and the client, or it receives private data from the client and acts on behalf of the client to hide certain properties from the server. This entity is called TTP. The term TTP originates from the field of cryptography, where TTPs represent parties that must be trusted for a cryptographic protocol to work. The fact that the TTP must be trusted, is also valid regarding privacy in this thesis. If the TTP could not be trusted, the privacy problems would be shifted from the server to the TTP.

The scope of the cloud for this thesis is illustrated in Figure 4.1. It emphasizes that clients represent a broad spectrum of end devices, ranging from desktop computers to devices in the IoT domain. The TTP is omitted in the figure, since it is part of the solution in the presented PPAPs. It is not part of the original cloud setting that is assumed in the first place.

4.2 Terminology

As discussed in Chapter 2, there are different definitions and different perspectives about the scope of a software architecture. The notions of Bass et al. and Martin Fowler are combined to obtain the definition of a software architecture used in this thesis [BCK03; Fow03]:

*A **software architecture** constitutes one or more structures of the system that allow reasoning about the system and have a high impact on the system. The structures consist of interacting software elements with externally visible properties. The stakeholders determine the degree of impact by weighting the quality attributes.*

The definition of an architectural pattern, presented in Chapter 2, is restated: "An architectural pattern is a package of design decisions that is found repeatedly in practice, has known properties that permit reuse, and describes a class of architectures" [BCK03].

With the identification of PPAPs, this thesis supports the implementation of the GDPR principles privacy by design (corresponding to the *privacy embedded into the system* principle in Ann Cavoukian's *privacy by design* framework [Cav+09]), and privacy by default (presented in Chapter 2). In the following, work discussed in Chapter 3 is taken into account, contributing to the definition of a PPAP.

PPAPs are architectural patterns, representing *connector and components* (C&C) structures as a class of architectures [BCK03]. The view of a software architecture definition above, in which the interests of the stakeholders play an important role, is also transferred to the definition of a PPAP. This thesis identifies architectural patterns, whose main objective is to satisfy the privacy quality attribute. The derivation of the new quality attribute privacy is given in Section 4.3.

This thesis adopts the approach of incorporating privacy properties and the application context into a privacy pattern [PB10; PS10]. The general pattern structure is extended with elements, providing information about privacy properties. These properties are adopted and adapted from the taxonomy for PETs: *Privacy aspect*, *aim*, *scenario* and *data* [HZNF15]. They are discussed in more detail within the element description in the PPAP template, given at the end of this section. Each identified PPAP satisfies these properties differently. These privacy properties, embedded into PPAPs, allow the selection of the suitable patterns in a given context. The architect, e.g., takes into account the various interests of the stakeholders and depending on this outcome, selects the appropriate PPAPs. Chapter 5 presents a methodology, supporting the architect in this process.

The distinction between technologies in the area of security made by Ventor and Eloff is applied to the privacy domain [VE03]. In terms of data protection, it is too late, when data breaches have already occurred (see Chapter 2). This thesis calls privacy patterns *proactive*, when they implement privacy requirements in a software system without the user noticing. The other type of privacy patterns are *reactive*. They are applied when the system is already running, e.g. via privacy policy notices [CRC05]. Being in conformity with the privacy by design and by default principles, this thesis acknowledges the assertion of Spiekermann and Cranor, and identifies privacy patterns that *proactively* embed privacy into the architecture, following the privacy-by-architecture principle [SC08]. Most privacy patterns, found in the literature, follow the privacy-by-policy principle [GWGT10; MC04; MCG06; PB10; RAH+06]. These patterns, e.g. UI patterns [GWGT10], are not considered for the definition of a PPAP.

This thesis claims, that privacy-by-architecture patterns contribute more to the privacy by design and default principle than privacy-by-policy patterns. They are considered to provide more abstraction and to be located earlier in the design phase of the SDLC than their counterparts. Therefore, a PPAP is defined to provide the level of abstraction of an architectural pattern and to embed privacy *proactively* into the architecture following the principle of privacy-by-architecture.

With the previous considerations in mind the following definition for a PPAP is used:

*A **privacy-preserving architectural pattern** (PPAP) is a collection of design decisions found repeatedly in practice, has known properties that permit reuse, describes one or more component and connector (C&C) structures of a software architecture, and achieves designated properties of the quality attribute privacy.*

A PPAP consists at least of the following parts: *Name*, *context*, *problem*, *privacy aspect*, *attacker model*, *privacy aim*, *data*, *solution* and *consequences*. Additional elements can be considered: *Alias*, *variants*, *known uses* and *relations*. Except for the four elements, corresponding to privacy properties, and the *variants* element, the patterns of the privacy pattern catalog of Doty et al. are examined, and the general structure inferred [DGZ15]. The *variants* element is taken from the book of Schumacher et al. [SFH+13]. Inspired by the *classification* element from the work of Pearson et al., four dimensions of Heurix et al.'s privacy taxonomy for PETs is adopted and adapted: *Privacy aspect*, *attacker model*, *privacy aim* and *data* [HZNF15; PB10; PS10].

A template with description for each element is given below, optional elements being marked accordingly within square brackets.

Name

This element represents the name of the pattern. It should be short and concise.

[Alias]

This element represents aliases which may be more common or meaningful to specific groups of people.

[Example]

An example can be given for better understanding. It provides more insight into a certain technique or provide an example context.

Context

The context defines the conditions under which the problem emerges. It describes in which part of a cloud service the problem occurs, usually where the solution is applied as well. But the solution does not have to emerge in the same place, see element *Data*.

Problem

This element formulates the problem, the pattern tries to solve. In most cases it concludes with a question to which the solution refers directly.

Privacy Aspect

This element corresponds to one privacy dimensions in the privacy taxonomy for PETs of Heurix et al. [HZNF15]. Privacy aspect is divided into *content*, *behavior* and *identity*.

The former refers to the protection of personal data from unauthorized access.

Behavior deals with user habits. A solution, addressing the behavioral aspect, protects sensitive metadata, generated during the data flow between server and client.

In contrast to *content*, the last aspect, *identity*, prevents personally identifiable information (PII) in the first place. It is further divided into *anonymity* and *pseudonymity*. The former yields solutions, enabling a client to use a service without the server or other clients receiving PII. Pseudonymity is a mitigation of anonymity, allowing other clients and the server to recognize clients by using pseudonyms, instead of identifiable information. Other clients or the server (depending on the *attacker model*, see next element) should not learn any PII except the pseudonym. The degree of linkability highly depends on the chosen pseudonym. Optimal pseudonyms would be random generated names. But for a better usability of the service, clients usually can name themselves. For example in an application of a social platform, it is more convenient for a client to memorize other clients, when they have a more meaningful name instead of a truly random generated one. The client has the responsibility for choosing a name which is *usable enough* and still provides as much unlinkability as possible.

For more details on the differences, see the taxonomy proposed by Pfitzmann and Köhntopp [PK01]. Heurix et al. additionally divide identity further, investigating directionality for both variants, as well as holder and cardinality for the variant pseudonymity [HZNF15]. In this thesis, their inclusion is assumed to make a pattern too specific, and thus harm the necessary abstraction needed for an architectural pattern. Therefore, this thesis does not consider them.

Attacker Model

Considering the other privacy constraints, a PPAP aims at solving the given privacy problem, assuming the server and the clients as potential attackers. In cryptography, an attack is a malicious exploitation of a vulnerability in a software system. This thesis assumes an attacker model, depicting the origin of potential attacks and its *intent*, determining the risk the attacker is willing to take. The interests of an attacker are referred from the principles for the protection of personal data, set out in Article 5 of the GDPR [EU16]. They suggest to use technologies offering as little personal data as possible to potential attackers. Inversely, this thesis assumes an attacker, attempting to gain more than it is necessary to run (server being the attacker) or use (client) the service.

In the scope of this thesis, different options for the origin are *server*, *client*, or both at the same time. In addition, the server is assumed to be malicious only at a particular time for a restricted duration. The attacker's *intent* is divided into two categories: *Malicious* and *honest-but-curious*. A malicious attacker tries everything in its power to gain more information about personal data with the risk of being caught. Honest-but-curious attackers are using the service as intended and attempt to obtain personal data, but without attracting attention.

As explained in Section 4.1, the server is represented by the CSP, allowing it to constantly monitor the processes in the cloud, without being conspicuous. This thesis assumes a malicious server, which accepts the risk of being caught, since it is minimized by the fact that it has control over the cloud.

In the case of a client being the attacker, this thesis further differentiate whether the clients are honest but curious or malicious. A PPAP assumes either malicious or honest but curious clients. The latter constitutes a weaker assumption for the respective solution.

The previous section defines the scope of this thesis, considering cloud service in the public cloud. This thesis assumes the server does not know its clients, and hence does not collude with them. Thus, in the case both server and some clients are assumed to be malicious, they can be considered separately.

This element corresponds to the privacy dimension *scenario* in the work of Heurix et al. [HZNF15]. In this thesis the name *attacker model* is assumed to be more common (in particular in the security field) and therefore preferred over *scenario*. The authors propose as last option an attacker from the outside, e.g. a criminal organization or government interference. In this thesis, an external gaining control is assumed to be equivalent to the case a server or a client being malicious and therefore is omitted. The protection against external attacks on the system falls under the security domain and is out of scope in this thesis.

Aim

While the privacy aspect specifies *what* to protect, *aim* specifies *how* to protect it. This thesis adopts three properties of the proposed dimension *aim* of Heurix et al.'s privacy taxonomy: Indistinguishability, unlinkability and confidentiality [HZNF15]. If the former *aim* is fulfilled, an attacker is not able to distinguish between multiple clients. Unlinkability is achieved, if a client from one dataset (e.g. a database table) cannot be linked to another entity from another dataset, e.g. if it is made difficult for the attacker to link a data entry from a person table to the address from the address table. In the case of the strictest view of confidentiality, the sensitive personal data is protected in such a way an attacker does not obtain any information at all. This thesis excludes *deniability*, assuming it to be too PET-specific, and thus not contribute to the abstraction needed for a PPAP.

Data

In contrast to the work of Heurix et al., this element specifies the state of the data that is affected when the problem occurs and not the data state of the solution [HZNF15]. Because a PPAP provides more abstraction than PETs, it is more likely that the data state of the solution differs from the data in the problem part of a pattern, for example, in having more states than the problem. As in Chapter 2 discussed, this data refers to personal data in the privacy domain. PPAPs only consider data of the clients, because they represent natural persons (in contrast to the CSP, which is assumed to be a company). Data can be considered in three states, forming the sub-components of this pattern element: *Stored*, *in transit* and *processed* [HZNF15]. Since there are multiple participants in cloud-based applications (see Figure 4.1), the solution can shift personal data from one participant to another, satisfying the privacy properties. Therefore, the sub-component stored is further divided: Data can be stored *at the cloud* or *at the client*. This distinction is in conformity with the principle

of network centricity proposed by Spiekermann and Cranor [SC08]. By moving personal data to the client, a solution contributes to the design of client-centric software architectures. Processed data is data used by the CSP or by a designated third party provider, for example to compute a common function with input data from multiple clients or for analysis purposes. In case of data in transit, the solution transfers data between the client and the server. A solution can combine the variants of stored data and data in transit.

Solution

The solution represents the solution description and optionally an example architectural structure for solving the problem stated before. It depicts a C&C structure, i.e. the important elements involved and the interaction between these elements are illustrated.

[Implementation and Variants]

A PPAP provides a solution in an abstract form, and can therefore vary in the implementation. This element suggests a list of potential concrete realizations.

Chapter 2 presents architectural tactics, referred as the building blocks of an architectural pattern. They can form concrete realizations in architectural patterns. In the privacy domain, they can be represented by PETs, which are proven technologies in practice.

In a PPAP, different variants (architectural tactics) represent different realizations, which may differ in a privacy property (*privacy aspect*, *attacker model*, *data* and *aim*). E.g. one tactic is addressing only stored data, while another concrete solution considers stored data and data in transit. Or they demonstrate compromises with respect to other quality attributes, e.g. sacrificing performance for an increase in the respective privacy property. Another possibility is that multiple tactics are applied, e.g. in a sequence where the output of one tactic is the input of another one.

The *TTP* dimension of Heurix et al. is taken by whether certain tactics require a TTP or not [HZNF15]. However, no further distinction is made between frequency, phase, and task. Instead, architectural tactics consider TTPs being present at some point in time.

This element serves on the one hand, as a help for the architects to choose the appropriate variant from a list of architectural tactics, and on the other hand, as a support for the implementation in the next phase of the SDLC.

Chapter 5 presents a methodology, where some variants are removed, depending on the context and the choice of the architect.

[Example Resolved]

If there is a running example (see element *Example*), this element specifies its specific solution (mostly representing one variant of the element stated before).

[Known Uses]

Known applications or systems, where this pattern is already employed effectively, can be listed here as examples for better understanding, or as a proof of concept. The pattern does not have to be used explicitly. It is sufficient that the respective PPAP can be employed to contribute to the software architecture of the known use.

Consequences

Strengths and weaknesses of the presented solution are highlighted by discussing the extent to which the problem and the privacy-related characteristics have been solved. In particular, if several variants are proposed in *Implementations and Variants*, they are compared and evaluated with respect to the contribution to the solution. Since privacy is introduced as a quality attribute, this element considers additionally the impact on other quality attributes.

[Related Patterns]

When identifying patterns, various authors involve their relationships to other patterns [DGZ15; Haf13; SFH+13]. This element specifies the relations to other elements, considering three types: *Dependencies*, *supplements* and *exclusions*.

The former group of relations lists the architectural patterns (not only PPAPs), which have to be covered in addition for applying this solution. Each PPAP identified in this thesis depends on the server-client architectural pattern. This pattern is presented in Chapter 2 and constitutes the cloud model presented in Section 4.1.

Some patterns are applied together to cover a wider range of privacy problems, or because they can be used together efficiently. The *supplements* group lists patterns being recommended to use in combination with this solution.

On the other hand there can also be patterns that are mutually *exclusive*. The latter group lists patterns, which cannot be used simultaneously.

[Sketch]

In the pattern domain, sketches are often used to provide the reader (in this thesis the architect) the operation of a pattern in an illustrated way. This is also true for (privacy-preserving) architectural patterns in the form of C&C views used here. Its elements and interaction can be visualized in a graphical representation.

In this thesis, a sketch is optionally presented at the beginning of a pattern definition.

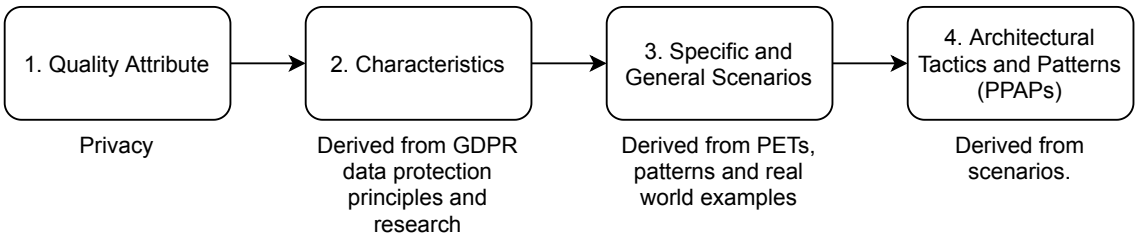

Figure 4.2: Deriving a new quality attribute for identifying new architectural patterns (PPAPs): Above the steps extracted from the book and below the approach chosen in this thesis.

4.3 Quality Attribute Privacy

Chapter 2 presents quality attributes, allowing to evaluate and design software architectures. In addition, a guideline for developing and deploying a new quality attribute is outlined [BCK03]. This section leverages this guidance for a new quality attribute: Privacy.

This guideline is adapted for identifying PPAPs in this thesis. In the following, it is applied to first characterize the new quality attribute, and second to set up its general scenario. In the subsequent sections, the identified PPAPs are proposed.

Bass et al. present an approach, laid out in textual form, for dealing with a new quality attribute, and using it to derive architectural tactics to create a *design checklist* for the respective quality attribute (illustrated in Figure 2.6) [BCK03]. This thesis adapts this approach with the goal of identifying PPAPs as a final step, instead of creating a design checklist. The modified approach, used in this thesis, is outlined in the following. It is illustrated in Figure 4.2, depicting its individual steps with the addition of comments on how each step is applied in this thesis.

4.3.1 Adapted Method for Deriving and Applying the Quality Attribute Privacy

Step 1: New Quality Attribute

The contribution of this thesis contains the definition and identification of architectural patterns, supporting the design of privacy-friendly systems. It is assumed that the quality attributes presented by Bass et al. are too different from privacy to allow the identification of PPAPs [BCK03]. The quality attribute security comes closest, but overlaps only in parts with the Privacy domain (see Chapter 2). Therefore, this thesis decides to introduce and derive the new quality attribute privacy.

Step 2: Establishing the Characteristics

Legal framework conditions and existing research are taken into account for the deduction of the various characteristics of the quality attribute privacy. They take over the role of stakeholders, with whom Bass et al. suggests to conduct interviews for identifying these characteristics [BCK03].

Since it is impossible to consider all legal positions regarding privacy, this thesis is limited to the GDPR [EU16], which is considered one of the strictest privacy laws worldwide[1]. The relevant characteristics identified for this thesis are enumerated and evaluated. The GDPR gives every EU citizen, the *data subjects*, the right to be informed, the right of access, the right to rectification, the right to erasure, the right to restrict processing, the right to data portability, the right to object and rights in relation to automated decision making and profiling. These privacy rights correspond to the privacy-by-policy notion (see Chapters 2 and 3) and are therefore excluded from consideration.

Privacy by design and privacy by default emphasize to consider privacy from early on in software development as a standard (see Chapter 2). The proposed PPAPs inherently support this by construction, but this does not represent a categorization of privacy in terms of a quality attribute.

Article 5 of the GDPR contains privacy principles considered in this thesis for characterizing privacy as a quality attribute. These include *data minimization*, *storage limitation*, and *integrity and confidentiality*. Other principles such as purpose *limitation*, *accountability*, *lawfulness*, *fairness and transparency*, *accuracy and accountability*, are excluded from consideration as they do not form principles that address software elements or embed privacy as a proactive approach in the software system.

The characteristic of *data minimization* states to collect and process as little data as possible to fulfill the purpose of the system. *Storage limitation* is the principle of keeping data in the system, only for a defined period of time and then deleting or anonymizing it. *Integrity and confidentiality* represent the part that overlaps with security and require those measures, protecting data from unauthorized access and modification.

The following characteristics are found in the literature. Hoepman derived 8 privacy design strategies, serving as a classification of privacy patterns by presenting a more abstract view [Hoe14]. Four of them fall under the privacy-by-policy principle and are therefore not considered. The remaining ones follow the principle of privacy-by-architecture and are therefore included for the creation of scenarios: *Minimise*, *hide*, *separate* and *aggregate*.

Figure 4.3 illustrates the deduced characteristics from current legislation and literature. These categorization types for privacy are used as keywords for finding real-world examples, and examining research for existing technologies and patterns. Although not considered for characterization due to the level of abstraction, the tactics of Colesky et al. [CHH16] and privacy properties included in the PPAP provide support for scenario building in the next step. Considering the categorization of the privacy quality attribute, and the specific scenarios, the general scenario is created.

Step 3: Building Scenarios

Similar to the book, scenarios are developed that play an important role in the derivation of the quality attribute. Using the set of attribute characterizations, crafted in the previous step, the specific scenarios are created first. There are different ways to create scenarios, and this thesis makes use of two of them. One option is to create scenarios from research by examining existing patterns, or PETs. Patterns may represent a different type than architectural patterns, and may occur in other domains. Provided they address privacy in one form, they can be used for the scenario creation.

[1] https://www.wired.com/story/europes-new-privacy-law-will-change-the-web-and-more/

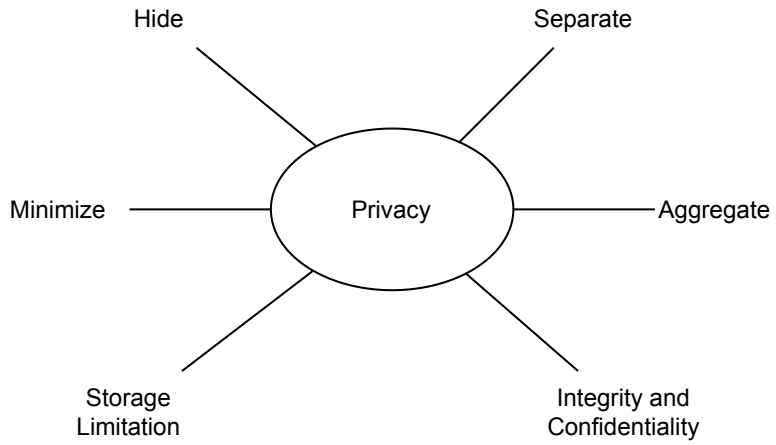

Figure 4.3: Deduced privacy characteristics

Another option involves the investigation of real world examples. The focus is placed on events where the quality attribute is or can be violated. The scenarios have the same structure similar to the book consisting of the same 6 parts. Following the book, a general scenario for the quality attribute is developed from these scenarios. Some specific scenarios emerge as tactics and are therefore presented in the individual PPAPs. For reasons of space and clarity, the presentation of all specific scenarios is omitted.

The derived general scenario for the quality attribute privacy is presented below. For each individual part, the relation to PPAPs is shown, e.g. the corresponding appearance within a PPAP. The general scenario is illustrated in Table 4.1.

The *source* of a data disclosure can be a client, the server, or an external threat. Errors in the system, that lead to the respective stimuli, originate in the server and are therefore associated with the server. In a PPAP, the source of stimuli is represented in the *attacker model* element. The external threat is not explicitly included there, since this thesis assumes a external impersonating the server, the client, or both (discussed in Section 4.2).

Principles, categorizing the quality attribute in this thesis, are introduced in the previous section: *Data minimization*, *storage limitation*, *confidentiality and integrity*, *hide*, *separate and aggregate*. The *stimulus* is any event compromising personal data by not complying with any of these principles. In a PPAP, stimuli are found in the *problem* description.

Artifact represents the elements or locations in the system, that are affected during the occurrence of the stimulus. Since this thesis considers the quality attribute (information) privacy, the artifact represents the elements where personal data is compromised, and hence the state of the data. Stimuli refer to data at rest, in transit, or processed data. In the former case, a further distinction is made between data that is placed on the server and data that is placed on the client device. The stimulus for data in transit can be the result of attacks during the data flow from the server to the client, and vice versa, or can take place at the server or client port. These elements are included in the example client-server architecture for the cloud, illustrated in Figure 2.3. In a PPAP, these states of data can be found in the *data* element.

Portion of Scenario	Possible Values
Source of Stimulus	Server, client, or external
Stimulus	Personal data is disclosed through corresponding source of stimuli
Artifact	At rest (client or server), at transit (at client port, server port, or intersected), or processed
Environment	The system involves one or multiple clients, using the service in an isolated fashion; client(s), using the service actively or not; a TPP is involved
Response	Data minimization, hide, aggregate, separate, storage limitation, or integrity and confidentiality
Response Measure	To which extent the data disclosure is prevented

Table 4.1: The general scenario for the quality attribute privacy.

The *environment* is described by the circumstances of the system: The stimulus may occur during default service operation, or while the client is not actively using the service; a third party may or may not be involved; or a client may be using the service in isolation or sharing the service with many other clients. The environment, constituting the cloud-specific context of an application, is found in the *context* of a PPAP.

The *response* is a realization of any principles, derived in the classification of the privacy, that is used to avoid or mitigate the stimulus: Data minimization, storage limitation, integrity and confidentiality, hide, separate, and aggregate. A PPAP is a realization of one or more of these principles. Considering the discussion about the levels of abstraction in Chapter 2, these principles provide more abstraction (illustrated in Figure 3.3). They can serve as classification for PPAPs, like in the work of Hoepman with privacy patterns in general (GDPR data protection principles are assumed to provide the same level of abstraction) [Hoe14].

To evaluate the success of the response, the extent of personal information the particular attacker (source of stimulus) can gain, is measured. *Response measures* are individually tied to the scenario, but are also tied to one or more of the principles presented. They can either measure, if the data has been minimized, hidden, aggregated or separated to the extent intended by the response (data minimization, hide, aggregate or separate, respectively), the data has been stored longer than necessary (storage limitation), or whether data is protected from unauthorized access and modification in the case of the principle of confidentiality and integrity. In a PPAP, this part of the general scenario is reflected in *consequences*, where the impact of the pattern's solution is discussed.

Step 4: Identifying Architectural Tactics and Patterns

Bass et al. suggest using the general scenario to categorize potential architectural tactics. This thesis deviates from this approach, including the main contribution of this thesis: In the process of developing the specific scenarios, and the general scenario, the PPAPs are identified. The abstraction

level of PPAPs is located between the specific scenarios and the general scenario. The different response options of the general scenario, discovered in Step 3, represent higher-level goals a PPAP follows. A PPAP is identified from the specific scenarios in one of the two following ways: Either a specific scenario results directly in a PPAP or different specific scenarios are merged to form one PPAP.

The first situation occurs, when a specific scenario is found and its *response* element maps the abstraction of an architectural pattern and follows the principle of proactively embedding privacy requirements (e.g. fulfilling the privacy properties of the PPAP definition) in the architecture. One example represents the *Pseudonymous Identity Management* pattern, presented in Section 4.8. Inspired by various related works, a specific scenario for the use of pseudonyms is created [Haf06; MPS13; RBE+12; Sch02]. It fulfills the required properties of following the privacy-by-architecture principle and providing the level of abstraction. Regarding the characteristics, using a *Pseudonymous Identity Management* falls under the *hide* and *minimizing* principles. The goal (*privacy aspect*) is to protect the identity of a client by using pseudonyms instead of PII. Additional architectural tactics can optionally be added, derived from other specific scenarios, or from the literature. In the example of *Pseudonymous Identity Management*, the pattern is augmented with architectural tactics in the form of variations, including *single sign-on*.

The second way of identifying PPAPs results from combining several specific scenarios. One reason is, if they do not reflect the level of abstraction of an architectural pattern, but are of the privacy-by-architecture type. Thus, they represent architectural tactics. Different specific scenarios form a PPAP, when these groupings pursue a goal that is more abstract than any individual one, but not as abstract as one of the overarching goals from the response part in the general scenario. This *intermediate* goal is identified as a PPAP, incorporating the different specific scenarios, e.g. into the *implementation and variants* element in the form of architectural tactics. One example is the identification of the two specific scenarios *Homomorphic Encryption* and *Trusted Execution Environment*. The *intermediate* goal both are pursuing, consists of processing personal data, while protecting its content from other participants. Hence, this thesis identifies the intermediate goal to be *Private Data Processing*, following the principles of *confidentiality* and *hide*. The *Private Data Processing* pattern is presented in Section 4.5.

In both cases, the next step is to put the discovered patterns in the form of an architectural pattern by specifying the individual elements. Section 4.2 presents the template for each PPAP, depicting each element, e.g. adopted dimensions of Heurix et al. for the respective privacy property elements [HZNF15]. The fulfillment of each privacy property element must be assessed and inserted accordingly. Relations to other patterns are considered and built in. In this thesis, each pattern is related to the *Client Server Pattern* under the element Relations. If a new pattern is added, and it results in new relations, the affected patterns have to be considered for adjustments. When all elements are described, the identification of a pattern is completed.

While specific scenarios are not presented in detail (they are included in form of architectural tactics in each PPAP), this thesis illustrates a PPAP-specific scenario in each corresponding section.

Omission of the Checklist

After identifying the architectural tactics, the approach of Bass et al. suggests the creation of a design checklist for the new quality attribute, divided into 7 categories (see Chapter 2). This checklist is intended to guide an architect, enabling him to consider the important decisions for the respective software system. In the presented guideline for deriving a quality attribute, the authors conclude with the creation of the design checklist. Architectural patterns are not created in their process.

In contrast to the work of Bass et al., this thesis identifies architectural patterns which is following similar goals as the *design checklist*. Different categories of the *design checklist* are represented in the architectural pattern. Second, Chapter 5 proposes a methodology supporting the architect in finding the appropriate PPAPs for the design of privacy-friendly software architectures. Hence, this thesis omits the creation of the *design checklist*.

This section introduces the new privacy quality attribute and its general scenario. The following sections present the proposed PPAPs, including the PPAP-specific scenario , the architectural tactics, and concluding the entire pattern according to the template of Section 4.2.

4.4 Client-Side Obfuscation

This section defines the PPAP *Client-Side Obfuscation*, following the principles of *data minimization*, *confidentiality* and *hide*.

4.4.1 Scenario

In the following, the PPAP-specific scenario is presented, illustrated in Figure 4.4. A CSP states in its service level agreements, that it ensures data integrity and recoverability. Thus, instead of saving the backup on an own external disk, a client decides to put the trust into the cloud service provider. Another scenario constitutes the storage highly sensitive data from multiple real identities, e.g. the data is a collection of medical records which must not be disclosed. In this scenario, a client wants to store data onto the cloud, e.g. doing a backup. This data can contain sensitive information about the client. The *stimulus* is the circumstance, in which the server (*source of stimulus*) learns more information about the client, than is necessary for the proper functional requirements of the service. The affected part of the system is therefore the cloud storage (*artifact*). Storing data on the cloud is a normal operation, hence does not constitute an unusual circumstance (*environment*). *Response measure* is described by the extent of obtaining personal data that the server should not obtain. In the best case (in terms of data protection), the server learns nothing about the personal information of a client. But depending on the cloud-based application, some personal data is necessary for properly running the service. In this case the extent of gaining personal information should be limited to the purpose of the service, following the *data minimization* principle [EU16]. Other data should be *hidden* or made *confidential*.

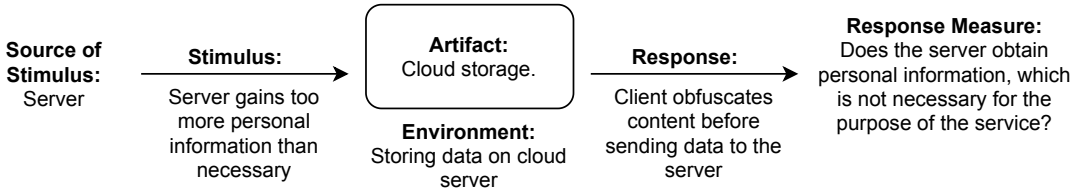

Figure 4.4: Specific scenario for *Client-Side Obfuscation*.

Responding to this problem, the client obfuscates either the whole content, or parts of it, containing the sensitive information. The most common form of obfuscating the client's data is to apply encryption. For encryption, the client uses a secret, which no one has access but itself. The measurement of the response is reduced to the binary case: The server either can infer the content behind the masked data, or not.

4.4.2 Architectural Tactics

Three obfuscation tactics are identified, making it difficult for the server to obtain the sensitive content of the client data.

Client-Side Encryption

The best known method to hide data is to mask it completely from the server. Typically, cryptographic primitive encryption is used for this purpose. This is also used extensively in the field of security to fulfill the property of confidentiality. A distinction can be made between asymmetric and symmetric encryption. The simplest case is where a service is only intended for a single client. In this case, it is sufficient for the client to generate a key pair, keep the private key for itself, and send data to the cloud encrypted, using the public key. If several clients use a service jointly, they are usually required to use asymmetric encryption first. Using the Diffie-Hellman key exchange, they can agree on a common private secret key. This offers significant performance advantages and is especially useful when a group of clients does not vary in size. Explaining encryption is beyond the scope of this thesis.

The corresponding scenario is extracted from the work of Heurix. et al. and Hafiz [Haf13; HZNF15]. For further details, this thesis refers to the work of Singla and Singh [SS13].

Tokenization

In contrast to encryption, tokenization is not based on a mathematical model that converts plaintext into meaningless (from the perspective of people without a private key) ciphertext, and vice versa. Also for increased security, the same plaintext must be encrypted into a different ciphertext every time. Tokenizing is converting a given value of a table (e.g. credit card number or social security number) or multiple table entries into one randomly different string of characters. Most of the time

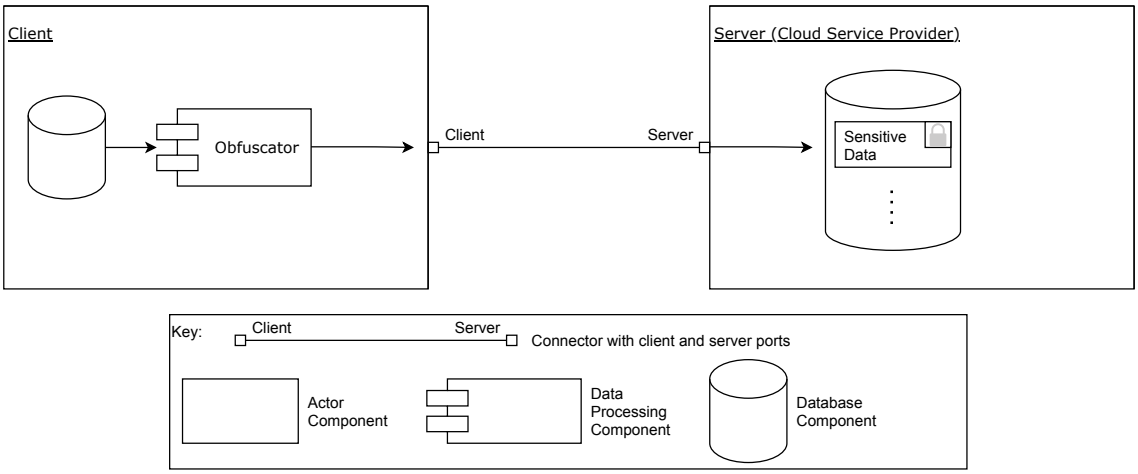

Figure 4.5: Sketch of *Client-Side Obfuscation*.

the length is preserved or normalized. This value does not change and is used in the process as usual. The goal of the process is to prevent joins over tables with a primary and foreign key, containing potentially sensitive information, and thus allowing an identification of persons.

There are variants, which differ in where the randomization of the plaintexts takes places and where the tokens are stored. This can be either on the client side, realized by a proxy (TTP) or on the server side just before the injection into databases. In the latter case, it is assumed that the server is not compromised at the time of tokenization. A potential malicious activity occurs at a later time and then only for a certain period of time.

For further details, this thesis refers to the work of Ang et al. [ATWW15].

Local Differential Privacy

Differential privacy aims at balancing the protection of privacy and the desire to gain as much knowledge as possible. The server add randomized noise to entries in a database to protect individual personal data while allowing statistically relevant analysis of the data [Dwo08].

In terms of privacy, *local differential privacy* provides a stronger concept, by randomizing the data at the client before sending it. For this pattern, the specific variant *local differential privacy* is adopted, since the protection of personal data is performed on the client side and the server never receives raw personal data.

For further details, this thesis refers to the work of Cormode et al. [CJK+18]

4.4.3 Architectural Pattern

In the following, the PPAP is written according to the template, given in Section 4.2. Its *Sketch* is illustrated in Figure 4.5.

Name

Client-Side Obfuscation of Information

Context

In a cloud-based application a client typically sends continuously data to the server, which is then stored in the database of the server.

Problem

The data sent to the server can include PII, or other kind of data that a client does not want to share with the server. How can the service prevent the (potential malicious) server from accessing personal data, which is not necessary for the proper operation of the service?

Privacy aspect

The goal the pattern tries to pursue, is to protect the *content*.

Attacker Model

The potential malicious adversary is assumed to be the *server* or a *TPP* getting data from the server for further processing.

Aim

The aim is to ensure that the content is stored in a *confidential* manner.

Data

Personal data is exposed on the *CSP storage*.

Solution

The content of data or critical parts of the data are hidden from the server by obfuscating the data. When data is obfuscated, a server cannot deduce the (complete) actual content.

An example architecture is illustrated in Figure 4.5. Obfuscation is performed in the *Obfuscator* component. It is part of the client component from the *Client Server Pattern*. In the example architecture, the *Obfuscator* takes entries from a database as input. Other types of input are also an option, for example generated data from IoT devices. Obfuscation takes place before the data is sent to the server component, via the client-server connector. Implementing the obfuscation of the content can vary. For different strategies see *Implementation and Variants*.

Implementation and Variants

The following variants are considered for implementation:

- Apply the architectural tactic *Client-Side Encryption*, using symmetric or asymmetric encryption.
- Apply the architectural tactic *Tokenization*. This solution variant may involve a TTP.
- Apply the architectural tactic *Local Differential Privacy*.

Known Uses

One application is the use of a cloud-based service as a cloud storage. Clients do not trust the cloud service provider, or are afraid of future data breaches. The critical data is stored permanently in the cloud and, thus masking of information adds a layer of security to the disclosure of such data. This can be achieved in the single client case, since it can store the necessary keys on its own device. With multiple clients it is possible to store data such that the information is hidden from the server, but all other included clients have access to the information (see architectural tactic *Client-Side Encryption)*.

The second variant is typically used to hide the credit card number [BH13].

Another use case includes data which the service needs for calculating on encrypted data. For different applications see the *Private Data Processing* pattern.

Consequences

By obfuscating sensitive information, the server does not learn what is inside the actual data to a certain extent. Thus, it is made more difficult for the server to draw any inferences about the clients from the data content perspective.

This comes with a price, though. Depending on the use case, the operation of the service may be limited or completely cut off. If the main purpose of the service is to calculate on sensitive information of one or more clients, the use of *Private Data Processing* is recommended (see *Supplements*). But if it is mandatory for the service to access the content of the data, other techniques have to be considered to preserve as much privacy as possible.

The architect can choose the degree of obfuscation and therefore reversely the degree of linkability by applying the respective variant presented here, or another variant not presented in *Implementation and Variants*. The consequences of the three variants presented are discussed in the following. The *Client-Side Encryption* implementation provides the greatest benefit in the degree of obfuscation. Using state-of-the-art encryption methods, it is impossible, even for attackers with supercomputers capabilities to access the data in a realistic amount of time. This strength can also become the greatest weakness: Performing calculations on encrypted data is difficult. For certain purposes, there are techniques that make it possible to work on them. These application areas are explained in the *Private Data Processing* pattern. However, if the server needs the data in plaintext in order to work with it, e.g., because private data processing is not possible or impractical, this architectural tactic is not applicable.

Comparing to the encryption method, *Tokenization* has the advantage that the values in databases can be used without the need for cryptographic keys, while in case of a data breach the substituted values (database entries in most cases) still remain irreversible. It reduces the linkability, but compared to encryption, data has a higher degree of linkability.

By allowing statistical analysis on the data, *Local Differential Privacy* provides a smaller effect on the confidential protection of the data than the other variants. It represents a trade-off between the wishes of the data controller (data to process) and the clients (data to be protected). The *Differential Privacy* approach works only with larger populations, i.e. many clients have to use the service.

Related Patterns

This solution depends on an existing cloud architecture, constituting a server component, offering a service to one or more client components. Therefore, it uses the *Client Server Pattern* as a basis. The client component is augmented with the *Obfuscator* component, adding one of the presented variants, or another tactic introduced by the architect.

This pattern complements the PPAPs *Private Data Processing* and *Private Information Exchange*. As mentioned in Section 4.4.2, *Private Data Processing* can be added on top of encryption. It enables the server to calculate on sensitive data, while it does not learn anything about the result and the input of different participating clients. This solution is applicable if the *Client-Side Encryption* variant is chosen. Combined with *Private Information Exchange*, clients can not only hide the content from the server, but additionally hide the behavioral patterns of single clients.

In the current catalog of PPAPs, there are no excluding patterns.

4.5 Private Data Processing

This section defines the PPAP *Private Data Processing*, following the principles of *confidentiality* and *hide*.

4.5.1 Scenario

There are applications with multiple clients, having the same class of sensitive data, wanting to compute a function together. They consider the use of a cloud as a central point of calculation. Each client wants to protect its sensitive data from both, other clients and the server. An example of this can be found in the area of health care, where various institutions store sensitive patient data. For improving predictions of whether patients will develop a certain disease, a more effective approach is to collect all data together. The collective analysis is assumed to achieve better results than each institution making predictions in its own smaller scope. But a institution does not want to (and often is not allowed to, because of strict regulations) disclose their highly sensitive medical data of patients in plaintext.

The scenario emerges as follows and is illustrated in Figure 4.6. The server represents the *source of the stimulus*, since it is gaining information about sensitive data. Clients are not considered as *source*. Depending on the application context, a client wants to protect its data from other clients. A

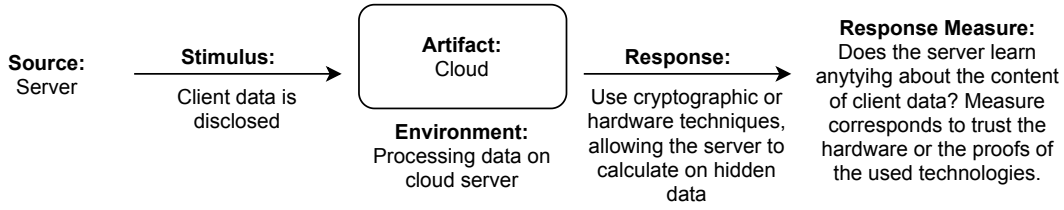

Figure 4.6: Specific scenario for *Private Data Processing*.

client gains more information about other clients, if the server colludes with it. Section 4.2 defines the attacker model and makes the assumption, that a server do not collude with clients. Therefore, a malicious client is not being able to gain more information. The server learns from performing calculations on this data, the result and the individual inputs of the respective clients. This *stimulus* takes place in the cloud (*artifact*), in the context of processing data (*environment*). Countermeasures are technologies, allowing a function to be calculated on the server, without him or other clients learning the inputs. In addition, the result should be sent to the clients without the server learning its content. *Responses* can be cryptographic, or hardware technologies. The *measurement* is binary: It is determined whether the server determines the content of the input or the result, or not. Since it is not measurable for the clients, they rely on the mathematical proofs in case of cryptographic technologies, or on the trustworthiness of hardware implementation, e.g. by referring to expert opinions.

The same process can be applied to the case, where a single client wants to use the clouds computing power to calculate computation-intensive tasks on sensitive data without revealing any content of the data to the cloud.

4.5.2 Architectural Tactics

Two architectural tactics in form of PETs are presented below. While the first is of cryptographic nature, the second represents a hardware-based solution.

Homomorphic Encryption (HE)

HE allows the analysis of encrypted data, and gives the same result as if the operations had been performed in plain text. The returned results are also encrypted.

For calculation on the client's inputs, each client provides its input in encrypted form to the server. Without having the secret key, the server is able to perform calculations on encrypted data, provided by the clients. Neither the server, nor other clients learn the input of a certain client.

Any function can be built from only two operations, addition and multiplication. Modern encryption methods, which support both operations in unlimited numbers, are called *Fully HE* methods. Making HE more practical, weakened variants have been developed with a smaller application scope, e.g. *Somewhat HE*. Nevertheless, in most applications, the use of HE is impractical. Following from the discussion in Chapter 3 [RBE+12; SFH+13], it is considered a potential implementation.

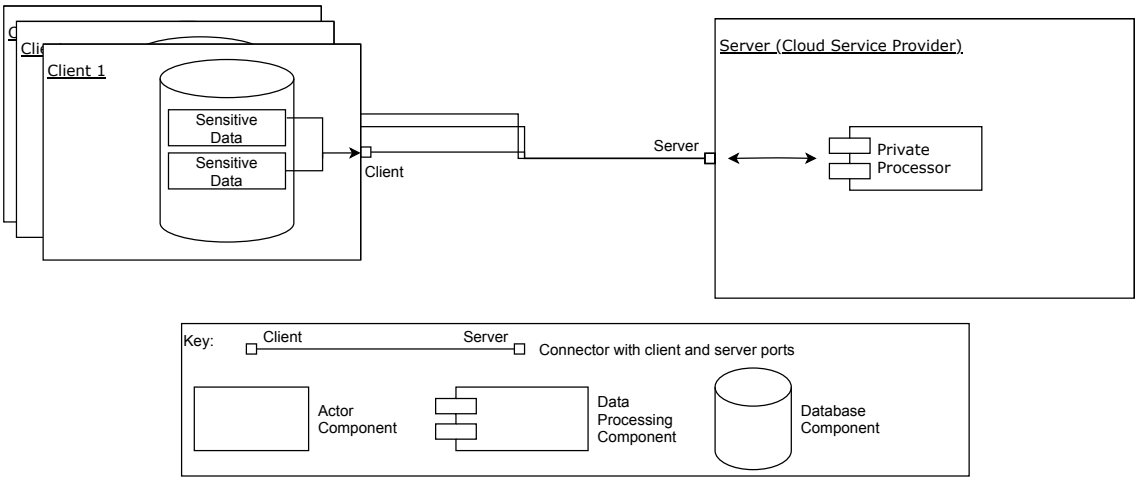

Figure 4.7: Sketch of *Private Data Processing*.

HE is intensively researched, and although there are many remaining open problems [MOO+14], Naehrig et al. provided serveral use cases where HE in form of *Somewhat HE* is applicable [NLV11]. Explaining HE in detail is beyond the scope of this thesis. For more information, the work of of Craig Gentry is referenced [Gen10].

Trusted Execution Environment (TEE)

A trusted execution environment (TEE) is designed to create an isolated environment, cut off from other applications and data, protecting the execution of applications or the storage of personal data [SAB15]. A TEE can be understood as a processor-in-processor, which can manage its own keys and only executes programs whose fingerprint remains unchanged from the original. The trust is placed in the manufacturer of the processor.

This can be used in the cloud to perform a function within the TEE, with all participating clients providing their input directly to the TEE. Within the TEE the function is then executed and the result is returned. Only the manufacturer could access the input of the clients, since it has the necessary keys.

For more information, the work of Sabt et al. is referenced [SAB15].

Regarding the attacker model, the following assumption is made. When the server (CSP) colludes with the manufacturer, it is always able to undermine the TEE, gaining the information placed in the environment. Thus, this thesis assumes this architectural tactic fulfilling the given privacy requirements in the existence of a malicious server, but requires a TTP in form of the manufacturer.

4.5.3 Architectural Pattern

In the following, the PPAP is written according to the template, given in Section 4.2. Its *Sketch* is illustrated in Figure 4.7.

Name

Private Data Processing

Alias

Secure Multiparty Computation

Example

Various companies from the same sector want to jointly process customer data for analysis purposes. They decide to outsource the processing to the cloud, requiring their own customer data to be protected from insight. How can they use the cloud to process the data of each company, while neither the server nor any other company gain information about their own customer data?

Context

A CSP provides a service for computing different kind of mathematical functions, utilizing its computational power to perform even computationally intensive calculations in short time. The service can be used by a single client, or by a group of clients. In the latter case, clients want to compute a collaborative function taking data from each client as an input.

Problem

Calculation with data may include personal data, which a client does not want to share. The problem is double-edged: First, the clients do not want to share their input or the result with the server. Furthermore the other clients are also not supposed to learn anything except the result. The question is: How can the cloud be used as a central processing unit, allowing the input of each client to be preserved from the server and other clients?

Privacy aspect

The general goal the pattern tries to achieve, is to protect the *content* of the inputs and the result.

Attacker Model

The main threat represents the malicious *server*. But depending on the context, the input has to be protected from malicious *clients* as well.

Aim

The data is to be processed in a *confidential* way.

Data

The data is *processed* on the server side.

Solution

Use of cryptographic or hardware technologies, allowing shared data to be computed without the server and clients learning more than they are supposed to, according to the constraints given by the problem.

An example architecture is illustrated in Figure 4.7. Private processing is performed in the *Private Processor* component. Since the solution takes place within the server, *Private Processor* is part of the server component from the *Client Server Pattern*. In the example architecture, it takes input data from different clients, which is transferred via the client-server connector. The *Private Processor* can take its input from a database, stored on the server side as well. In this case, the data content must be hidden from the server, e.g. by encryption. For different implementations of the *Private Processor* component, see *Implementation and Variants*.

Implementation and Variants

The following variants are considered for implementation:

- Apply the architectural tactic *Homomorphic Encryption* (HE).
- Apply the architectural tactic *Trusted Execution Environment* (TEE). This solution variant involves a TTP.

Example Resolved

The companies decide to use a cloud service to perform calculations on the customer data, without disclosing the specific content of the data to the analytics engine.

Consequences

This pattern allows the processing and storage of data to be outsourced to cloud environments. The solution performs calculations on client's data, while preserving the confidentiality of the information: Neither the server, nor other clients learn the content of a client's own data.

On the downside, the requirements for being able to apply this pattern are so high, that limitations are emerging in other areas. In the first variant the approach is secured by mathematical proofs, but the complexity of their applications is large. In contrast, the hardware solution of the second variant is dependent on the trust, put on the hardware manufacturer (TTP).

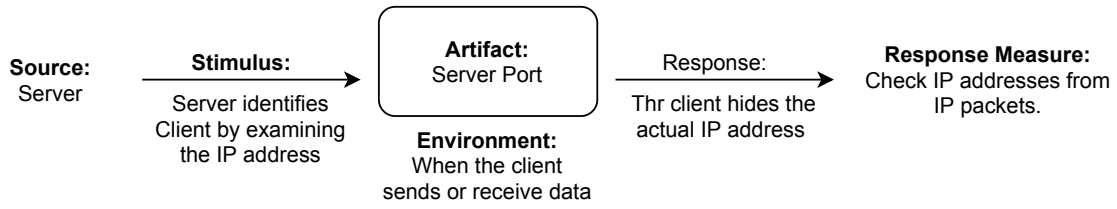

Figure 4.8: Specific scenario for *Private Network Access*.

Related Patterns

This solution depends on an existing cloud architecture, constituting a server component, offering a service to one or more client components. Therefore, it uses the *Client Server Pattern* as a basis. The server component is augmented with the *Private Processor* component, adding one of the presented variants, or another tactic introduced by the architect.

Applying the variant *Homomorphic Encryption*, the client has to encrypt the data beforehand. In this case, this pattern complements the *Encryption* implementation of the *Client-Side Obfuscation* pattern, augmenting the client component with the *Obfuscator* component.

In the current catalog of PPAPs, there are no excluding patterns.

4.6 Private Network Access

This section defines the PPAP *Private Network Access*, following the second strategy of Hoepman [Hoe14], *hiding* personal data (origin of the client request) from the server.

4.6.1 Scenario

One way the server can improve its user profiling capabilities is to observe the IP traffic [KM09]. The server examines the source IP addresses of a client, using the provided service for a certain time.

The scenario for the *Private Network Access* pattern emerges as follows, and is illustrated in Figure 4.8. The attacker of this scenario corresponds to the *source of the stimulus*, which is in this case the malicious server. He tries to re-identify the origin of different clients of the service by monitoring the IP traffic (*stimulus*). Since it is about IP packets, the scenario is located at the server port, where the packets enter and leave the system (*artifact*). The *environment* is composed of common communication processes between the server and its clients. The *response* is to prevent re-identification by IP traffic monitoring, the client can hide its address from the server by sending and receiving packets about other stations. The success of this response can be determined by inspecting the IP packets: Whether the actual address is stored or the address of an intermediary (*response measure*). Since the client sends and receives the IP packets via relays, it has to rely on the technology or intermediary used for this process.

4.6.2 Architectural Tactics

This thesis identifies two architectural tactics being used in practice. In critical contexts, these tactics can help to circumvent high levels of censorship, e.g. for bypassing the *Great Firewall of China* [And12].

Virtual Private Network (VPN)

A virtual private network (VPN) is a virtual network that allows the client to use services on the Internet without revealing its IP address.

In the setting of this thesis, it implies first sending data from the client to a VPN server (the intermediary). Accordingly, the VPN server forwards this data to the server, containing its source IP address. The CSP can only see the VPN server as the source and at most assume that the client is located in the area around the VPN server location. To avoid that the VPN server learns the content of the client's data, the data is first encrypted with the shared symmetric or public key of the server. For the response of the cloud server, the same applies in reverse sequence. The connection between the VPN and the client is encrypted, the *encrypted VPN tunnel*. It prevents attackers (e.g. eavesdropper) from determining which connections clients establish, and which destination these connections have.

For more details about the implementation of VPNs, this thesis refers to the work of Seid and Lespagnol [SL98].

Onion Routing (TOR)

Applying the *onion routing* technique, data is encrypted and the traffic sent to the destination server via multiple nodes, each adding one layer of encryption. Receiving the traffic, the destination cannot deduce the original source of the data.

One implementation of onion routing is *Tor network* [2]. Tor encrypts a user's traffic and routes it through three random nodes of the Tor network while routes are changed every 10 minutes (Hidden Service Protocol). This makes it almost impossible to track, who the sender of the request is, except for the first node which knows the IP address of the origin (the *entry server*)—but these are not stored by the Tor server. While the Tor browser[3] is mainly used for connecting to the Tor network, the Tor project team gives instructions, how the Tor connectivity can be added to an application[4].

For more information about onion routing, the work of Goldschlag et al. is referenced [GRS99], or see the pattern *Onion Routing* in the online catalog of privacy patterns created by Doty et al.[5] [DGZ15]. McCoy et al. describe the onion routing implementation Tor network in more detail [MBG+08].

[2] https://www.torproject.org/
[3] https://www.torproject.org/download/
[4] https://gitlab.torproject.org/legacy/trac/-/wikis/doc/TorifyHOWTO
[5] https://privacypatterns.org/patterns/Onion-routing

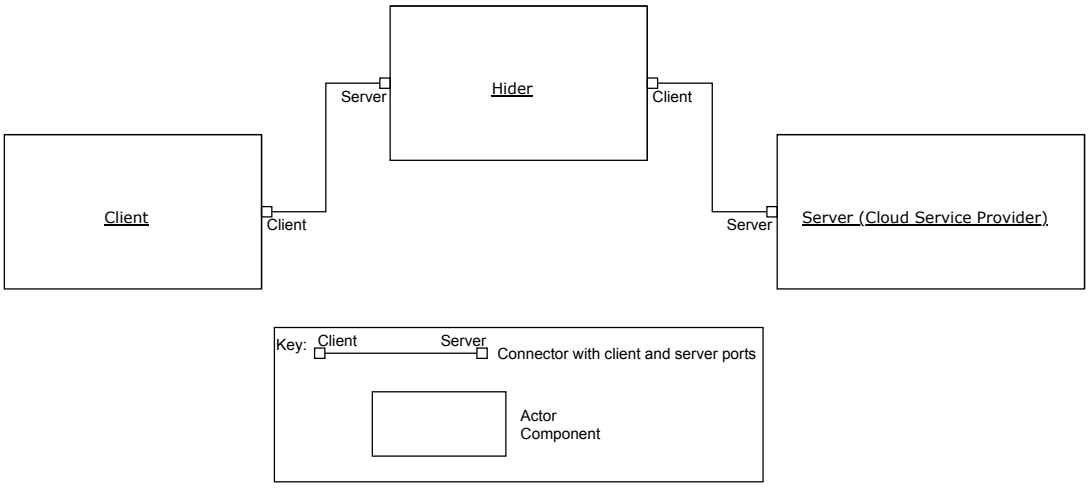

Figure 4.9: Sketch of *Private Network Access*.

4.6.3 Architectural Pattern

In the following, the PPAP is written according to the template, given in Section 4.2. Its *Sketch* is illustrated in Figure 4.9.

Name

Private Network Address

Alias

Anonymous Network Communication

Example

Clients authorize themselves to a video streaming service using privacy *Privacy-Preserving Trust Establishment* to validate their age (see *Anonymous Authorization*). They watch videos at irregular times and in varying patterns to avoid further user profiling.

Context

Clients intend to access a cloud-based service while remaining anonymous. Reasons for this can be the use of the service including sensitive data, that an individual client does not want to disclose. Other anonymization solutions are already in use to protect privacy, such as *Anonymous Authorization* and *Location Granularity*.

Problem

Despite the fact that other privacy-preserving solutions are in use helping to make clients more anonymous, the server can nevertheless obtain information about the identity of the client by learning the origin of each client using the IP address. The following question arises: How to prevent the server from gaining sensitive information about a client by inspecting the IP address while the client is using the service (communicating with the server)?

Privacy aspect

The goal the pattern tries to achieve, is to protect the *identity* of the client by providing *anonymity*.

Attacker Model

In the scoped of cloud-based applications, as used in this thesis, clients connect directly to the server. Also in case of interaction with other clients, the messages are first sent to the *server*, which in most cases are processed and then sent to the other clients in some form. Thus the server is the main attacker in this problem.

Aim

The aim is to achieve anonymity by making properties, here the origin of the data, *unlinkable*.

Data

This problem addresses the origin of data packets *transmitted* by the clients arriving at the server node.

Solution

Hide the IP address of the clients by connecting to the cloud through an intermediary.

An example architecture is illustrated in Figure 4.7, using the *Client Server Pattern*. Instead of direct connection between server and client, a *Hider* component is placed in between. It splits the connector into two connectors, which connect the client and the server to the *Hider*, respectively. The *Hider* can be implemented by the architectural tactics VPN or Tor variants. It forwards the data packets in the appropriate direction without revealing the IP address of a client. Any traffic is routed through the *Hider*, which prevents the server from learning the client's IP address.

Implementation and Variants

The following variants are considered for implementation:

- Apply the architectural tactic *Onion Routing*.
- Apply the architectural tactic *Virtual Private Network (VPN)*. This solution variant involves a TTP.
- Combine the previous variants implementing the *Onion over VPN* approach by implementing the clients application that it first connects to a VPN server and uses Tor from then on.

Example Resolved

The problem in the video platform example is that even though other anonymous solutions have been applied, the server nevertheless can learn the origin of the clients packets. One solution is to build the architecture with a VPN Server between the CSP and the clients. All future connections from the clients, in form of IP addresses of the clients, are hidden from the server.

Consequences

Using this solution, clients are able to access cloud-based services without revealing their IP address. However, the implementation of this solution also has disadvantages in terms of performance or trustworthiness.

Although improvements can be made, the speed of using the Tor network is reduced compared to the connection without Tor. It becomes even less as the ratio of Tor users to relays bandwidth increases [DM09].

As far as trust is concerned, this thesis differentiates between two types.

In case of Onion Routing, trust is put into the technology, which is in this case open-source software (OSS). A large OSS community for comprehensive review by community experts and continuously code improvements are major components to provide high quality software [Abe07], which are present int the Tor project. Regarding trusting the exit nodes in Tor, the probability that these are malicious or incorrectly configured is negligible [WKM+14].

With VPNs, the trust is placed in the VPN Server resulting in a scenario including TTP.

Related Patterns

This solution depends on an existing cloud architecture, constituting a server component, offering a service to one or more client components. Therefore, it uses the *Client Server Pattern* as a basis.

This pattern complements patterns aiming to enhance the anonymity of clients regarding the *privacy aspect* identity, e.g. *Anonymous Authorization*. This solution does not protect against a server analyzing the behavior. To protect the clients' behavior, *Private Information Exchange* supplements this solution.

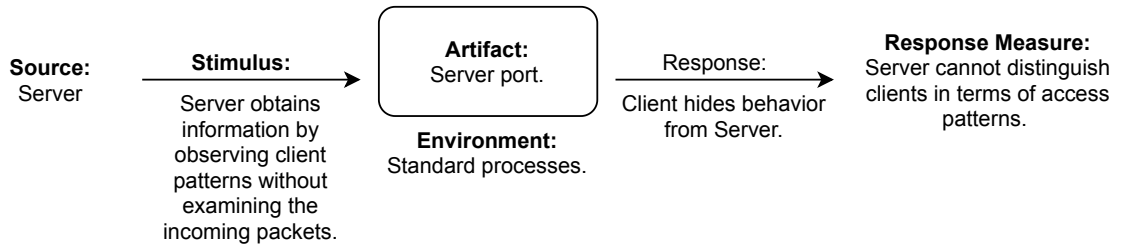

Figure 4.10: Specific scenario for *Private Information Exchange*.

In the current catalog of PPAPs, there are no excluding patterns.

4.7 Private Information Exchange

This section defines the PPAP *Private Information Exchange* following the second strategy of Hoepman [Hoe14], *hiding* PII (client behavior) from the server. In addition, some variations apply the principle of *separate*.

4.7.1 Scenario

Even if the server cannot read the content of incoming messages (e.g. by applying the client-side obfuscation pattern) and the origin of the IP address is obfuscated (Private Network Access pattern), client behavior patterns can reveal sensitive information about the person behind the client.

The scenario for the *Private Information Exchange* pattern emerges as follows and is illustrated in Figure 4.10. In this scenario, the source of the threat is the server. The stimulus emerges when the server can extract sensitive information from the behavior of the clients, and occurs during default sending and receiving (environment) of data on the server port (artifact). Clients can avoid the stimulus by concealing their behavior from the server by excluding it from communication to a certain degree, compared to normal communication. The response is successful, if the server is not able to create a profile from the behavior or at least only as far as the response allows.

4.7.2 Architectural Tactics

Below, architectural tactics are presented preventing the server from analyzing the behavioral patterns of clients.

Broker

One way to hide the clients behavior from the server, is to put a broker between clients and the server, which is considered to be trustworthy. It receives all messages from the clients and acts as a client towards the server by accessing data in the cloud on behalf of the respective client. It then forwards the responses to the respective clients.

Client-Centric Control and Communication

This architectural tactic aims at ensuring that the server does not receive the sensitive data. It suggests to move parts of the cloud service to the client side by means of direct client communication, or processing of data at the clients devices.

The cloud scope described in Section 4.1 is modified, allowing direct communications between clients: Clients can interact by not transmitting all messages exclusively through the malicious server. Thus, control is placed in the hands of the clients by communicating directly with each other. Another approach is to avoid communication with the server, by storing and processing parts of the personal data on the client side.

This architectural tactic is extracted by the specific scenario explored in the German *Corona Warn App*[6], which implements direct communication via Bluetooth in local proximity. Moving the control of personal data towards the client complies with the approach of enhancing privacy by building *client-centric* software architectures [SC08].

Metadata-Hiding Communication

Metadata-Hiding communication allows client communication with the server, without revealing when an actual data transfer takes place. A client sends either messages with meaningful content or fake ones (*dummy traffic* [BL02]). The effectiveness of this approach depends on the server not being able to distinguish between the fake and real messages. Thus, data is obfuscated by the clients in such a way that the server cannot tell when it is fake entries and when it is not. Client-Side obfuscation is presented in the respective pattern in Section 4.4.

This architectural tactic is extracted by the specific scenario explored in the work of Eskandarian et al. [ECM20]. They propose a payment splitting application, enhancing privacy by having all participants upload entries at specific times [ECM20]. When no entry needs to be uploaded, a fake messages is being sent instead.

Information Broadcasting

This architectural tactic prevents the server from knowing when a client receives personal data. It sends a bundle of data with personal information to multiple clients in regular intervals. Respective clients can select the data they are interested in from the bundle, making it difficult for the server to find out which client has received which data. In contrast to the previous tactic, it protects the behavior of clients in the event of data flows from the server to the clients.

This architectural tactic is extracted by the specific scenario explored in the German *Corona Warn App*[7]. The server provides a list of information, containing the corresponding keys of infected persons. In the process of *Direct Client Communication*, clients exchange keys with other clients nearby. These obtained keys allow a client to detect an infected person in the list it has encountered in the past. The application differs in that the list is made available online and clients can download

[6] https://www.coronawarn.app/en/
[7] https://www.coronawarn.app/en/

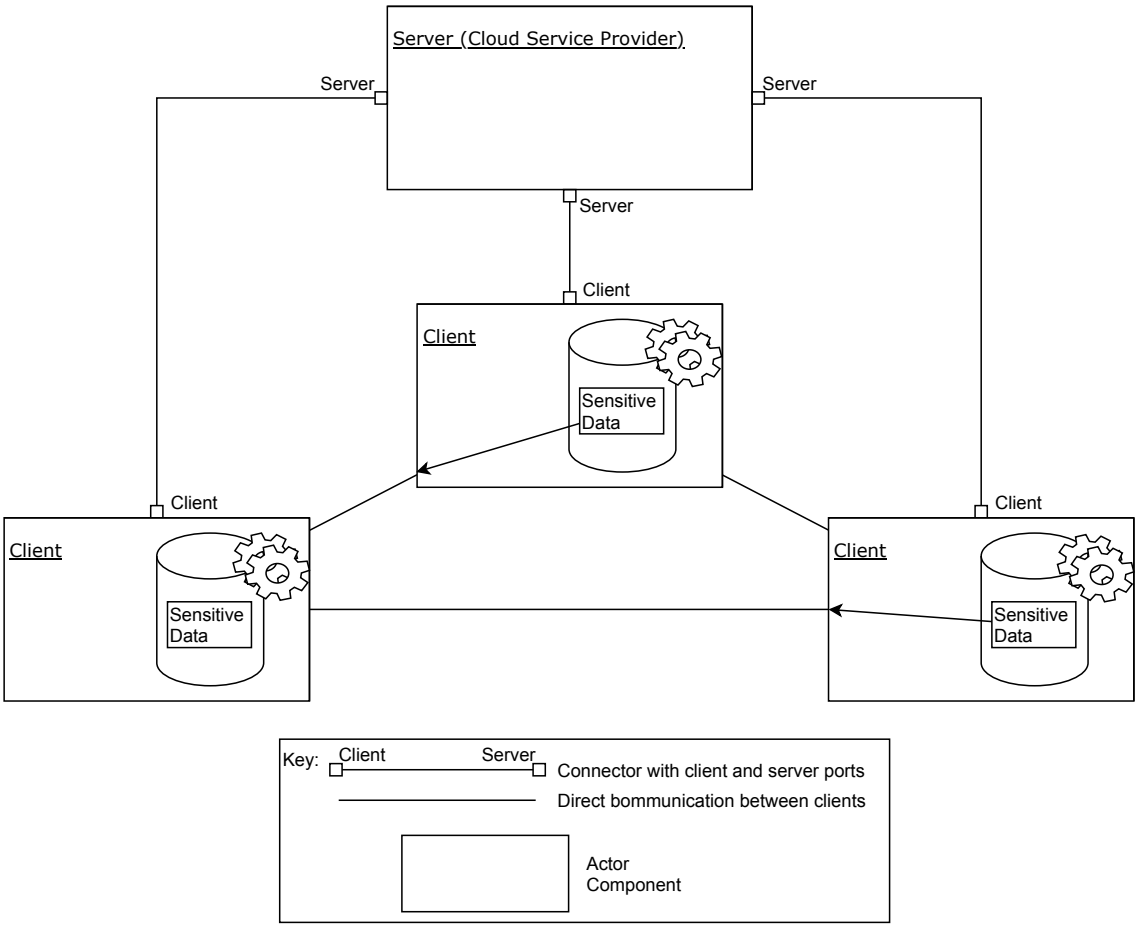

Figure 4.11: Sketch of *Private Information Exchange*.

it anytime. Since there is no login mechanism, it is difficult for the server to figure out which clients are accessing the list. Therefore, the clients' behavior patterns are considered to be protected in the *Corona Warn App* as well.

4.7.3 Architectural Pattern

In the following, the PPAP is written according to the template, given in Section 4.2. Its *Sketch* is illustrated in Figure 4.11.

Name

Private Information Exchange

Context

A cloud service is offered whose purpose includes the interaction of multiple clients. All communication takes place via the server as intermediary.

Problem

Acting as intermediary, the server is able to gain sensitive information about each client by analyzing the behavioral (service access) pattern. When within a group of clients, for example, one client is sending significantly more messages or messages at certain times, the server can use this metadata to improve its user profiling of the clients.

The question arises: How can the server be prevented from monitoring the behavioral pattern of its clients by examining the communication flows?

Privacy aspect

The goal this pattern tries to achieve, is to protect information disclosed by *behavioral* patterns.

Attacker Model

The attacker in this pattern is the *malicious server*, which tries to gain information by analyzing the behavioral pattern.

Aim

The solution aims to make the accesses of a client *indistinguishable* from other accesses of the same client or others.

Data

This pattern deals with the information flow between clients and server; thus addresses data *in transit*.

Solution

Clients hide their behavior from the server. There are several ways to achieve this. One is to exclude the server from the client's service usage to some extent. Another way is to modify the server's perceived view by modifying the client's behavior artificially.

An example architecture is depicted in the following, using the *Client Server Pattern* and the *Client-Centric Control and Communication* variant. In addition to the connectors between each client and the server, connectors are established between clients. These connectors enable *direct communication* between clients to the exclusion of the server. Each client is assigned an additional memory component for storing personal data (*client-centric control*).

Implementation and Variants

The following variants are considered for implementation:

- Apply the architectural tactic *Broker*.
- Apply the architectural tactic *Client-Centric Control and Communication*.
- Apply the architectural tactic *Metadata-Hiding Communication*.
- Apply the architectural tactic *Information Broadcasting*.

There are various combinations possible for simultaneous implementation, e.g. *Information Broadcasting* for protecting data from the server to the client and *Client-Centric Control and Communication* vice versa.

Known Uses

At the time of writing, the nationwide available *Corona warn App* is used to reduce the spread of *SARS-CoV-2* in Germany. People using the app are informed when they had contact with a person who is tested positive. Here the second variant comes into play. The disclosure of peoples' behavior is protected by only communicating with nearby persons via Bluetooth. Only a person found positive is sending her or his encrypted key to the server while the others do not upload anything at all. Thus the server never knows which persons had contact or where in the same place at the same time. In the corona case, this solution is complemented by the pattern *Information Broadcasting*.

Consequences

All variants ensure that the clients can run the service while, with respect to the clients behavior, keeping the server in the dark.

Using a *Broker* as intermediary can completely prevent the server from gaining information by the behavioral pattern. In case the broker handles each client separately, it can buffer messages and send it at a later time. When it sends all receiving messages from the clients, acting as one client itself, their behavior is protected since the server never knows which clients are connecting. The *broker* approach comes also with some downsides. First it can be a single point of failure meaning

in case of a breakdown, no message can be received and forwarded to the server; and vice versa. Second, the broker carries great responsibility by acting on behalf of the clients. Therefore, it has to be considered trustworthy from the clients point of view. In addition, The scope of possible applications for a service may be limited.

The second variant, *Client-Centric Control and Communication*, protects from the server analyzing the pattern and does not need a TTP. However the decreased controllability of the server comes also with a price. If the service requires the processing of this sensitive data, excluded from the communication with the server, it is not an option.

While the degree of indistinguishability is less, the *Metadata-Hiding* variant on the other hand allows the server to use the data sent by clients. Since *Client-Side Obfuscation* is needed (see *related patterns*), this solution only works if the server still can process the encrypted data or it does not have to process it at all. For processing on encrypted data, see the *Private Data Processing* pattern. Using this pattern, though, probably introduce new downsides like reduced performance of the service. Thus, a service with too many clients may be not applicable.

Information Broadcasting protects the behavior for data sent from the server to the clients. Clients may unintentionally learn about sensitive data from others, which is why additional mechanisms such as encryption can be employed, see supplements. Another disadvantage is that the server sends more than in normal operation, where individual messages are sent to clients. Furthermore, the size of the bundle could be a problem, for example for low-powered devices.

Related Patterns

This solution depends on an existing cloud architecture, constituting a server component, offering a service to one or more client components. Therefore, it uses the *Client Server Pattern* as a basis.

The server can detect fake messages by looking at the content if the *metadata-hiding* approach is chosen. In the case of *Information Broadcasting*, the same holds true for other clients which could learn more than it is intended. Therefore, this pattern complements *Client-Side Obfuscation* hiding the actual content can be supplemented for these two variants.

In the current catalog of PPAPs, there are no excluding patterns.

4.8 Pseudonymous Identity Management

This section defines the PPAP *Pseudonymous Identity Management*, adhering the following strategies [Hoe14]: *Hiding* and *minimizing* the disclosure of PII, by using pseudonyms.

4.8.1 Scenario

One possible scenario for this PPAP is explained in the following, and illustrated in Figure 4.12.

A client subscribes to a service and interacts with other clients. During registration, he provides PII to the server, for example, full name and location. The *stimulus* consists of revealing this personal information to other clients (*source of stimulus*), while they are communicating. Interaction with

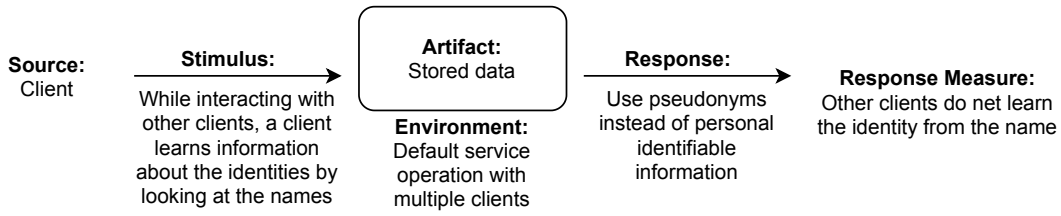

Figure 4.12: Specific scenario for *Pseudonymous Identity Management*.

other clients represents a regular process within the service operation (*environment*) and concerns data stored on the cloud (*artifact*). The problem is eliminated by the server allowing clients to use pseudonyms (*response*). During registration, a client can choose a pseudonym. When interacting with other clients, this pseudonym is presented instead of other PII. The strength of the response is measured by the degree of linkability (*response measure*). E.g., a client chooses a pseudonym, consisting of the first letter of his first name and the complete last name. The linkability is potentially greater than with a randomized pseudonym.

The server can form the *source of the stimulus* as well. Pseudonyms can provide a more anonymous registration. Therefore, this pattern can enhance privacy, increasing unlinkability of the client. However, in some applications, personal data is needed for the service to operate properly, giving the server access to this data regardless. This thesis assumes the client to be the main attacker. Since there are use cases, where pseudonymization also protects against the server (e.g., registration in forums), the server is included in the pattern as a possible attacker.

4.8.2 Architectural Tactics

Three architectural tactics are identified and presented below, establishing pseudonyms in different ways. Choosing the pseudonym is done by the server (e.g. random created user name) or by the client.

Pseudonymous Registration on-Premises

The pseudonym is chosen and stored during registration on the server side.

Pseudonymous Registration via Single Sign-On

The server implements single sign-on (SSO), and acquires the pseudonym from the identity provider.

Pseudonymous Registration via Privacy-Preserving Trust Establishment

During privacy-preserving trust establishment, the server adopts the token id as a pseudonym. For example, privacy attributed-based credentials can be used and the pseudonym is given in the process of trust establishment.

4.8.3 Architectural Pattern

Name

Pseudonymous Identity Management

Alias

Use Pseudonyms.

Context

Clients using a service interact with each other. Representative names have to be used for interactions in the future to resort to these names allowing a client to recognize other clients by their names. E.g. on social networking platforms a client can look up a client by remembering its name or scroll through a list of names.

Problem

This name can contain sensitive information usable for the re-identification of a client. For example, if the first and last name is used in combination with other sensitive information of its profile, these can be combined to link the respective client to the real person behind.

The following question arises: How to prevent other clients from seeing personal information of a client while interacting with it.

Privacy aspect

The goal this pattern tries to achieve, is to protect the *identity* by providing *pseudonymity*.

Attacker Model

One source of threats is represented by other *clients* which can gain sensitive information. They are assumed to follow the protocol as it is intended.

In addition, the attacker model includes the *server*. It can access PII, provided by the clients.

Aim

This pattern aims to provide a solution for supporting *unlinkability* of a client to the real identity.

Data

The names are *stored* on the *server*.

Solution

The server stores a pseudonym for each client with which it logs into the system and interacts with other clients. Typically, a database is used in which the pseudonym and an associated password are stored. The client can log in and interact with other clients without revealing PII. From an architectural perspective, for example, a database as well as a login and registration mechanism are represented by corresponding components on the server side. The way a pseudonym is established can vary, depicted in the *Implementation and Variants* element.

Implementation and Variants

For the implementation of the pseudonym generation the following variants are considered:

- Apply the architectural tactic *Pseudonymous Registration on-Premises*.
- Apply the architectural tactic *Pseudonymous Registration via Single Sign-On*.
- Apply the architectural tactic *Pseudonymous Registration via Privacy-Preserving Trust Establishment*.

Consequences

Using pseudonyms mitigates the disclosure of sensitive information, enhancing unlinkability.

As discussed in the consideration of the *privacy aspect* in Section 4.2, the usage of pseudonyms impact the degree of unlinkability. The following trade-off can occur: Providing high unlinkability by choosing a random pseudonym results in decreased usability, when clients are interacting with each other.

The attacker model comprises clients and the server. But some applications require personal information for running the service. This data the server can access regardless.

A hybrid approach is possible, where pseudonyms are only used for a single session. Then the degree of linkability is between pseudonymity and anonymity. Anonymity is provided between sessions and pseudonymity within a session.

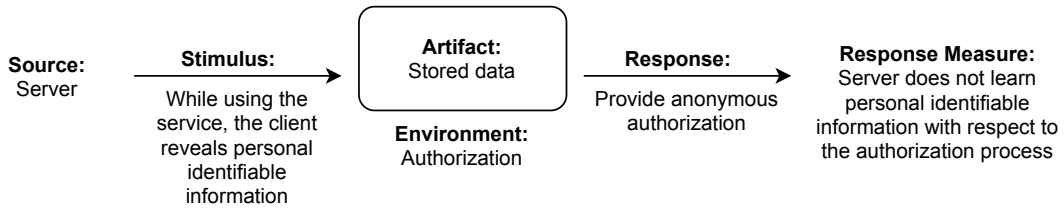

Figure 4.13: Specific scenario for *Anonymous Authorization*.

Related Patterns

This solution depends on an existing cloud architecture, constituting a server component, offering a service to one or more client components. Therefore, it uses the *Client Server Pattern* as a basis. If the third variant, *Pseudonymous Registration via Privacy-preserving Trust Establishment* is used, the pattern *Anonymous Authorization* has to be applied in addition. It provides the usage of privacy attribute-based credentials.

In the current catalog of PPAPs, neither excluding, nor complementing patterns are found. Pseudonyms are often used in practice, independent of other technologies (or patterns).

4.9 Anonymous Authorization

This section defines the PPAP *Anonymous Authorization*. It adheres to the following strategies: *Minimizing* personal data, by *hiding* it from the server.

4.9.1 Scenario

One possible scenario for this PPAP is depicted in the following, and illustrated in Figure 4.13.

To use a cloud-based service, a client must verify certain characteristics, e.g. the age. During registration, a client discloses personal data, which can be exploited by the server to infer the identity (*stimulus*). The *source of stimulus* is therefore the server. Since the client has to verify properties, the stimulus concerns the *environment* of authorization. The concerned data is located on the server (*artifact*). *Responding* to the stimulus, the server provides registration via anonymous authorization. With such techniques, clients can use the service without revealing personal data. To enable anonymous authorization, a TTP is usually assumed. The *response measure* is binary: Either the server learns PII in the authorization process through credentials, or it does not.

Other clients are not involved in the authorization process and therefore not considered as *source of stimulus*. Unless they collude with the server, or exploit a vulnerability in the system, they are not able to gain personal data in that process. Both variants are excluded in this thesis: Section 4.2 states that collusion between server and clients is neglected due to the scope defined in Figure 4.1 (public cloud). Eliminating vulnerability is the responsibility of the server and falls into the security domain.

4.9.2 Architectural Tactics

Two architectural tactics are identified, preventing the server to gain personal information of the client in the authorization process.

Privacy-Preserving Trust Establishment (PPTE)

Attribute-based credentials allow servers to grant clients the use of a service by revealing only data (the attributes) necessary to be permitted the use of the service (e.g., being of a certain age). Although this architectural tactic requires a TTP for setting up the credentials in the first place, the presence of the TTP in the authorization process is not mandatory.

For implementations see the work of Camenisch and Van Herreweghen, Persiano and Visconti, or Paquing and Zaverucha [CV02; PV04; PZ11].

Single Sign-On (SSO)

The server implements single sign-on (SSO), putting the trust to a third party (the *identity provider*). SSO describes a logon procedure that grants a user access to multiple services and resources after a one-time authentication at a trusted third party (identity provider). For more details about SSO, this thesis refers to the work of De Clercq [De 02].

4.9.3 Architectural Pattern

Name

Anonymous Authorization

Example

A service provides a video streaming platform with some movies containing mature content. Only people that are at least 18 years old are allowed to watch these videos.

Context

Clients registering at the provided cloud-based service.

Problem

Registering disclose sensitive information of the clients. How to prevent the server from gaining sensitive information about the clients when they register?

Privacy aspect

Anonymity (identity). The goal this pattern tries to achieve, is to protect the *identity* by providing *anonymity*.

Attacker Model

The main threat represents the *malicious server* since it is involved in the process of registering.

Aim

The solution aims for increasing the *indistinguishability* and *unlinkability* of clients.

Data

Sensitive information is *stored* in the cloud (*server*). This solution addresses the problem of storing personal information.

Solution

Instead of having to register with personal data, clients use anonymous authorization techniques that allow them to prove properties without revealing other personal data.

From an architectural point of view, an authorization component is added to the connector port on the server side. Verifying the required properties varies depending on the implementation variant.

Implementation and Variants

The following variants are considered for implementation:

- Apply the architectural tactic *Privacy-Preserving Trust Establishment (PPTE)*.
- Apply the architectural tactic *Single Sign-On (SSO)*.

Example Resolved

Clients authorize with *attribute-based credentials*. The server only learns that a user is of accepted age or not.

Consequences

Clients stay anonymous and are not linkable between different sessions, because from the server's point of view, each client is different.

The first simple variant is the easiest to implement, but does not provide the control of clients fulfilling specific properties.

Another downside is the trade-off with usability. There are only few use cases where this solution can be applied. Since it is the nature of services themselves, they often rely on user interactivity and storing information. In this case, it is recommended to consider the *Pseudonymous Identity Management* pattern.

For both variants, usability can be increased by using cookies. But especially when the same cookies are used between different sessions, the privacy aspect vanishes.

Independent of the variant, the server is able to analyze the behavioral pattern of each client, for example when clients are watching the same type of videos at the same time everyday. If cookies are used, the degree of linkability increases as well.

Related Patterns

This solution depends on an existing cloud architecture, constituting a server component, offering a service to one or more client components. Therefore, it uses the *Client Server Pattern* as a basis.

This pattern complements patterns aiming to enhance the anonymity of clients regarding the *privacy aspect* identity, e.g. *Private Network Access*. In conjunction with the pattern *Pseudonymous Identity Management*, anonymous authorization can be chosen as a starting point for a client to prove certain properties and then to continue using the system with a pseudonym and password.

In the current catalog of PPAPs, there are no excluding patterns.

4.10 Access Control

This section defines the PPAP *Access Control* adhering to the following strategies: Preserving the *confidentiality* of a clients data, and *hiding* it from other clients.

4.10.1 Scenario

The PPAP-specific scenario is depicted in the following, and illustrated in Figure 4.14.

Multiple clients use a service. There is the risk of a client being able to access more data from another client, than is necessary for the regular operation of the service (*stimulus*). Therefore, clients constitute the *source* of the scenario. The *artifact* concerns elements, accessing data on the server side. The *environment* involves the regular operation of multiple clients, where the clients are

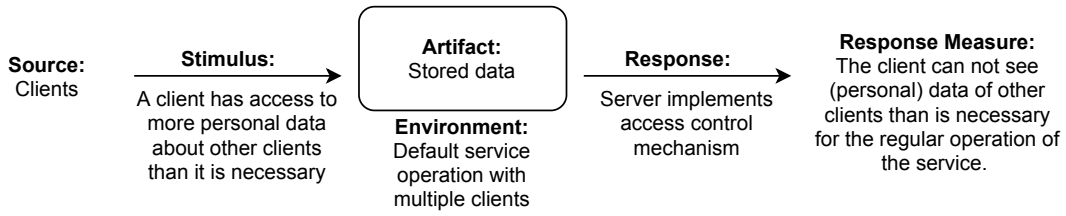

Figure 4.14: Specific scenario for *Access Control*.

considered honest but curious. The stimulus is solved by the server, implementing access control mechanisms (*response*). The *response measure* is the degree, to which a client *might* access data from another client that it should not be able to access.

4.10.2 Architectural Tactics

One architectural tactic is identified and presented in the following: *Limited Access*.

This thesis assumes privacy and security forming two different fields overlapping in partial areas. For example, some security measures can be applied to protect personal data, as in the case of confidentiality and integrity. Schumacher et al. presented patterns for the field of security in software engineering [SFH+13]. Part of these are the *system access control architecture patterns*, including the *Limited Access* pattern. This pattern is used to design architecture components, restricting users to access only certain data and deny the access to other data. In case of this thesis, this pattern is adopted as an architectural tactic, restricting access to personal data, that a client may or may not access.

The pattern involves the usage of other patterns as prerequisite (corresponding to the *dependency* property of the *Related Patterns* element). For detailed descriptions of the patterns, this thesis references to the work of Schumacher et al. [SFH+13].

4.10.3 Architectural Pattern

Name

Access Control

Context

Multiple clients use a service provided by the CSP by provisioning individual physical IT resources to multiple clients.

Problem

Multiple clients can access the same storage and use the same functions other clients use. The risk arises that a client can use this to access resources and functions of other clients containing or providing sensitive data, respectively. The question arises: How can the server prevent one client from gaining sensitive information from another client, despite sharing individual IT resources with multiple clients?

Privacy aspect

Unauthorized access is restricted to gain sensitive *content* of another client.

Attacker Model

The attackers constitute *clients* that are assumed to be malicious.

Aim

The content is kept *confidential* so that other clients do not gain access to sensitive information they are not supposed to.

Data

The solution is targeted at data, which is stored on the server.

Solution

Access control mechanisms are used to logically separate the IT resources used by the clients and to ensure that a client can only access the data that the service involves.

From an architecture perspective, components are added on the server side that restrict client access to certain areas, e.g. components by the *Limited Access* pattern.

Implementation and Variants

The architectural tactic *Limited Access* is considered for implementation.

Consequences

Access control allows to set access permissions for clients, allowing clients to access only the personal data within their access permissions. With the required security measures in place, this makes it significantly more challenging for malicious clients to access unauthorized data.

A disadvantage is the higher costs for implementation and maintenance compared to other architecture measures.

Related Patterns

This solution depends on an existing cloud architecture, constituting a server component, offering a service to one or more client components. Therefore, it uses the *Client Server Pattern* as a basis.

In the current catalog of PPAPs, neither excluding, nor complements patterns are found.

5 Methodology

The architect's task is to incorporate the various interests of the stakeholders into the software architecture. Addressing the quality attribute privacy, Chapter 4 establishes the definition of a PPAP, and identifies 7 PPAPs. The architect can leverage their structure to select the appropriate patterns, for building a privacy-friendly software system. The catalog of PPAPs is not complete, implying more patterns can be added in the future. Each Pattern can contain variants in the form of architectural tactics. Therefore, an architect has potentially many combinations of solutions available. The question arises how an architect chooses the appropriate patterns and variations, that can be used in a specific application context. The context is given by the stakeholders' interests, and the scope of the system. This chapter proposes a methodology for PPAP selection, solving this problem.

This chapter is structured as follows: First, the objectives of the methodology are presented. Then, *Functional Principle and an Overview* outlines the structure of the methodology and includes incorporation of the considerations from Chapter 3. The subsequent sections describe the individual steps in detail.

5.1 Objectives

The methodology aims at fulfilling the following objectives: *Reproducibility*, *effectiveness*, low *complexity* in use, and *transferability* to other pattern domains.

The methodology should be *reproducible*, implying it outputs the same selection of patterns when the architect reruns it in the same environment. The environment constitutes the privacy requirements, application context and the pattern catalog. It should always select PPAPs and output the ones that are needed and applicable in the architect's context (*effectiveness*). The application of the methodology is supposed to be straightforward and not to require much expertise (*complexity*). The methodology ought to be applicable *independent* of the type of pattern and domain the patterns address.

Chapter 7 discusses the achievement of these objectives, considering the application to the methodology on the use case in the next chapter.

5.2 Functional Principle and Overview

The first part of the methodology consists of the pattern elicitation, illustrated in Figure 5.1. There are 3 major components in the elicitation process: The pattern catalog, the architect, and the *pattern elicitation*. The pattern catalog consists of the 7 identified PPAPs and provides one input to the

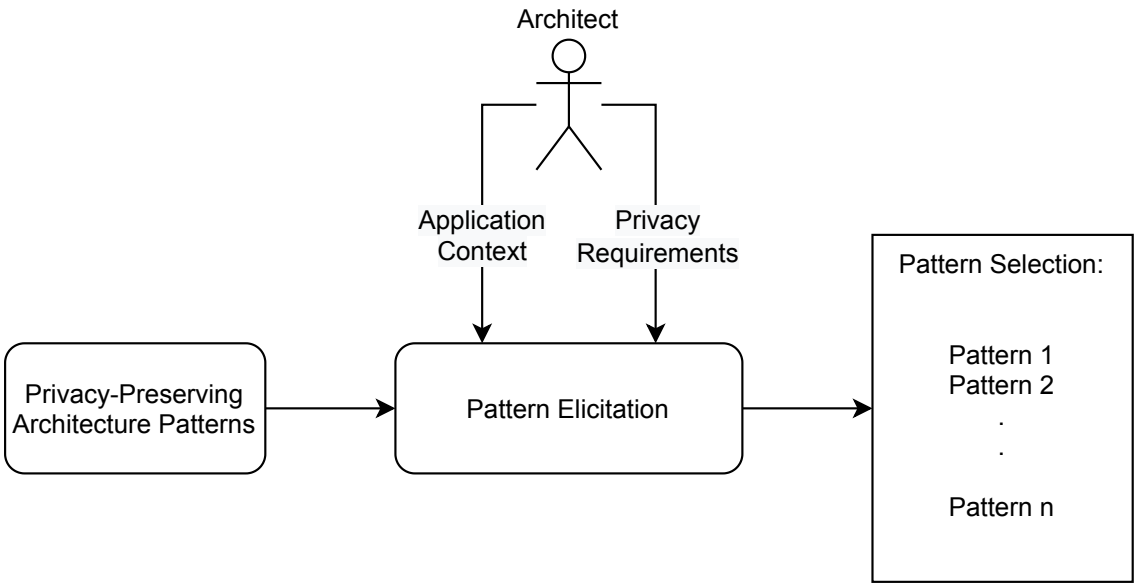

Figure 5.1: Abstract View of the Pattern Elicitation.

pattern elicitation. The architect provides the scope as well as the privacy requirements of the system to the *pattern elicitation*. The *pattern elicitation* then automatically performs its selection part of the process and returns a subset of the patterns given from the catalog. The architect decides on the final selection of the remaining patterns and on their implementation variant. In the following, the functional principle of the methodology is presented first; and then the individual steps are presented.

The definition of a PPAP includes privacy properties, adapting the privacy dimensions of Heurix et al. to match the abstraction level of architectural patterns: *Privacy aspect*, *attacker model*, *privacy aim* and *data* [HZNF15]. Each PPAP addresses these embedded privacy properties in a different way, which allows the selection of the appropriate patterns, depending on the given privacy requirements. Selecting PPAPs comprises four steps, illustrated in Figure 5.2.

The systems of Pearson et al. address privacy patterns that follow the privacy-by-policy principle and do not have the abstraction level of an architectural pattern [PB10; PS10]. This thesis proposes a methodology for the selection of PPAPs, constituting a semi-automated approach. It considers the privacy properties a PPAP fulfills, takes into account the contextual environment of the software system, and saves the architect from having to go through guidelines. Following the different stakeholder's interests, the architect decides which privacy properties the software system has to fulfill, e.g. protecting the *identity* in case of *privacy aspect*. The contextual environment specifies the scope of the system, e.g. the involvement of third parties or the amount of clients.

In cloud computing, Pearson claims that PIAs are preferable to design patterns, because the template structure of a pattern cannot cover the situation-specific context [Pea09]. The selection process presented here addresses this issue, by incorporating the context and privacy properties with the support of the architect. The architect eventually decides which patterns to implement and considers the architectural tactics for the implementation. The choice of implementation also affects the trade-offs with respect to other quality attributes, and is addressed within the PPAPs *consequences* element.

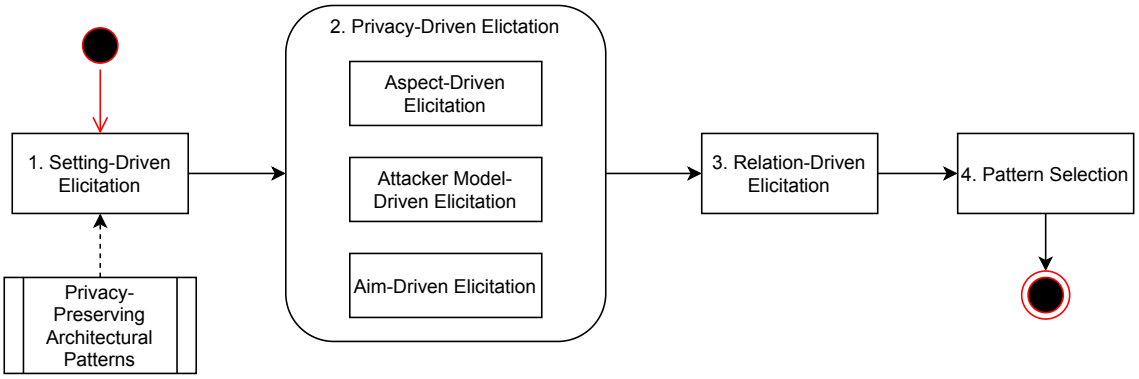

Figure 5.2: Overview of the methodology: Steps of the pattern elicitation (1-3) and the final step of the architect's selection (4).

The first step considers the situation-specific context, referring to the scope, presented in Section 4.1. Step 2 determines the privacy properties: *Privacy aspect*, *attacker model*, and *aim*. They are combined in one step, since their order can be chosen arbitrarily. The sequence of first considering the *privacy aspect*, then the *attacker model* and finally the *aim*, is chosen, as it is assumed to represent the properties in descending order of importance. The *data* element is not considered, as it is assumed to be irrelevant for the selection of PPAPs. It is intended to give additional information about the problem description, for the architectural view and the implementation in the next SDLC phase. The second last step evaluates the PPAPs' relations, included in the respective *related patterns* element. Finally, the architect decides on the selection of the remaining patterns, including the choice of implementation (listed in the *implementation and variants* element).

If the outcome is not satisfying, the architect can trigger a new run of the process, or jump back to a specific step to change the specific parameters. For example, the architect first attempts to design the system providing anonymity, and select the *privacy aspect* to be fulfilled only with patterns providing anonymity (identity). The methodology does not output any satisfying patterns. After consulting with the stakeholders, the architect re-runs the process, changing the *privacy aspect* to be fulfilled by protecting the identity through pseudonyms.

In the beginning, the complete pattern catalog is available. Each elicitation step reduces this set of patterns, depending on the decisions set by the architect. After an elicitation, the remaining patterns are passed on to the next step.

In each elicitation of step 2, the architect has the possibility to maintain the pattern set, by not deciding on any specific value for the respective privacy property. This action is called *skip*, because maintaining the current set of patterns is the same as skipping the particular step.

In the following, all steps are described individually, concluding with an illustration of the remaining patterns for each step, except 4. For clarity, each elicitation assumes the full pattern catalog as input, avoiding the presentation of all potential combinations. The same result is achieved by intersecting (in terms of the set theory) the outputs of each elicitations.

Pattern	|C| = 1	|C| > 1	TPP	TTP
Client-Side Obfuscation	No	Yes	No	No
Private Data Processing	Yes	Yes	Yes	No
Private Network Access	Yes	Yes	Yes	No
Private Information Exchange	No	Yes	Yes	No
Pseudonymous IDM	Yes*	Yes	Yes	No
Anonymous Authorization	Yes	Yes*	Yes	No
Access Control	Yes*	Yes	Yes	Yes

Table 5.1: Setting-Driven Elicitation: Passing on variants dependent on the scope settings $|C| = 1$ (exactly one client), $|C| > 1$ (multiple clients), TPP (is involved in the system), and TTP (is considered for the solution). A cell marked with *yes* implicates the forwarding of all variants, and *no* otherwise. Cells marked with a star (*) are special cases being discussed in the respective section.

5.3 Setting-Driven Elicitation

Section 4.1 defines the scope of this thesis, depicting the involved participants. This step examines the situation-specific variation of this scope. In contrast to the other steps, which reduces entire patterns, the variations (architectural tactics) of patterns are investigated. These *settings* are not embedded in a PPAP, since they depend on the specific situation of the considered software system and on specific architectural tactics.

The first two settings examine the amount of clients using a cloud service in isolation: Either *one client* uses a service in isolation, or *multiple clients*. Involvement of third parties determines the remaining two settings. There are two types of third parties: *TPPs* for further processing of personal data *and TTPs* that have to be involved to fulfill a certain privacy property.

In the following, for each of the four settings, all pattern variants are examined and inappropriate ones are eliminated. Step 5 uses this elicitation to select variants for implementation. Table 5.1 provides an overview about the results of each setting. It specifies for each pattern in the respective setting, whether this step passes on all variants (marked with *yes*) or eliminates at least one variant of the respective pattern (marked with *no*).

Elicitation Depending on Number of Clients

In the context of applications where *one client* uses the service in isolation, two variants are found that cannot be applied. The variant *Direct Client Communication*, which is included in the pattern *Privacy Information Exchange*, excludes the server in parts from communication with other clients in order to protect the behavior of the clients from the server. Since there are no other clients to interact with, this variant can be excluded. The second variant is the *Local Differential Privacy* tactic of the *Client-Side Obfuscation* pattern. Its performance is mitigated, since it depends on a larger number of clients.

Pattern	Tactics to be removed
Client-Side Obfuscation	Tokenization with TTP
Private Data Processing	Trusted Execution Environment
Private Network Access	Virtual Private Network
Private Information Exchange	Broker
Pseudonymous Identity Management	1) Privacy-Preserving Trust Establishment (PPTE) 2) Single Sign-On
Anonymous Authorization	Privacy-Preserving Trust Establishment (PPTE)

Table 5.2: TTP-Driven Elicitation: Eliminated Variants when a TTP is not considered.

There are two patterns worth considering if they should be applied in the context of isolated clients: *Pseudonymous Identity Management* and *Access Control*. Both act in the attacker model of malicious clients, i.e. they are supposed to protect against other clients. Since *Access Control* also protects against access by (unauthorized) clients who access the same service despite isolation, the pattern is maintained. *Pseudonymous Identity Management* is also maintained, because in case of data leakage, it makes linking more difficult if a pseudonym is disclosed instead of personal identifiable information. This is the consequence of the decision made in the attacker model to exclude external threats, assuming to be covered by malicious clients or servers. E.g., a malicious client represents an external threat, when it attempts to gain access to personal data that is not within its scope of service.

All variants contained in the identified patterns can be used in applications that involve *multiple clients*. Several clients can use a service together, despite *Anonymous Authorization*, which is the reason for keeping the pattern and its variants.

Elicitation Depending on Third Party Involvements

If the setting involves a *TPP*, the *Tokenization* variant is excluded from the *Client-Side Obfuscation* pattern. It replaces personal identifiable data with a randomized string or number.

If the architect decides not to include a *TTP*, each PPAP of the current catalog, except *Access Control*, is truncated by at least one variant that requires a TTP to work. Table 5.2 lists for each pattern the tactics that are excluded from consideration.

5.4 Privacy-Driven Elicitation

Elicitations are presented below, each addressing a different privacy property: *Privacy aspect*, *attacker model* and *aim*. These are embedded into the definition of a PPAP, allowing a semi-automatic selection.

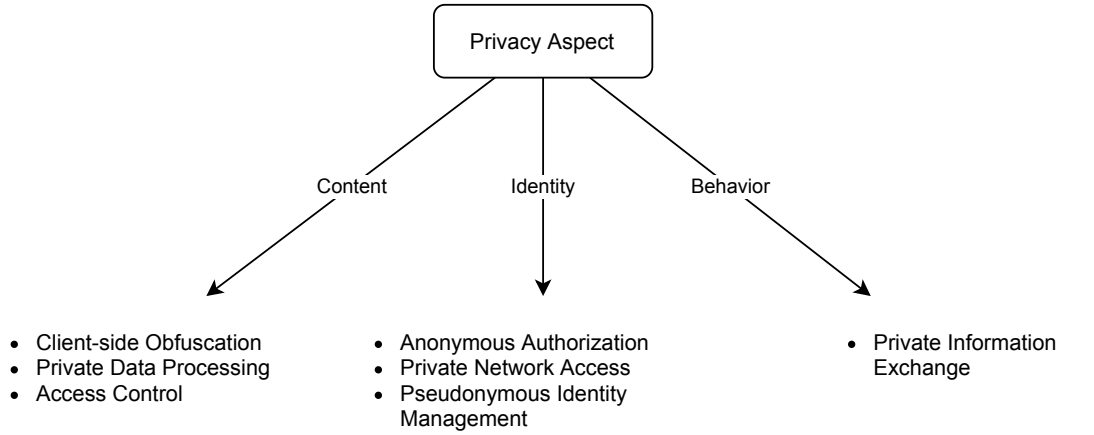

Figure 5.3: Elicitation of patterns with respect to the privacy properties of *privacy aspect*: Content, identity, behavior.

5.4.1 Privacy Aspect-Driven Elicitation

This elicitation step selects patterns by their privacy properties. In this step the architect has to decide which privacy aspect is of most importance, and/or makes most sense in the specific context: *Content*, *identity* or *behavior*. If *content* is chosen, the patterns *Client-Side Obfuscation*, *Private Data Processing* and *Access Control* are preserved. The rest is removed from consideration. In the case of the *Behavior* aspect, one pattern remains: *Private Information Exchange*. With *identity* the patterns *Anonymous Authorization*, *Private Network Access* and *Pseudonymous Identity Management* are available.

The architect is also given the option to select a combination of the aspects, then the selected patterns are aggregated. The skip operation described above corresponds to selecting all three aspects.

Figure 5.3 illustrates the pattern selection possibilities, when choosing the respective privacy aspect value.

5.4.2 Attacker Model-Driven Elicitation

In Chapter 4 the attacker model is defined, involving clients or the server as adversaries, which are either malicious or honest-but-curious. This step aims at selecting patterns that match the adversary model chosen by the architect. Those patterns are maintained that build architectural parts to protect against the corresponding adversary. If the client represents the attacker, the patterns *Access Control*, *Pseudonymous Identification* and *Private Data Processing* are selected. In case of the server being the attacker, the other four remaining PPAPs are used for the remaining process, and additionally *Private Data Processing*, since it protects against malicious clients. In Figure 5.4, the patterns are listed graphically according to the chosen adversary.

There is one possible combination to mark both participants as adversaries, implying that all patterns are forwarded.

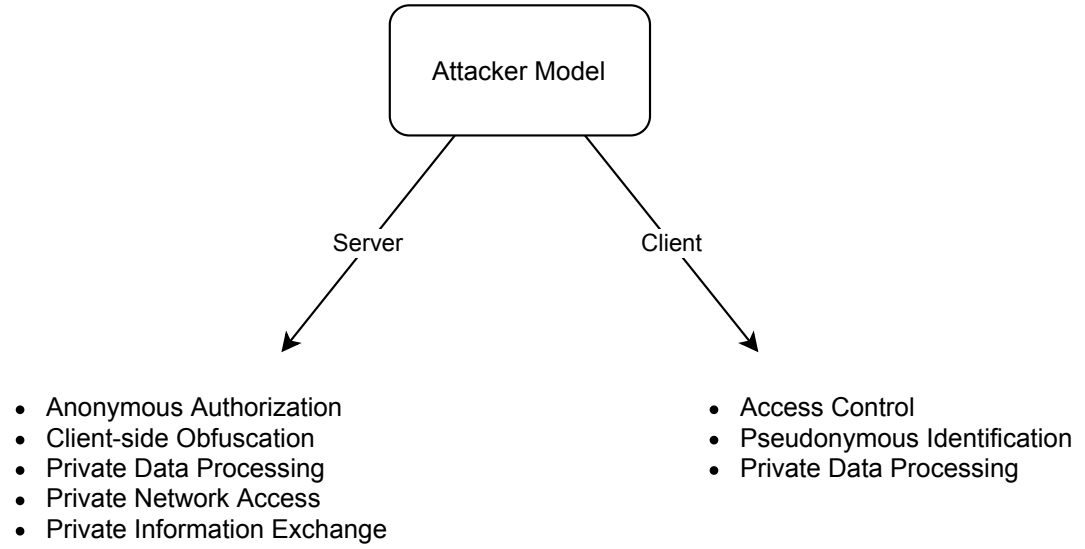

Figure 5.4: Elicitation of patterns with respect to the the privacy properties of *attacker model*: Server and client.

5.4.3 Aim-Driven Elicitation

An additional way to distinguish between patterns comprises the privacy property aim. This thesis divides aim into unlinkability, indistinguishability and confidentiality. The pattern *Anonymous Authorization* is a special pattern, since it supports the first two aims mentioned above. Therefore, it is included in both cases. If the architect chooses unlinkability, the patterns *Anonymous Authorization*, *Private Network Access* and *Pseudonymous Identification* remain. If the aim indistinguishability is adopted, the two patterns *Anonymous Authorization* and *Private Information Exchange* must be retained. Choosing Confidentiality results in the patterns *Access Control*, *Client-Side Obfuscation* and *Private Data Processing*. The 3 variants are illustrated in Figure 5.5.

Four further combinations are possible, in which the respective patterns are combined to a set (without duplicates). The methodology skips this step, if the architect selects all three aim variants.

5.5 Pattern Relation-Driven Elicitation

In the penultimate step the relations of patterns, which are elicited so far, to other patterns are examined. Relations are embedded in the *related patterns* element of a PPAP. The possible relations are *dependencies*, *supplements* and *exclusions*.

Every pattern is inspected individually for these elements, as shown in the following. Patterns listed in *Dependencies* must be applied in order to be able to use the current inspected pattern. This step attaches these necessary patterns to the current one.

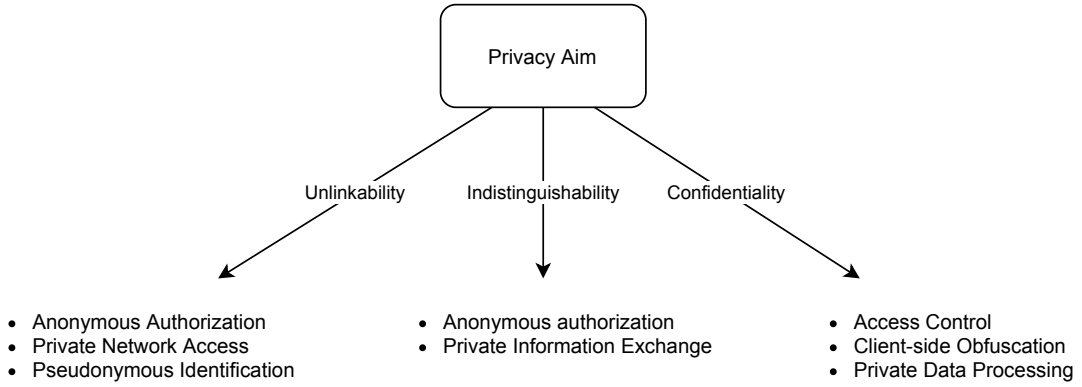

Figure 5.5: Elicitation of patterns with respect to privacy properties of *privacy aim*: Unlinkability, indistinguishability, and confidentiality.

The *Supplements* item proposes patterns whose application is useful in combination with the current pattern ,or which are often used together in practice. The supporting patterns, from all patterns that have been elicited so far, are combined in a separate set, called *Patterns in Consideration*.

Patterns in *Exclusions* represent conflicts and have to be resolved by the architect. At first, the patterns elicited so far are examined for mutual exclusion. Resulting conflicts are listed in the set *Conflicts* and assigned to *high priority*. Subsequently, these patterns are compared with the *Patterns in Consideration* and resulting conflicts are listed in *Conflicts*, given priority *low*.

Unlike the others, this step outputs three different sets. The first set consists of the patterns elicited so far, appended with the respective patterns they depend on. *Conflicts* constitutes the second set and represents key-value pairs. Key is a pattern being elicited so far, and values containing conflicting patterns with accompanying priorities. The last set constitutes the *Patterns in Consideration*, built analogously to the previous key-value style but without attached priorities.

5.6 Architect-Driven Selection

In the last step, the architect chooses the appropriate patterns, depending on the context. The architect can decide about each patterns' variant, whether it should be eliminated, or highlighted for the next phase of the SDLC, the implementation. Two scenarios are identified and possible solutions are proposed. They are presented in the following.

Scenario 1: Non-Empty Pattern Set

In the first scenario, the final set of patterns available for selection, is not empty. The architect decides which patterns can be applied in the context of the application. In addition, collisions from the second and third sets have to be solved (identified in the previous step). For each conflict the architect weighs up and selects the most appropriate one(s). Or the architect find a compromise applying both patterns of a conflict, e.g. by fulfilling their privacy goals in a weakened way. The following criteria to decide on one pattern, or against, can be considered.

PPAPs are constructed to take into account different privacy properties. To get the maximum possible privacy protection, the architect should try to apply each PPAP available. In particular those that are complementary (expressed in *related patterns*) are recommended to use simultaneously.

For each pattern the architect decides if its applicable, e.g. considering the skill of the development team, time and cost efforts for the implementation, other (architectural) parts of the system, or trade-offs with different quality attributes.

Once the setting comprises clients, using the service in isolated form, the architect should decide whether it is possible to offer the system in an anonymous way. If this is the case, *Anonymous Authorization* should be preferred over *Pseudonymous Identity Management*, because it offers greater protection against linkability.

If the final set of patterns is selected, the architect decides on the variants for each one. Variants are selected, taking into account the discussion in a patterns' *consequences* element, or considering other implementation variants not present in the pattern (e.g. by consulting experts in the privacy domain). The selection of variants can be determined, e.g. by discussing the impacts on the quality attributes with different stakeholders.

If no suitable pattern or implementation is found, the following scenario takes place.

Scenario 2: No Remaining Pattern

If no patterns remain after the last step or if no realization is found for a pattern, the following considerations can be made.

Although Spiekermann and Cranor prefer the principle of privacy-by-architecture, they do not exclude the other case [SC08]. It may not be feasible to design the appropriate architecture, proactively implementing privacy requirements. In the lack of architectural solutions, the architect can consider privacy-by-policy measures, e.g. using other privacy patterns. They are less abstract, but still found in the design phase, and thus follow the principle of privacy by design [EU16]. Research, concerning privacy-by-policy patterns, is discussed in Chapter 3. One example is *informing* (privacy design strategy [Hoe14]) the client, by use of privacy policy notices [CRC05; GWGT10], e.g. via collecting the consent of the users about the processing of their data.

Considerations for each Scenario

Independent of the scenario, the architect can take the following considerations into account.

The pattern catalog is incomplete, implying there can always be solutions which are not yet examined in the current pattern catalog, although they are more suitable for the context. Using the architect's experience and expertise, the architect should try to find solutions, not presented in the current pattern catalog.

The methodology presented in this chapter follows an iterative approach. If no satisfactory solution of patterns and variants has been found, the process can be re-run, and executed with a different setting, or other privacy properties. In addition, emerging conflicts can be solved differently in the new run. The architect is encouraged to follow this idea, before choosing the next best solution.

New runs can be executed for different parts (data) in the system. Patterns that would otherwise be mutually exclusive, when applied in the same context, can be used simultaneously.

This chapter presented a methodology for selecting PPAPs. It is constructed to achieve the objectives *reproducibility*, *effectiveness*, low *complexity* in use, and *transferability* to other pattern domains. Taking into account privacy properties and the situational context, it supports the architect in finding the suitable patterns for the given scope of the system. In the next chapter, this method is applied in a use case. Chapter 7 discusses the approach of using PPAPs, and the methodology, examining the fulfillment of the objective set.

6 Use Case

This chapter presents a use case, demonstrating the applicability of the PPAPs and the selection methodology of Chapters 4 and 5, respectively. It consists of a company's plan to analyze emojis and provide emoji predictions in a privacy-friendly manner in the scope of smartphone keyboard applications. This thesis considers two companies with different privacy requirements, attempting to realize the use case with the help of the methodology.

This chapter first describes the context and motivation that emerges. Subsequently, the methodology is performed for both companies. It selects the PPAPs and their variations, for implementing the privacy requirements in the software architecture, respectively.

6.1 Context

In everyday life, the keyboard of a smartphone is constantly in use, e.g. when chatting with friends, writing documents or surfing the web browser. Keyboards on Android devices are apps, and can therefore be programmed by anyone and made available in the Google Play Store. When acquiring a smartphone, at least one keyboard app is preinstalled. In case of Android devices, this can be Google's own developed keyboard *Gboard*, or a manufacturer-specific one, for instance the *Samsung Keyboard*.

Besides Google's own keyboard app, there are plenty of others for Android phones offered by third-party providers. A distinction can be made between proprietary and open source applications. Examples of proprietary keyboards are Gboard[1] (Google), or Microsoft SwiftKey Keyboard[2]. Open Source alternatives are AnySoftKeyboard[3] and Simple Keyboard[4].

In the post-T9 era of Android, keyboard input is supplemented by word predictions and completions, allowing the user to work faster with a predefined selection of words. How keyboard providers perform analysis and make predictions varies. A privacy risk arises when using proprietary keyboards, where it is not ensured what happens to the persons' keyboard inputs. The keyboards can secretly record every keystroke and misuse the sensitive data. Regardless of the app's license (whether proprietary or open source), privacy issues arise if the cloud is connected to process personal data for input analysis.

[1] https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin
[2] https://play.google.com/store/apps/details?id=com.touchtype.swiftkey
[3] https://play.google.com/store/apps/details?id=com.menny.android.anysoftkeyboard
[4] https://play.google.com/store/apps/details?id=rkr.simplekeyboard.inputmethod

Gboard for example provides next word prediction, auto-correction, word completion and animation suggestion. Google presents the *federated learning* approach used in *Gboard* for predicting words [HRM+18]. They claim to protect the privacy of its clients by using the global model to train their inputs locally and only sharing model updates with the server. But security experts advise caution: *Federated learning* approaches are vulnerable to specific kinds of attacks, e.g. model poisoning [BCMC19].

6.2 Motivation

The use case is motivated by the desire to provide a privacy-friendly software system for analyzing and predicting emojis in a keyboard application.

Let *Company A* and *Company B* be two companies trying to deploy such a software system in the cloud. Both companies share the business goals of analyzing and predicting input data. Since text input is inherently sensitive, they decide to limit the analysis and prediction to emojis. The companies are located in various countries of the EU and are aimed at citizens who speak the local language. Their ambition is to meet as many privacy requirements as possible within the scope of their business objectives and expertise.

Being based in EU countries, the companies need to follow the terms of the GDPR to make their respective systems privacy-friendly. A software architecture is built, meeting the business goals, and the requirements of the GDPR. The architect therefore decides to use PPAPs, and the methodology presented in Chapter 5. For the development on the client side, both companies choose the open source application *AnySoftKeyboard*, on which the application is built. *AnySoftKeyboard* has local word predictions (on bigrams) implemented. Since it is open source software and has no cloud connection, it is assumed to be privacy-friendly, making it suitable for the enterprise.

6.3 Applying the Methodology

For selecting the PPAPs appropriate for their application, the companies' architects apply the methodology presented in Chapter 5. Each elicitation step and the final pattern selection are examined in tandem for each company below. The process is illustrated in Figure 6.1.

6.3.1 Setting-Driven Elicitation

In the first step, the setting is examined, potentially excluding variants or entire PPAPs: In this use case, clients are not directly connected by interaction, but they do not work completely isolated from each other. The offered services collect the data of individual clients to train it in their machine learning models, respectively.

In both companies there is no TPP involved. While the architect of Company A decides to exclude the participation of a TPP, the other company may consider one.

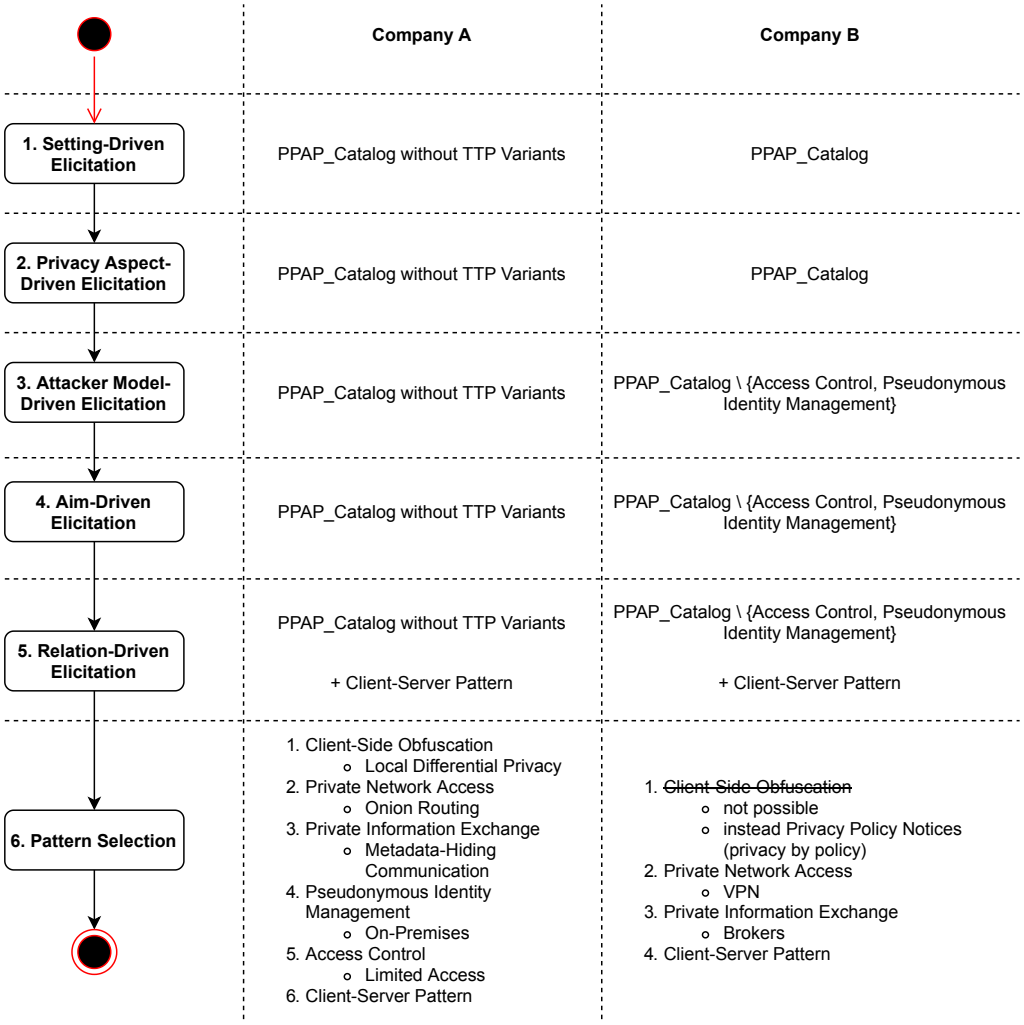

Figure 6.1: The PPAP selection process for both companies: The left column depicts the individual steps and the other two columns display the selection of patterns (and variants in the last step) for the respective step and the respective company.

Therefore, all patterns and variants regarding the client setting are preserved in both companies. Taking into account the involvement of third parties, variants requiring the presence of TTP are eliminated for Company A. Apart from these variants, all patterns are passed to the next step unchanged for both companies.

6.3.2 Privacy Aspect-Driven Elicitation

In this step, the decision about the *privacy aspect* is made. Both companies consider all three options to be relevant in the use case setting: The *content* of data and the *identity* of the respective client should be protected; the examination of user *behavior* should be prevented.

All patterns are passed to the next step for both companies.

6.3.3 Attacker Model-Driven Elicitation

Next, the architects examine the *attacker model*. The application of Company B does not involve any interaction, leading the architect to consider the *server* as a source of threat only. While clients could compromise the effectiveness of the training model by providing false information, they are assumed to be unable to gain sensitive data.

Company A's application provides extra features and a forum for their clients, requiring a login. Since it includes interaction, the architect decides to consider both, clients and the server, as threats.

Therefore, the following patterns are sorted out for Company B: *Access Control* and *Pseudonymous Identification*. Consequently, 5 patterns are passed on. In case of Company A, all patterns are passed on.

6.3.4 Aim-Driven Elicitation

In the next step, the architects decide to consider all *privacy aims*: The *content* of data should be sufficiently protected against unauthorized access (confidentiality), the usage should not be linkable to the identity of the respective client (*unlinkability*), and it should not be possible to distinguish between individual data (*indistinguishability*).

Accordingly, the pattern selection remains unchanged.

6.3.5 Relation-Driven Elicitation

The penultimate step examines the *related patterns* element of a PPAP: Dependencies, supplements, and occurring conflicts (exclusions) are automatically generated and passed on to the last step. The dependency that all PPAPs share, is the *Client Server Pattern*.

The following scenario arises for Company A: The first set that is passed to the last step consists of all patterns of the catalog (since no pattern has been excluded so far) and the *Client Server Pattern*. Since there are no exclusions in the current PPAP catalog, set 2 is empty. Set 3 consists of all pattern recommendations.

Company B's input from the previous step contains 5 patterns. Set 1 consists of all patters elicited so far and the *Client Server Pattern*. Set 2 is empty. In *Anonymous Authorization*, the pattern *Pseudonymous Identity Management* is considered as a possible supplement, but it has already been excluded. All other supplements are added to set 3.

Pattern Selection

In the last step, the architect selects the suitable patterns and their implementation.

Company A makes the following selection: It excludes *Anonymous Authorization*, since the application is intended to be made available for each user and does not require an authorization mechanism. The architect is satisfied with the remaining patterns and makes the following implementation

decisions: The company chooses the variant *Local Differential Privacy* for the *Client-Side Obfuscation* pattern. To protect the clients' IP addresses, the architects decides to implement the *Private Network Access* via the *Onion Routing* variant. The *Metadata-Hiding Communication* implementation is chosen for the *Private Information Exchange* pattern to avoid client user profiling. The corresponding recommendation to use client-side obfuscation in addition, is already in place. Since no TTP is included, the architect decides to implement *Pseudonymous Identity Management* in the form of *Pseudonymous Registration On-Premises*. The implementation of *Access Control* is *Limited Access*. The *Client Server Pattern* is listed as a dependency in all PPAPs and is applied accordingly.

Company B decides to remove the *Anonymous Authorization* pattern, since there is no login needed. The architect is satisfied with the remaining four patterns and makes the following implementation decisions: After consulting the stakeholders, the architect concludes that the skill level of the development team is not sufficient for a *Local Differential Privacy* implementation. The other implementation variants are not practical for their machine learning model. If a PPAP is not found or a found one impractical, scenario 2 of the current step recommends applying privacy-by-policy measures. Therefore, the architect decides to apply *Privacy Policy Notices*, informing the clients about the disclosure of their personal data [CRC05]. Protecting the clients' IP addresses, Company B implements the *Private Network Access* pattern via the *VPN* variant. For *Private Information Exchange*, the architect decides to implement the *Broker* variant, since the VPN server can take over its part. The *Client Server Pattern* is listed as a dependency in all PPAPs and is applied accordingly.

For selecting PPAPs in a given context, this chapter applied the methodology presented in this thesis. A use case was defined, analyzing and predicting emojis on keyboard inputs of Android smartphones. For two companies that want to deploy the same application but have different privacy requirements, the methodology resulted in different sets of architectural patterns to support the design of an privacy-friendly software system. Chapter 7 discusses the use case against the set objectives of the methodology.

7 Discussion

The new class of patterns, *PPAPs*, introduced in Chapter 4, and the methodology presented in Chapter 5 are examined below. First, this thesis assesses the objectives defined for the methodology. Limitations of PPAPs, and the methodology are discovered thereafter.

7.1 Objectives

Chapter 5 determines the objectives for the methodology: *Reproducibility*, *effectiveness*, low *complexity* in use, and *transferability* to other pattern domains. They are discussed in the following.

Reproducibility

The methodology is a systematic approach that selects patterns according to the architect's preferences. If the pattern catalog remains unchanged, and the architect selects the same preferences for the privacy properties, the methodology is reproducible up to the last step. In the last step, the architect selects the appropriate patterns according to the specific context and expertise. If these have not changed, the same result is produced.

In the actual implementation variant the result may change, e.g. when a new PET evolves. The *implementation and variants* element provides support for the implementation phase and does not affect the abstract view of the pattern.

Effectiveness

Considering the use case, Company B cannot implement a pattern (*Client-Side Obfuscation*) due to the missing skill level (*Local Differential Privacy*) and due to the application context (remaining variants). Since the approach of proactively implementing privacy requirements in the system cannot be followed, the weaker variant of privacy-by-policy is implemented (informing the clients). In both cases, the companies' architects decide not to use the *Anonymous Authorization* pattern, because they assume it to be irrelevant in their scope.

Therefore, it is not always possible for an architect to realize a pattern proposed for the last step. This thesis defines and identifies PPAPs to cover as many combinations of the presented privacy properties as possible. It is possible that the selection process does not output PPAPs. Either potential patterns have not been identified (yet) or the elicited ones are not applicable in the considered system context. If no architectural solution is found, the architect considers other measures, e.g. patterns following the privacy-by-policy principle.

The final implementation of the patterns, selected semi-automatically before the last step, in the system depends on the architect's decisions. Therefore, the selection process is dependent on the expertise of the architect.

Whether the selection process covers all patterns that are needed, is difficult to answer. PPAPs are designed to include different privacy properties that each pattern satisfies in different ways. Therefore, if the architect makes the appropriate settings, the patterns with the corresponding privacy requirements are available for the final selection of the architect. In addition, the architect can re-run the selection process with different preference to get more patterns for selection, if necessary.

Low Complexity

Executing the first three steps is straightforward, and represents a semi-automated process. In the last step, the architect has to decide which patterns to adopt. Since the architect's expertise is required, automating the entire process is not possible. A need for expertise is not considered problematic, because the architect is present during the design phase of the SDLC.

Transferability

In Section 5.2, the functional principle of the selection process is presented. The methodology is transferable to patterns found in other fields, e.g. in the security domain. It selects patterns, considering properties that are embedded in the patterns, and involves the architect of the relevant software system. Two possible ways of transferring the methodology to the corresponding domain are explained.

The first approach is to adapt the patterns of the new domain, by extending the patterns with specific properties. If the domain corresponds to a quality attribute, for example security, patterns can be extended with properties used to fulfill this particular quality attribute. In this example, the common classification into confidentiality, integrity and availability can be used. Another option could be to adopt the categories extracted by Bass et al.: *Detect attacks*, *resist attacks*, *react to attacks*, and *recover from attacks* [BCK03].

An alternative approach is to proceed in reverse order: The methodology is adapted according to the already existing patterns. Properties from the patterns are extracted and integrated into the selection process of the methodology. If the patterns include relations, they can be applied as it is done in the second last step of the methodology presented in thesis.

The degree of application context (here e.g. third parties), that can be included, must be examined individually for each pattern domain.

7.2 Limitations

This thesis highlights and discusses problems encountered with the use of PPAPs below. They are taken into consideration for future work in Chapter 8.

Patterns Inherent Incompleteness

Examining the objective *effectiveness*, an observed problem is the absence of potential PPAPs in the catalog. The current catalog, consisting of 7 identified patterns, is not complete: The size of the pattern catalog, the architectural patterns, and the architectural tactics do not guarantee completeness. Potential reasons are outlined in the following: Within the time and space constraints of this thesis, patterns are identified that conform to the definition given in Chapter 4. Shen and Pearson observe the difficulty to conduct a comprehensive study on PETs [SP11]. By analogy, patterns may not have been identified despite their existence.

Chapter 2 outlines the increasing importance and prioritization of privacy, potentially leading to an increase in relevant research areas. Existing technologies may become more practicable through improvements or may be deprecated since new state-of-the-art technologies emerge. Fernandez identified patterns, that are not yet deployable according to current time and technology, but may become so in the near future [Fer13] . The same is true for variants in PPAPs: Current technologies (in the form of architectural tactics/ PETs) may be modified or new ones added. The architectural tactic *Homomorphic Encryption* is currently only implementable to a certain degree. Assumptions are made, allowing the technology to operate with limited impact in certain contexts, for example by mitigating other quality attributes, such as performance and scalability. However, the lack of architectural tactics is not a limitation. They serve as an assistance for implementations, and for solving possible trade-offs. A PPAP represents a more abstract view of the solution and can be used more universally.

Another reason for incompleteness is a change in the existing structure of a pattern or of individual elements over time. New privacy properties can be introduced or existing properties can be redefined to cover the current state of the catalog more *effectively*. E.g., this thesis considers anonymity to be the property of hiding the client's identity and existence from the server. Anonymity within a service does not guarantee anonymity between different services. The use of *SSO* supports to achieve anonymity in the current system, but the identity provider (for example Facebook or Google) learns which other services are used, and thus gains more sensitive data. Modifying the pattern structure can help to address this issue.

The incompleteness of a catalog lies in the nature of patterns [DG13]. The incompleteness does not diminish the usefulness of the catalog. The SDLC is an iterative process, where designing the architecture is part of the design phase (see Chapter 2). Architectural patterns support the design of software system from a software architecture view. Other measures that are executed in the SDLC afterwards, addressing more specific parts of the system, can be applied in addition. An example is suggested in the methodology: Applying privacy patterns following the privacy-by-policy principle. There are other methods that implement privacy in the system in a different way and can be applied in addition, e.g. risk model approaches (see Chapter 3). These take place at the earliest when the architecture is in place. One example is the privacy threat methodology *LINDDUN* [WJ15].

Therefore, an incomplete PPAP catalog can contribute to the construction of privacy-friendly systems, e.g., as a starting point or complement to other methods.

Lack of Comparison With Other Patterns and Quality Attributes

The identified patterns address the quality attribute privacy. The fulfillment of the privacy quality attribute has positive or negative effects on those of others [BCK03]. Emerging trade-offs are made primarily at the level of architectural tactics. They arise in the suggested variants within the *Implementations and Variants* part of a PPAP and are discussed in the *Consequences* part. In the scope of this thesis, PPAPs outline these trade-offs, covering a range of a few other quality attributes: In most cases, these are usability and performance.

The variants are suggestions and vary. They are listed to the best of the authors' knowledge and serve as an initial proposal of a PPAP. The architect, however, has to reflect on them and to take into consideration alternative technologies to deliver the implementation, most suited to the system. Therefore, trade-off analysis of the superior PPAP is difficult to carry out: An architectural pattern is assumed to be too abstract to make detailed final statements about the impact on other quality attributes.

Due to the incompleteness of patterns, it is impossible to be able to solve all trade-off issues. The issues arise not only between quality attributes, but also within a quality attribute. Employing privacy patterns in the IoT domain, Pape and Rosenberg question whether the system better meets privacy requirements when data is being moved from the cloud to the clients [PR19]. This is synonymous with the question whether the achievement of privacy by moving parts of the cloud service to the end devices, could have a negative effect on securing the data. Confidentiality of personal data pertains to the quality attributes privacy (and security). There are arguments for both approaches and the architect ultimately decides on the solution and the corresponding pattern.

So far, the trade-off regarding quality attributes between PPAPs is discussed. Architectural patterns in general, can be related to PPAPs. The *Related Patterns* and *Consequences* elements examine exclusively PPAPs.

When applying architectural patterns, it is common practice to make trade-offs on the level of architectural tactics [BCK03]. The architectural patterns of this thesis are designed to mainly satisfy the quality attribute privacy, implications on other quality attributes are mainly expressed with different tactics. Applying architectural patterns from other domains, the architect must consider emerging trade-offs primarily at the level of architectural tactics (keeping in mind the incompleteness of architectural patterns and tactics, as discussed above).

The difficulty, in comparing PPAPs with other patterns in terms of quality attributes, lies in the nature of architectural patterns and their degree of abstraction. Therefore, it is not a specific limitation of PPAPs, but a property of using architectural patterns in general. An architect is responsible for weighing and balancing the different quality characteristics. Hence, comparing PPAPs with other architectural patterns, is beyond the scope of this thesis.

Limitations Related to the Abstraction of a PPAP

Finding the right abstraction is the key for identifying privacy patterns [RBE+12]. This thesis leverages the abstraction level of architectural patterns to implement privacy requirements in a software architecture.

In the definition of a PPAP, only those dimensions of Heurix et al.'s work are considered providing the necessary level of abstraction for building architectures [HZNF15]. Furthermore, the extent to which a dimension is adopted (e.g., excluding *deniability* for *aim*) is examined. If all dimensions would be adopted unchanged, instead of architectural patterns, only PETs or lower privacy design patterns would be identified in terms of abstraction. Regarding the trade-off between abstraction of PPAPs and the *effectiveness* of the methodology, this thesis encounters and discusses two limitations.

First, PPAPs can satisfy multiple privacy property values of a privacy property, e.g. *client* and *server* for the *attacker model* in the *Private Data Processing* pattern. As a consequence, this ambiguity makes the selected methodology less precise. The more ambiguities arise, the less granularity the methodology provides, and the greater the number of patterns are available in the last step of the methodology. In extreme case, the second last step outputs all patterns of the catalog and provides minor support for an architect.

There is always a trade-off, and thus the option of a PPAP meeting multiple property values. It is important to choose the abstraction level in such a way that good decisions about the selection of patterns can be made. In this thesis, the presented selection process makes the decisions on the selection of patterns, with the help of the present architect. The PPAP abstraction is chosen, being more abstract than PETs, and representing architectural elements. In addition, they are not as abstract as Colesky et al.'s tactics or Hoepman's privacy design strategies, which would make the methodology less practical [CHH16; Hoe14]. When more PPAPs are added to the catalog, the privacy properties may need to be readjusted, or new ones to be embedded into the definition of a PPAP to maintain the practicability of the methodology. One option is to refine the elicitation driven by the *attacker model* to distinguish malicious from honest but curious clients, when more patterns are added to the catalog. PETs are incorporated into PPAPs in form of architectural tactics, allowing the architect to make decisions on the abstraction level of PETs as well.

The second limitation arises from the non-uniform definitions and views of software architectures. PPAPs adhere to the definitions of a software architecture and an architectural pattern given in Chapter 4. For other architects, the definition may differ or be interpreted differently. E.g., the assumption, adopted by Martin Fowler, that a software architecture constitutes elements that are important in the given context [Fow03] has been attempted to be defined more precisely. In this thesis, privacy is from importance, and therefore it is included in the definition supported by legal background and related work. What is important to a certain architect may still differ, depending which definition and scope are chosen. As a result, PPAPs may be less useful for certain architects.

It is not a specific limitation of PPAPs, but the limitation of the ambiguous definitions and scopes of a software architecture. According to the transferability as described before, the patterns can be transferred to other domains. A different view of a software architecture can be considered as a different domain and, therefore, the patterns or the methodology can be adapted accordingly.

Lack of Relations

In a PPAP, relationships to other patterns are linked by the *related patterns* element. It allows additional patterns to be considered or excluded. Except for the *Client Server Pattern*, which is involved in every pattern, relations are scarce in the current state of the pattern catalog. Relations

are often used in the field of patterns and considered one of the key points of patterns, increasing their expressiveness [DGZ15; Haf13; SFH+13]. Potential reasons for the lack of relations in PPAPs are highlighted in the following.

The first limitation observes the incompleteness of the catalog. With 7 patterns included, the size of the current catalog may be too small to provide a sufficiently large basis for relations. This thesis assumes the amount of the relations increase, when new patterns are added or individual PPAPs are modified.

As discussed in the second limitation, this work is limited to architectural patterns in the privacy domain. The lack of consideration of other patterns that fulfill other quality attributes is assumed to lead to fewer options of architectural patterns that could be related.

The major cause is observed in the construction of a PPAP: They are defined and identified to satisfy various privacy properties in various ways. E.g., the *Privacy Aspect* property can be satisfied by content, identity, or behavior. Thus, each PPAP can be considered to operate in its own privacy domain. When the abstraction addressed in the previous limitation is reduced, more possibilities of relationships arise (but the *effectiveness* of the methodology decrease).

The problem with the lack of relationships (the simultaneous application or exclusion) of multiple patterns is partially solved by the methodology. Appropriate patterns are selected in the context of the system where the architect is present. These are the patterns constituting the relation with the context of an application: They are important and should be applied together. Patterns that do not meet the respective privacy properties are excluded. This corresponds to the attitude of relations of a pattern.

8 Conclusion and Outlook

This thesis presents an approach to design software architectures with privacy in mind, utilizing architectural patterns. Using the presented patterns provide a way to comply with the *privacy by design and default* principle (GDPR) to attack the problem of the recent increasing privacy concerns [EU16]. The contributions of this thesis consist of the identification of privacy-preserving architectural patterns (PPAPs) as well as a methodology for the selection of suitable (architectural) patterns in a given context of the software system.

In the following, the contributions are discussed with respect to their fulfillment of the research questions (RQs). This thesis concludes with open questions, which can be the starting point for future work.

8.1 Contributions to Research Questions

Chapter 1 defines two RQs. Their fulfillment by the presented contributions is assessed in the following. This thesis represents one way to address these questions. Other work is being carried out with similar objectives. Some of these are presented and discussed in Chapter 3.

RQ 1: Which Architectural Patterns Exist That Implement Privacy Requirements in a Software Architecture?

By identifying PPAPs, this thesis proposes architectural patterns allowing the design of a privacy-friendly software architecture. Compared to most privacy patterns found in the literature, PPAPs share the abstraction level of an architectural pattern to enable the view of a software architecture. The privacy quality attribute is introduced to identify PPAPs. The identified patterns are brought into the structure defined in Chapter 4 to implement privacy properties and to represent an architecture class. A PPAP forms the architecture class of a *component-and-connector (C&C) structure*, describing its elements and their interaction [BCK03].

Thus, architectural patterns exist, implementing privacy requirements in a software architecture: PPAPs. This thesis identifies 7 PPAPs.

The discussion in Chapter 7 reveals that PPAPs may not represent the complete architecture. The designed architecture can be leveraged as a starting point for further methods, taking place in the SDLC afterwards, e.g. privacy threat modeling in the form of LINDDUN [WJ15].

RQ 2: How Can These Patterns Be Selected in a Given Application Context?

This thesis presents a methodology for selecting patterns in the context of the application. One of the goals of this methodology is the transferability to other pattern domains, in order to make it more universally applicable.

PPAPs are designed to incorporate privacy features, leveraged in the selection process in Chapter 5. The selection includes the scope, e.g. involvement of third parties. Another part of the selection is the evaluation of relations between patterns. After these selection steps, the architect has the patterns available that can be used. In a final step, the architect decides which patterns to adopt for designing the software architecture.

The answer to this research question is, that a methodology for selecting patterns can be constructed, by including quality attribute related properties of a pattern and situational environments (the scope of the software system) for the selection process.

8.2 Open Questions and Future Work

The discussion in Chapter 7 discovers open questions, not being addressed by this thesis. They may be the starting point for future work, and are presented below.

Universal PPAPs

Data processed in the cloud is expected to account for 50 percent of the world's data (see Chapter 1). Therefore, this thesis focuses on identifying PPAPs, representing software architectures in the cloud. The question is whether the patterns are applicable to other privacy domains or are too specific to the cloud context.

Relevance for New SDLC Models

This thesis considers the general SDLC model, where the design of the software architecture takes place in the second step: The design [Rup10]. In contrast to traditional models, such as the *Waterfall model*, the agile development approach does not follow a sequential approach, allowing the overlapping of individual SDLC steps. Therefore, the use of PPAPs for designing software architectures in such models may be complicated.

As discussed before, PPAPs represent parts of the architecture, not the whole system, and can be used in conjunction with other methods (e.g., *LINDDUN* [WJ15]). This thesis assumes that PPAPs can be applied in other SDLC models. Future work can be done, investigating the relevance of PPAPs in multiple SDLC models and validating this assumption.

Extension of the Catalog and Patterns

Chapter 7 discusses the incompleteness of the catalog of PPAPs and PPAPs themselves.

Future work can be done to extend the catalog, implying the identification of new PPAPs and the adaption to the methodology. Furthermore, the privacy properties can be studied and new ones developed; or the existing ones can be modified. By adapting the privacy properties, the methodology can maintain its *effectiveness*, despite the increasing pattern catalog.

Impact on other Architectural Patterns

Besides the *Client Server Pattern*, this thesis does not consider other architectural patterns. The discussion in Chapter 7 reveals, that the application of architectural patterns depends on the emerging trade-offs between quality attributes. The impact on other architectural patterns, with respect to other quality attributes, can be investigated in future work.

Composition

This thesis incorporates privacy properties into architectural patterns. The privacy taxonomy for PETs is taken and adapted to map the abstraction level of architectures [HZNF15]. Funke et al. extend the work of Heurix et al., comparing the enforced privacy of PETs [FWD17]. They propose a mathematical approach that can be used to measure the enforced privacy. This idea can be adapted to PPAPs, potentially enhancing the relation between patterns.

Bibliography

[Abe07] M. Aberdour. “Achieving quality in open-source software”. In: *IEEE software* 24.1 (2007), pp. 58–64 (cit. on p. 68).

[AFT06] A. Acquisti, A. Friedman, R. Telang. “Is there a cost to privacy breaches? An event study”. In: *ICIS 2006 Proceedings* (2006), p. 94 (cit. on p. 17).

[AGM+15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash. “Internet of things: A survey on enabling technologies, protocols, and applications”. In: *IEEE communications surveys & tutorials* 17.4 (2015), pp. 2347–2376 (cit. on p. 13).

[Ale77] C. Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 1977 (cit. on p. 23).

[And12] D. Anderson. “Splinternet behind the great firewall of china”. In: *Queue* 10.11 (2012), pp. 40–49 (cit. on p. 65).

[APT13] R. Arora, A. Parashar, C. C. I. Transforming. “Secure user data in cloud computing using encryption algorithms”. In: *International journal of engineering research and applications* 3.4 (2013), pp. 1922–1926 (cit. on p. 19).

[ATWW15] G. W. Ang, D. J. Townsend, J. H. Woelfel, T. P. Woloszyn. *System and method for tokenization of data for storage in a cloud*. US Patent 9,021,135. Apr. 2015 (cit. on p. 56).

[BC11] F. Bélanger, R. E. Crossler. “Privacy in the digital age: a review of information privacy research in information systems”. In: *MIS quarterly* (2011), pp. 1017–1041 (cit. on p. 17).

[BCK03] L. Bass, P. Clements, R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003 (cit. on pp. 3, 4, 14, 22–26, 29, 30, 33, 34, 38, 42, 43, 49, 102, 104, 107).

[BCMC19] A. N. Bhagoji, S. Chakraborty, P. Mittal, S. Calo. “Analyzing federated learning through an adversarial lens”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 634–643 (cit. on p. 96).

[BH13] K. B. Bomar, G. E. Harper. *Tokenized data security*. US Patent 8,595,812. Nov. 2013 (cit. on p. 58).

[BJKL04] S. Bellman, E. J. Johnson, S. J. Kobrin, G. L. Lohse. “International differences in information privacy concerns: A global survey of consumers”. In: *The Information Society* 20.5 (2004), pp. 313–324 (cit. on p. 17).

[BL02] O. Berthold, H. Langos. “Dummy traffic against long term intersection attacks”. In: *International Workshop on Privacy Enhancing Technologies*. Springer. 2002, pp. 110–128 (cit. on p. 70).

[BR01] J. J. Borking, C. Raab. “Laws, PETs and other technologies for privacy protection”. In: *Journal of Information, Law and Technology* 1 (2001), pp. 1–14 (cit. on p. 19).

[BRD+15] B. T. Bailey, J. Romer, C. Doyle, J. Gifford, K. Zibart. *Tokenizing sensitive data*. US Patent 8,943,574. Jan. 2015 (cit. on p. 19).

[BW90] L. Brandeis, S. Warren. “The right to privacy”. In: *Harvard law review* 4.5 (1890), pp. 193–220 (cit. on p. 17).

[Cav+09] A. Cavoukian et al. “Privacy by design: The 7 foundational principles”. In: *Information and privacy commissioner of Ontario, Canada* 5 (2009) (cit. on pp. 13, 18, 43).

[CC18] M. Colesky, J. C. Caiza. “A System of Privacy Patterns for Informing Users: Creating a Pattern System”. In: *Proceedings of the 23rd European Conference on Pattern Languages of Programs*. 2018, pp. 1–11 (cit. on p. 37).

[CCD+18] M. Colesky, J. C. Caiza, J. M. Del Alamo, J.-H. Hoepman, Y.-S. Martín. “A system of privacy patterns for user control”. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018, pp. 1150–1156 (cit. on p. 37).

[CG18] C. Cadwalladr, E. Graham-Harrison. “Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach”. In: *The guardian* 17 (2018), p. 22 (cit. on pp. 3, 4).

[CHH16] M. Colesky, J.-H. Hoepman, C. Hillen. “A critical analysis of privacy design strategies”. In: *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2016, pp. 33–40 (cit. on pp. 33, 34, 50, 105).

[CJK+18] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, T. Wang. “Privacy at scale: Local differential privacy in practice”. In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 1655–1658 (cit. on p. 56).

[CRC05] G. E. Clayton, K. I. Robertson, H. T. Carneal. *Policy notice method and system*. US Patent 6,904,417. June 2005 (cit. on pp. 28, 43, 93, 99).

[CV02] J. Camenisch, E. Van Herreweghen. “Design and implementation of the idemix anonymous credential system”. In: *Proceedings of the 9th ACM conference on Computer and communications security*. 2002, pp. 21–30 (cit. on p. 79).

[De 02] J. De Clercq. “Single sign-on architectures”. In: *International Conference on Infrastructure Security*. Springer. 2002, pp. 40–58 (cit. on p. 79).

[DG13] N. Doty, M. Gupta. “Privacy Design Patterns and Anti-Patterns Patterns Misapplied and Unintended Consequences”. In: *Trustbusters Workshop at the Symposium onUsable Privacy and Security* (2013) (cit. on pp. 36, 39, 103).

[DGZ15] N. Doty, M. Gupta, J. Zych. *privacypatterns. org-Privacy Patterns.(2015)*. 2015. URL: https://privacypatterns.org/ (cit. on pp. 14, 36, 37, 39, 44, 48, 65, 106).

[DM09] R. Dingledine, S. J. Murdoch. “Performance Improvements on Tor or, Why Tor is slow and what we’re going to do about it”. In: *Online: http://www. torproject. org/press/presskit/2009-03-11-performance. pdf* (2009) (cit. on p. 68).

[Dom07] J. Domingo-Ferrer. “A three-dimensional conceptual framework for database privacy”. In: *Workshop on Secure Data Management*. Springer. 2007, pp. 193–202 (cit. on p. 30).

[Dwo08] C. Dwork. “Differential privacy: A survey of results”. In: *International conference on theory and applications of models of computation*. Springer. 2008, pp. 1–19 (cit. on pp. 19, 56).

[ECM20] S. Eskandarian, M. Christodorescu, P. Mohassel. “Privacy-Preserving Payment Splitting”. In: *Proceedings on Privacy Enhancing Technologies* 2020.2 (2020), pp. 67–88 (cit. on p. 70).

[EU16] R. 2. (EU). *Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC. European Parliament and Council*. Luxembourg: Office for Official Publications of the European Communities. Apr. 2016 (cit. on pp. 3, 18, 19, 45, 50, 54, 93, 107).

[Fer13] E. Fernandez-Buglioni. *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013 (cit. on p. 103).

[Fow03] M. Fowler. “Who needs an architect?” In: *IEEE SOFTWARE* 20.5 (2003), pp. 11–13 (cit. on pp. 22, 42, 105).

[FWD17] S. Funke, A. Wiesmaier, J. Daubert. “Constrained PET Composition for Measuring Enforced Privacy”. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. 2017, pp. 1–10 (cit. on p. 109).

[Gen10] C. Gentry. “Computing arbitrary functions of encrypted data”. In: *Communications of the ACM* 53.3 (2010), pp. 97–105 (cit. on p. 61).

[GHJV93] E. Gamma, R. Helm, R. Johnson, J. Vlissides. “Design patterns: Abstraction and reuse of object-oriented design”. In: *European Conference on Object-Oriented Programming*. Springer. 1993, pp. 406–431 (cit. on pp. 23, 24).

[GRS99] D. Goldschlag, M. Reed, P. Syverson. “Onion routing”. In: *Communications of the ACM* 42.2 (1999), pp. 39–41 (cit. on pp. 19, 65).

[GWGT10] C. Graf, P. Wolkerstorfer, A. Geven, M. Tscheligi. “A pattern collection for privacy enhancing technology”. In: *The 2nd Int. Conf. on Pervasive Patterns and Applications (PATTERNS 2010)*. 2010, pp. 21–26 (cit. on pp. 36, 39, 43, 93).

[Haf06] M. Hafiz. “A collection of privacy design patterns”. In: *Proceedings of the 2006 conference on Pattern languages of programs*. 2006, pp. 1–13 (cit. on pp. 35, 39, 53).

[Haf13] M. Hafiz. “A pattern language for developing privacy enhancing technologies”. In: *Software: Practice and Experience* 43.7 (2013), pp. 769–787 (cit. on pp. 35, 36, 39, 48, 55, 106).

[HH10] M. H. Hugos, D. Hulitzky. *Business in the cloud: what every business needs to know about cloud computing*. John Wiley & Sons, 2010 (cit. on p. 13).

[Hoe14] J.-H. Hoepman. “Privacy design strategies”. In: *IFIP International Information Security Conference*. Springer. 2014, pp. 446–459 (cit. on pp. 14, 30, 32, 50, 52, 64, 69, 74, 93, 105).

[HRM+18] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, D. Ramage. “Federated learning for mobile keyboard prediction”. In: *arXiv preprint arXiv:1811.03604* (2018) (cit. on p. 96).

[HV18] K. A. Houser, W. G. Voss. “GDPR: The end of Google and facebook or a new paradigm in data privacy”. In: *Rich. JL & Tech.* 25 (2018), p. 1 (cit. on p. 13).

[HZNF15] J. Heurix, P. Zimmermann, T. Neubauer, S. Fenz. “A taxonomy for privacy enhancing technologies”. In: *Computers & Security* 53 (2015), pp. 1–17 (cit. on pp. 19, 28, 29, 38, 43–47, 53, 55, 86, 105, 109).

[KBL17] H. Khazaei, H. Bannazadeh, A. Leon-Garcia. “Savi-iot: A self-managing containerized iot platform”. In: *2017 IEEE 5th international conference on future Internet of Things and Cloud (FiCloud)*. IEEE. 2017, pp. 227–234 (cit. on p. 37).

[KM09] M. Kumpošt, V. Matyáš. “User profiling and re-identification: case of university-wide network analysis”. In: *International Conference on Trust, Privacy and Security in Digital Business*. Springer. 2009, pp. 1–10 (cit. on p. 64).

[LGL+15] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, L. Sun. “Fog computing: Focusing on mobile users at the edge”. In: *arXiv preprint arXiv:1502.01815* (2015) (cit. on p. 21).

[MBG+08] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, D. Sicker. “Shining light in dark places: Understanding the Tor network”. In: *International symposium on privacy enhancing technologies symposium*. Springer. 2008, pp. 63–76 (cit. on p. 65).

[MC04] G. R. Milne, M. J. Culnan. “Strategies for reducing online privacy risks: Why consumers read (or don’t read) online privacy notices”. In: *Journal of interactive marketing* 18.3 (2004), pp. 15–29 (cit. on pp. 28, 43).

[MCG06] G. R. Milne, M. J. Culnan, H. Greene. “A longitudinal assessment of online privacy notice readability”. In: *Journal of Public Policy & Marketing* 25.2 (2006), pp. 238–249 (cit. on pp. 28, 43).

[MFP07] D. Mellado, E. Fernández-Medina, M. Piattini. “A common criteria based security requirements engineering process for the development of secure information systems”. In: *Computer standards & interfaces* 29.2 (2007), pp. 244–253 (cit. on p. 30).

[MG+11] P. Mell, T. Grance, et al. “The NIST definition of cloud computing”. In: (2011) (cit. on p. 20).

[MOO+14] C. Moore, M. O’Neill, E. O’Sullivan, Y. Doröz, B. Sunar. “Practical homomorphic encryption: A survey”. In: *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2014, pp. 2792–2795 (cit. on p. 61).

[Moo08] A. D. Moore. “Defining privacy”. In: *Journal of Social Philosophy* 39.3 (2008), pp. 411–428 (cit. on pp. 3, 4, 13).

[MPS13] A. Martínez-Ballesté, P. A. Pérez-Martínez, A. Solanas. “The pursuit of citizens’ privacy: a privacy-aware smart city is possible”. In: *IEEE Communications Magazine* 51.6 (2013), pp. 136–141 (cit. on pp. 29, 53).

[NLV11] M. Naehrig, K. Lauter, V. Vaikuntanathan. “Can homomorphic encryption be practical?” In: *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. 2011, pp. 113–124 (cit. on p. 61).

[PB10] S. Pearson, A. Benameur. “A decision support system for design for privacy”. In: *IFIP PrimeLife International Summer School on Privacy and Identity Management for Life*. Springer. 2010, pp. 283–296 (cit. on pp. 35, 36, 39, 43, 44, 86).

[Pea09] S. Pearson. “Taking account of privacy when designing cloud computing services”. In: *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE. 2009, pp. 44–52 (cit. on pp. 34, 35, 86).

[PH10] A. Pfitzmann, M. Hansen. “A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management”. In: *URL: http://dud. inf. tu-dresden. de/literatur/Anon_Terminology_v0* 34 (Jan. 2010) (cit. on p. 32).

[PK01] A. Pfitzmann, M. Köhntopp. “Anonymity, unobservability, and pseudonymity—a proposal for terminology”. In: *Designing privacy enhancing technologies*. Springer. 2001, pp. 1–9 (cit. on pp. 19, 35, 45).

[PR19] S. Pape, K. Rannenberg. “Applying privacy patterns to the internet of things’(iot) architecture”. In: *Mobile Networks and Applications* 24.3 (2019), pp. 925–933 (cit. on pp. 38, 39, 104).

[PS10] S. Pearson, Y. Shen. “Context-aware privacy design pattern selection”. In: *International Conference on Trust, Privacy and Security in Digital Business*. Springer. 2010, pp. 69–80 (cit. on pp. 35, 36, 39, 43, 44, 86).

[PS11] P. A. Pérez-Martínez, A. Solanas. “W3-privacy: the three dimensions of user privacy in LBS”. In: *12th ACM Int’l. Symp. Mobile Ad Hoc Networking and Computing*. 2011 (cit. on p. 30).

[PV04] G. Persiano, I. Visconti. “An efficient and usable multi-show non-transferable anonymous credential system”. In: *International Conference on Financial Cryptography*. Springer. 2004, pp. 196–211 (cit. on p. 79).

[PZ11] C. Paquin, G. Zaverucha. “U-prove cryptographic specification v1. 1”. In: *Technical Report, Microsoft Corporation* (2011) (cit. on p. 79).

[RAH+06] S. Romanosky, A. Acquisti, J. Hong, L. F. Cranor, B. Friedman. “Privacy patterns for online interactions”. In: *Proceedings of the 2006 conference on Pattern languages of programs*. 2006, pp. 1–9 (cit. on pp. 34, 39, 43).

[RBE+12] J. van Rest, D. Boonstra, M. Everts, M. van Rijn, R. van Paassen. “Designing privacy-by-design”. In: *Annual Privacy Forum*. Springer. 2012, pp. 55–72 (cit. on pp. 17, 32, 53, 60, 104).

[RR98] M. K. Reiter, A. D. Rubin. “Crowds: Anonymity for web transactions”. In: *ACM transactions on information and system security (TISSEC)* 1.1 (1998), pp. 66–92 (cit. on p. 35).

[Rup10] N. B. Ruparelia. “Software development lifecycle models”. In: *ACM SIGSOFT Software Engineering Notes* 35.3 (2010), pp. 8–13 (cit. on pp. 21, 108).

[Ryd18] D. R.-.-J. G.-.-J. Rydning. “The digitization of the world from edge to core”. In: *Framingham: International Data Corporation* (2018) (cit. on p. 13).

[SAB15] M. Sabt, M. Achemlal, A. Bouabdallah. “Trusted execution environment: what it is, and what it is not”. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE. 2015, pp. 57–64 (cit. on p. 61).

[SC08] S. Spiekermann, L. F. Cranor. “Engineering privacy”. In: *IEEE Transactions on software engineering* 35.1 (2008), pp. 67–82 (cit. on pp. 28, 36, 38, 43, 47, 70, 93).

[Sch02] M. Schumacher. “Security Patterns and Security Standards.” In: *EuroPLoP*. Citeseer. 2002, pp. 289–300 (cit. on pp. 30, 34, 35, 37, 39, 53).

[SFH+13] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013 (cit. on pp. 31, 39, 44, 48, 60, 82, 106).

[Sil15] J. Siljee. “Privacy transparency patterns”. In: *Proceedings of the 20th european conference on pattern languages of programs*. 2015, pp. 1–11 (cit. on pp. 37, 39).

[SL98] H. A. Seid, A. Lespagnol. *Virtual private network*. US Patent 5,768,271. June 1998 (cit. on p. 65).

[Sol02] D. J. Solove. “Conceptualizing privacy”. In: *Calif. L. Rev.* 90 (2002), p. 1087 (cit. on p. 17).

[Sol05] D. J. Solove. “A taxonomy of privacy”. In: *U. Pa. L. Rev.* 154 (2005), p. 477 (cit. on pp. 17, 18).

[SP11] Y. Shen, S. Pearson. “Privacy enhancing technologies: A review”. In: *HP Laboratories* 2739 (2011), pp. 1–30 (cit. on p. 103).

[Spi12] S. Spiekermann. “The challenges of privacy by design”. In: *Communications of the ACM* 55.7 (2012), pp. 38–40 (cit. on p. 18).

[SS13] S. Singla, J. Singh. “Cloud data security using authentication and encryption technique”. In: *Global Journal of Computer Science and Technology* (2013) (cit. on p. 55).

[Uni17] C. M. University. *What is your definition of software architecture?* 2017. URL: https://resources.sei.cmu.edu/asset_files/FactSheet/2010_010_001_513810.pdf (cit. on p. 22).

[VE03] H. Venter, J. H. Eloff. “A taxonomy for information security technologies”. In: *Computers & Security* 22.4 (2003), pp. 299–307 (cit. on pp. 27, 38, 43).

[Wei91] M. Weiser. “The Computer for the 21 st Century”. In: *Scientific american* 265.3 (1991), pp. 94–105 (cit. on p. 13).

[WJ15] K. Wuyts, W. Joosen. “LINDDUN privacy threat modeling: a tutorial”. In: *CW Reports* (2015) (cit. on pp. 27, 103, 107, 108).

[WKM+14] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, E. Weippl. “Spoiled onions: Exposing malicious Tor exit relays”. In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2014, pp. 304–331 (cit. on p. 68).

[WMAH15] B. D. Weinberg, G. R. Milne, Y. G. Andonova, F. M. Hajjat. “Internet of Things: Convenience vs. privacy and secrecy”. In: *Business Horizons* 58.6 (2015), pp. 615–624 (cit. on p. 38).

[WYH+19] H. Washizaki, N. Yoshioka, A. Hazeyama, T. Kato, H. Kaiya, S. Ogata, T. Okubo, E. B. Fernandez. “Landscape of iot patterns”. In: *2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT)*. IEEE. 2019, pp. 57–60 (cit. on pp. 37, 39).

[XXL18] L. D. Xu, E. L. Xu, L. Li. “Industry 4.0: state of the art and future trends”. In: *International Journal of Production Research* 56.8 (2018), pp. 2941–2962 (cit. on p. 13).

[YB97] J. Yoder, J. Barcalow. “Architectural patterns for enabling application security”. In: *Proceedings of the 4th Conference on Patterns Language of Programming (PLoP’97)*. Vol. 2. Citeseer. 1997 (cit. on p. 30).

All links were last followed on January 15, 2021.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature