applied sciences

MDPI

Article

Towards a Collection of Security and Privacy Patterns

Manos Papoutsakis [1,2,*], Konstantinos Fysarakis [3], George Spanoudakis [2], Sotiris Ioannidis [1] and Konstantina Koloutsou [3]

[1] Institute of Computer Science, Foundation for Research and Technology, 700 13 Heraklion, Greece; sotiris@ics.forth.gr

[2] Department of Computer Science, City University of London, London EC1V 0HB, UK; g.e.spanoudakis@city.ac.uk

[3] Sphynx Technology Solutions AG, 6300 Zug, Switzerland; fysarakis@sphynx.ch (K.F.); koloutsou@sphynx.ch (K.K.)

* Correspondence: paputsak@ics.forth.gr or Emmanouil.Papoutsakis@city.ac.uk

Featured Application: Internet of Things and Industrial Internet of Things.

Abstract: Security and privacy (SP)-related challenges constitute a significant barrier to the wider adoption of Internet of Things (IoT)/Industrial IoT (IIoT) devices and the associated novel applications and services. In this context, patterns, which are constructs encoding re-usable solutions to common problems and building blocks to architectures, can be an asset in alleviating said barrier. More specifically, patterns can be used to encode dependencies between SP properties of individual smart objects and corresponding properties of orchestrations (compositions) involving them, facilitating the design of IoT solutions that are secure and privacy-aware by design. Motivated by the above, this work presents a survey and taxonomy of SP patterns towards the creation of a usable pattern collection. The aim is to enable decomposition of higher-level properties to more specific ones, matching them to relevant patterns, while also creating a comprehensive overview of security- and privacy-related properties and sub-properties that are of interest in IoT/IIoT environments. To this end, the identified patterns are organized using a hierarchical taxonomy that allows their classification based on provided property, context, and generality, while also showing the relationships between them. The two high-level properties, Security and Privacy, are decomposed to a first layer of lower-level sub-properties such as confidentiality and anonymity. The lower layers of the taxonomy, then, include implementation-level enablers. The coverage that these patterns offer in terms of the considered properties, data states (data in transit, at rest, and in process), and platform connectivity cases (within the same IoT platform and across different IoT platforms) is also highlighted. Furthermore, pointers to extensions of the pattern collection to include additional patterns and properties, including Dependability and Interoperability, are given. Finally, to showcase the use of the presented pattern collection, a practical application is detailed, involving the pattern-driven composition of IoT/IIoT orchestrations with SP property guarantees.

Keywords: pattern-based engineering; security patterns; privacy patterns; pattern taxonomy; Internet of Things (IoT); Industrial Internet of Things (IIoT); dependability; interoperability

check for updates

Citation: Papoutsakis, M.; Fysarakis, K.; Spanoudakis, G.; Ioannidis, S.; Koloutsou, K. Towards a Collection of Security and Privacy Patterns. *Appl. Sci.* **2021**, *11*, 1396. https://doi.org/10.3390/app11041396

Received: 22 December 2020
Accepted: 29 January 2021
Published: 4 February 2021

1. Introduction

Massive advancements in computing and communication technologies have allowed for the ubiquitous presence of smart devices that communicate in an effortless manner. These hyper-connected smart environments have allowed for the transition to what is typically referred to as the IoT era—and the IIoT era in the industrial domain—becoming a driver for innovation and promising significant improvements to all aspects of modern life. However, these new developments come with a number of challenges, stemming from the intricacies of the IoT environment such as its dynamicity, scalability, and heterogeneity, as well as the need for end-to-end security and privacy [1–4].

Motivated by the above, the EU H2020 SEMIoTICS (Smart End-to-end Massive IoT Interoperability, Connectivity and Security; https://www.semiotics-project.eu/ (accessed on 2 December 2020)) project is developing a pattern-driven framework, built upon existing IoT platforms, to enable and guarantee secure and dependable actuation and semi-autonomic behavior in IoT/IIoT applications [5]. Patterns are re-usable solutions to common problems and building blocks to architectures [6,7], the foundations of which were laid by the architect Christopher Alexander in his seminal work "The Timeless Way of Building" [8]. In other words, patterns describe a recurring problem that arises in a specific context and present a well-proven, generic solution for it. In the context of security and privacy, patterns can capture the knowledge of field experts as applied solutions to known security and privacy problems. In that way, patterns can be leveraged by researchers and developers with different levels of security/privacy expertise, facilitating the design of solutions that are secure and privacy-aware by design. In SEMIoTICS, patterns encode proven dependencies between the security, privacy, dependability, and interoperability (SPDI) properties of individual smart objects and the corresponding properties of orchestrations (compositions) involving them. The encoding of such dependencies enables (i) the verification that a smart object orchestration satisfies certain SPDI properties and (ii) the generation (and adaptation) of orchestrations in ways that are guaranteed to satisfy required SPDI properties.

There are a plethora of security and privacy patterns in the form of books and catalogues and as part of the academic literature, as is thoroughly presented in Section 2. In most of these cases, patterns are presented with various limitations, such as the number of patterns, the properties they cover, or the specific applications they are tailored to. Moreover, there is often no effort for any classification or organization in any way, which hinders the development of practical implementations that leverage said patterns. In this context, the work presented herein presents a number of key contributions, which include:

- Aggregation, adaptation, and specification of patterns based on a novel pattern specification language defined for that purpose (as detailed in our previous works [9–11]), focusing on the core properties of Security and Privacy, while also briefly covering additional properties related to Dependability and Interoperability. These patterns include adaptations of concepts identified by surveying the relevant literature as well as original patterns developed within the SEMIoTICS project. The patterns presented in this work offer a complete coverage of all security and privacy properties and their sub-properties, also covering all data states (data in transit, data at rest, and data in processing) and all cases of platform connectivity (i.e., patterns within an IoT as well as across IoT platforms).
- Creation of an SP Pattern Collection featuring classification of said patterns based on the individual properties they cover, being categorized using a hierarchical taxonomy (the most widely accepted approach to tackle this issue, as shown by Hafiz et al. [12]). The proposed approach for pattern organization allows classifying patterns based on provided property, context, and generality, while also facilitating pattern relationships' specification, the retrieval of patterns, and, consequently, the verification of the associated properties. When visualized, this results in tree-form graphs connecting the defined properties and associated patterns, as will be presented in the Sections that follow.
- Design and development of a pattern-driven approach for composing IoT orchestrations where security, privacy, dependability and interoperability (SPDI) properties are guaranteed, leveraging the presented Pattern Collection. The proposed approach is capable of defining and managing IoT applications based on patterns, allowing for design-time and run-time verification that an IoT system/orchestration satisfies certain SPDI properties, triggering adaptations (if needed) in ways that are guaranteed to satisfy said properties.

To present the above, the remainder of this paper is organized as follows: Section 2 provides an overview of related works; Sections 3 and 4 present the pattern collection, focusing on security and privacy patterns, respectively; Section 5 provides an overview of the identified patterns and pointers to potential collection extensions to support additional properties (i.e., dependability and interoperability) and also demonstrates how the presented pattern collection can be leveraged for the specification and adaptation of IoT/IIoT orchestrations with SPDI guarantees by translating them to a machine-processable form that allows for automated property reasoning; finally, Section 6 provides the concluding remarks and pointers to future work.

2. Related Works

There are a plethora of security patterns provided by researchers, developers, and domain experts; these are included in books, catalogues, and the academic literature. Some notable works are highlighted in this section.

Steel et al. [13] present a catalog of 23 security patterns. However, they only focus on Java 2 Platform Enterprise Edition (J2EE) applications, Web services, and identity management, taking a hands-on, developer-centric approach to patterns' specification. Schumacher, M. et al. [14] in their security pattern book include 46 patterns, covering areas such as enterprise security and risk management, identification and authentication, access control, accounting, firewall architecture, and secure internet applications. Nevertheless, their focus is on system security, without considering additional areas and properties. The Security Patterns Repository Version 1.0 [15] consists of 26 patterns and three mini-patterns focusing exclusively on the domain of web application security. The presented patterns are organized into two broad categories, namely structural and procedural patterns. A book published by Microsoft's Patterns and Practices group [16] includes 18 security-related patterns, focusing once again on web application security. This book adopts an implementation-focused approach and does not take into consideration the relationships among the different patterns.

A recent literature study on privacy patterns research [17] identified 148 privacy patterns in total, recognizing that privacy patterns are a relatively young field compared to others. Some more recent works continue to expand the available privacy patterns (e.g., the work of Gabel et al. [18] on pseudonymity patterns). Other indicative resources for the retrieval of privacy-related patterns include works from Chung et al. [19], Hafiz [20,21], Drozd [22], Schümmer [23], Schumacher [24], and Graf et al. [25].

Authors in [19] developed an initial pattern language focusing on the ubiquitous computing domain, accompanied by 45 pre-patterns describing application genres, physical-virtual spaces, interaction, and system techniques for managing privacy and techniques for fluid interactions. Even though the evaluation of their pre-patterns showed statistically significant differences with respect to usefulness and time for task completion, those differences were between expert and novice designers rather than between pairs that used patterns and those that did not. Hafiz et al. in [20] present four design patterns that can potentially assist designers of privacy protecting systems to decide which privacy solutions they are going to apply. The presented patterns, while limited in number and property coverage, can be used for the design of anonymity systems for online communication, data sharing, location monitoring, voting, and electronic cash management. In [21], the first pattern language for developing Privacy Enhancing Technologies (PETs) is described. This language is used to define 12 patterns, focusing on assisting developers during the design of PETs. Drozd [22] presents an interactive online privacy pattern catalogue, aiming to address the gap of a comprehensive privacy pattern catalogue that could be used during the software development process. The classification used is according to the description of the privacy principles within the international standard ISO/IEC 29100:2011 [26]. Schümmer [23] introduces six patterns referring to a very specific issue: information received from and transmitted to other users, pointing out the need for controlling the provided information. This narrow scope leads to the specification of a relatively limited number of patterns. Schumacher [24] describes how security standards such as the Common Criteria can help the creators of patterns, providing indicative patterns on privacy aspects. A proof of concept is provided, while the overall process is described as "an ongoing experiment".

Authors in [25] present a method for creating User Interfaces Patterns for PETs. However, they provide a very short overview of the whole pattern collection and present only two of the patterns in detail.

In addition to the above literature resources, a number of online repositories can be found focusing on aggregating and providing patterns, such as the one from Arcticura (https://patterns.arcitura.com/ (accessed on 2 December 2020)), encompassing various ICT aspects including security and other privacy-specific repositories (https://privacypatterns.eu; https://privacypatterns.org/; http://privacypatterns.wu.ac.at:8080/catalog/ (accessed on 2 December 2020)).

3. Security Patterns

Security is typically broken down into the core properties of Confidentiality, Integrity, and Availability [27]. Additional supporting security properties are identified in the literature [28] such as Non-reputation, Auditability, Accountability, and Authenticity. The Security properties covered in this work and their relationships are depicted in Figure 1, creating a hierarchical taxonomy. As can be seen in said figure, Security decomposes to Confidentiality, Integrity, and Availability, while it also extends to additional properties: Non-repudiation, Auditability, and Accountability, which imply Authorization (in order to have control and auditable evidence over critical functions); and Authorization implies Authentication (authentication is a pre-requisite for any authorization decision to take place).

In the sections that follow, these overarching properties will be described in detail while later sections provide more specific security patterns that may cover one or more of these fundamental security properties.

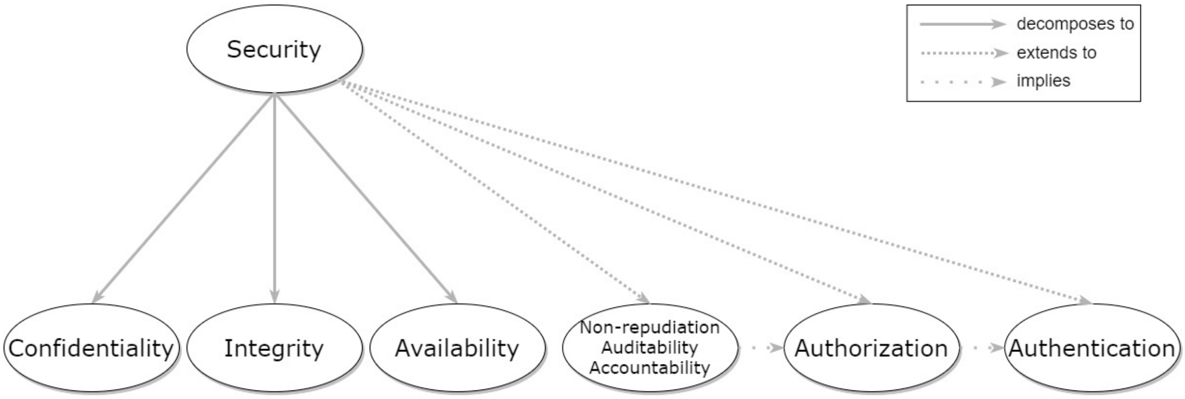


Figure 1. Hierarchical graph of Security and relevant patterns.

3.1. Confidentiality

Confidentiality refers to the "property that information is not made available or disclosed to unauthorized individuals, entities, or processes" (ISO/IEC 15408-2008 [29]). Thus, the preservation of Confidentiality requires that the disclosure of information happens only in an authorized manner, i.e., non-authorized access to information should not be possible. This implies leveraging a number of security controls to protect the confidentiality of data, which prominently include encryption mechanisms and, depending on the specific use case, will also typically encompass authentication and authorization provisions.

Formal definitions of Confidentiality are typically based on the concept of information flow (IF) [30], separating users in classes with different access rights to the system's information and distinguishing the information flows within the system according to the user classes to which they should be accessible. Herein, Confidentiality is considered a property that refers to a high-level problem and, thus, depends on specific patterns to solve lower-level problems, creating the context for them. The relationships of Confidentiality with the specific patterns are depicted in Figure 2. Three core patterns have been identified for achieving Confidentiality: Encrypted Channel, Encrypted Storage, and Encrypted Processing, respectively covering all data states, i.e., data in transit, data at rest, and data in processing, respectively.

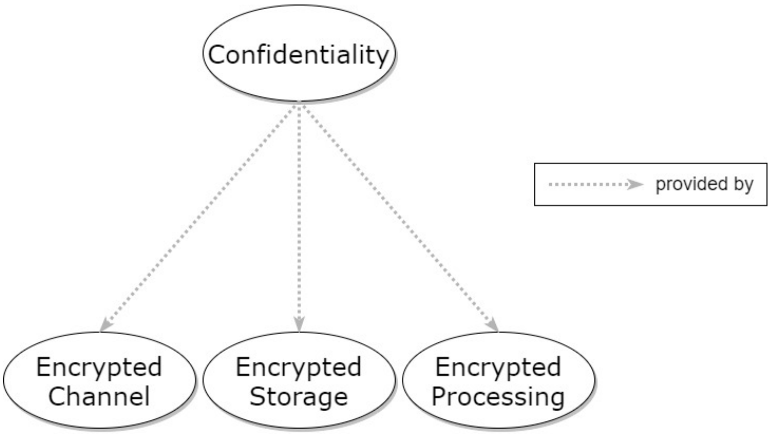


Figure 2. Hierarchical graph of Confidentiality property and relevant patterns.

3.1.1. Encrypted Storage

The Encrypted Storage pattern [15] provides a second line of defense against unauthorized disclosure of data since, even if other controls are bypassed, the most sensitive data will remain safe.

Examples of this pattern include (i) the mechanisms integrated in Operating Systems that allow encryption of the system's files (e.g., Microsoft's Bitlocker (https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview (accessed on 2 December 2020)) integrated into the Windows operating system); (ii) web sites that use encryption to protect the most sensitive data that must be stored on their backend databases (e.g., databases storing credit card numbers).

An overview of the pattern is provided in Figure 3. As can be seen in the figure, the Data generator creates the unencrypted data and sends it for storage. Right before the Storage component, the Encryption Module transforms the clear-text version of the sensitive data. Finally, the sensitive data are stored in the Storage unit in an encrypted form.

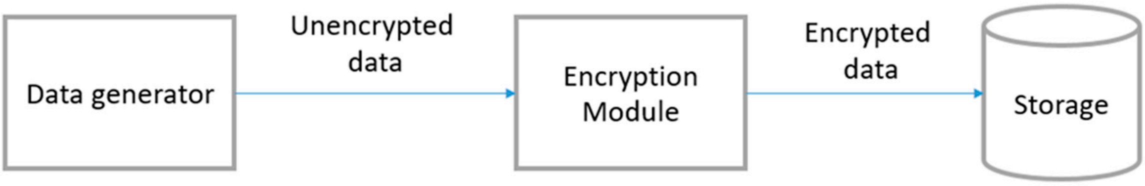


Figure 3. Descriptive overview of the Encrypted Storage pattern.

3.1.2. Encrypted Channel

The Encrypted Channels pattern [14] ensures that sensitive data remain confidential during transmission across networks. Information that passes across, e.g., public networks is open to eavesdropping and can be intercepted.

The Secure Sockets Layer (SSL)/Transport Layer Security (TLS) (https://tools.ietf.org/html/rfc8446 (accessed on 2 December 2020)) family of protocols is the most common implementation of the Encrypted Channels Pattern across the Internet. Web browsers, Web Servers and development platforms offer SSL/TLS capabilities. Other implementations of the pattern include protocols such as IPSec (https://tools.ietf.org/html/rfc4301 (accessed on 2 December 2020)) and various Virtual Private Network (VPN) applications leveraging said protocols.

Cryptographic information exchanged between two communicating parties (typically referred to as a "handshake") allows them to set up encrypted communication. A shared encryption (session) key is used for the encryption and decryption of the sensitive data. For the exchange of that key, a (typically, asymmetric encryption-based) protocol is used. Figure 4 shows the key exchange between the two communication participants.

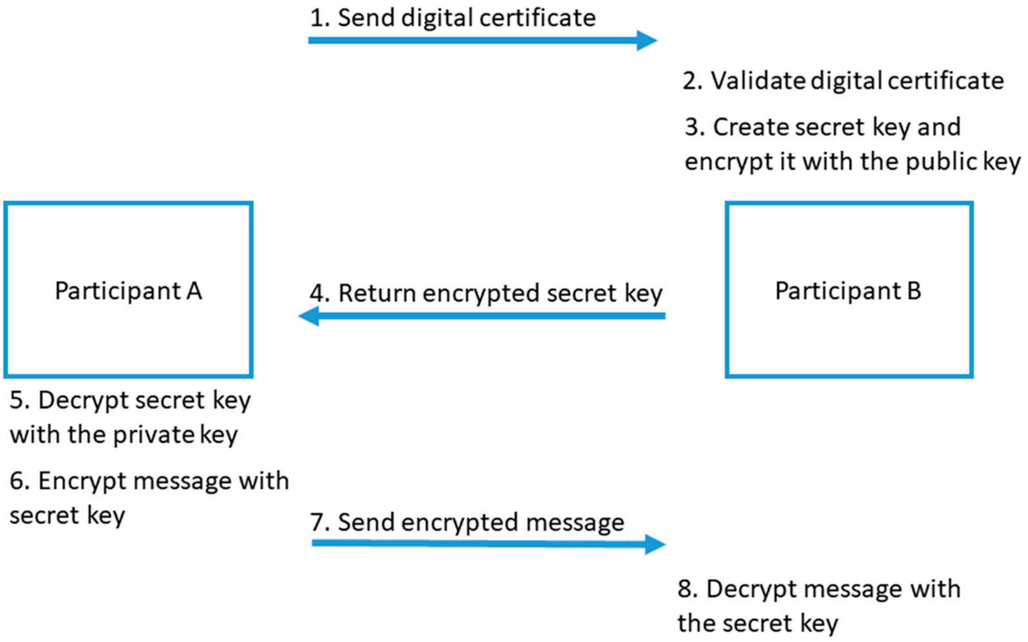

1. Send digital certificate
2. Validate digital certificate
3. Create secret key and encrypt it with the public key
Participant A
4. Return encrypted secret key
Participant B
5. Decrypt secret key with the private key
6. Encrypt message with secret key
7. Send encrypted message
8. Decrypt message with the secret key

Figure 4. Set-up of encrypted communication used in the Encrypted Channels pattern.

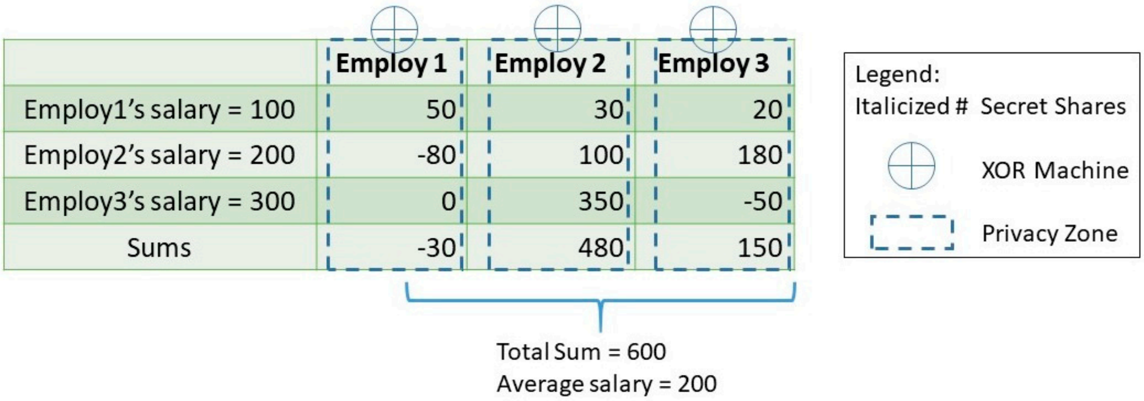
3.1.3. Encrypted Processing

Although encryption during transfer and storage does offer protection of sensitive data, it is typically decrypted while being processed, leaving it exposed to unauthorized disclosure if the processing host is compromised. The Encrypted Processing pattern aims to address this concern. The most significant advantages from the adoption of this approach include [31] strengthening of data security; considerable savings in computer time, and savings in the costs of the handling part of the security problems of the operating system.

Practical implementations of the Encrypted Processing pattern are mainly based on the use of homomorphic functions [31,32] and secure multi-party computation (SMC) [33,34]. The latter takes advantage of additive secret sharing, which refers to the division of a secret and its distribution among a group of independent, willing participants. As an example, let us consider a scenario where three employees want to calculate their average salary without revealing each person's own salary. Figure 5 depicts how the salary of each employee is randomly split into three pieces (secret shares) and distributed to the three employees. Therefore, the sums of the corresponding pieces allow the calculation of the total sum and the average salary, without disclosing any useful information about each employee's salary.

3.2. Integrity

Integrity refers to maintaining the consistency, accuracy, and trustworthiness of data over their entire life cycle. It is a "property of accuracy and completeness" according to ISO/IEC 15408-2008 [29].



	Employ 1	Employ 2	Employ 3
Employ1's salary = 100	50	30	20
Employ2's salary = 200	-80	100	180
Employ3's salary = 300	0	350	-50
Sums	-30	480	150

Figure 5. Secure multi-party computation (https://www.inpher.io/ (accessed on 2 December 2020))—overview of the average salary scenario.

To achieve this, we must be sure that data are not altered in an unauthorized manner at any state. Firstly, data must not be altered in transit; to achieve this, a secure protocol integrating security checks can be used. After reaching their destination, data must not be altered by unauthorized users; file permissions and user access controls are mechanisms that are commonly used towards that direction. However, even authorized users may cause accidental changes. Version control can be the solution to that problem. Finally, changes may occur as a result of events such as an electromagnetic pulse (EMP) or server crashes. Checksums can be used for verification of integrity, and available backups or redundancies are useful for the restoration of affected data.

Integrity, as was the case for Confidentiality, is considered a property that refers to a high-level, complex problem, thus creating the context for more specific patterns. The relationships of Integrity with the specific patterns are depicted in the graph of Figure 6. Three high-level patterns (Safe Channel, Safe Storage, and Safe Processing) can be utilized for achieving integrity and three lower-level patterns can be used for the implementation of the above, as sketched in the figure.

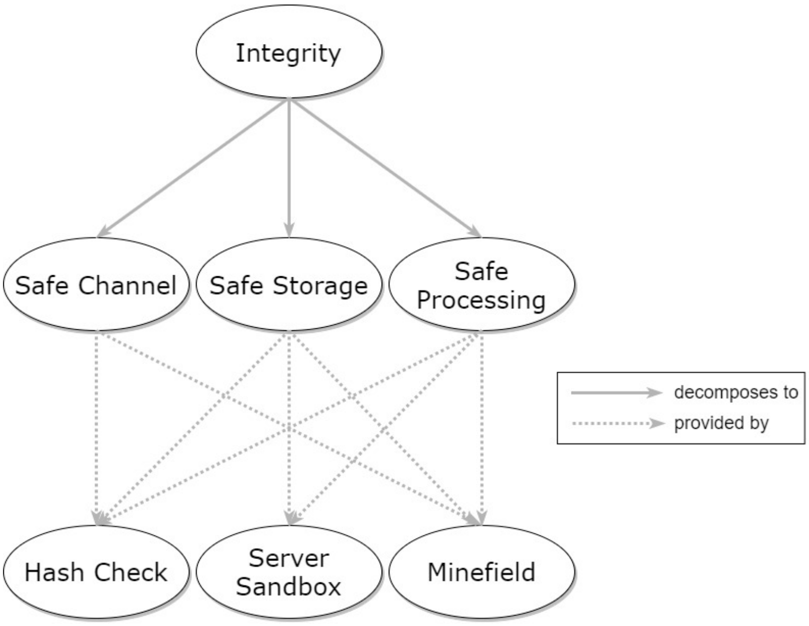


Figure 6. Hierarchical graph of Integrity property and relevant patterns.

3.2.1. Safe Channel

The Safe Channel pattern [28] ensures integrity on the transport layer, making use of certificates. Communicating through public (i.e., unsafe) networks allows for interceptions, making the transmitted messages open to tampering.

The file channel integrity tool in Apache Flume (https://flume.apache.org/ (accessed on 2 December 2020)), a distributed system for efficiently collecting, aggregating, and moving large amounts of log data from different sources to a centralized data store, is an example of implementation of the Safe Channel pattern. The Transmission Control Protocol (TCP) protocol also provides a (non-cryptographic) checksum to aid in detecting data corruption, while not protecting against reordering of data nor recalculation of the checksum. Finally, the Minefield is a relevant lower-level pattern that will be detailed below.

3.2.2. Safe Storage

The Safe Storage pattern [28] ensures integrity at storage level, i.e., for data at rest. When data integrity holds for a system, the information stored locally will remain complete, accurate, and reliable no matter how long it is stored for, or how often it is accessed. This form of integrity also encompasses the safety of data in regard to regulatory compliance (i.e., auditable data).

Lower-level patterns that can be used to implement these aspects of integrity are the Hash Check, Server Sandbox, and Minefield patterns, as shown in Figure 6.

3.2.3. Safe Processing

The Safe Processing pattern [28] targets integrity at the processing level, thus encompassing the protection of data that are being used in processing. Data integrity must hold during and after the processing of data by an application.

The Minefield is a relevant lower-level pattern which will be defined below, but other implementations can easily be derived, e.g., by decorating any processing code with integrity checks.

3.2.4. Hash Check

A direct implementation of the Safe Channel pattern is the Hash Check pattern. Let us assume a sequential orchestration P with two activity placeholders, A and B, whereby B is executed after A.

We assume that for each x in {P, A, B}, the following hold:

- IN^x and OUT^x are the sets of inputs and outputs of x;
- D^x (i) are the data of x at the given time t_i;
- Hash(D^x (i)) are the cryptographic hash function results applied to data $D^x(i)$.

Further conditions that define P include:

- The inputs of A are the inputs of the orchestration P;
- The inputs of B are the outputs of A;
- The outputs of the orchestration P are the outputs of B.

Based on the above, a pattern for preserving integrity for data that are in processing and in transit on the service orchestration P can be defined as follows:

- Hash (IN^P) = Hash (IN^A);
- Hash (IN^B) = Hash (OUT^A);
- Hash (OUT^P) = Hash (OUT^B).

Interpreting the pattern above, we have to—for all data that are transmitted, not only through datalinks but also through inter-process communication—evaluate that the data that activity A sends to activity B are not changed.

Moreover, based on the above specification. we can define a generic pattern for integrity of data at rest as the following:

$$\text{Hash}\ (D^x\ (i)) = \text{Hash}\ (D^x\ (i-1)) \tag{1}$$

which means that whenever we check data at rest, that data must not be changed.

3.2.5. Server Sandbox

What the Server Sandbox [15] pattern does is to build a barrier around the Web server. In that way, for example, damage that could result from an undiscovered bug in the server software is contained (Figure 7). To achieve sandboxing, privileges that the Web application components possess at run time must be strictly limited by creating a user account that is to be used only by the server. Said accounts' privileges are, then, limited by the underlying operating system's access control mechanisms to those that are needed to execute but not to administrate or otherwise alter the server.

As a result, integrity is enhanced since component vulnerabilities are prevented from causing the entire server to be compromised.

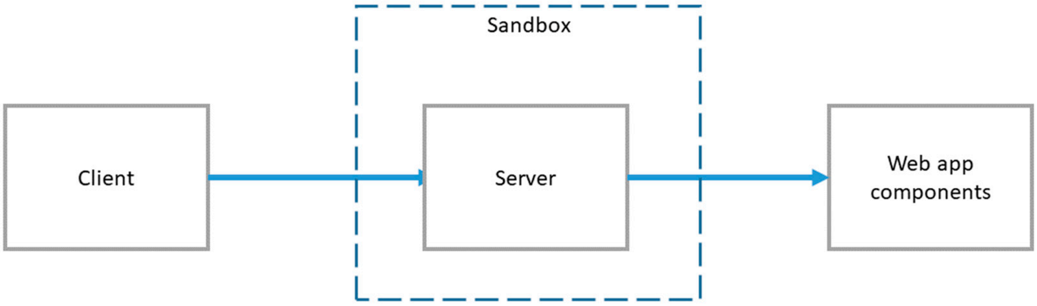


Figure 7. Descriptive overview of Server Sandbox pattern.

3.2.6. Minefield

The Minefield pattern [15] tricks, detects, and blocks attackers during a break-in attempt. Security aspects of standard components are often well-known to attackers. Variations that are introduced by Minefield counter the attackers' advantage and aid in their detection.

There is no specific way to implement the Minefield pattern. If there was, that would defeat the purpose. However, some basic approaches are the following:

- Rename common, exclusively administrative commands on the server and replace them with instrumented versions that alert the administrator to an intruder before executing the requested command.
- Alter the file system structure.
- Introduce controlled variations using tools such as the Deception Toolkit [35].
- Add application-specific boobytraps that will recognize tampering and prevent the application from starting.

Through the above, integrity is protected by increasing confidence that scripted attacks will fail and that human attackers will be detected before they can cause damage.

Identification and monitoring of security threats in complex network infrastructures can be performed with Intrusion Detection Systems (IDSs) utilizing two methods; signature-based and profile-based (or anomaly-based), e.g., as described in [36].

3.3. *Availability*

According to [37], availability is defined as "readiness for correct system service", whereby a service is deemed to be "correct" if it implements the specified system function. The term Readiness of a system, in this context, refers to obtaining a correct response to a request for accessing some information or using a resource. An alternative definition of Availability refers to the "property of being accessible and usable on demand by an authorized entity", as defined in ISO/IEC 15408-2008 [29].

A more information technology-focused definition of Availability [38] refers to a system that is continuously operational for a desirably long length of time. Availability can be measured relative to “100% operational” or “never failing.” In actual practice, Availability goals are expressed and measured in the number of nines of availability, typically ranging from 99.9% (3NINES) to 99.999% (5NINES) and even up to 99.9999% (6NINES).

Considering the above, we consider Availability as a property that refers to a high-level problem and depends on specific lower-level patterns, creating the context for them. The relationships of Availability with the specific patterns are depicted in Figure 8, in the form of a graph. Three core patterns have been identified for achieving Availability: Uptime, Redundancy, and Fault Management.

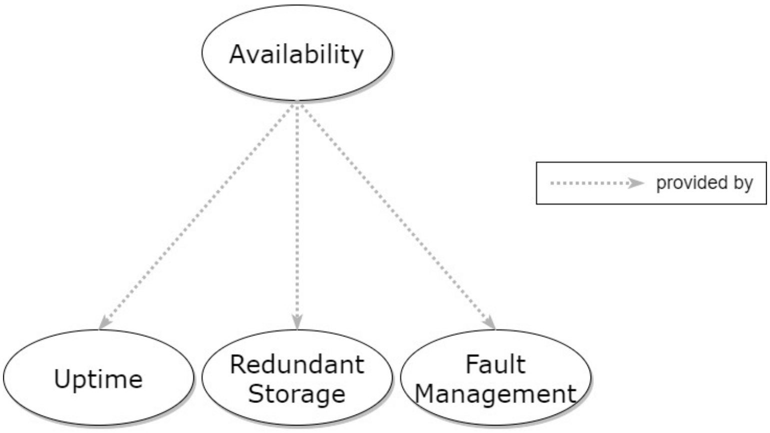


Figure 8. Hierarchical graph of Availability property and relevant patterns.

3.3.1. Uptime

As mentioned, one way to interpret Availability is Uptime, i.e., the time that a system (e.g., a machine, a computer, or a website) is working and is available to the users over a given period. Motivated by this, an Uptime pattern can be defined, which can be used to monitor the Uptime of the resource of interest and provide satisfaction or violation evidence in reference to the desired value.

One way to define Uptime is as an absolute value (e.g., in minutes, days, and months) or as a percentage (e.g., 99.8%). In the first case, what it is measured is the length of time a system has been running since it was initialized. In the second case, the percentage is calculated by dividing the total time the system has been available by the time it has been active.

3.3.2. Redundant Storage

Traditional data storage media, such as solid-state storage, are subject to failure or data errors due to wear-out. A solution to these problems is the use of a distributed redundant array of independent drives (RAID) or other redundant data systems [39]. In such systems, if a data device fails, the data of the failed device are restored with the help of parity devices. On the other hand, if a parity device fails, the data in the data devices are used to restore the parity data.

The Redundant Storage pattern [40] is presented as a solution to the problem of failure of cloud storage devices. In cases that consumers are unable to access data and/or services become unavailable, a secondary redundant cloud storage device is synchronizing its data with the data in the primary cloud storage device. Thus, when the latter fails, a storage service gateway diverts requests to the former, ensuring uninterrupted service delivery.

3.3.3. Fault Management

According to [41], Fault Management is “the management of the abnormal operation of a computer system” and “the purpose of fault management is to return a computer system to normal operation”. Fault Management includes all the necessary steps from the detection of a fault to its removal from the system.

Authors in [42] present a series of such steps as part of the so-called "flow of fault management" in networks: (i) collect alarms, (ii) maintain customer satisfaction through immediate action, (iii) filter and correlate the alarms, (iv) diagnose faults through analysis and testing, (v) determine a plan for correction and implement it, (vi) verify the fault is eliminated, and (vii) record data and determine the effectiveness of the current fault management function.

Finally, widespread fault-tolerant techniques for distributed networks, along with more efficient methods, are available in the literature [43]. Such techniques include Simple Network Management Protocol (SNMP) and TCP/IP, clustering SNMP agents, and replication of crashed agents by peer cluster [44], application of k-nearest neighbor algorithm for poison message failure [45], usage of MultiRouter Traffic Graphing for the creation of a network monitoring tool [46], a policy-based application for fault identification and action decision [47], and an eXtensible Markup Language (XML) notation for the specification of dependencies between managed resources [38].

3.4. Non-Repudiation, Accountability, and Auditability

A set of additional key properties integral to the provision of security are non-repudiation, accountability, and auditability. Non-repudiation refers to the "ability to prove the occurrence of a claimed event or action and its originating entities" (ISO/IEC 27000:2018 [48]). Auditability from a security perspective requires the ability to recognize, record, store, and analyze information related to security relevant activities, producing "audit records that can be examined to determine which security relevant activities took place and whom (which user) is responsible for them" (ISO/IEC 15408-2008 [29]). Finally, accountability, as the name suggests, refers to the ability of assigning actions and decisions to entities, while having the means (e.g., using authentication, authorization, auditing, and non-repudiation mechanisms) to hold said entities accountable for those actions and decisions.

Since the aforementioned properties are considered to refer to high-level problems, they depend on lower-level patterns, creating the context for them. Their relationships to those patterns are depicted in Figure 9, in the form of a graph. Signed Message and Audit Log patterns can be utilized to provide non-repudiation, accountability, and auditability.

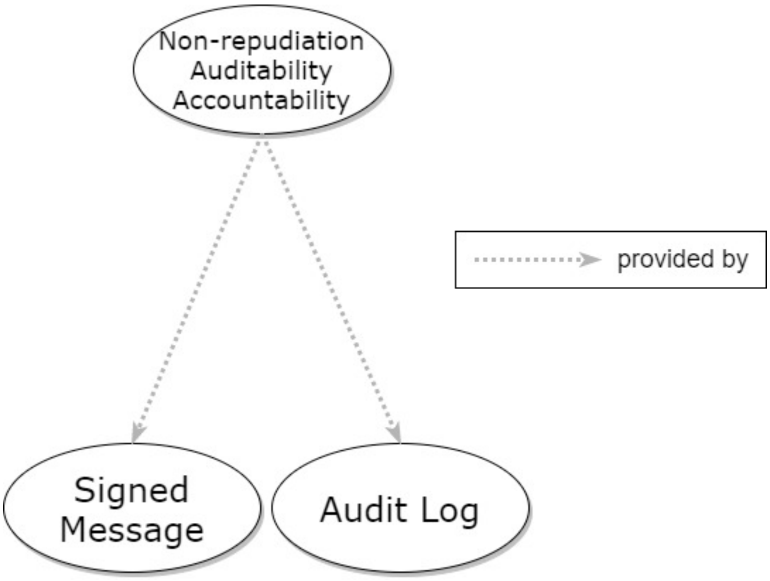


Figure 9. Hierarchical graph of Non-repudiation, Accountability, and Auditability properties and relevant patterns.

3.4.1. Signed Message

The Signed Message pattern ensures a message's authenticity, integrity, and non-repudiation. The message sender may sign the message using a digital signature, and in that way, the signer is securely associated with the generation and/or transmission of the message. Digital signatures use Public Key Infrastructures (PKIs) to provide the highest levels of security along with universal acceptance. They are a specific signature technology implementation of electronic signature (eSignature).

Three algorithms are involved with the digital signature process:

- Key generation: This algorithm provides a private key along with its corresponding public key.
- Signing: This algorithm produces a signature (typically on the digest of the message) upon receiving a private key and the message that is being signed.
- Verification: This algorithm checks the authenticity of the message by verifying it along with the signature and the public key.

Figure 10 shows the processes that take place and the communications between the involved parties. The signer uses their private key to sign the message, while the verifier uses the corresponding public key to verify the signature and, by extension, the message. The signature is valid when the two hash values (i.e., message digests), the one stored and the one calculated, are equal.

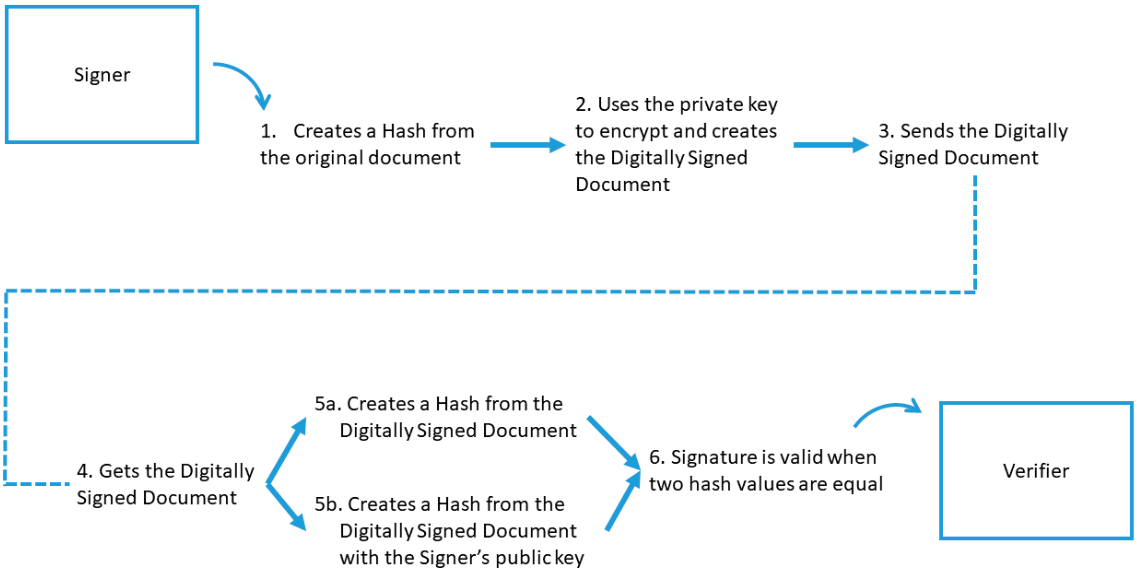

Signer
1. Creates a Hash from the original document
2. Uses the private key to encrypt and creates the Digitally Signed Document
3. Sends the Digitally Signed Document
4. Gets the Digitally Signed Document
5a. Creates a Hash from the Digitally Signed Document
5b. Creates a Hash from the Digitally Signed Document with the Signer's public key
6. Signature is valid when two hash values are equal
Verifier

Figure 10. Descriptive overview of Signed Message pattern (https://medium.com/@meruja/digital-signature-generation-75cc63b7e1b4 (accessed on 2 December 2020)).

3.4.2. Audit Log

The Audit log is the simplest, yet very effective, form of tracking temporal information. The idea is that whenever something significant happens, you write a record indicating what happened and when it happened.

An audit log can have many implementations. The most common one is a file-based log. A database table can also be used as an audit log. If you use a file, you need a format, such as ASCII, which makes it readable to humans without special software. If it is a simple tabular structure, then text is simple and effective. XML can be used for more complex structures.

At the system level, UNIX offers a variety of standard log files, directed to the/var partition. The "cron" program performs the rotation of log files. Old logs are compressed to save space and are renamed with a unique suffix. Moreover, at the network level, tools such as the NetCool unified logging system offer a number of different "probes" that can be used to collect data from a variety of sources, including conventional text log files, UNIX syslog streams, Private Internet Exchange (PIX) firewall events, and Oracle table insertions [15].

3.5. Authorization

Authorization defines who is allowed to access specific resources in a system, and the way in which a resource is accessible by a specific user. Authorization implies the existence of authentication (before checking if entity "X" is allowed to access resource "Y", we have to confirm that the entity interacting is indeed "X"—hence the need to authenticate "X"). In this context, Figure 11 presents an authorization process pattern, which also shows the role of authentication within.

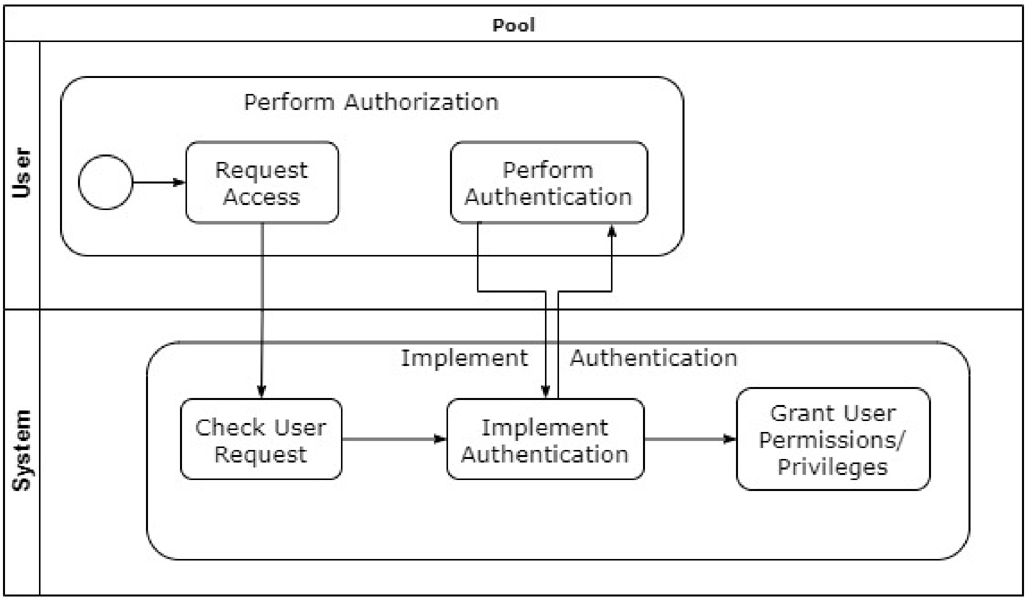


Figure 11. Conceptual overview of Authorization Process pattern [49].

We consider Authorization as a property that refers to a high-level problem and depends on specific lower-level patterns for implementation, creating the context for them. Two such patterns that can be utilized for achieving Authorization are Single Access Point and Authorization Enforcer, with the latter actually implying the existence of the former. The relationship of Authorization to these specific patterns is depicted in the graph of Figure 12.

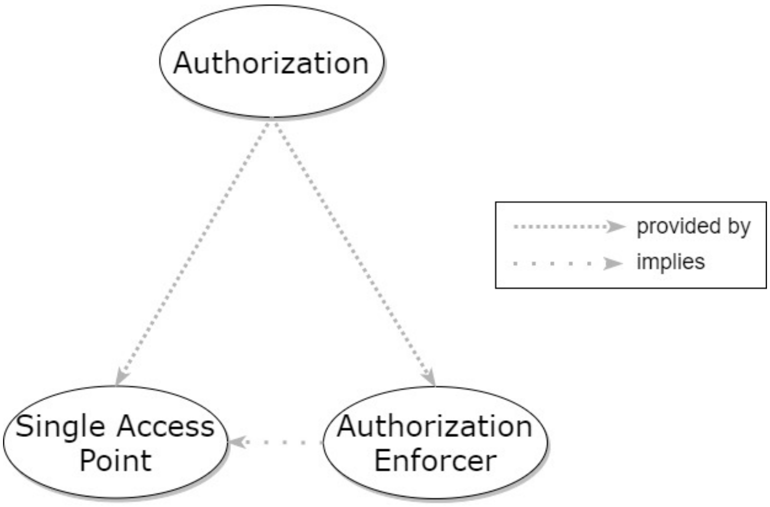


Figure 12. Hierarchical graph of Authorization property and relevant patterns.

3.5.1. Single Access Point

The Single Access Point [14] grants or denies entry to the system after checking the client requiring access. In that way, external access is provided along with protection from misuse or damage. The Single Access Point pattern is easy to apply, defines a clear entry point to the system, and can be assessed when implementing the desired security policy. Through a Single Access Point, we can provide authorization and also support the overall system security posture.

Concrete implementations are elements such as a Login Window or a Validation Screen. The client logs in at the Single Access Point and then uses the protected system. As an entry point is created, it can be used for the enforcement of different policies such as authentication, authorization, validity of incoming data, known offenders, and replay policies. Therefore, from an implementation perspective, the establishment of a Single Access Point must be supported with the introduction of a check point [14] (including, e.g., a Policy Decision Point and a Policy Enforcement Point, as typically found in policy-based authorization management solutions [50]).

3.5.2. Authorization Enforcer

The Authorization Enforcer [14] describes who is authorized to access specific resources in a system, in an environment in which we have resources whose access needs to be controlled. It indicates, for each active entity that can access resources, which resources it can access and how it can access them.

As depicted in Figure 13, the Subject class describes an active entity that attempts to access a resource in some way. The Protected Object class represents the resource to be protected. The association between the subject and the object defines an authorization, from which the pattern gets its name. The association class Right describes the access type (for example, read and write) the subject is allowed to perform on the corresponding object. Through this class, one can check the rights that a subject has on some object, or who is allowed to access a given object.

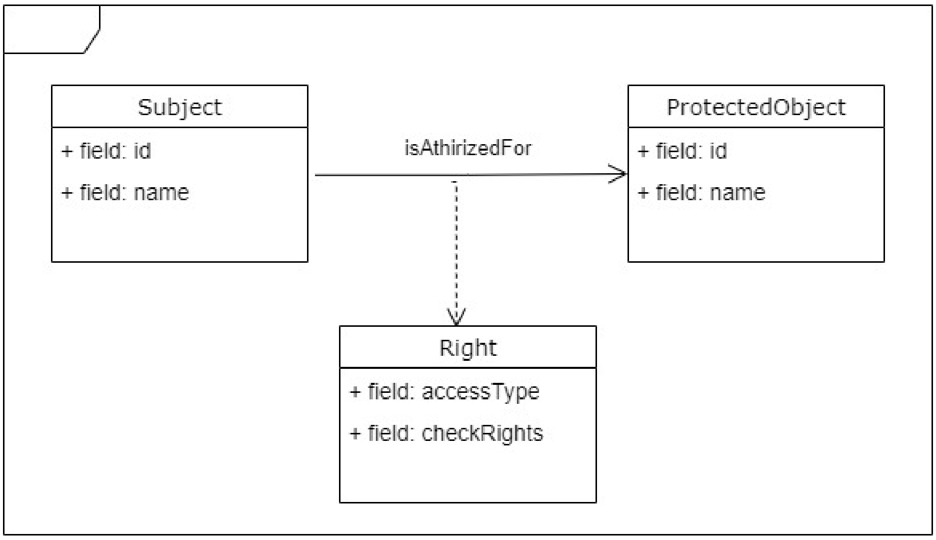


Figure 13. Essential components of Authorization Enforcer pattern—class diagram [14].

The two most common implementations of this pattern are Access Control Lists and Capabilities. Access Control Lists (ACLs) are kept with the objects to indicate who is authorized to access them, while Capabilities are assigned to processes to define their execution rights. Access types should be application-oriented.

3.6. *Authentication*

Authentication is the "provision of assurance that a claimed characteristic of an entity is correct" (ISO/IEC 15408-2008 [29]). According to [51], Authentication enforces the verification and validation of the identities and credentials exchanged between the Web services provider and the consumer. A requester must be authenticated, i.e., to prove its identity with credentials that are considered reliable (e.g., X.509 digital certificates [52], Kerberos tickets [53], or any kind of security token). An Authentication process pattern presenting the above in a structured manner is provided in Figure 14.

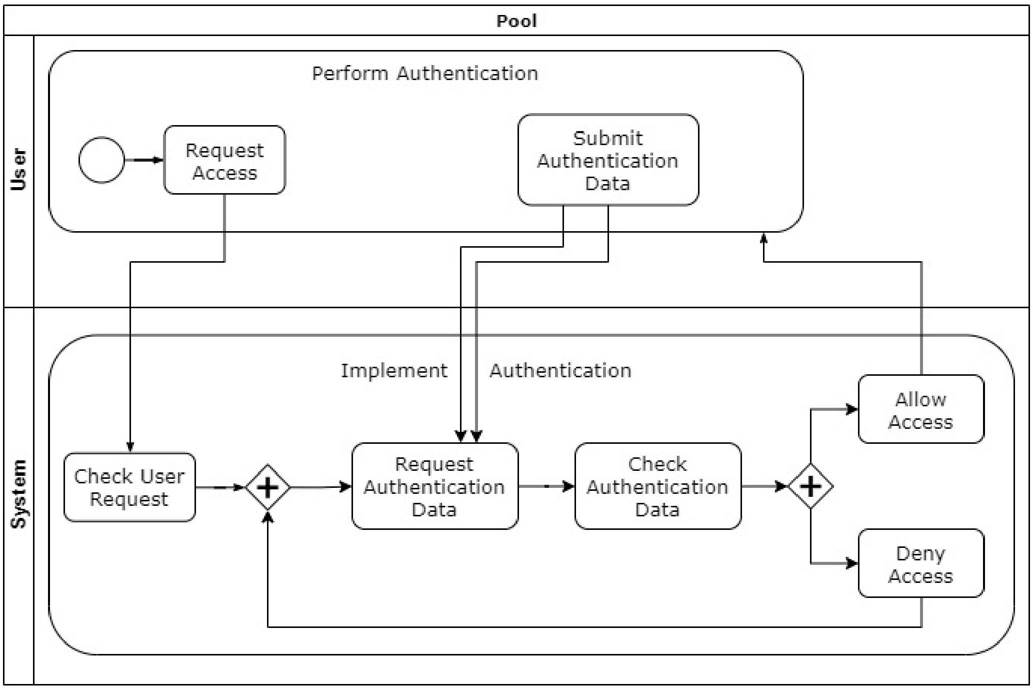


Figure 14. Conceptual overview of Authentication Process pattern [49].

Similarly to Authorization, Authentication is considered a property that refers to a high-level problem and depends on specific lower-level patterns for its implementation, creating the context for them. A set of patterns, in two levels, can be utilized for achieving Authentication. The relationship of Authentication to the specific patterns is shown in the graph of Figure 15.

3.6.1. Authenticated Channel

The Authenticated channel ensures authenticity on the channel level. According to [54], a secure authenticated channel should have the following characteristics:

- Mutual authentication of the peers;
- Key confirmation (at least one peer is able to verify that the common secret is indeed common);
- Forward secrecy (old session keys cannot be calculated even when long-term secret keys (such as certificate Secret keys are known).

Implementation examples of an Authenticated channel are channels established through Kerberos authentication [53] and the two-way Secure Sockets Layer (SSL) authentication [55].

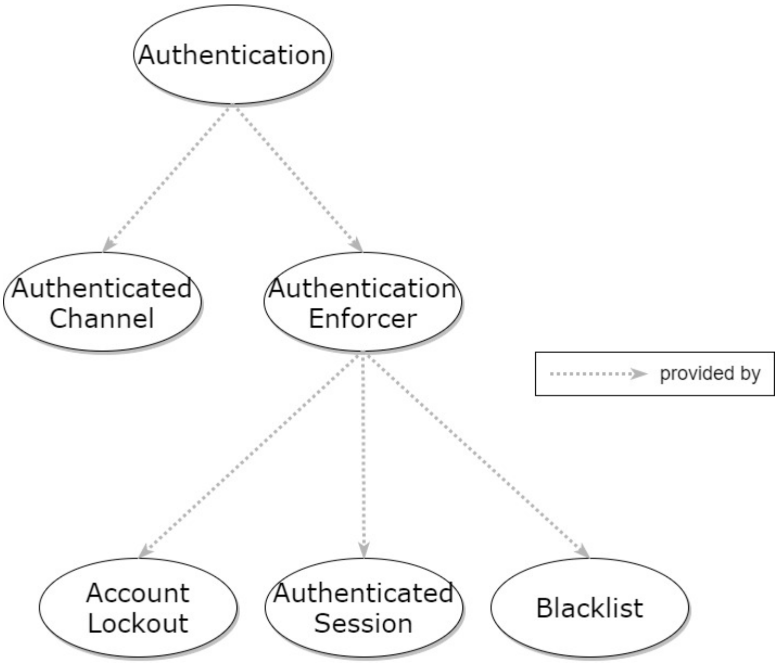


Figure 15. Hierarchical graph of Authentication property and relevant patterns.

3.6.2. Authentication Enforcer

Operating systems have legitimate users that use it to host their files. However, there is no way to make sure that a user who is logged in is a legitimate user. Users can impersonate others and gain illegal access to their files. The Authentication Enforcer pattern addresses the problem of how to verify that a subject is who it says it is.

A Subject, typically a user, requests access to system resources. The Authentication Enforcer receives this request and applies a protocol using some Authentication Information. If the authentication is successful, the Authentication Enforcer creates a Proof of Identity, which can be explicit—for example, a token—or implicit (see Figure 16).

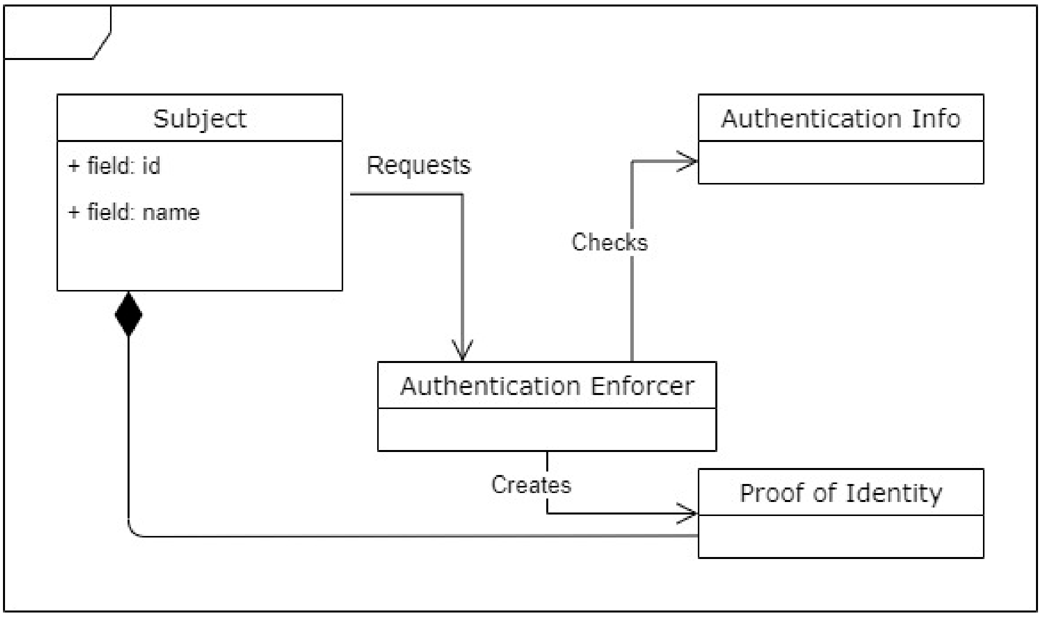


Figure 16. Essential components of Authentication Enforcer pattern—class diagram [14].

The implementation of such a pattern includes (i) the definition of the authentication requirements (number of users, degree of security, etc.); (ii) the selection of the authentication approach (e.g., the use of passwords); and (iii) the building of the authentication information.

The Authentication Enforcer allows user data to be protected from unauthorized disclosure. The Authentication Enforcer can itself be considered as a pattern that creates the context for other more specific patterns to solve lower-level problems. It creates an Authentication Session and acts as an enforcement point for the Account Lockout and Blacklist patterns.

3.6.3. Account Lockout

Account Lockout protects user accounts from automated password-guessing attacks by implementing a limit on incorrect password attempts before further attempts are disallowed.

A high-level diagram is provided in Figure 17, while the steps of an authentication attempt are presented below:

1. The client is provided with a transaction form or login screen requiring both a username and a password.
2. The user provides the username and password and submits the request for a protected resource.
3. The mediator checks the username and, if valid, retrieves the account information. If the username is invalid, it returns a generic failed login message.
4. The mediator checks if this user's account is locked out (number of successive failed logins exceeds the threshold) and not yet cleared (last failed login time + reset duration > current time). If locked, it returns a generic failed login message.
5. If the user's account is not locked, the mediator checks the validity of the password. If the password is not valid, it increments the number of failed login attempts against the account and sets the last failed login time to the current time. It then returns a generic failed login message.
6. If the password is valid, it resets the number of failed logins to 0 and executes the request against the protected resource.

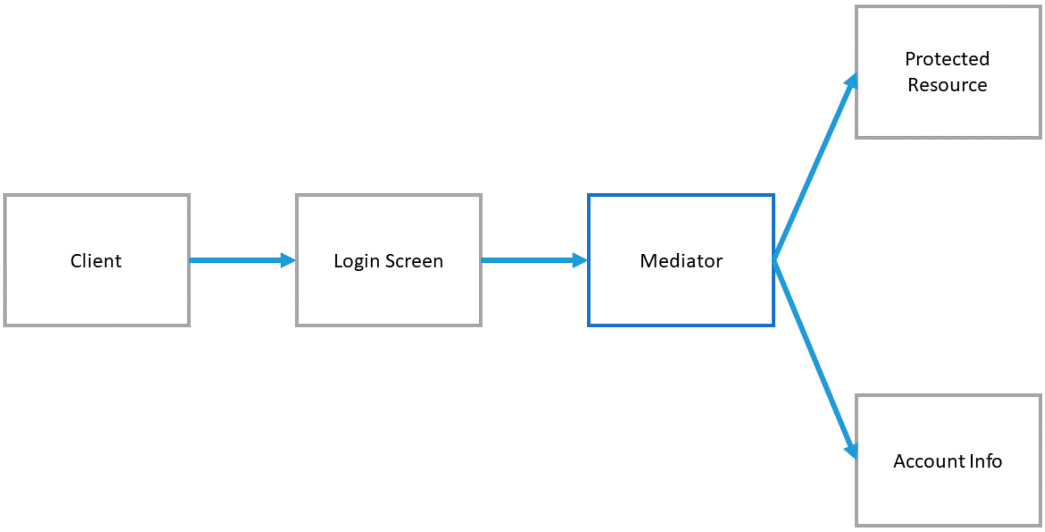


Figure 17. Conceptual overview of Account Lockout pattern [15].

3.6.4. Authenticated Session

An Authenticated Session keeps track of a user's authenticated identity through the duration of a Web session. A user is allowed to access multiple protected pages on a website authenticating themselves just once. It keeps track of the last page access time and causes the session to expire after a predetermined period of inactivity.

The Authenticated Session pattern caches the user's authenticated identity on the server. As a result, the application is more confident that the user has not tampered with it. The only way that the authenticated identity session attribute can be set is if the user is successfully authenticated to the authentication module. The Authenticated Session pattern utilizes existing session mechanisms to associate the client with a particular session.

The repeated authentication in this automated manner creates accountability, while the security posture of the system is, consequently, increased.

3.6.5. Blacklist

A Blacklist (Figure 18) is used to keep track of network addresses from where hacking attempts and other malicious actions have originated from. Requests from an address on the blacklist are ignored.

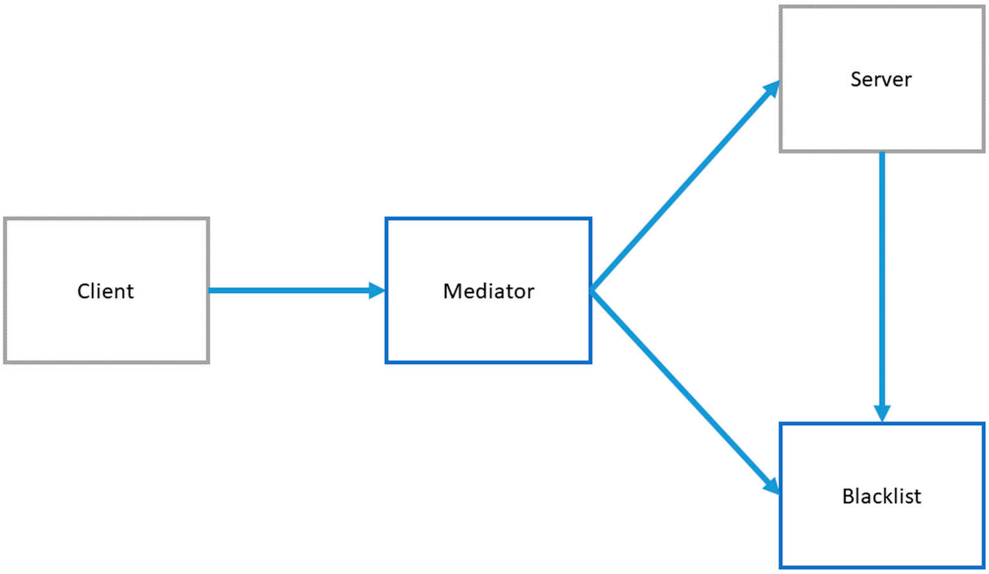


Figure 18. Conceptual overview of Blacklist pattern [15].

Such blocking mechanisms can be deployed at the network and at the application (endpoint) level.

Blocking at the network level: Normally, a client with an Internet Protocol address (IP) that is not on the blacklist will make a request to the server and the blocking mechanism will let the request pass. The server will respond with normal functionality. However, a suspicious request from a non-blacklisted client will cause the server to keep a record of the network address and the nature of the request. If the request is sufficiently suspicious, the server will request that the address be blacklisted. The blocking mechanism will then be configured to deny further requests from that address. Moreover, when a blacklisted client makes a request, the blocking mechanism will not just drop the request but also make a log of it. A blacklisted address may be removed from the blacklist after a period of inactivity or because of manual administrator intervention.

Blocking mechanism within the application: Similar to the above case, when an application request is received (e.g., access to a specific exposed interface), it is first checked against the blacklist before being dispatched to the appropriate interface.

If the Blacklist pattern is implemented effectively, it enhances the system's security level as it will dissuade or prevent attempts to misuse the system.

4. Privacy Patterns

Privacy is a complex topic with significant legal and regulatory implications. The latter comprise a dynamic landscape, as ever-stricter privacy laws are being introduced in many countries around the world, with the EU being at the forefront of such efforts, mainly through General Data Protection Regulation (GDPR) [56], and the push for "Privacy by design".

There have been various efforts to provide architectural and design patterns for the protection of privacy (e.g., [21,24,25,57]), though these are significantly less common than other topics such as patterns covering software development and cloud architectures, or even security properties, such as the ones defined in the previous sections. Still, this is expected to improve as privacy comes into the limelight, being recognized as an important standalone requirement.

An important obstacle when defining privacy patterns is the fact that while privacy properties themselves are relatively well understood, there is a lack of established terminology for referring to some of the properties [58] and also a lack of a clear taxonomy in defining the relationships between said properties [59]. Therefore, a literature review reveals conflicting definitions of the terms as well as their relationships, even among privacy-related standards. For example, some works group anonymity and pseudonymity together (e.g., [60]), while others highlight the significant differences between the two in terms of linkability (e.g., see Pfitzmann et al. [58] and ISO/IEC 29100:2011 [26]). Another example of such discrepancies is that some works group unobservability and undetectability (e.g., see ISO/IEC 15408-2:2008 [29] and Kalloniatis et al. [60]), while others differentiate between the two (e.g., Pfitzmann et al. [58] and Diamantopoulou et al. [49]).

Considering the above landscape and consolidating the various views, the work presented herein covers the eight key privacy concepts as identified and defined by the consensus of works in the area [31,58,61], namely (i) data protection; (ii) authentication; (iii) authorization; (iv) anonymity; (v) pseudonymity; (vi) unlinkability; (vii) undetectability, and (viii) unobservability.

From the above, we can derive two subgroups of properties: properties (i–iii) are, in fact, security properties (which is expected, since security is required to achieve privacy, considering the need for protection of personal data), while properties (iv–viii) are solely privacy-related. Thus, for the first group of properties, we refer the reader to the analysis of the corresponding properties above, while the properties of the second group will be defined in the sections that follow.

The sections below also highlight enablers (typically referred to as Privacy-Enabling Technologies—PETs—in the context of Privacy) allowing the satisfaction of said properties from an implementation perspective, while providing patterns for some characteristic examples of them. PETs are defined as "a system of ICT measures protecting informational privacy by eliminating or minimizing personal data thereby preventing unnecessary or unwanted processing of personal data, without the loss of the functionality of the information system" [62,63]. In the context of this work, PETs can be considered as specific technological building blocks that can be used to implement a pattern.

It should be noted that anonymity, pseudonymity, unlinkability, undetectability, and unobservability depend on the system/attacker model considered. Said model defines the capabilities of the attacker (if the attacker eavesdrops upon or controls the network and/or some of the endpoints, if the attacker is colluding with some of the senders or some of the receivers or third parties, and other such variations). A full coverage and analysis on the satisfaction of these properties under all envisioned cases is covered by the rich literature on the matter (e.g., the seminal works of Pfitzmann and Hansen [58], Pfitzmann and Waidner [64], Pfitzmann [65], and other resources on the topic [66]).

The above-mentioned privacy properties, concepts, and related patterns, their relationships, and the enabling patterns that will be provided herein are depicted in the hierarchical graph of Figure 19.

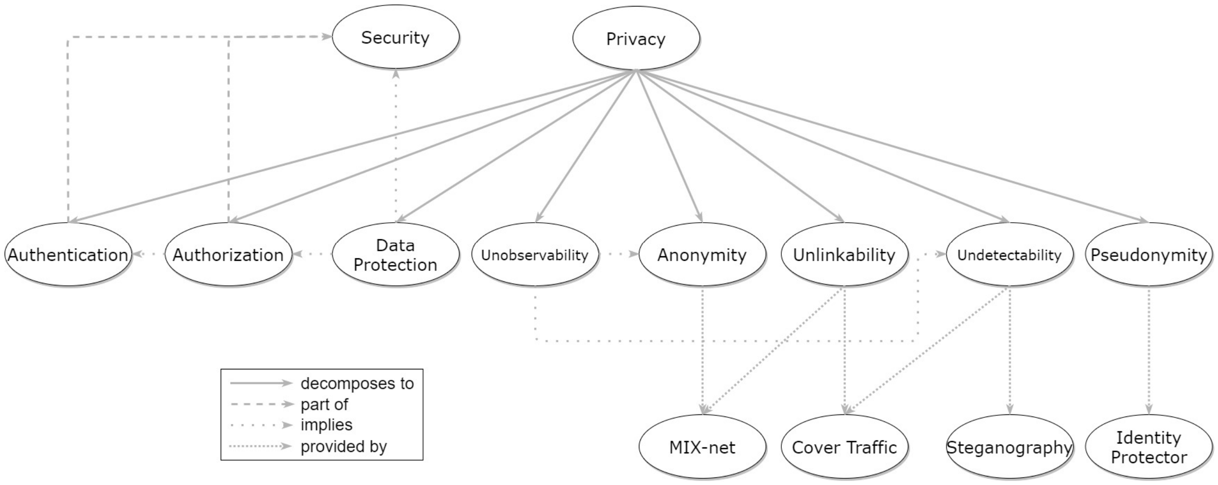


Figure 19. Hierarchical graph of Privacy properties and relevant patterns.

4.1. Anonymity

Anonymity "ensures that a user may use a resource or service without disclosing the user's identity" (ISO/IEC 15408-2008 [29]). A definition more evidently considering the current legislative privacy landscape (e.g., GDPR), thus focusing on personally identifiable information (PII), states that, "Anonymity is a characteristic of information that does not permit a personally identifiable information principal to be identified directly or indirectly" (ISO/IEC 29100:2011 [26]). Nevertheless, a more generic view of anonymity is provided in the following: "Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set" [58], whereby "anonymity set" is the set of all possible subjects.

A formal definition of anonymity is as follows (originally defined in [65], adapted in [66]):

Let R_U denote the event that an entity U (e.g., a user) performs a role R (e.g., as a sender or receiver of a message) during an event E (e.g., communication event or a service access). Let A denote an attacker, and let NC_A be the set of entities that are not cooperating with A.

An entity U is called anonymous in role R for an event E against an attacker A, if for each observation B that A can make, the following holds:

$$\forall U' \in NC_A: 0 < P(R_{U'} \mid B) < 1$$

A less strict requirement could be that the above relation does not have to hold for all but does for at least most non-cooperating entities from the set NC_A. However, anonymity for an entity U in the role R can only be guaranteed if the value $P(R_U \mid B)$ is not too close to the values 1 or 0. Thus, an additional requirement should be:

$$0 \ll P(R_U \mid B) \ll 1 \text{ [A << B means that A is much smaller than B]}$$

As mentioned above, anonymity also depends on the model considered, leading to different types of anonymity: sender anonymity means the user is anonymous in their role as a sender (the receiver might not be), and vice versa for receiver anonymity. Elaborating on the above:

An entity U is defined as perfectly anonymous in role R for an event E against an attacker A, if for each observation B that A can make:

$$\forall U' \in NC_A: P(R_{U'}) = P(R_{U'} \mid B)$$

That is, observations give an attacker no additional information.

Thus, in perfect sender (or receiver) anonymity, the attacker cannot distinguish the situations in which a potential sender (receiver) actually sent (received) communications and those in which he did not.

As such, anonymity facilitates user access to services and resources without revealing their identity or other sensitive information (e.g., exact location), blocking tracking and profiling attempts. Nevertheless, some constraints have to be considered—e.g., strong anonymity implies a large anonymity set (i.e., a large set of users, which is not always possible), while strong anonymity also constrains the usability of data (e.g., location-based or personalized services cannot be as accurate).

A significant number of PETs can be leveraged to implement anonymity, as the development of anonymous communication networks (ACNs) comprises one of the most prominent areas of privacy-related research and development efforts. Solutions include the use of anonymizing proxies and trusted third parties (e.g., CATS (https://www.custodix.com/index.php/cats (accessed on 2 December 2020))), Mix networks [67] (e.g., the Mixmaster anonymous remailer), DC-nets [68], Onion Routing [69] (e.g., the Tor browser [70]), *k*-Anonymity [71], *L*-diversity [72], *t*-Closeness [73] schemes (e.g., for location privacy [74]), attribute-centric anonymization [75], and others [76].

4.2. Unlinkability

Unlinkability "ensures that a user may make multiple uses of resources or services without others being able to link these uses together" and "requires that users and/or subjects are unable to determine whether the same user caused certain specific operations in the system" (ISO/IEC 15408-2008 [29]). A more universal definition, not focusing specifically on users but on items of interest (IOIs) in general, has been proposed by Pfitzmann et al. [58]: "Unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, ...) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not".

More formally, unlinkability can be defined as follows (originally defined in [65], adapted in [66]):

Let $X_{E,F}$ denote the event that the events E and F have a corresponding characteristic X with. Two events, E and F, are unlinkable in regard of a characteristic X (e.g., two messages are unlinkable with a subject or with a transaction) for an attacker A, if for each observation B that A can make, the probability that E and F are corresponding in regard of X given B is greater than zero and less than one:

$$0 < P(X_{E,F} \mid B) < 1.$$

A stricter requirement for unlinkability is:

$$0 << P(X_{E,F} \mid B) << 1$$

E and F are perfectly unlinkable if:

$$P\,(X_{E,F} \mid B) = P\,(X_{E,F})$$

As such, unlinkability is related to anonymity but is a more "fine-grained" property; it is a sufficient condition for anonymity, but not a necessary one [58]. The benefits from unlinkability are important for users who do not wish to be tracked in terms of the services and other resources they access, thus minimizing risks that may arise from the correlation of such activities (e.g., to derive PII by combining seemingly non-personal data) and user profiling in general. Nevertheless, as it is the case with anonymity, whereby strong anonymity requires a large anonymity set, strong unlinkability also requires a large unlinkability set. To strengthen unlinkability, often, equal or near-equal distribution of traffic between all potential senders and all potential receivers is pursued (often through the generation of cover "dummy" traffic), which is not always practical and leads to excessive traffic overheads.

Implementation techniques that can be used to provide the unlinkability property are, for the most part, common to those providing anonymity (e.g., Mix networks [46], DC-nets [68], and Onion Routing [69]). Other pertinent tools include various track and evident erasers [49] (e.g., spyware detection and removal tools, browser cleaning tools, activity traces erasers, and hard disk data erasers), as well as identity federation and data fragmentation techniques [18]. Finally, the use of dummies (be it dummy traffic, dummy activity traces, and any other dummy data and actions) can be used to provide unlinkability between a user and their actions [21,57,66].

Mix Network

The concept of Mix networks (often referred to as "Mix-Nets") was introduced by D. Chaum [67] and has had a very significant impact in the area of anonymous networking. It provides sender anonymity against the receiver (optionally recipient anonymity), as well as unlinkability of sender and receiver [66]. Combined with dummy traffic, they can also provide unobservability [58] (more on that in the sections that follow). The popular concept of Onion Routing [69] is based on mixes but adds layered encryption.

The concept, as shown in Figure 20, relies on the existence of a chain of independent mix stations (or "mixes") between senders and receivers. Said stations are responsible for mixing traffic from each user with traffic from other users, collecting and storing the user messages, decrypting and re-encrypting them (to change their appearance), and outputting them in a different order than the one received.

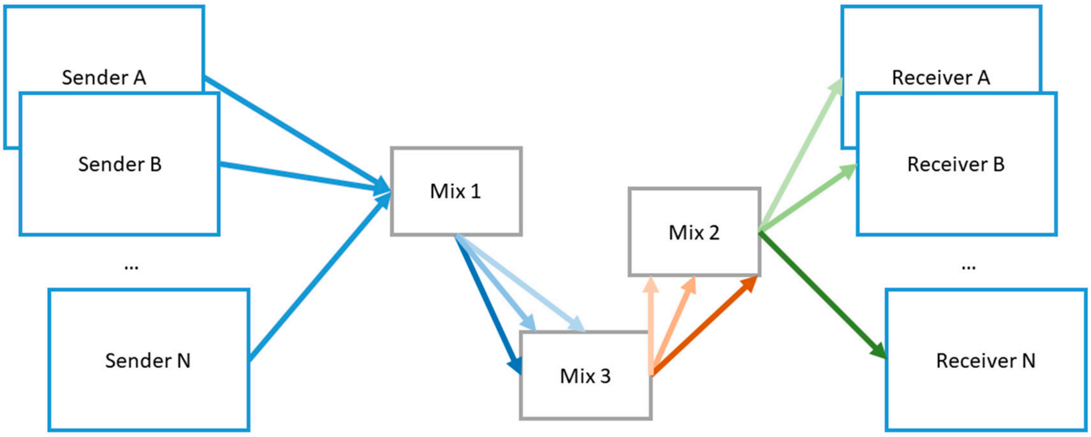


Figure 20. Conceptual overview of Mix Network offering sender anonymity [20].

4.3. Undetectability

Undetectability is "the inability for a third party to distinguish who is the user (among a set of potential users) using a service" [49]. A less user-focused and, thus, more general definition of undetectability is as follows: "Undetectability of an item of interest (IOI) from an attacker's perspective means that the attacker cannot sufficiently distinguish whether it exists or not" [58].

It should be noted that in the privacy-related literature, the undetectability property is not always considered as a standalone aspect (e.g., there is no reference to it in ISO/IEC 15408-2008 [29], ISO/IEC 29100:2011 [26], the Common Criteria for Information Technology Security Evaluation [77], nor in a large part of the pertinent academic works); in fact, it is often merged with unlinkability or unobservability. Nevertheless, as highlighted by A. Pfitzmann's seminal works on privacy terminology and the iterative enrichment of said terminology (e.g., see additions from [58,78] and more recent works from other authors adopting this approach [49]), the differentiation between these terms is clear and needed for a thorough analysis of said aspects under the different attacker models. More specifically, in the case of unlinkability (and anonymity, as well), the focus is on protecting (hiding) the relationship between subjects and other IOIs (other users, services, resources, etc.), while in the case of undetectability, the focus is on protecting the IOIs as such [58]. Unobservability, on the other hand, is a much stronger property, as will be analyzed bellow.

A process pattern for the application of unlinkability and undetectability, providing a structure for the application of lower-level patterns of individual primitives, is shown in Figure 21.

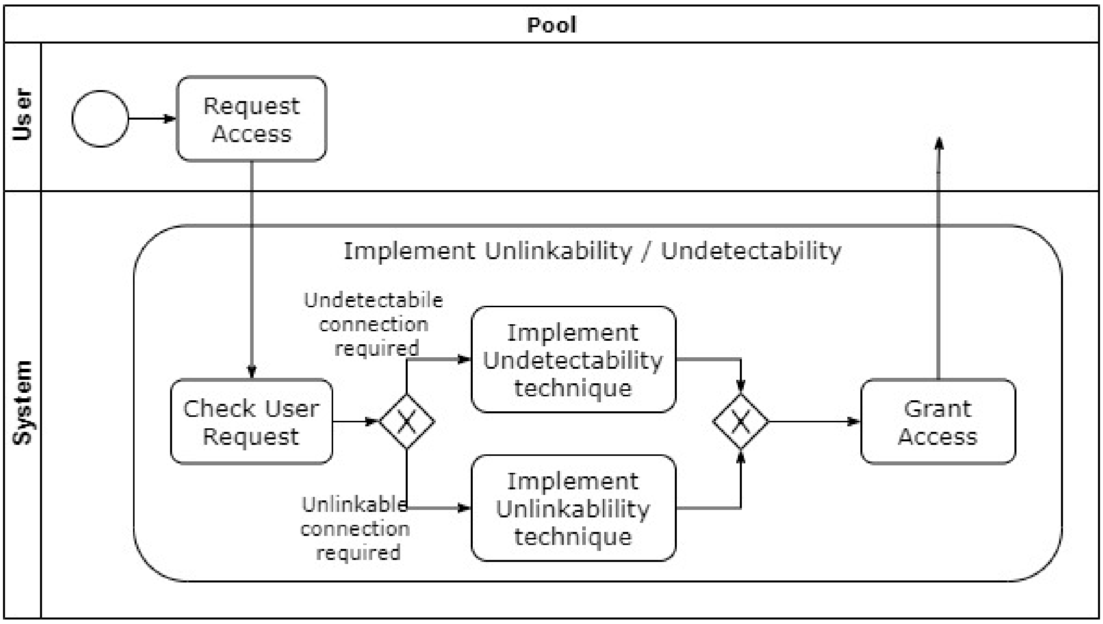

Pool
User
System
Request Access
Implement Unlinkability / Undetectability
Undetectabile connection required
Implement Undetectability technique
Check User Request
Grant Access
Unlinkable connection required
Implement Unlinkablility technique

Figure 21. Conceptual overview of Unlinkability and Undetectability process pattern [49].

Through undetectability, users' privacy is protected, since an attacker (or any third party) cannot detect the IOI; e.g., assuming the IOI is a message, maximal undetectability (or "perfect undetectability") would mean that the message is completely indistinguishable from no message at all. As is the case with other properties defined, the strength of undetectability depends on the number of IOIs belonging to the undetectability set.

The main method of providing undetectability in communication channels is through the adoption of mechanisms to achieve statistical independence of all discernible phenomena at lower layers of the communication infrastructure stacks (i.e., at a lower communication layer) [58]. This can be achieved, for example, by using dummy traffic to maintain a constant flow of messages that look random (typically by leveraging encryption to achieve this randomness) for everyone except the entities communicating. Therefore, the communication between the entities will go undetected by external parties as they will see a random and constant flow of traffic. In this context, the Cover Traffic generation techniques mentioned for the unlinkability property are also applicable. Other well-known techniques for achieving unlinkability include the use of steganography [79], as well as spread-spectrum [80] and spread-time [81] stegosystems.

4.3.1. Cover Traffic

Considering the importance of generating dummy traffic to increase the unlinkability set and negate any attempts for linking users to their actions, a valuable addition in any orchestration requiring unlinkability is one of a proxy generating fake requests and dummy traffic. Such a proxy will typically be placed between the user and the services of interest (Figure 22), ensuring that third parties cannot map specific users to specific requests of access to a service or a resource.

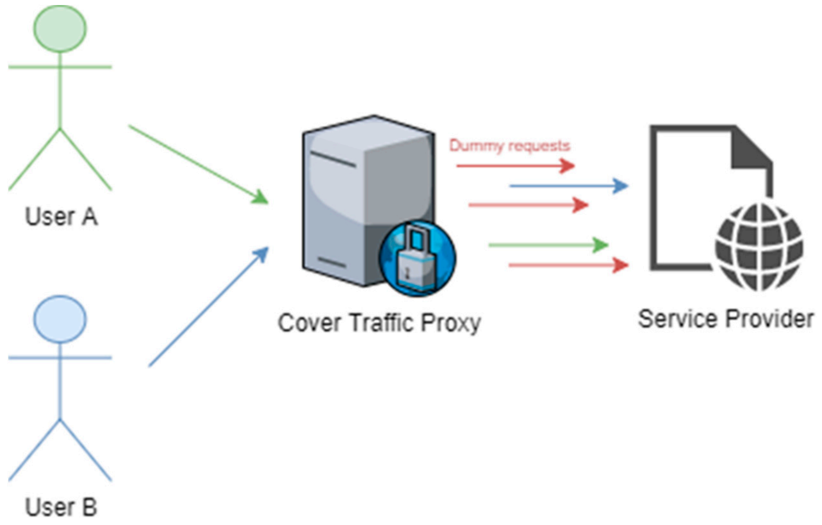


Figure 22. Conceptual overview of Cover Traffic—dummy traffic is generated to increase unlinkability set [21].

A similar strategy can be followed to protect the data at rest—e.g., leveraging dummy files' generation through the technique provided in [82], whereby authors describe a fake online repository generation engine for cyber deception.

4.3.2. Steganography

A key enabler of undetectability is the use of steganography, selecting specific steganographic techniques depending on the medium used for communication. Steganography (from the Greek word "Στεγανογραφία", translated to "covert writing") focuses on techniques allowing undetectable transmission of messages by embedding them into innocuous carriers. These carriers are referred to as "stego-mediums" and they may include text messages, images and video, audio, and others (e.g., using unused space in storage devices or the noise in communication channels). Depending on the stego-medium, different steganographic techniques can be employed.

Therefore, in contrast to cryptography, which is used to hide the contents of a message, steganography focuses on hiding the message itself.

4.4. Pseudonymity

Pseudonymity "ensures that a user may use a resource or service without disclosing its user identity but can still be accountable for that use" (ISO/IEC 15408-2008 [29]). A definition focusing on PII refers to pseudonymization as the "process applied to personally identifiable information (PII) which replaces identifying information with an alias" (ISO/IEC 29100:2011 [26]). Again, a more generic definition is provided by Pfitzmann et al. [58]: "Pseudonymity is the use of pseudonyms as identifiers"; whereby a pseudonym is "an identifier of a subject other than one of the subject's real name".

As such, pseudonymity allows the use of services without disclosing the user's real identity or other identifiable information, while allowing the use of resources that allow the user to be accountable for their actions (i.e., allowing the use of authenticated services, billing, logging, auditing, etc.); this is often referred to as linkability. A process pattern that provides a generic structure for the use of pseudonymity or anonymity (depending on the context and application requirements) is depicted in Figure 23.

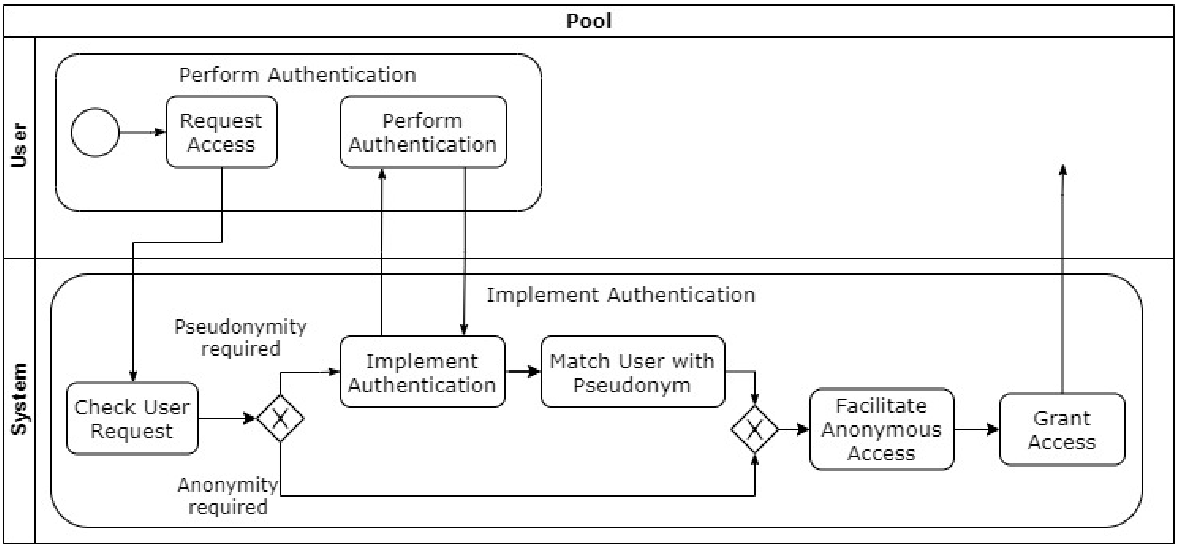


Figure 23. Conceptual overview of Anonymity and Pseudonymity process pattern [49].

It is important to reiterate that while a number of works group pseudonymity and anonymity together (e.g., [37]), the two properties are not equivalent [58]. In fact, pseudonymization contrasts with anonymization, as the latter destroys linkability. This is evident in the context of the handling of PII, as anonymized data are no longer PII (ISO/IEC 29100:2011 [26]).

There are numerous degrees of pseudonymity depending on various factors (number of pseudonyms per subject, guaranteed uniqueness, transferability to other subjects, frequency of changeover, etc. [66,83,84]) and classifications of pseudonyms (public and non-public personal pseudonyms, role-based pseudonyms, transaction pseudonyms, etc. [65]), but again, a full analysis of these is beyond the scope of this paper and we defer the reader to the related works cited herein.

A number of PETs can be leveraged to implement pseudonymization, varying in their features and capabilities depending on the context and application requirements that they were designed to accommodate. The most prominent application of pseudonyms is user-generated public keys encoded on self-signed certificates—e.g., as in the Pretty Good Privacy (PGP) system. Other pseudonymization PETs include the use of browsing pseudonyms, virtual email addresses, or pseudonymous remailers (e.g., Mixminion [85]), trusted third parties (such as Identity Protectors/Brokers), the use of Anonymous Credential Systems [83] (e.g., idemix [86]), Customer Relationship Management (CRM) personalization [87], and authentication methods that can facilitate user registration using pseudonyms (e.g., the use of smart cards or Radio-frequency identification (RFID) cards).

Identity Protector

A key pseudonymity enabler, particularly in applications where the presence of a trusted third party can be assumed, is the deployment of an Identity Broker. An Identity Broker acts as a proxy that handles the linkability between users and external services/resources that they access and the interactions between users. As defined in [58]: "Since anonymity can be described as a particular kind of unlinkability, the concept of identity broker can be generalized to linkability broker. A linkability broker is a (trusted) third party that, adhering to agreed rules, enables linking IOIs for those entities being entitled to get to know the linking".

An implementation of such an Identity Broker is the "Identity Protector" [66], which generates pseudo-identities, handles the translation between real and pseudo-identities, maps pseudo-identities to other pseudo-identities, and controls all instances when the real identity is disclosed. Such a proxy can be installed on a part of the network and create two domains within the information system: (i) an "Identity Domain", where the actual identity of the user(s) is known, and (ii) one or more Pseudo Domain(s), where the real identity is secret and pseudonyms are used. A high-level view of this concept is shown in Figure 24.

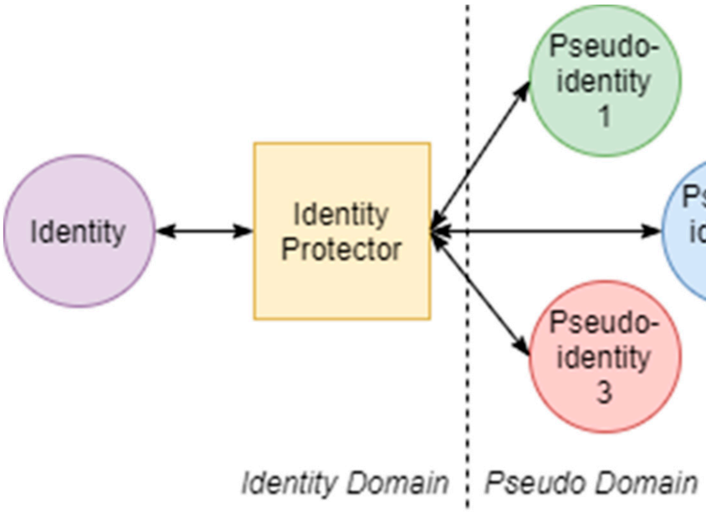


Figure 24. Conceptual overview of Identity Protector separating identity and pseudo domains [66].

4.5. Unobservability

Unobservability "ensures that a user may use a resource or service without others, especially third parties, being able to observe that the resource or service is being used" (ISO/IEC 15408-2008 [29]). Pfitzmann et al. [58] provide a definition that also shows the relationship (reliance) of unobservability to the co-existence of two other "weaker" properties defined above, undetectability and anonymity: "Unobservability of an item of interest (IOI) means undetectability of the IOI against all subjects uninvolved in it and anonymity of the subject(s) involved in the IOI even against the other subject(s) involved in that IOI."

A formal definition of unobservability is as follows (originally defined in [65], adapted in [66]):

An event E is unobservable for an attacker A, if for each observation B that A can make, the probability of E given B is greater zero and less one:

$$0 < P(E \mid B) < 1$$

A stricter requirement, which prevents that the value $P(E \mid B)$ is too close to either 1 or 0, could be:

$$0 << P(E \mid B) << 1$$

If, for each possible observation B that A can make, the probability of an event E is equal to the probability of E given B, that is $P(E) = P(E \mid B)$, then E is called perfectly unobservable.

As such, unobservability can be considered the strongest property of the ones previously defined and requires the provision of these underlying properties in order to be achieved. The unobservability process pattern is depicted in Figure 25, whereby the fact that unobservability implies the existence of anonymity or pseudonymity and undetectability is visualized. Here, it should be clarified that even though the terms of anonymity and pseudonymity are used interchangeably in the figure, as highlighted in [49], the pseudonymity in this context is only acceptable if a certain degree of unlinkability can be provided (e.g., through the use of transaction pseudonyms that offer a high degree of unlinkability).

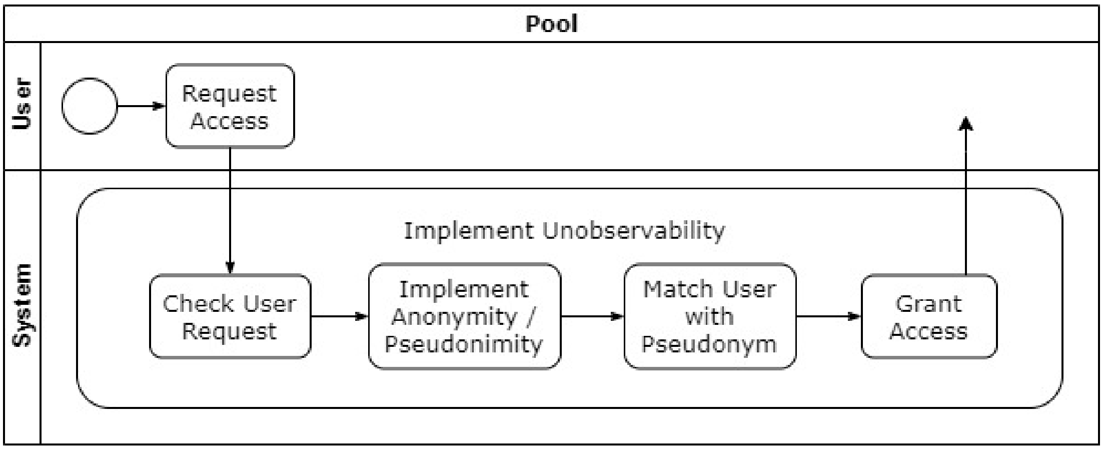

Pool
User
System
Request Access
Implement Unobservability
Check User Request
Implement Anonymity / Pseudonimity
Match User with Pseudonym
Grant Access

Figure 25. Conceptual overview of Unobservability process pattern [49].

5. Collection Overview and Practical Application

Aggregating the patterns presented in this paper, Table 1 presents the coverage that these patterns offer in terms of the considered properties (Security and Privacy), data states (data in transit, at rest, and in process), and platform connectivity cases (within the same IoT platform and across different IoT platforms).

Table 1. Summary of the presented patterns and their coverage.

Pattern		Property		Data State Coverage			Platform Connectivity		Concept or Pattern Source
#	Name	S	P	In Transit	At Rest	In Processing	Within	Across	
1	Overall Security	✓		✓	✓	✓	✓	✓	Known concept
2	Overall Confidentiality	✓	✓	✓	✓	✓	✓	✓	Ritter et al. [28]
3	Encrypted Storage	✓	✓		✓		✓		Kienzle et al. [15]
4	Encrypted Channels	✓	✓	✓			✓	✓	Schumacher et al. [14]
5	Encrypted Processing	✓	✓			✓	✓		Ahituv et al. [31]
6	Overall Integrity	✓	✓	✓	✓	✓	✓	✓	Ritter et al. [28]
7	Safe Storage	✓	✓		✓		✓		Ritter et al. [28]
8	Safe Channel	✓	✓	✓			✓	✓	Ritter et al. [28]
9	Safe Processing	✓	✓			✓	✓		Ritter et al. [28]
10	Hash Check	✓	✓	✓	✓	✓	✓	✓	known concept
11	Server Sandbox	✓	✓	✓	✓	✓	✓		Kienzle et al. [15]
12	Minefield	✓		✓	✓	✓	✓		Kienzle et al. [15]
13	Overall Availability	✓		✓	✓		✓	✓	Avizienis et al. [37]
14	Uptime	✓		✓	✓		✓	✓	Known concept
15	Redundant Storage	✓			✓		✓	✓	Erl et al. [40]
16	Fault Management	✓			✓	✓	✓	✓	Known concept
17	Non-repudiation/ Auditability/Accountability	✓	✓	✓	✓	✓	✓	✓	Ritter et al. [28]
18	Signed Message	✓	✓	✓			✓	✓	Ritter et al. [28]
19	Audit Log	✓	✓	✓	✓	✓	✓		Kienzle et al. [15]
20	Overall Authorization	✓	✓	✓	✓	✓	✓	✓	Schumacher et al. [14]
21	Single Access	✓	✓	✓	✓	✓	✓	✓	Schumacher et al. [14]
22	Authorization Enforcer	✓	✓	✓	✓		✓	✓	Schumacher et al. [14]
23	Overall Authentication	✓	✓	✓	✓	✓	✓	✓	Schumacher et al. [14]
24	Authentication Enforcer	✓	✓	✓	✓	✓	✓	✓	Schumacher et al. [14]
25	Authenticated Channel	✓	✓	✓			✓	✓	Durand et al. [54]
26	Account Lockout	✓	✓	✓	✓	✓	✓	✓	Kienzle et al. [15]
27	Authenticated Session	✓	✓	✓	✓		✓	✓	Kienzle et al. [15]
28	Blacklist	✓	✓	✓			✓	✓	Kienzle et al. [15]
29	Overall Privacy		✓	✓	✓	✓	✓	✓	Pfitzmann et al. [58]

Table 1. *Cont.*

Pattern		Property		Data State Coverage			Platform Connectivity		Concept or Pattern Source
#	Name	S	P	In Transit	At Rest	In Processing	Within	Across	
30	Mix Network		✓	✓				✓	M. Hafiz [21] and S. Fischer-Hübner [66]
31	Pseudonymity		✓	✓	✓	✓	✓	✓	Diamantopoulou et al. [49]
32	Identity Protector		✓	✓	✓	✓	✓	✓	S. Fischer-Hübner [66]
33	Unlinkability		✓	✓	✓			✓	S. Fischer-Hübner [66]
34	Cover Traffic		✓	✓			✓	✓	M. Hafiz [21]
35	Undetectability		✓	✓			✓	✓	Diamantopoulou et al. [49]
36	Steganography		✓	✓			✓	✓	Zöllner et al. [88] and S. Fischer-Hübner [66]
37	Unobservability		✓	✓	✓		✓	✓	Pfitzmann et al. [58] and Diamantopoulou et al. [49]

Data state refers to state of the data of an IoT orchestration placeholder (see Section 5.2). If such a placeholder is an Orchestration, then the state of the data will be "in_transit". If it is bound to a storage service, then the state of the data could also be "at_rest". If it is bound to a service or device that transforms data, then the state of the data could be "in_processing".

Regarding the platform connectivity cases, two distinct cases are taken under consideration: (i) when applications or services access resources from one and only an IoT platform; (ii) when applications or services access resources from multiple platforms though common interfaces.

The categorization of patterns based on these additional properties can further assist in pattern selection (e.g., "select all patterns from the collections that refer to data in transit").

Furthermore, while a full view of the pattern collection in its hierarchical graph form cannot be provided herein due to page size limitations (it would have to be minimized to an extent that it becomes unreadable), the reader may refer to the Appendix A, whereby a full view of said graph is provided. This provides a graphical representation of the pattern collection's taxonomy, including the relationships between the included properties and sub-properties, and is the envisioned means of navigating through the pattern collection.

5.1. Collection Expansion

The pattern collection and classification presented herein can be expanded to include additional security and privacy patterns, e.g., leveraging the breadth of work already existing, as identified in the Section 2.

Nevertheless, of particular interest is also the extension of the collection in terms of properties' coverage, to encompass properties that are related to Security and Privacy, such as Dependability and Interoperability. Other than the significant interplay and dependencies between these properties, they are also extremely pertinent in the context of the IoT/IIoT and the various application domains. That is why these properties are often studied and pursued together, and this is also evident in various research efforts. For example, in addition to the SEMIoTICS project motivating the work presented herein, past research efforts such as the nSHIELD [89] and SERENITY [90] projects also considered the Security, Privacy, Dependability, and Interoperability properties together in the context of embedded systems and ambient intelligence applications. Pointers to such future extensions and related considerations are provided in the sections that follow.

Furthermore, extensions to address additional domains such as smart vehicles or smart agriculture could also be introduced. This would demand the extension of the proposed model to cover the devices and interactions intrinsic to each of the targeted domains. This is not an obstacle and is supported by the (by design) extensible approach that follows: the system model is, by design, extensible and it is trivial to define additional classes and interactions. Any extensions to the model are later transferred to the language, enabling it to support additional expressive constructs as needed. Thus, the language itself is equally volatile and adding more concepts to newer versions of the language can be done easily.

5.1.1. Dependability

Dependability typically refers to the provision of an expected service towards task accomplishment in a reliable and trustworthy manner, and it entails reliability, safety, availability, and security [91]. Nevertheless, the concept of security is covered separately above, and in modern computer engineering, security is considered to encompass availability (along with confidentiality and integrity). Therefore, in the context of this work, dependability properties mainly focus on reliability, fault tolerance, and safety aspects. These aspects, along with their relationship to some enabling patterns, are depicted in Figure 26.

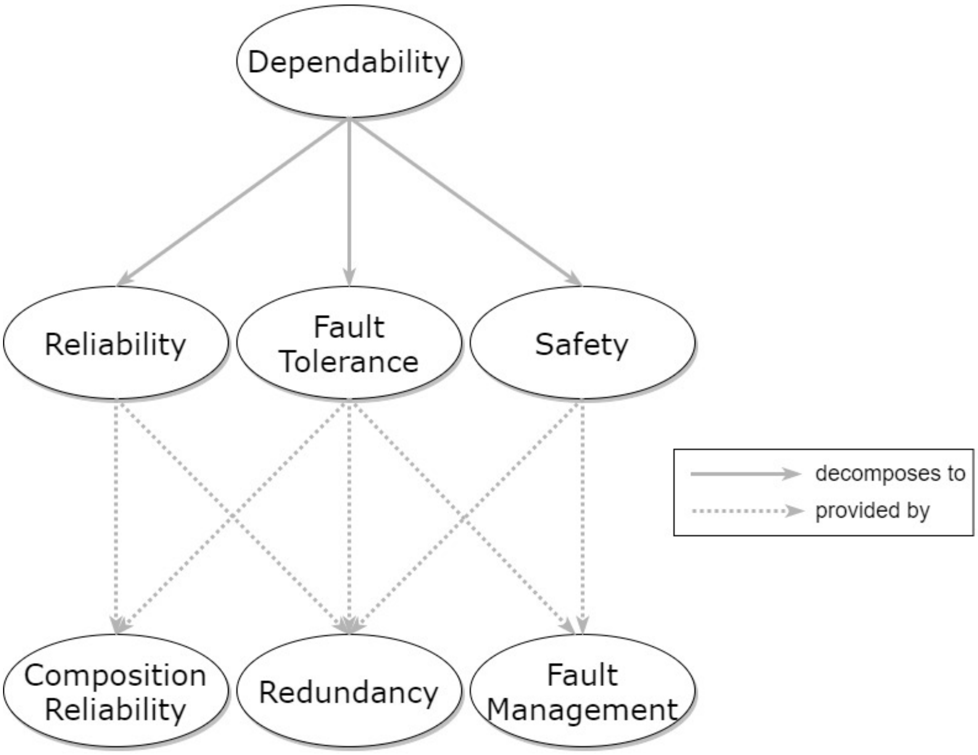


Figure 26. Hierarchical graph of Dependability properties and relevant patterns.

In order to guarantee end-to-end dependability properties for a system, suitable component orchestrations should be found. If they do not exist, the substitution or the addition of current components with other atomic ones or orchestrations is required to guarantee dependability. This is also related to the components and the topology of the composition.

However, it should be noted that the composition of two components which preserve a dependability property does not necessarily guarantee that the composition will also preserve the same property. In addition, if a composition guarantees the conditions of a property, the atomic components may not preserve the property.

Moreover, certain dependability properties will need to hold at the component level to enable the end-to-end properties to be achieved. One of the most important issues for a system designer is to validate the system dependability of components as a critical condition for the design of complex network infrastructures and to identify the weakest components in order to replace, redesign, and find alternative solutions.

5.1.2. Interoperability

Four levels of interoperability are identified and taken into consideration: technological, syntactic, semantic, and organizational interoperability. An approach towards a pattern rule definition for these cases is detailed in the sections that follow.

In more detail, from the bottom up, the following types of interoperability can be distinguished [92]:

- Technical interoperability—enables seamless operation and cooperation of heterogeneous devices that utilize different communication protocols on the transmission layer.
- Syntactic interoperability—establishes clearly defined formats for data, interfaces, and encoding.
- Semantic interoperability—settles commonly agreed information models and ontologies for the used terms that are processed by the interfaces or are included in exchanged data.
- Organizational interoperability—cross-domain and cross-platform service integration and orchestration through common semantic and programming interfaces.

It is important to note that higher levels of interoperability assume the existence of lower ones, otherwise they cannot be achieved—e.g., to have syntactic interoperability, you need to have established technical interoperability first. This is denoted by the "implies" relationship in the resulting graph.

The interoperability properties and their relationships with some enabling patterns that could be further documented are visualized in Figure 27.

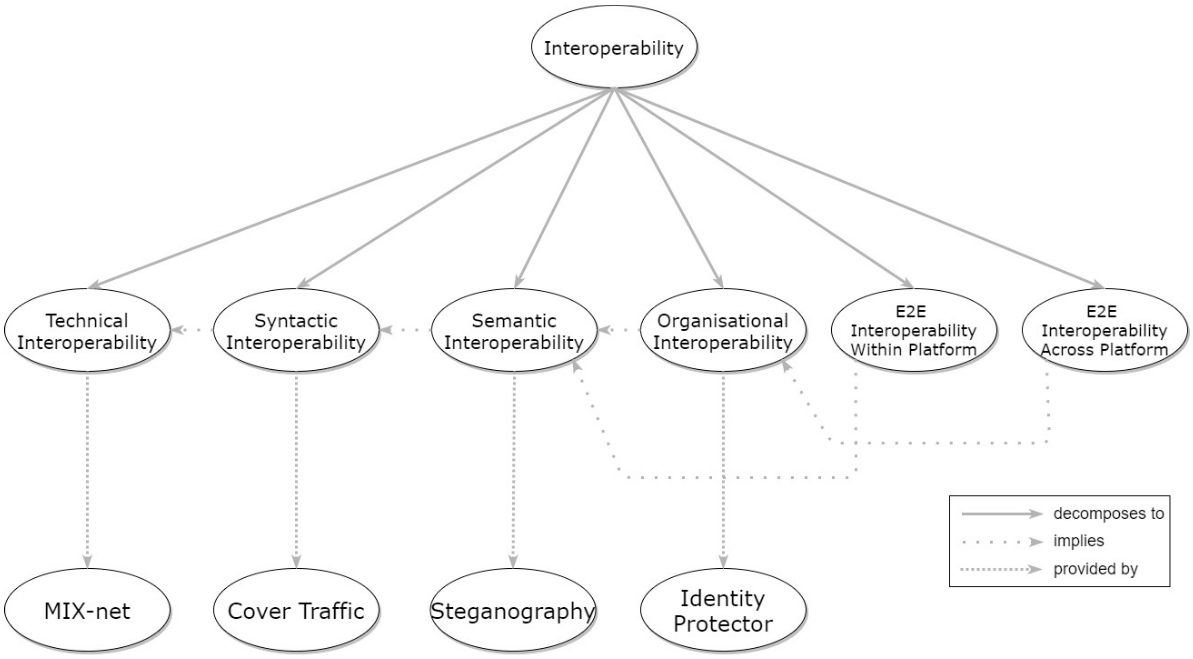

Interoperability
Technical Interoperability
Syntactic Interoperability
Semantic Interoperability
Organisational Interoperability
E2E Interoperability Within Platform
E2E Interoperability Across Platform
MIX-net
Cover Traffic
Steganography
Identity Protector
decomposes to
implies
provided by

Figure 27. Hierarchical graph of Interoperability properties and relevant patterns.

5.2. Application—The SEMIoTICS Approach to Pattern-Driven IoT/IIoT Orchestrations with SPDI Guarantees

Motivated by the significance of overcoming the security, privacy, dependability, and interoperability barriers to the adoption of IoT/IIoT applications and aiming to leverage the potential of patterns in this regard, efforts within project SEMIoTICS focused on the design and development of a pattern-driven approach for composing IoT systems/orchestrations where security properties are guaranteed [93]. The overall objective was to develop a framework that will be capable of managing IoT applications based on patterns and one that allows for (i) verification that an IoT system/orchestration satisfies certain SPDI properties and (ii) the generation of such systems/orchestrations in ways that are guaranteed to satisfy required SPDI properties.

A brief overview of the SEMIoTICS framework's use of patterns will be presented in this section as it is an archetypical implementation that leverages, in practice, the pattern collection presented herein, while also validating its results in three key IoT use cases, covering renewable energy, healthcare, and smart sensing.

To achieve the pattern-driven management objective of the project, the need for the development of a language for specifying the IoT orchestration components along with their interface and interactions became apparent. Using this language, it should also be possible to specify SPDI properties that may be required of such components and their orchestrations. Thus, an IoT orchestration model with such characteristics was defined to form the basis for the language specification. This, in conjunction with the patterns' specification in the same language, allowed us to enable the reasoning required for determining the applicability of particular SPDI patterns in specific IoT orchestrations.

In more detail, in the SEMIoTICS model, the IoT systems/orchestrations are decomposed into individual "placeholders" implementing miscellaneous "activities". SPDI patterns encode proven dependencies between SPDI properties of these placeholders (activity-level properties) and SPDI properties of the IoT systems/orchestrations themselves (orchestration-level properties). The main classes of the model, which are used in the pattern specification, include Placeholder, Orchestration, Orchestration Activity, and Property. Placeholders act as predefined positions for different IoT application components. Figure 28 depicts the attributes of the Placeholder class and its subclasses.

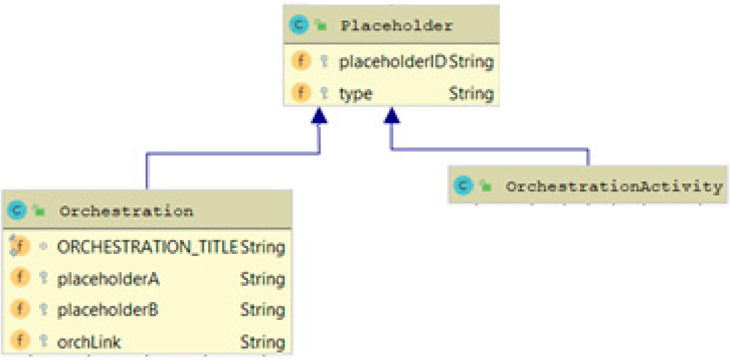


Figure 28. Descriptive overview of Placeholder class and its subclasses.

The components themselves are described with the help of the Orchestration Activities class. Subclasses of Orchestration Activity include IoTSensor, IoTActuator, IoTGateway, SoftwareComponent, NetworkComponent, LinkedActivity, and UnAssignedActivity classes. Each one of those is able to describe the unique characteristics of the corresponding components in the form of attributes. Placeholders of the type UnAssignedActivity make our model parametric, since it does not have to refer to exact placeholders. Figure 29 shows all the subclasses of the OrchestrationActivity class.

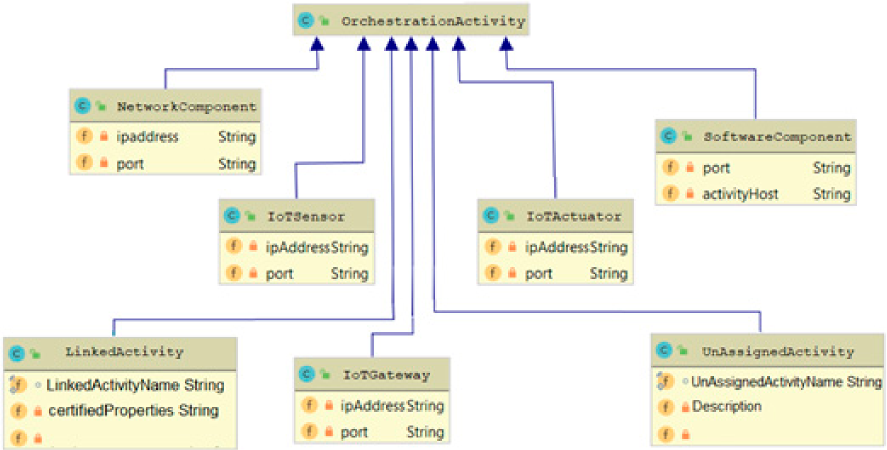


Figure 29. Descriptive overview of OrchestrationActivity class and its subclasses.

Placeholders are orchestrated depending on the order in which the corresponding IoT application components must be executed. As a result, an Orchestration can be defined as Sequence, Merge, Choice, Parallel, or Split (subclasses of the Orchestration class). The meaning of these types of orchestrations is as follows:

1. Sequence is a segment of a process instance in which several activities are executed in sequence under a single thread of execution.
2. Parallel is a segment of a process instance where two or more activity instances are executing in parallel within the workflow, giving rise to multiple threads of control.
3. Merge is a point in the workflow where two or more parallel executing/alternative activities converge into a single common thread of control.
4. Choice is a point within the workflow where a single thread of control makes a decision on which branch to take when encountered with multiple alternative workflow branches, based on a choice condition.
5. Split is a point within the workflow where two or more branches are created and taken. After a Split, the activities of the branches that are taken are executed in parallel.

In Figure 30, the Orchestration class' attributes and its subclasses are depicted.

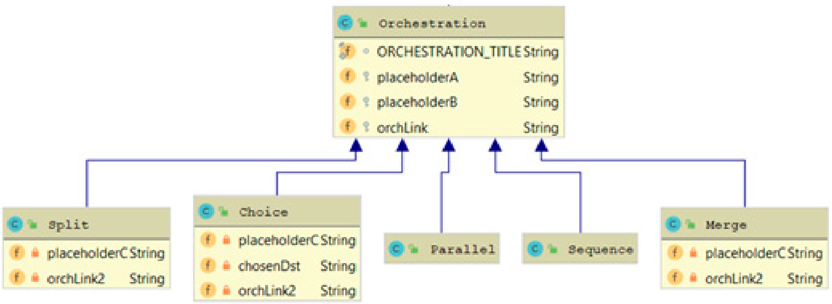


Figure 30. Descriptive overview of Orchestration class and its subclasses.

Finally, Properties characterize placeholders, expressing security and privacy requirements. The state of a property can be required or confirmed. A required property is a property that a placeholder must hold in order to be included (considered for) a specific IoT composition. On the other hand, a confirmed property is a property that is verified at runtime through predefined means. Figure 31 shows all the attributes of the Property class along with the Verification class, which is dedicated for the verification of each property. Verification is a class that describes the way a property of a placeholder is verified. The verification process can be conducted through monitoring, testing, a certificate, or via a pattern. This means that the existence of a monitoring service or a testing tool allows the verification of the SPDI property of a placeholder activity. Such a monitoring service could, for example, justify that a service or a device is available at specific time windows if the desirable property is a specific target for availability. Another way of verifying SPDI properties could be a repository with certificates that are able to justify that a certain placeholder satisfies a certain property. In case of a pattern, the Mean of verification is the pattern itself; in all the other cases, we need an interface to a corresponding monitoring tool, testing service, or certificate repository through which the verification can take place.

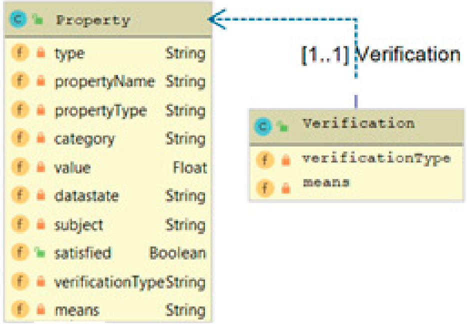


Figure 31. Descriptive overview of Property class.

From an implementation perspective, the presented model was derived using the Eclipse Modeling Framework (EMF (https://www.eclipse.org/modeling/emf/ (accessed on 2 December 2020))), visualizing the Ecore part of the EMF metamodel, which contains the information about the defined classes. A detailed representation of the pattern language and model is presented in our previous work [9,10].

Based on the aforementioned IoT orchestration model, we created a corresponding language, the constructs of which are described using Extended Backus–Naur Form (EBNF (https://tomassetti.me/ebnf/ (accessed on 2 December 2020))) grammar. EBNF is considered a metalanguage that is used to specify the grammar for a language with precise structure. Using this language, we can define activities as well as basic control flow operations (namely sequential, parallel, choice, and merge), enabling their composition into complex orchestrations, and define the associated individual and composition properties. Upon instantiation of the orchestration, the abstract definition of the orchestration structure is replaced with the actual components implementing said orchestration. The produced pattern language (i) provides constructs for expressing/encoding dependencies between SPDI properties at the component and at the composition/orchestration level; (ii) is structural, without the need to prescribe exactly how the functions should be executed (nor, e.g., how the ports ensure communication); (iii) supports the static and dynamic verification of SP properties, and (iv) is automatically processable so that IoT applications can be adapted at runtime. For the sake of brevity, an excerpt of the EBNF grammar of the pattern language is presented in Box 1. What this excerpt depicts is the different types of orchestrations that the IoT orchestration model allows for. Additionally, a lexer and a parser have been created utilizing the Eclipse ANTLR4 plugin (https://www.antlr.org/tools.html (accessed on 2 December 2020)); ANother Tool for Language Recognition (ANTLR) is a parser generator that allows for reading, processing, and executing structured text and binary files. The ANTLR4 lexer recognizes keywords in any input created with the pattern language and transforms them into tokens. The created tokens are used by the ANTLR4 parser for creating logical structure, i.e., the parse tree. In this way, any input can be checked for compliance with the defined grammar.

Box 1. An excerpt of the SEMIoTICS language Extended Backus–Naur Form (EBNF) grammar.

```
grammar EBNF;
 ...
orchestration : sequence | parallel | choice | merge | split ;
 ...
```

Furthermore, a formal way to define patterns is needed. In this context, the specification of an SPDI pattern consists of four parts:

1. The Activity Properties (AP) part that represents the SPDI properties of the individual placeholders of a system/orchestration.
2. The Orchestration (ORCH) part that represents the abstract form of the orchestration that the pattern applies to.
3. The Conditions part that describes requirements, constraints, and reactions of the system/orchestration to specific inputs.
4. The Orchestration Properties (OP) part that represents the orchestration-level SPDI properties that the pattern can guarantee for the orchestration specified in the ORCH part.

Consequently, an SPDI pattern with the above structure can be interpreted as follows: If the AP properties of the activity placeholders in the orchestration of the pattern and the conditions of the pattern hold, then the OP property specified in the pattern also holds for the whole ORCH. Alternatively, in the form of a formula, such a pattern be expressed as:

$$\text{AP} \bigwedge \text{ORCH} \bigwedge \text{CONDITIONS} \models \text{OP}$$

where $\models$ denotes the entailment relation that has been established by the proof of the pattern. APs are materialized using the Property class described above. Property name uniquely identifies the SPDI property and the Property Subject depicts the placeholder that implements the activity for which the property is required or verifiable (PropertyType). In the latter case, PropertyVerification depicts how the verification takes place. PropertyCategory classifies the SPDI property, while DataState show the state of the data used by the placeholder. ORCH is an Orchestration object including placeholders of the type UnassignedActivity, making our model parametric since it does not have to refer to exact placeholders. Conditions are materialized using the Operation and Parameters classes. Inputs and outputs of the activity placeholders of the SPDI pattern are defined in the objects of those two classes. Finally, OP is an orchestration-wide Property object.

One of the goals of the SEMIoTICS approach is the support of automated processing of defined SPDI patterns, to enable autonomic and semi-autonomic behavior in IoT/IIoT ecosystems with SPDI guarantees. Towards this goal, and to achieve the machine processable-pattern specification, SPDI patterns are expressed as Drools business production rules (https://www.drools.org/ (accessed on 2 December 2020)), inserted in the associated Drools rule engine, making use of the Rete algorithm [94]. Drools is a business rules management system (BRMS) solution and allows for the construction, maintenance, and enforcement of business policies in an organization, application, or service. Such a Drools production rule has the following generic structure:

rule name <attributes>*

when <conditional element>*

then <action>* **end**

The **when** part of the rule specifies a set of conditions, and the part **then** of the rule, a list of actions. When a rule is applied, the Drools rule engine checks whether the rule conditions (defined within the <conditional element> above) match with the facts in the Drools Knowledge Base (KB), and if they do, it executes the actions (i.e., "<action>") of the rule. Rule actions are typically used to modify the KB by inserting, retracting, or updating the objects (facts) in it through the standard Drools actions “insert”, “retract”, and “update”, respectively. The conditions of a rule are expressed as patterns of objects that encode the facts in the Drools KB. These patterns define object types and constraints for the data encoded in objects, which may be atomic or complex. Complex Drool object constraints are defined through logical operators (e.g., and, or, not, exists, forall, contains).

The Dependability pattern will be used as an example to showcase how a pattern is translated to a Drools rule; this pattern and its relationship to enabling patterns has been shown in Figure 26, depicting the Dependability property being decomposed to Reliability, Fault Tolerance, and Safety properties. The corresponding Drools rule is shown in Box 2.

Box 2. Dependability pattern in the form of Drools rule.

```
rule “Dependability Pattern”
when
Placeholder($sensor1:=placeholderid)
Placeholder($sensor2:=placeholderid)
Merge($merge1:=placeholderid,$sensor1:=placeholdera,$sensor2:=placeholderb)
$PR: Property ($merge1:=subject, category==“Dependability”, satisfied==false)
then
Property s1Property = new Property();
s1Property.setCategory(“Reliability”);
s1Property.setSubject($merge1);
s1Property.setSatisfied(false);
insert(s1Property);

Property s2Property = new Property();
s2Property.setCategory(“FaultTolerance”);
s2Property.setSubject($merge1);
s2Property.setSatisfied(false);
insert(s2Property);

Property s3Property = new Property();
s3Property.setCategory(“Safety”);
s3Property.setSubject($merge1);
s3Property.setSatisfied(false);
insert(s3Property);
end
```

The orchestration (ORCH) that is chosen here is a merge between two placeholders, as can be seen in the first three lines of the **when** part of the rule. The next line declares the OP property of the pattern in question and concludes the **when** part. In the **then** part, three new properties are created (Reliability, Fault Tolerance, and Safety), which correspond to the AP properties of the pattern. Figure 32 shows graphically what is happening when a Drools rule, as the one above, is triggered. In this figure, placeholders are depicted as sensors that merge into a gateway, and after that, the whole system communicates with the rest of the world (Cloud). The oval shapes depict properties that apply to the system parts that they enclose. Consequently, the Dependability pattern in Section 5.1.1 can be interpreted as follows: If the Reliability, Fault Tolerance, and Safety properties of the orchestration hold, then the Dependability property specified in the pattern also holds for the whole ORCH.

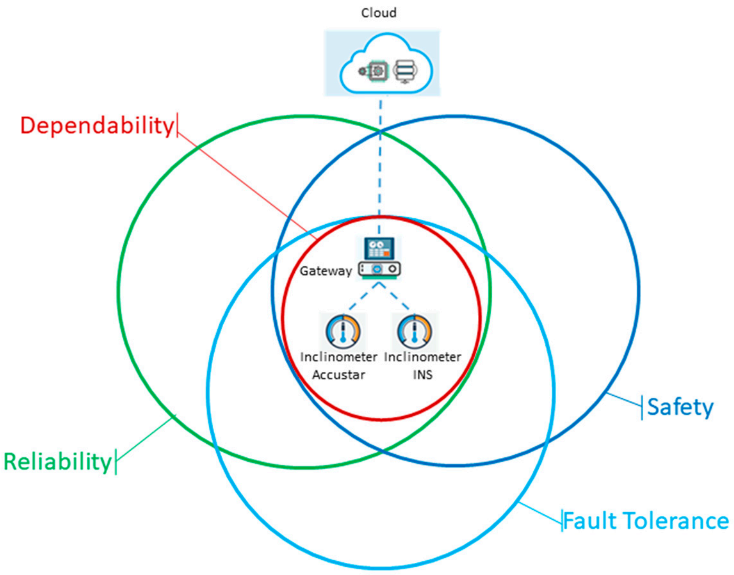


Figure 32. Conceptual view of Dependability pattern—If the Reliability, Fault Tolerance, and Safety properties of the orchestration hold, then the Dependability property specified in the pattern also holds.

The Drools business rule management system functionality was implemented by a set of reasoning agents, referred to as Pattern Engines. An implementation of the described approach has been presented in [95], where the mechanisms for automated translation of IoT orchestrations into network configurations, as well as a mechanism for the monitoring of these configurations, are described. Definition of the IoT orchestrations is performed by an extended version of the Distributed Node-RED (https://nodered.org/ (accessed on 2 December 2020)) (DNR) application, which allows for user-friendly specification of IoT orchestrations, along with definition of the desired SPDI properties on said orchestrations. Node-RED is a programming tool for wiring together hardware devices, APIs, and online services, offering a browser-based flow editor that makes it easy to wire together flows using a wide range of predefined nodes. The created flows can then be deployed at runtime.

Following deployment, the created IoT orchestrations are communicated to the Pattern Engines, transformed into the pattern language, and are then inserted into knowledge sessions of Drools Engine as Drools facts. These Drools facts are used by Drools rules, which are fired when their conditions are met. Upon the arrival of a Drools fact, a new Knowledge Is Everything (KIE) session is created. This session is used for insertion of the Drools fact into the working memory of the Drools Rule Engine. Drools Rules are contained in the RuleBase, ready to be used. Such rules preexist in the Pattern Engine or can be sent to it. Drools facts are used to satisfy the "when" part of the Drools Rules (conditional elements) aiming to execute a rule (action).

6. Conclusions and Future Work

This paper presented a collection of SP patterns, classifying patterns based on the provided property, context, and generality and considering the relationships between the properties. The provided patterns include adaptations of common concepts taken from relevant literature as well as original patterns, creating a comprehensive overview of the associated security and privacy properties that are of interest in IoT/IIoT environments. Moreover, the coverage that these patterns offer in terms of the considered properties, data states (data in transit, at rest, and in process), and platform connectivity cases (within the same IoT platform and across different IoT platforms) was also highlighted.

In addition to the above, the manuscript included pointers to extensions to cover Dependability and Interoperability properties (leading to "SPDI" coverage), along with an overview of how these patterns are practically leveraged in the SEMIoTICS research project towards a framework that provides pattern-driven SPDI management of IoT/IIoT applications. This approach allows for the verification that an IoT orchestration satisfies certain SPDI properties and the generation (and adaptation) of orchestrations in ways that are guaranteed to satisfy the required SPDI properties.

The overarching objective of these efforts is to facilitate the development of a usable way to adopt relevant patterns in the IoT/IIoT domain. By encoding proven dependencies between SP properties of individual smart objects and corresponding properties of orchestrations (compositions) involving them, the development of IoT/IIoT applications and services that are secure and privacy-aware by design is facilitated, as showcased through the SEMIoTICS project.

In terms of next steps, there are two main focus areas: (i) maintenance and expansion of the collection, and (ii) enhancement of its practicality/usability. Regarding (i), a key concern will be the expansion of the collection, both in depth (i.e., number of patterns) as well as in breadth (i.e., coverage of additional properties and specification of relationships between said properties). Furthermore, the authors plan to provide the collection in an open online repository to which interested parties can refer, navigating through the collection using the tree form hierarchy and retrieving their patterns of interest. In terms of (ii), the focus will be on the design and implementation of a mechanism allowing access to the presented patterns and their hierarchical categorization in an automated manner, allowing to select patterns depending on the corresponding needs. This will enable different types of automated IoT orchestration adaptations at design time (e.g., recommending alternate orchestrations to system designers) and at run time (e.g., replacing devices) when a desired SPDI property is not satisfied.

Finally, the above will be validated in practice in the context of real IIoT environments, within the C4IIoT research project (Cyber security 4.0: protecting the Industrial Internet Of Things; https://www.c4iiot.eu/ (accessed on 2 December 2020)), towards a novel and unified Cybersecurity 4.0 framework that provides an end-to-end holistic and disruptive security-enabling solution for minimizing the attack surfaces in Industrial IoT systems.

Author Contributions: Conceptualization, M.P., K.F., G.S., S.I. and K.K.; methodology, G.S. and S.I.; software, M.P.; validation, K.F. and K.K.; formal analysis, G.S., K.F. and S.I.; investigation, M.P. and K.F.; data curation, M.P., K.F. and K.K.; writing—original draft preparation, M.P. and K.F.; writing—review and editing, G.S., S.I. and K.K.; visualization, M.P. and K.F.; supervision, G.S. and S.I.; funding acquisition, G.S. and S.I. All authors have read and agreed to the published version of the manuscript.

Funding: This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780315 (SEMIoTICS), and the European Union's Horizon 2020 innovation programme under grant agreement No. 833828 (C4IIoT)

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Full View of Pattern Collection Graph

A full view of the Security and Privacy patterns collection (including all properties, sub-properties, and their relationships) is provided in Figure A1 below.

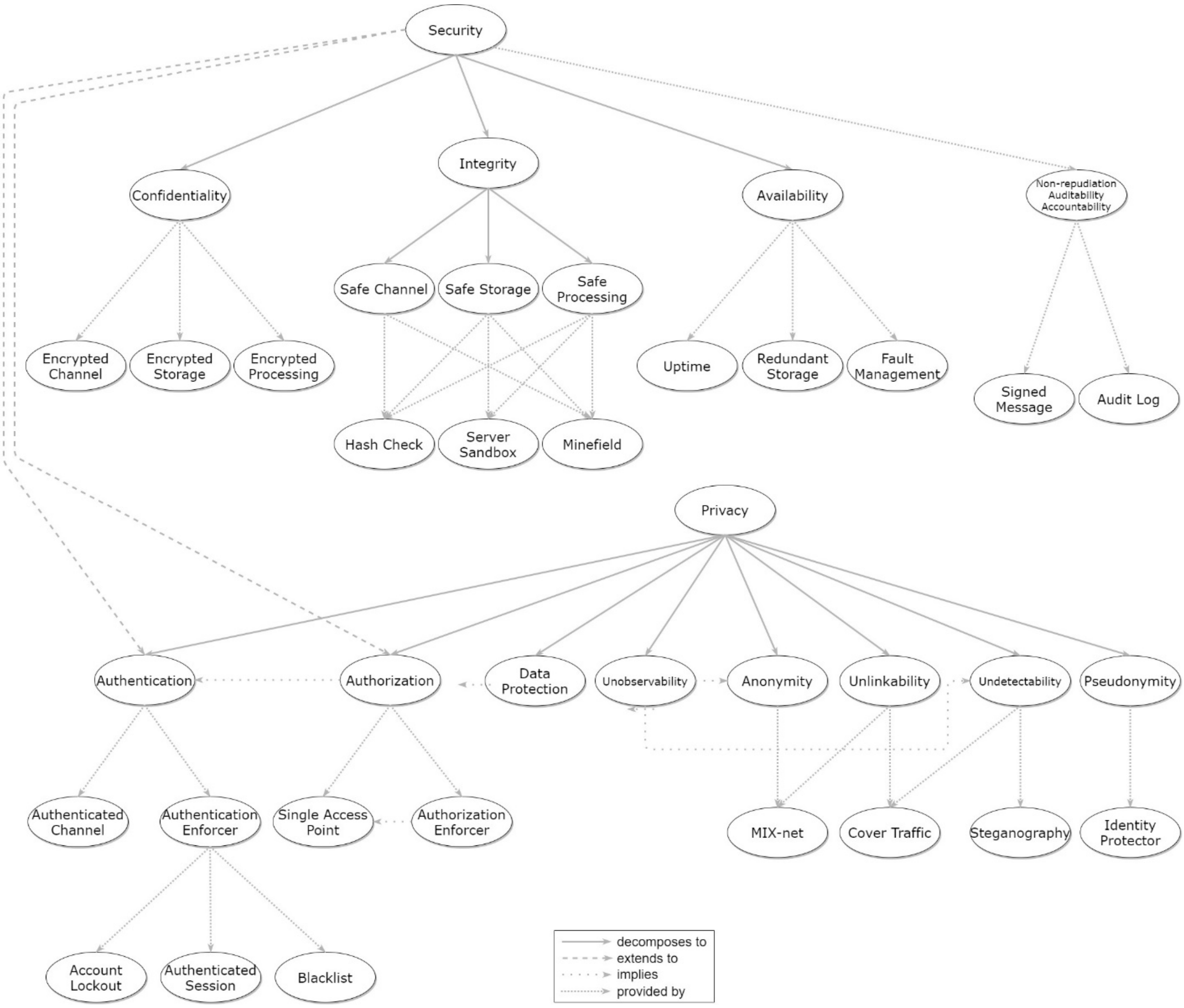


Figure A1. The Security and Privacy patterns collection.

References

1. Tawalbeh, L.A.; Muheidat, F.; Tawalbeh, M.; Quwaider, M. IoT Privacy and security: Challenges and solutions. *Appl. Sci.* **2020**, *10*, 4102. [CrossRef]
2. Botta, A.; De Donato, W.; Persico, V.; Pescapé, A. Integration of cloud computing and internet of things: A survey. In *Future Generation Computer Systems*; Elsevier: Amsterdam, The Netherlands, 2016; Volume 56, pp. 684–700.
3. Alaba, F.A.; Othman, M.; Hashem, I.A.T.; Alotaibi, F. Internet of Things security: A survey. *J. Netw. Comput. Appl.* **2017**, *88*, 10–28. [CrossRef]
4. Khan, M.A.; Salah, K. IoT security: Review, blockchain solutions, and open challenges. *Future Gener. Comput. Syst.* **2018**, *82*, 395–411. [CrossRef]
5. Petroulakis, N.; Lakka, E.; Sakic, E.; Kulkarni, V.; Fysarakis, K.; Somarakis, I.; Serra, J.; Sanabria-Russo, L.; Pau, D.; Falchetto, M.; et al. *SEMIoTICS Architectural Framework: End-to-end Security, Connectivity and Interoperability for Industrial IoT*; Global IoT Summit (GIoTS'19): Aarhus, Denmark, 2019.
6. Abowd, G.D.; Allen, R.; Garlan, D. Formalizing style to understand descriptions of software architecture. *ACM Trans. Softw. Eng. Methodol. TOSEM* **1995**, *4*, 319–364. [CrossRef]
7. Schumacher, M. *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2003; p. 2754.
8. Alexander, C. *The Timeless Way of Building*; Oxford University Press: Oxford, UK, 1979. [CrossRef]
9. Fysarakis, K.; Papoutsakis, M.; Petroulakis, N.; Spanoudakis, G. Towards IoT Orchestrations with Security, Privacy, Dependability and Interoperability Guarantees. In Proceedings of the IEEE Global Communications Conference (GLOBECOM 2019), Waikoloa, HI, USA, 9–13 December 2019.

10. Fysarakis, K.; Spanoudakis, G.; Petroulakis, N.; Soultatos, O.; Bröring, A.; Marktscheffel, T. *Architectural Patterns for Secure IoT Orchestrations*; Global IoT Summit (GIoTS): Aarhus, Denmark, 2019; pp. 1–6. [CrossRef]
11. Soultatos, O.; Papoutsakis, M.; Fysarakis, K.; Hatzivasilis, G.; Michalodimitrakis, M.; Spanoudakis, G.; Ioannidis, S. Pattern-driven Security, Privacy, Dependability and Interoperability management of IoT environments. In Proceedings of the 24th IEEE International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks (CAMAD 2019), Limassol, Cyprus, 11–13 September 2019.
12. Hafiz, M.; Adamczyk, P.; Johnson, R.E. Towards an Organization of Security Patterns. *IEEE Softw. Spec. Issue Softw. Patterns* **2007**, *24*, 52–60.
13. Steel, C.; Nagappan, R.; Lai, R. *Core Security Patterns: Best Practices and Strategies for J2EE(TM), Web Services, and Identity Management*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2005.
14. Schumacher, M.; Fernandez-Buglioni, E.; Hybertson, D.; Buschmann, F.; Sommerlad, P. *Security Patterns: Integrating Security and Systems Engineering*; John Wiley: Hoboken, NJ, USA, 2006.
15. Kienzle, D.M.; Elder, M.C.; Tyree, D.; Edwards-Hewitt, J. *Security Patterns Repository Version 1.0*; DARPA: Washington, DC, USA, 2002.
16. Hogg, J.; Smith, D.; Chong, F.; Taylor, D.; Wall, L.; Slater, P. *Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0*; Microsoft Press: Redmond, WA, USA, 2006.
17. Lenhard, J.; Fritsch, L.; Herold, S. A Literature Study on Privacy Patterns Research. In Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, Austria, 30 August–1 September 2017; pp. 194–201. [CrossRef]
18. Gabel, A.; Schiering, I. Privacy Patterns for Pseudonymity. *IFIP Adv. Inf. Commun. Technol.* **2019**, 155–172.
19. Chung, E.S.; Hong, J.I.; James, L.; Prabaker, M.K.; Landay, J.A.; Liu, A.L. Development and evaluation of emerging design patterns for ubiquitous computing. In Proceedings of the 5th conference on Designing interactive systems: Processes, practices, methods, and techniques, Cambridge, MA, USA, 1–4 August 2004; pp. 233–242. [CrossRef]
20. Hafiz, M. A collection of privacy design patterns. In Proceedings of the 2006 conference on Pattern languages of programs—PLoP '06, Portland, OR, USA, 21–23 October 2006; p. 1. [CrossRef]
21. Hafiz, M. A pattern language for developing privacy enhancing technologies. *Softw. Pract. Exp.* **2013**, *43*, 769–787. [CrossRef]
22. Drozd, O. Privacy pattern catalogue: A tool for integrating privacy principles of ISO/IEC 29100 into the software development process. In *IFIP International Summer School on Privacy and Identity Management*; Springer: Cham, Switzerland, 2015; pp. 129–140.
23. Schümmer, T. The public privacy—Patterns for filtering personal information in collaborative systems. *Proc. CHI Work Hum. Comput. Hum. Interact. Patterns* **2004**, *1963*, 1–35.
24. Schumacher, M. Security Patterns and Security Standards—With Selected Security Patterns for Anonymity and Privacy. In Proceedings of the European Conference on Pattern Languages of Programms (EuroPLoP '2003), Irsee, Germany, 25–29 June 2003.
25. Graf, C.; Wolkerstorfer, P.; Geven, A.; Tscheligi, M. A Pattern Collection for Privacy Enhancing Technology. In Proceedings of the Second International Conference on Pervasive Patterns and Applications, Lisbon, Portugal, 21–26 November 2010.
26. ISO/IEC 29100:2011. Information Technology—Security Techniques—Privacy Framework. 2011. Available online: https://www.iso.org/obp/ui/#iso:std:iso-iec:29100:ed-1:v1:en (accessed on 2 December 2020).
27. Stallings, W.; Brown, L.; Bauer, M.D.; Bhattacharjee, A.K. Computer security: Principles and practice. In *Upper Saddle River*; Pearson Education: Upper Saddle River, NJ, USA, 2012; p. 978.
28. Ritter, D.; Rinderle-Ma, S. Towards a Collection of Cloud Integration Patterns. *arXiv* **2015**, arXiv:1511.09250.
29. ISO/IEC 15408-2:2008. *Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 2: Security Functional Components*; ISO: Geneva, Switzerland, 2008.
30. Denning, D.E. A lattice model of secure information flow. *Commun. ACM* **1976**, *19*, 236–243. [CrossRef]
31. Ahituv, N.; Lapid, Y.; Neumann, S. Processing encrypted data. *Commun. ACM* **1987**, *30*, 777–780. [CrossRef]
32. Ding, W.; Yan, Z.; Deng, R.H. Encrypted data processing with Homomorphic Re-Encryption. *Inf. Sci.* **2017**, *409–410*, 35–55. [CrossRef]
33. Lindell, Y. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*; IGI Global: Hershey, PA, USA, 2005; pp. 1005–1009.
34. Bogetoft, P.; Christensen, D.L.; Damgård, I.; Geisler, M.; Jakobsen, T.; Krøigaard, M.; Toft, T. Secure multiparty computation goes live. In Proceedings of the International Conference on Financial Cryptography and Data Security; Springer: Berlin, Heidelberg; pp. 325–343.
35. Cohen, F.; Lambert, D.; Preston, C.; Berry, N.; Stewart, C.; Thomas, E. A framework for deception. In *National Security Issues in Science, Law, and Technology*; Taylor and Francis: London, UK, 2001.
36. Albanese, M.; Erbacher, R.F.; Jajodia, S.; Molinaro, C.; Persia, F.; Picariello, A.; Subrahmanian, V.S. Recognizing unexplained behavior in network traffic. In *Network Science and Cybersecurity*; Springer: New York, NY, USA, 2014; pp. 39–62.
37. Avizienis, A.; Laprie, J.C.; Randell, B. Fundamental Concepts of Dependability. LAAS-CNRS: Tech. Rep. N01145. 2001. Available online: http://www.cs.cmu.edu/~{}garlan/17811/Readings/avizienis01_fund_concp_depend.pdf (accessed on 2 December 2020).
38. Ahluwalia, K.S.; Jain, A. High availability design patterns. In Proceedings of the 2006 Conference on Pattern Languages of Programs—PLoP '06, Portland, OR, USA, 21–23 October 2006; p. 1. [CrossRef]

39. Thatcher, J.; Flynn, D.; Aune, J.; Fillingim, J.; Inskeep, B.; Strasser, J.; Vigor, K. U.S. Patent No. 8,281,227, 2012. Available online: https://www.uspto.gov/web/offices/com/sol/og/2012/week33/TOC.htm (accessed on 2 December 2020).
40. Erl, T.; Cope, R.; Naserpour, A. *Cloud Computing Design Patterns*; Prentice Hall Press: Upper Saddle River, NJ, USA, 2015.
41. Gardiner, J.L.; Heider, G.K.; Emlich, L.W.; Luhrs, B.H.; Li, M.C.; Masters, M.R.; Seger, M.J. Apparatus, System, and Method to Increase Data Integrity in a Redundant Storage System. U.S. Patent No. 5,740,357, 2 October 2012.
42. Gürer, D.W.; Khan, I.; Ogier, R.; Keffer, R. An artificial intelligence approach to network fault management. *Sri Int.* **1996**, *86*, 1–10.
43. Paradis, L.; Han, Q. A survey of fault management in wireless sensor networks. *J. Netw. Syst. Manag.* **2007**, *15*, 171–190. [CrossRef]
44. Dos Santos, A.L.; Duarte, E.P.; Keeni, G.M. Reliable distributed network management by replication. *J. Netw. Syst. Manag.* **2004**, *12*, 191–213. [CrossRef]
45. Du, X. Identifying control and management plane poison message failure by k-nearest neighbor method. *J. Netw. Syst. Manag.* **2006**, *14*, 243–259. [CrossRef]
46. Hong, J.W.; Park, S.; Kang, Y.; Park, J. Enterprise network traffic monitoring, analysis, and reporting using web technology. *J. Netw. Syst. Manag.* **2001**, *9*, 89–111. [CrossRef]
47. Lutfiyya, H.L.; Bauer, M.A.; Marshall, A.D.; Stokes, D.K. Fault management in distributed systems: A policy-driven approach. *J. Netw. Syst. Manag.* **2000**, *8*, 499–525. [CrossRef]
48. ISO/IEC 27000:2018. *Information Technology—Security Techniques—Information Security Management Systems—Overview and Vocabulary*; ISO: Geneva, Switzerland, 2018.
49. Diamantopoulou, V.; Argyropoulos, N.; Kalloniatis, C.; Gritzalis, S. Supporting the design of privacy-aware business processes via privacy process patterns. In Proceedings of the International Conference on Research Challenges in Information Science, Brighton, UK, 10–12 May 2017. [CrossRef]
50. Rantos, K.; Papanikolaou, A.; Fysarakis, K.; Manifavas, C. Secure policy-based management solutions in heterogeneous embedded systems networks. In Proceedings of the International Conference on Telecommunications and Multimedia (TEMU), Heraklion, Crete, Greece, 30 July–1 August 2012; pp. 227–232. [CrossRef]
51. Zakinthinos, A.; Lee, E.S. A general theory of security properties. In Proceedings of the Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097), Oakland, CA, USA, 4–7 May 1997; pp. 94–102. [CrossRef]
52. X.509 Digital Certification—Microsoft. Available online: https://docs.microsoft.com/en-us/windows/win32/seccrypto/x-509-digital-certification (accessed on 10 December 2020).
53. Kerberos Authentication Overview—Microsoft. Available online: https://docs.microsoft.com/en-us/windows-server/security/kerberos/kerberos-authentication-overview (accessed on 10 December 2020).
54. Durand, A.; Andreaux, J.P.; Sirvent, T. Vague Feelings or An MBA Framework Fact? U.S. Patent No. 7,545,932, 9 June 2009.
55. Two-Way Authentication—IBM. Available online: https://www.ibm.com/support/knowledgecenter/SSRMWJ_7.0.1/com.ibm.isim.doc/securing/cpt/cpt_ic_security_ssl_scenario.htm (accessed on 1 December 2020).
56. European Union. Regulation 2016/679 of the European parliament and the Council of the European Union. *Off. J. Eur. Communities* **1995**, *2014*, 1–88.
57. Maidi, M. The common fragment of CTL and LTL. *Found. Comput. Sci.* **2000**, 643–652. [CrossRef]
58. Pfitzmann, A.; Hansen, M. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. *Tech. Univ. Dresden* **2010**.
59. Caiza, J.C.; Martín, Y.S.; Del Alamo, J.M.; Guamán, D.S. Organizing design patterns for privacy: A taxonomy of types of relationships. In *ACM International Conference Proceeding Series*; Association for Computing Machinery: New York, NY, USA, 2017. [CrossRef]
60. Kalloniatis, C.; Kavakli, E.; Gritzalis, S. Addressing privacy requirements in system design: The PriS method. *Requir. Eng.* **2008**, *13*, 241–255. [CrossRef]
61. Kuhn, C.; Beck, M.; Schiffner, S.; Jorswieck, E.; Strufe, T. On Privacy Notions in Anonymous Communication. *Proc. Priv. Enhancing Technol.* **2019**, *2019*, 105–125. [CrossRef]
62. Van Blarkom, G.W.; Borking, J.J.; Olk, J.G.E. Handbook of Privacy and Privacy-Enhancing Technologies The case of Intelligent Software Agents. *Priv. Inc. Softw. Agent PISA Consort. Hague* **2003**, *198*, 14.
63. Commission of the European Communities. Communication from the Commission to the European Parliament and the Council on Promoting Data Protection by Privacy Enhancing Technologies (PETs). COM 2007 228 Final 2007. Available online: https://eur-lex.europa.eu/legal-content/EN/LSU/?uri=CELEX%3A52007DC0228 (accessed on 2 December 2020).
64. Pfitzmann, A.; Waidner, M. Networks without user observability. *Comput. Secur.* **1987**, *6*, 158–166. [CrossRef]
65. Pfitzmann, A. Diensteintegrierende Kommunikationsnetze mit teilnehmerüberprüfbarem Datenschutz. In *Informatik-Fachberichte 234*; Springer: Berlin/Heidelberg, Germany, 1990.
66. Fischer-Hübner, S. *IT-Security and Privacy: Design and Use of Privacy-Enhancing Security Mechanisms*; Springer: Berlin/Heidelberg, Germany, 2001.
67. Chaum, D.L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* **1981**, *24*, 84–90. [CrossRef]
68. Chaum, D.L. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptol.* **1998**, *1*, 65–75. [CrossRef]
69. Goldschlag, D.; Reed, M.; Syverson, P. Onion routing. *Commun. ACM* **1999**, *42*, 39–41. [CrossRef]

70. Dingledine, R.; Mathewson, N.; Syverson, P. Tor: The second-generation onion router. In Proceedings of the 13th USENIX Security Symposium, San Diego, CA, USA, 9–13 August 2004.
71. Sweeney, L. k-Anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* **2002**, *10*, 557–570. [CrossRef]
72. Machanavajjhala, A.; Kifer, D.; Gehrke, J.; Venkitasubramaniam, M. ℓ-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data* **2007**. [CrossRef]
73. Li, N.; Li, T.; Venkatasubramanian, S. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In Proceedings of the IEEE 23rd International Conference on Data Engineering, Istanbul, Turkey, 11–15 April 2007; pp. 106–115. [CrossRef]
74. Fysarakis, K.; Manifavas, C.; Papaefstathiou, I.; Adamopoulos, A. A lightweight anonymity and location privacy service. *IEEE Int. Symp. Signal Process. Inf. Technol.* **2013**, 000124–000129. [CrossRef]
75. Majeed, A. Attribute-centric anonymization scheme for improving user privacy and utility of publishing e-health data. *J. King Saud Univ. Comput. Inf. Sci.* **2019**, *31.4*, 426–435. [CrossRef]
76. Majeed, A.; Sungchang, L. Anonymization Techniques for Privacy Preserving Data Publishing: A Comprehensive Survey. *IEEE Access.* **2020**, *9*, 8512–8545. [CrossRef]
77. Common Criteria. *Common Criteria for Information Technology Security Evaluation—Part 2: Security Functional Components. Version 3.1 Rev. 5*; ISO/IEC: New York, NY, USA, 2017.
78. Pfitzmann, A.; Kohntopp, M. Anonymity, unobservability, and pseudonymity—A proposal for terminology. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Berlin/Heidelberg, Germany, 2001. [CrossRef]
79. Provos, N.; Honeyman, P. Hide and seek: An introduction to steganography. *IEEE Secur. Priv.* **2003**, *1*, 32–44. [CrossRef]
80. Nutzinger, M.; Fabian, C.; Marschalek, M. Secure Hybrid Spread Spectrum System for Steganography in Auditive Media. In *Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*; IEEE: New York, NY, USA, 2010; pp. 78–81. [CrossRef]
81. Korzhik, V.; Morales-Luna, G.; Loban, K.; Marakova-Begoc, I. Undetectable spread-time stegosystem based on noisy channels. In Proceedings of the International Multiconference on Computer Science and Information Technology, Wisła, Poland, 18–20 October 2008; pp. 723–728. [CrossRef]
82. Chakraborty, T.; Jajodia, S.; Katz, J.; Picariello, A.; Sperli, G.; Subrahmanian, V.S. FORGE: A fake online repository generation engine for cyber deception. *IEEE Trans. Dependable Secur. Comput.* **2019**. [CrossRef]
83. Chaum, D. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM* **1985**, *28*, 1030–1044. [CrossRef]
84. Chaum, D. Showing credentials without identification transferring signatures between unconditionally unlinkable pseudonyms. In *Advances in Cryptology—AUSCRYPT*; Springer: Berlin/Heidelberg, Germany, 1990; pp. 245–264.
85. Danezis, G.; Dingledine, R.; Mathewson, N. Mixminion: Design of a type III anonymous remailer protocol. *IEEE Symp. Secur. Priv.* **2003**. [CrossRef]
86. Camenisch, J.; Van Herreweghen, E. Design and implementation of the idemix anonymous credential system. In Proceedings of the ACM Conference on Computer and Communications Security, Washington, DC, USA, 18–22 November 2002. [CrossRef]
87. Mulvenna, M.D.; Anand, S.S.; Büchner, A.G. Personalization on the Net using Web mining: Introduction. *Commun. ACM* **2000**, *43*, 122–125. [CrossRef]
88. Zöllner, J.; Federrath, H.; Pfitzmann, A.; Westfeld, A.; Wicke, G.; Wolf, G. Über die Modellierung steganographischer Systeme. In *Proceedings GIFachtagung 'Verlässliche Informationssysteme' VIS'*; Müller, G., Rannenberg, K., Reitenspieß, M., Stiegler, H., Eds.; Vieweg-Verlag: Freiburg, Germany, 1997; p. 30.9.-2.10.97.
89. Fiaschetti, A.; Noll, J.; Azzoni, P.; Uribeetxeberria, R. (Eds.) *Measurable and Composable Security, Privacy, and Dependability for Cyberphysical Systems: The SHIELD Methodology*; CRC Press: Boca Raton, FL, USA, 2017.
90. Spanoudakis, G.; Kokolakis, S. (Eds.) *Security and Dependability for Ambient Intelligence*; Springer: Berlin/Heidelberg, Germany, 2019. [CrossRef]
91. Laprie, J.C. *Dependability: Basic Concepts and Terminology*; Springer: Berlin/Heidelberg, Germany, 1992.
92. Hatzivasilis, G.; Askoxylakis, I.; Anicic, D.; Broring, A.; Kulkarni, V.; Fysarakis, K.; Spanoudakis, G. The Interoperability of Things: Interoperable solutions as an enabler for IoT and Web 3.0. In Proceedings of the 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Barcelona, Spain, 17–19 September 2018; pp. 1–7.
93. Papoutsakis, M.; Fysarakis, K.; Mixalodimitrakis, E.; Lakka, E.; Petroulakis, N.; Spanoudakis, G.; Ioannidis, S. A Pattern–Driven Adaptation in IoT Orchestrations to Guarantee SPDI Properties. In *International Workshop on Model-Driven Simulation and Training Environments for Cybersecurity*; Springer: Cham, Switzerland, 2020; pp. 143–156.
94. Forgy, C. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artif. Intell.* **1982**, *19*, 17–37. [CrossRef]
95. Bröring, A.; Seeger, J.; Papoutsakis, M.; Fysarakis, K.; Caracalli, A. Networking-Aware IoT Application Development. *Sensors* **2020**, *20*, 897. [CrossRef]