The σ-Calculus - A process calculus for privacy-preserving protocols in location-based service systems

Vom Promotionsausschuss der
Technischen Universität Hamburg
zur Erlangung des akademischen Grades
Dr. rer. nat.
genehmigte Dissertation

von
Kai Bavendiek

aus
Braunschweig

2022

1. Gutachter: Prof. Dr. Sibylle Schupp
2. Gutachter: Prof. Dr. Hannes Federrath

Datum der mündlichen Prüfung: 04.03.2022

Acknowledgements

First and foremost, I want to thank Sibylle Schupp for her constant support – both, intellectual and moral. You read through so many drafts and tirelessly gave pointed feedback, which is by no means something to be taken for granted!

Furthermore, I would like to thank everyone from my research project who helped me broaden my horizon with inter-disciplinary talks about privacy and the world. This goes especially for Hannes Federrath who kindly agreed to review my dissertation.

I would like to thank the whole STS institute as I have very much enjoyed working with each of the members. Especially my fellow research assistants made each day better with our discussions about random stuff. Sharing a room with Mathias was probably not always improving our productivity but it was fun.

Finally, I would like to thank my family for their support. They always believed in me and at the right times acted like they didn't care, which I am very grateful for!

Abstract

Location privacy-preserving mechanisms (LPPM) use different computational approaches to protect personal data in queries to location-based services. These LPPMs give different privacy claims that are not comparable per se. We propose the σ-calculus, a process calculus to model LPPMs and their privacy guarantees. Furthermore, we present LP3Verif, which is a model checking tool to verify location privacy properties of LPPMs using the σ-calculus.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

Glossary

k-Anonymity Is a privacy model originating from databases where in a table containing sensitive data with n rows each row with its set of identifying attributes cannot be distinguished from at least $k-1$ other rows.

l-Diversity Is an extension to k-Anonymity where additionally each k-block must contain at least l different values of its sensitive attributes to avoid background-knowledge attacks.

Caches Is a category of privacy protection mechanisms that reduces the number of queries that have to be issued to the service provider by remembering requests and responses.

Cloaking In location privacy describes the hiding of spatial data and while cloaking does not specify an approach, it is often used in context of Mix Zones.

Cryptography Is a category of privacy protection mechanisms that utilize mathematical approaches to make it computationally infeasible to derive identities or locations.

Dummies Is a category of privacy protection mechanisms that use fake user data or completely fake requests to confuse service providers.

Generalization Is a category of privacy protection mechanisms that generalizes a location (spatial) or a point in time (temporal) to a region or time span, respectively.

Mix Zones Is a category of privacy protection mechanisms that use black box areas in which no locations are revealed and identities may be swapped.

Model Checking Is an approach to check a system against specifications by modeling the system as formal state-machine or automaton.

Obfuscation In location privacy describes the obscuring of spatio-temporal data and while obfuscation does not specify an approach, it is often used in context of Perturbation.

Perturbation Is a category of privacy protection mechanisms that perturbs locations, usually using noise.

Process Calculus Is a formal model of concurrent systems where usually order of execution and information exchange are described.

Witness Is a specific value assignment to demonstrate certain behavior, e.g., a counterexample.

Acronyms

ACP	Algebra of Communicating Processes.
API	Application Programming Interface.
ASP	Anonymization Service Provider.
ATEL	Alternating-Time Temporal Epistemic Logic.
ATL	Alternating-Time Temporal Logic.
BDD	Binary Decision Diagram.
BNF	Backus Naur Form.
CAA	Context-Aware Ambient.
CCS	Calculus of Communicating Systems.
CEGAR	Counterexample-Guided Abstraction Refinement.
CSP	Communicating Sequential Processes.
CTL	Computation Tree Logic.
CVC4	Cooperating Validity Checker 4 (SMT solver).
DTMC	Discrete-Time Markoc Chain.
GDPR	General Data Protection Regulation.
GPS	Global Positioning System.
HIPAA	Health Insurance Portability and Accountability Act.
IoT	Internet of Things.
LBS	Location-Based Services.
LP3	Location Privacy-Preserving Protocol.
LPPA	Location Privacy-Preserving Algorithm.
LPPM	Location Privacy-Preserving Mechanism.
LTL	Linear(-Time) Temporal Logic.
LTS	Labeled Transition System.
MANET	Mobile ad hoc Network.
MAS	Mulit-Agent System.
MBB	Minimal Bounding Box.
MC	Model Checking.
MDP	Markov Decision Process.
OBDD	Ordered Binary Decision Diagram.
P2P	Peer to Peer.
PoI	Point of Interest.
QoS	Quality of Service.

SAT	Boolean Satisfiability Problem.
SMT	Satisfiability Modulo Theories.
SOS	Structural Operational Semantics.
TL	Tense Logic.
TLA	Temporal Logic of Actions.
VPN	Virtual Private Network.

1. Introduction

Location-Based Services (LBS) are ubiquitous in times in which virtually everyone carries a smart device equipped with location sensor systems like GPS. LBS systems can be very useful for various applications such as navigation, social networks, or finding near points of interest (PoI). However, the use of LBS implies the trading of location data for the desired service. Hence, the use of LBS systems can lead to the leaking of personal data and even the intended location data might be more fine-grained than necessary for a specific application. In recent years, data privacy has become a globally much discussed topic and location privacy is a sub-field recognized by privacy advocates, e.g., the European Union General Data Protection Regulation (GDPR) names in Article 4 (1) "location data" explicitly as an example of personal data that needs to be protected [13]. Research shows that, for instance, location data can be used to infer other personal data like home addresses and even names [14]. As location data can be used to infer personal (and even identifying) data, it is itself categorized as identifying personal data and consequently, needs special protection under the GDPR.

The issue of data privacy violation in LBS systems is well known in the literature and many different approaches have been proposed over the past years to prevent the leaking of personal data. Many of these approaches are based on the assumption that users do not want to trust the corporations behind LBS with their sensitive data. Therefore, Location Privacy-Preserving Mechanism (LPPM) were introduced. These are oftentimes trusted third parties that sit in between the user and the LBS and modify the data flow to protect the users' privacy. LPPMs are, even though in academia a much discussed topic, in practice still relatively new and not widely adopted. Only a handful LPPM applications exist to protect user locations in LBS communication. Because this field is quite new in practice, tools and theory supporting software designers in developing new applications are in fact needed for establishing more practical implementations.

LPPMs are generally described via pseudo-code algorithms and usually give (formal) guarantees based on different metrics and definitions of location privacy. The protection goals of LPPMs are foremost protection of spatio-temporal data, i.e., the time and location of a service request, but also include protection of the identity of the users as well as protection of the contents of their service requests, which may allow derivation of sensitive data like religious confession or medical condition. While the protection goals of LPPMs only differ slightly, the approaches of the mechanisms are manifold. For instance, to both protect the anonymity and the location of a single user, an LPPM can gather multiple user queries and then send a group query to the LBS where identities are hidden and the locations are generalized to an area.

All location privacy-preserving mechanisms have in common that they promise some location privacy guarantee their algorithms satisfy. However, a common language is lacking to formally describe all algorithms and privacy guarantees. Such a common language enables the verification of formal privacy guarantees, which in turn enables the comparison of mechanisms based on the verified guarantees. Without a common formal language and verification LPPMs cannot be compared according to their location privacy

protection. For example, the privacy impact of an approach which decreases the accuracy of location cannot easily be compared to an approach which sends additional dummy requests to confuse the LBS. Therefore, we identify the need for a uniform intermediate representation. Furthermore, LPPM papers usually define their own privacy metrics and promise more or less formal privacy guarantees. The metrics are usually specific to the application or certain attacks and the privacy promises may be misleading. The validation and comparison of these guarantees requires a common formal language and logic.

We introduce the term *location privacy-preserving protocols* (LP3), which are models of online query-based LBS systems with LPPMs protecting the users' privacy. LP3s differ from LPPMs in that LP3s are a formal description of the communication between the user side (including optional trusted servers) and the LBS where we focus on the messages, i.e., LP3s model the queries sent to an LBS containing personal data that is modified by an LPPM. LPPMs, on the other hand, are algorithms missing the context of the whole queries, e.g., an LPPM describes how to manipulate the location, while LP3s describe additionally how the location is communicated with the LBS. This is an important context because an LBS query does not consist only of location data. For example, an LPPM generalizing a position to an area with multiple users in it cannot guarantee anonymity if the query contains a unique identifier. We also present our taxonomy of location privacy as we view the field and its sub-categories. We derive a set of four ground requirements that can be used to compare LP3s based on their privacy guarantees. These ground requirements can be used as a benchmark set of privacy properties that represent all aspects of privacy in location-based services. LP3s can be compared by their coverage of these requirements.

We introduce a new modeling language for LP3s as well as location privacy requirements, and a logic that enables the verification of location privacy of LP3 models. Two challenges of a framework that entails a modeling language and verification logic are first to find an abstraction of protocol models which captures the differences in the underlying LPPMs but also is simple enough to allow comparison, and second to create a logic that allows both meaningful and computable statements about the models. We decided on a process calculus to describe LP3 models and a temporal-epistemic logic to verify non-probabilistic location privacy properties. We call the framework, containing the formal description languages and the verification logic, the σ-calculus. The name is derived from the set-oriented logic and the state-keeping of the σ-calculus. We also present in this dissertation the tool LP3Verif, which is an explicit-state model checker that takes an LP3 model and location privacy properties as input and verifies these. If a property does not hold, LP3Verif returns a "witness." A witness is an execution of the protocol that demonstrates a potential violation of the property. LP3Verif is written in Java and uses the SMT solver CVC4 as back end.

We evaluate the expressiveness of the σ-calculus by modeling the most-cited LPPM papers with non-probabilistic guarantees and compare them by means of our ground requirements set. The set of modeled LP3s is categorized on the basis of our LPPM taxonomy into six categories of mechanisms. This set serves as a "look-up table" for software developers who want to design a protocol and need inspiration from mechanisms

with already evaluated privacy properties. For further evaluation we present a case study that demonstrates a step-by-step application of LP3Verif and the σ-calculus in practice as well as an example that shows the process for a software designer using LP3Verif for designing a new LPPM protocol.

In summary, we make the following contributions:

- A location-query process calculus, called σ-calculus, based on set-centered terms for describing location privacy preserving protocols (LP3)
- A temporal modal logic with dynamic epistemic properties to support formal reasoning about location privacy, enabling the formal verification of privacy properties of LP3 models
- An explicit-state model checking procedure for automatic verification and an open source software, called LP3Verif, implementing the σ-calculus framework
- Models of common LPPMs from the literature, our own location privacy property benchmark set (consisting of four ground requirements), as well as the verification results of all these protocols and properties

The σ-calculus demonstrates that a light-weight process calculus, a reduced variant of a pi-calculus, suffices to model LBS communication as well as location-privacy preservation. Furthermore, a simple temporal modal logic with dynamic epistemic properties (we call "temporal-K-less-epistemic modal logic") is sufficient to reason about location privacy properties in this context, and an explicit-state model checking algorithm suffices to efficiently verify location privacy properties of LP3 models. LP3Verif – source code and binaries – and the LPPM models are available at `https://www.tuhh.de/sts/research/data-protection-machine-learning/lp3verif.html`.

Outline

In Chapter 2 we provide the necessary (informal) background on location privacy in location-based service systems. Here, we set the scope of our understanding of LBS, LPPMs, and location privacy requirements and derive the important features this view implies for our language and logic. We provide examples of LPPMs for the better understanding of the readers. One of these protocols, called PrivacyGrid, is introduced as a running example that serves as a reference protocol for the rest of the dissertation.

Chapter 3 introduces the formal background for this dissertation, including the basic concepts of process calculi, modal logics, and model checking. This chapter highlights the common foundations of the formal methods required to understand the remainder of this dissertation. This chapter also presents the combination of process calculi and modal logics and characterizes a potential model checking algorithm of such a combination.

In Chapter 4 we present related research building on the state of the art presented in the previous chapter. We focus on publications with a similar goal of verifying location privacy or similar formal methods such as process calculi and temporal-epistemic modal

logics. This chapter places the contributions of this dissertation in the context of related work and also serves as an overview of the research in the intersection of process calculi and modal logics with a focus on verification. We compare our contributions, listed in this chapter, to previous work and highlight important similarities and differences.

Afterwards, Chapter 5 presents the σ-calculus. We first set the basis of the framework by stating ground assumptions for the LBS setting and the attacker model. We then describe the syntax of the protocol language that models LP3s as processes. The semantics of the process calculus follow with a focus on the reduction of processes and their impact on the location state, which represents the internal knowledge of the protocol. The second part of this chapter first presents a property syntax that enables expressing location privacy requirements as motivated in Chapter 2. We then present the semantics of properties, which are given by a satisfaction relation of LP3 models and properties.

In Chapter 6 we show how model checking is implemented in LP3Verif. First, we present model checking algorithms that enable the automated verification of location privacy properties. The described approach explores the (comparably small) state space explicitly but uses structural properties of the models to reduce the number of checks. We then discuss our implementation of the described model checking algorithms and present our tool, called LP3Verif. We describe the scope and functionality of the tool and discuss design decisions as well as alternative approaches.

With Chapter 7 an evaluation of expressiveness of the σ-calculus and performance of LP3Verif follows. We first assess the expressiveness of the σ-calculus based on our own benchmark set of protocols and ground requirements, which are comprised of the most common LPPMs and their protection goals from the literature. We then evaluate the performance of LP3Verif in respect of timing and scalability. Furthermore, we measure the impact of certain design decisions such as caching. Afterwards, we discuss possible threats to validity regarding our basic assumptions and decisions.

Chapter 8 presents a case study application that demonstrates the complete work flow of modeling a protocol, formalizing its privacy properties, and the subsequent verification. But first, we introduce location privacy-preserving mechanisms in practice and show the results of our survey of LPPM applications. Then, we choose one real-world LPPM application, which is a browser extension called *Location Guard*, to present in form of a case study. We model *Location Guard* as a location privacy-preserving protocol and verify its claimed privacy guarantee using LP3Verif. For a further demonstration of the σ-calculus and how it can be used in a scenario where a software designer wants to implement an LPPM, we present the example of Alice who designs a new mechanism to use with Google Places.

Finally, in Chapter 9 we summarize the findings of this dissertation, have a concluding discussion, and suggest future work that could build on our contributions.

2. On Location Privacy

In this chapter, we give a general overview of location-based services, location privacy-preserving mechanisms, and location privacy. The overview serves as an informal categorization of the privacy threat scenario introduced in the previous chapter. Furthermore, we derive from the introduced location privacy-preserving mechanisms requirements for formal modeling languages of these mechanisms and their corresponding location privacy guarantees. To this end, we perform a survey for both fields, location privacy-preserving mechanisms and location privacy, and derive a taxonomy comprising categories of LPPMs and privacy guarantees. From these we derive the requirements and properties that a formal language (like the one we introduce in Chapter 5) should meet. Furthermore, our literature survey shows that location privacy is a term that is not well defined but used ambiguously in the field of location-based services. The term represents either the broad field of user data protection in the area of location-based services or a location-specific guarantee (like location k-anonymity) [15]. For the scope of this paper we understand location privacy as "the data protection of all personal user data in the context of an LBS."

2.1. Location-Based Services

The above definition of location privacy uses location-based services as context. Therefore, we define the term *LBS* in this section to narrow down the setting we consider in this dissertation. The field of LBS is huge and many types of applications fall into this category. In 2004 Spiekermann, who defines location services as "services that integrate a mobile device's location or position with other information so as to provide added value to a user," names 22 types of LBS applications ranging from news, gaming, and advertisement to navigation and point of interest (PoI) [16]. She, furthermore, categorizes LBS applications according to two distinctions: *person-oriented* or *device-oriented* as well as *push services* or *pull services*. Person-oriented LBS are applications which use the position of a person for their service, while device-oriented LBS might also indirectly use the positions of persons but generally track objects like cars. An example of an LBS application in the category of person-oriented pull services is the search for a nearest PoI like a restaurant.

In 2005 Gratsias et al. proposed a different taxonomy of LBS [17]. The authors distinguish between *stationary* and *mobile data objects* and *query objects*, where the query object is the user of the LBS, and the data object is the target of a position request. For instance, a navigation app that guides towards a fixed destination is an application in the category *mobile query object* and *stationary data object*. Whereas a *stationary query object* and *mobile data object* application can be a delivery tracking app where the user waits at home and tracks the delivery truck.

For this dissertation, we consider LBS applications that involve the transmission of sensitive personal data, hence the LBS in this dissertation fall into the category of person-oriented services. Furthermore, we consider applications where users issue queries

to receive the requested service, placing our understanding of LBS applications in the category of person-oriented pull services according to the taxonomy by Spiekermann. Although, in theory, all four categories in the taxonomy by Gratsias et al. are possible scenarios in our understanding of LBS applications, we focus in this dissertation on stationary data objects and mobile query objects, i.e., a potentially moving user issues queries for services with a fixed location.

A more recent survey of LBS systems by Primault et al. from 2019 includes LPPMs into the taxonomy and categorizes LPPM application scenarios into *online* and *offline* and furthermore places online LPPMs into the categories *real-time* and *batch* [1]. As Figure 2.1 from their survey shows, online LPPMs are in between the user and the LBS, whereas offline LPPMs sit between the LBS and data analysts. Real-time LPPMs are used in scenarios where users request services based on single locations and expect immediate responses, while batch-based LPPMs operate in a setting where users send multiple locations over a course of time and expect aggregated service results in a certain interval. An example of an application of an offline LPPM is the anonymization of a data base view by removing names and generalizing post codes to 3 digits before giving the table to data analysts. An example of a real-time LPPM is the perturbation of the true location before querying a find-nearest-restaurant LBS. Finally, an LPPM that adds dummy locations to the real ones before submitting a batch to a crowd-sensing application is an example of a batch-based LPPM. We argue that online, real-time LPPMs are the category that involve the most challenging privacy preservation properties as they neither have the time and full information of offline LPPMs nor do they have the context of multiple data points and the certainty that the data is aggregated, which simplifies the task of protecting privacy significantly. For this dissertation we, consequently, focus on real-time LPPMs as they are the most interesting category.

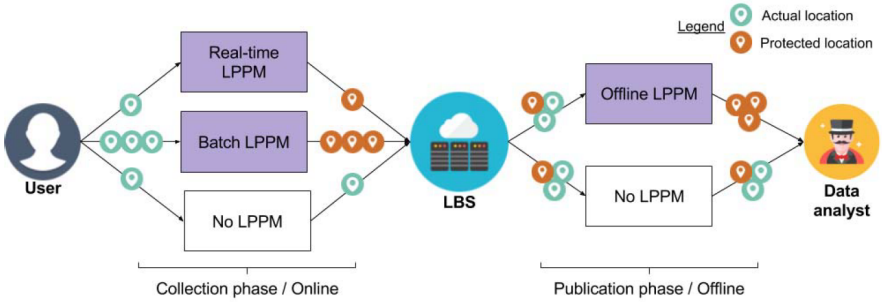


Figure 2.1.: Categories of LPPMs in LBS scenario [1]

The taxonomy in the paper by Primault et al. further categorizes LPPMs by their protection mechanisms. In Section 2.4, we discuss alternative taxonomy approaches and explain how we performed our own literature survey to derive requirements of a modeling language. In the following section we present our survey of LPPM papers and motivate the categories we derive from it.

2.2. Location Privacy-Preserving Mechanisms

The idea of privacy mechanisms that protect the users' data is nearly as old as location-based services themselves. Krumm performed a survey of location privacy threats and countermeasures [15]. This survey included an early taxonomy of LPPMs with a focus on anonymity.

With the goal of identifying the requirements of a modeling language, we performed our own literature survey with 62 papers [2, 4–8, 12, 18–71] in the field of location privacy preserving mechanisms that propose algorithms or protocols to protect the location privacy of users in LBS. All papers have in common that they propose ways to break the link between user identities and locations: by modifying the locations, the identities, or the link itself. Locations can either be generalized to break a one-to-one link or can be perturbed. Identities can either be exchanged with others or be hidden under a large number of other identities. The link between identity and location, finally, can be hidden either via mathematical operations or by reducing the number of linkable requests. Accordingly, in our taxonomy, LPPMs fall into six categories based on their approach to protect the location of LBS users. We differentiate between: Generalization, Perturbation, Mix Zones, Dummies, Cryptography, and Caches. Table 2.1 shows the six categories, highlights their respective approach, and gives an exemplary protocol falling into said category.

Table 2.1.: Categories of LPPMs

Category	**Approach**	**Example**
Generalization	coarsen spatial/temporal resolution	PrivacyGrid
Perturbation	report shifted location	CAP
Mix Zones	change identities in black box area	MobiMix
Dummies	generate additional fake requests	MobiPriv
Cryptography	mathematically hide real locations	MaPIR
Caches	reduce number of queries	MobiCrowd

Before we describe in more detail how the six category differ in their approach to preserve location privacy, first a few concepts and terms have to be introduced. A key term in location privacy preservation is *obfuscation* as it describes the process of modifying location data before reporting it to an LBS. Technically, not all categories of LPPMs we derived use obfuscation as caching only reduces the number of queries and not the data itself. Most commonly, obfuscation is used for perturbation and generalization mechanisms but it can also describe the general approach of hiding locations and identities.

The concept of *k-anonymity* is a very common protection goal of group-based privacy approaches and describes that a person is anonymous in a group of size k. This principle originates in database privacy where a table is k-anonymous if each person contained cannot be distinguished from $k-1$ other persons in the table [72]. In the context of location-based services, this principle is often called *location-k-anonymity* and usually refers to regions, where a region is said to be location-k-anonymous if at least k users

are inside this location at the same time. Other interpretations of k-anonymity, e.g., k locations or PoI in a region, exist but the general idea is always the same: one individual hides in a group of k.

The privacy guarantee of k-anonymity is usually achieved via generalization and suppression of data. That is, identifying data is omitted and pseudo-identifying data is generalized until at least $k-1$ other individuals share the same set of attributes. In location privacy the process of generalization is also called *cloaking*, where *spatial cloaking* describes the process of generalizing a location to a location-k-anonymous region and *temporal cloaking* the process of generalizing a time stamp to a time frame that fulfills k-anonymity.

As regions are usually described as sets of connected discrete locations, a standard approach to getting from a set of possibly distributed locations to a region is using so-called *bounding boxes*. Finding a minimal bounding box, i.e., the smallest rectangular region that contains all locations of a set, is a common approach in Generalization mechanisms.

The concept of mix zones is based on the idea that certain locations are more sensitive than others from a privacy perspective, e.g., hospitals, shopping malls, churches, etc. If one only wants to protect against disclosure of these sensitive locations, the areas around these locations can be declared as black boxes where no positions are reported inside these black boxes and only the entering and exiting time are shared with an LBS. These black boxes are called *mix zones* if users inside the black box area swap identifiers to prevent the LBS from using the timing information to deduce whether an individual has only passed by or stayed in an area for a longer time.

Cryptographic mechanisms use mathematical characteristics to prevent, for instance, linking of identities and locations. Cryptographic protection usually relies on computationally cheap operations to hide these links and computationally expensive operations to reverse them. The concept of spatial *redundancy* is a common approach, where users and service provider agree on a 'map' which consists of a grid of columns and rows. Instead of reporting a location, a user reports a number which is the result of a cheap computation using the column and row of the location. However, the reported number can also belong to different combinations of columns and rows, hence the name redundancy. The service provider answers the query with service results for each of the redundant locations. Spatial redundancy is considered a crypto-based mechanism if the redundancy is achieved solely via logical computations. This concept of redundancy should not be confused with redundancy approaches that require physical redundancy of data.

2.3. Location Privacy-Preserving Protocols

Evaluating location privacy of location privacy-preserving mechanisms requires context of location-based services and its users. This section puts LPPMs in the context of LBS queries from users to the LBS and introduces location privacy-preserving protocols (LP3). LP3s describe the communication between users and the LBS where LPPMs are

used to obscure the personal data in queries. In the following, we go over the example LPPMs from Table 2.1 in more detail. Furthermore, we show LP3 formalizations of these examples modeled already in our protocol language, which we will introduce in Chapter 5. The full syntax of the protocol language of the σ-calculus will be shown in Subsection 5.1.1. Here we only point out the important parts of each protocol.

Generalization mechanisms such as PrivacyGrid [2] generalize the location of the user by reporting an area instead or generalize the point in time by reporting a time frame, i.e., a span of time in between two time stamps t_0 and t_n. Usually the goal of generalization mechanisms is location-k-anonymity, i.e., at least k users are in the reported area. Figure 2.2 shows the LP3 formalization of the PrivacyGrid (cf. Table 2.1) mechanism. Line 3 indicates that this protocol uses the data of other users for the anonymization. Lines 4 and 6 show that this protocol follows a Generalization approach because a region R is reported and this region is the minimum bounding box (MBB) that contains the locations of the gathered users. Furthermore, line 5 indicates that this protocol also demands a diverse set of service requests and the query parameter "servs" in line 6 indicates that the query requests the set of service requests of the group.

```
1 process PrivacyGrid
    !
3       if k_users
            Compute(R=MBB(locs))
5           if s_diverse
                Query(pid,R,servs,t)
7           end
        end
9   end
end
```

Figure 2.2.: PrivacyGrid as location privacy-preserving protocol (LP3)

Perturbation mechanisms, next, like CAP [5] are based on random noise on (a discrete interpretation of) locations such that the user does not send the real position to the LBS. Figure 2.3 shows the location privacy-preserving protocol formalization of the CAP (cf. Table 2.1) mechanism. Line 3 indicates that this protocol is based on a Perturbation mechanisms as it uses noise to hide the location of the user and line 4 shows that only the location is perturbed in the query.

Mix Zone approaches like MobiMix [21] are a category of protection mechanisms that achieve location privacy by defining areas as mix zones, in which no locations are reported and identities are exchanged. Figure 2.4 shows the location privacy-preserving protocol formalization of the MobiMix (cf. Table 2.1) mechanism. Line 5 indicates that this protocol is based on a Mix Zone mechanism as it swaps pseudonyms of users. This form of protection breaks the link between identity and location of users for continuous service use. Furthermore, the two queries in lines 4 and 6 also indicate a Mix Zone as

```
process CAP
    !
        Compute(R=noise(loc))
        Query(pid,R,serv,t)
    end
end
```

Figure 2.3.: CAP as LP3

these queries model the reporting before entering and after leaving of an area where no locations are reported. The parameter "t" in both queries indicates that the unprotected time stamp is reported

```
process MobiMix
    !
        if k_users
            Query(pid,loc,serv,t)
            Compute(P=swap(pids))
            Query(P,loc,serv,t)
        end
    end
end
```

Figure 2.4.: MobiMix as LP3

Dummy-based mechanisms such as MobiPriv [41] either send single dummy queries to the LBS to confuse an attacker or enrich another mechanism by adding fake users to a group of real users. Figure 2.5 shows the location privacy-preserving protocol formalization of the MobiPriv mechanism. Lines 3-5 indicate that this protocol is in its core based on a Generalization approach and line 7 shows that the protocol in fact uses a Dummy mechanism on top. MobiPriv (cf. Table 2.1) is a protocol that uses dummy locations only if not enough real users can be found, which is visible from the two nested *if* structures.

The category of Crypto encompasses all cryptography-based approaches, such as MaPIR [7], that protect the communication with an LBS by making it computationally infeasible for an attacker to derive the location of the user. Figure 2.6 shows the location privacy-preserving protocol formalization of the MaPIR (cf. Table 2.1) mechanism. Line 4 indicates that this protocol is based on a Crypto mechanism as the location is protected via redundancy, i.e., the location is represented by an identifier that maps to multiple, distributed locations. Line 3 shows that the protocol also protects the identity with a simple hash function.

Cache-based approaches like MobiCrowd [56] either solely base their protection on

```
process MobiPriv
    !
        if k_users
            Compute(R=MBB(locs))
            Query(pids,R,serv,t)
        else
            if dummies
                Compute(R=MBB(locs))
                Query(pids,R,serv,t)
            end
        end
    end
end
```

Figure 2.5.: MobiPriv as LP3

```
process MaPIR
    !
        Compute(h=hash(pid))
        Compute(R=redund(loc))
        Query(h,R,serv,t)
    end
end
```

Figure 2.6.: MaPIR as LP3

a reduced number of queries achieved by caching sent queries and responses or are combined with another approach to improve performance. Figure 2.7 shows the LP3 formalization of the mechanism MobiCrowd (cf. Table 2.1). Caching is not modeled in LP3 as in the worst case, the desired service is not cached or outdated and must be requested from the LBS. Lines 3 to 5 model this scenario where the protocol is executed without any cache lookups.

In the following, using our taxonomy of LPPMs, we derive the corresponding requirements for our σ-calculus framework. Each category from Table 2.1 necessitates its own set of language and logic features. A language to describe LPPMs must be able to model the movement in an LBS (as the LBS can link queries to gain additional knowledge) and describe location-specific operations. All categories from Table 2.1 share this need, however, their differences lie in location-specific obfuscation operations. As described above spatial generalization approaches generalize single locations to areas. Similarly, temporal generalization expands time stamps into time frames. The category of Generalization mechanisms requires a language that can model locations and areas (as well as time stamps and time frames). In discrete models of locations and regions, a region R can

```
process MobiCrowd
    !
        Query(pid,loc,serv,t)
    end
end
```

Figure 2.7.: MobiCrowd as LP3

be seen as a **set** of locations l_i: $R = \{l_1, l_2, ..., l_n\}$. Similarly, time frames F can be described by sets of time stamps t_i: $F = \{t_1, t_2, ..., t_m\}$. Sets are not only a good basis for representing Generalization mechanisms but can also be used to model other types of mechanisms as well. In the following, we explain how our decision for a set-based language and logic affects other types of mechanisms, especially as we view these mechanisms from the perspective of the adversary, which we consider to be the location-based service provider (or someone with similar access).

The Perturbation approaches add noise to the (discretized) real locations. For the location-based service provider itself, who knows the underlying mechanism, i.e., noise, such a perturbed location is an area of all locations in the radius of the maximum noise around the reported location. As we model knowledge of locations from the perspective of the adversary, Perturbation mechanisms can be modeled via areas. Mix Zone approaches take their protection from the changing of identities. From the LBS' point of view, this can be seen as an assignment problem: which identity belongs to whom. As our logic has a single representative user (more on that in Assumption 3 in Section 5.1), the LBS has several options in a set of potential identities of which only one is the specific user. This choice set can, hence, represent the identity switching from the LBS's point of view. Dummy-based mechanisms hide real queries behind a number of dummy requests. Similar to the scenario of Mix Zones, the LBS is here confronted with several options in a set of potential real queries, making a set, again, a possible formalization of this protection mechanism. Caches do not per se change the queries sent to an LBS. They are oftentimes combined with other approaches and bring no modification to the queries which are not already covered by the other types. We, therefore, conclude that caches can be modeled with a language that can model the other five categories.

We describe LP3 models with the intention to reason about the knowledge gain the LBS, in its role as adversary, can get from received user queries. The goal of this dissertation, among others, is to create a description language to model LPPMs from all categories. To achieve this goal, we need a common formalization of all obfuscation approaches. As can be seen from the previous paragraph, the six categories have in common that one can describe their protection approach with the help of sets when considering the perspective of the LBS. In Subsection 5.1.1 we will introduce the syntax of a process calculus that is based on a set-centered term language as motivated in this section. The two main components of the language are a query with a quadruple of sets with identity, location, service, and time data as well as a computation that does the

actual obfuscation, e.g., generalizing a location to a set of locations. The computation allows the definition and modification of sets of locations or identities. In Section 2.6, we link the four user-data parameters of queries (identity, location, service, and time) to potential privacy threats.

2.4. Alternative Taxonomies

While the above described taxonomy reflects our view of LBS scenarios (online, real-time, pull), others have a different focus and hence derive different taxonomies from similar sets of papers. As we focus on online LPPMs and some authors also include offline protection mechanisms, the paper surveys are not identical. Chow and Mokbel, for instance, have a trajectory-oriented view on LPPMs and come to a different categorization of mechanisms compared to our taxonomy or the one by Primault et al.. Chow and Mokbel do not differentiate between online and offline applications but their categorization is solely based on the approaches to break trajectories: *spatial cloaking*, *mix zones*, *vehicular mix zones*, *path confusion*, and *dummy trajectories* [73]. While these categories look similar to ours from Table 2.1, their categories reflect the focus on protecting trajectories. In the following we describe the categories which are different from ours. Spatial cloaking is by the definition of Chow and Mokbel a category that encompasses spatial generalization and perturbation mechanisms. Vehicular mix zones are a special type of mix zones which have an underlying road network and usually only contain one street crossing. Path confusion is an approach to make trajectories unpredictable for maximum likelihood detection attacks by only reporting location updates when the trajectory is not predictable with a high probability. Mechanisms of this category eliminate certain queries and leave the others as is. Dummy trajectories are a specialization of our Dummies category where complete trajectories of positions corresponding to each dummy user are generated to confuse the LBS, instead of single queries. Chow and Mokbel do not exclude offline mechanisms and focus on trajectories, which leads to a different categorization than ours.

Wernke et al. have an attack-oriented perspective and categorize approaches by their vulnerability to specific attacks [74]. They distinguish six categories: *position dummies*, *mix zones*, *k-anonymity*, *obfuscation*, *cryptography*, and *position sharing*. Again, we briefly summarize the new concepts or the ones differing from our terminology. The category k-anonymity encompasses all mechanisms that provide group-based privacy guarantees. Obfuscation mechanisms, by the definition of Wernke et al., are mechanisms that reduce the precision of spatial data. As in practice mechanisms of both categories, k-anonymity and obfuscation, generalize positions to areas, the union of these two categories is comparable to Generalization in our taxonomy. Mechanisms from the category "position sharing" split their accurate position into so-called 'shares' which combined reveal the position and in subsets only reveal imprecise locations. This way users have different precision levels. Wernke et al. picked a different subset of papers for their survey and, partly due to their focus on known attacks, derived a taxonomy with a few different categories.

2.5. Example Protocol: PrivacyGrid

We choose the protocol PrivacyGrid to take a closer look at by discussing it in more detail. The protocol from the paper "PRIVACYGRID: Supporting Anonymous Location Queries in Mobile Environments" by Bamba et al. [2] will serve as running example throughout the rest of the dissertation and will be referenced in Chapter 5 and Chapter 6. While the previous examples demonstrated different approaches to location-privacy protection, in this section we show a typical LPPM paper and how such a paper describes an LPPM in a pseudo-code algorithmic style. By describing the algorithm from the paper, we demonstrate that a protocol formalization has to abstract details of mechanisms.

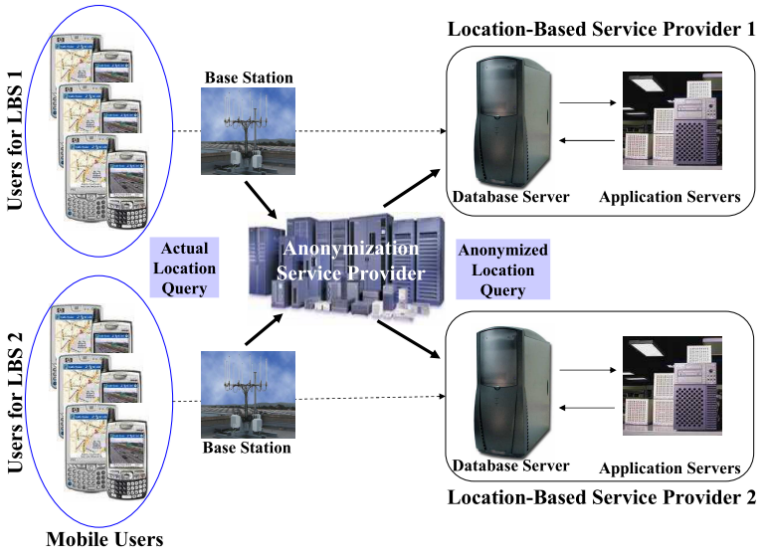


Figure 2.8.: PrivacyGrid system architecture [2]

Figure 2.8 shows the architecture of the PrivacyGrid LPPM, two users, and two location-based services. The Anonymization Service Provider (ASP) in the middle is the trusted third party that performs the privacy protection of the LPPM. Instead of directly querying the LBS, users can query the ASP with a privacy profile consisting of a triple $(k, l, \{d_x, d_y, d_t\})$, where k and l define the k-anonymity and l-diversity levels, respectively, (d_x, d_y) define the maximum spatial resolution, and d_t defines the maximum temporal resolution. While the first two parameters represent the privacy level the user wants to achieve, the maximum spatial and temporal resolution represent QoS restrictions, i.e., d_x, d_y, d_t describe the maximum tolerable distance to the actual location and time stamp of the query.

With a configured privacy profile users can make queries to the ASP, where a query is a sixtuple $m_s = \langle object_{id}, request_{id}, \{x, y, t\}, F, k, l, \{d_x, d_y, d_t\}\rangle$. The parameter $object_{id}$ is an identifier representing the user, $request_{id}$ is an identifier representing the query, the triple $\{x, y, t\}$ is the spatio-temporal information of the user, F represents a content filter for nearest-PoI queries, e.g., gas stations, Italian restaurants, hospitals, and

the remaining parameters are the privacy profile of the user as just introduced. The ASP applies an LPPM and transforms the user query to a perturbed query $m_t = \langle h(object_{id}||request_{id}), \{X, Y, I\}, F\rangle$. The identifier $h(object_{id}||request_{id})$ is a hash of both, the user and the query identifier, the spatio-temporal region $\{X, Y, T\}$ is the minimum bounding box containing at least k users in the time frame (also satisfying l-diversity and the QoS requirement), and the service request F is just passed on. Even though the query transformation is not part of the algorithm, the paper describes (informally) how to construct queries to an LBS. In the case of PrivacyGrid, the paper already provides a good description of the protocol but most papers neglect the aspect of queries and focus only on location protection. The following algorithm is a typical representation of LPPMs in research papers. Furthermore, it is also typical for these paper to only informally (or not at all) describe the protocol, i.e., complete queries and potential server-side computations.

Algorithm 1 Quad Grid Cloaking (shortened) from [2]

Require: $\{object_{id}, request_{id}, x, y, t\}$ original request, $\{d_x, d_y, d_t\}$ QoS requirement, $\{k, l\}$ privacy requirement
Ensure: Minimal spatial cloaking box

```
1:  cid ← GridIndexSearch(object_id, x, y)            ▷ returns cell index of the location
2:  procedure QuadGridCloaking(k, l, {x, y}, {d_x, d_y}, cid)
3:      (MN, SN) ← CellObjectCount(cid)               ▷ number of mobile and static nodes
4:      if MN ≥ k && SN ≥ l then
5:          CheckCloakingBoxValidity(x, y, d_x, d_y)
6:          return cid
7:      end if
8:      MN_v, MN_h, SN_v, SN_h ← #nodes including neighbors ▷ v vertical, h horizontal
9:      if MN_v ≥ k && SN_v ≥ l and MN_v ≥ MN_h then
10:         CheckCloakingBoxValidity(x, y, d_x, d_y)
11:         return cid, cid_v
12:     else
13:         if MN_h ≥ k && SN_h ≥ l then
14:             CheckCloakingBoxValidity(x, y, d_x, d_y)
15:             return cid, cid_h
16:         else
17:             return QuadGridCloaking(k, l, {x, y}, {d_x, d_y}, Parent(cid))
18:         end if
19:     end if
20: end procedure
```

Algorithm 1 shows how the privacy protection via generalization is achieved. The pseudo-code algorithm depicts the actual location-privacy preserving mechanism as we defined it above. The algorithm QuadGridCloaking actually is only a spatial generalization mechanism and neglects the temporal aspect of the queries. The mechanism

uses a hierarchical grid for spatial generalization, where level 0 comprises only one cell, level 1 four cells, level 2 sixteen cells, and so on. These levels are linked such that each cell of a level can be linked to one of the cells on the previous level. Figure 2.9 shows the hierarchy, the grid index table, and the cell object count map, where the grid index table maps object identities, coordinates, and cells, and the cell object count map contains the number of mobile (users) and static (PoI) nodes in a cell.

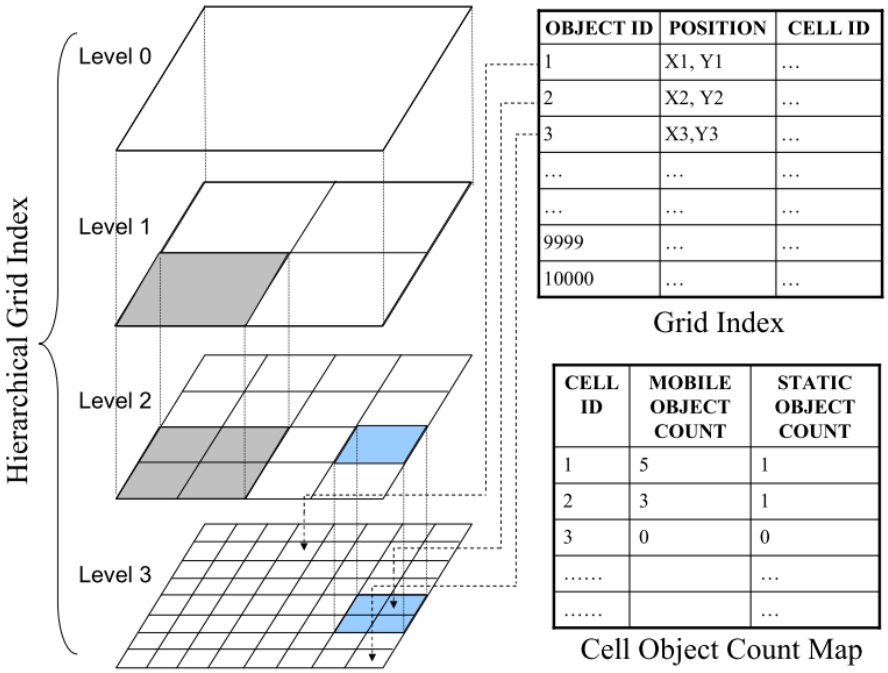


Figure 2.9.: PrivacyGrid grid hierarchy [2]

QuadGridCloaking, starting at the (numerically) highest level, recursively checks whether the current cell already satisfies the user's requirements. If not, the neighboring cell is included and the requirements are checked for the horizontal and the vertical neighbor cell. If any of the two fulfills the requirements, the current cell and its neighboring cell are returned as the region. If both are valid options, the one with the higher resulting k-anonymity is chosen. If both neighbors are not enough, the method QuadGridCloaking is called with the parent cell, i.e., the cell in the next lower level containing the current cell.

As mentioned before, the algorithm QuadGridCloaking does not perform temporal generalization and hence our LP3 model does neither. At this point we repeat our LP3 formalization of PrivacyGrid as depicted in Figure 2.2. We abstract the important parts from the algorithm and omit the implementation details like the hierarchical grid. The relevant facts we take from the algorithmic description are: first, the mechanism is based on finding k-user groups, second, the mechanism searches for a minimum bounding box containing these users, and third, service diversity is also checked. From these three facts, we can derive a model as in Figure 2.2, which includes the grouping of users, a

spatial generalization via minimum bounding box, and a diverse service set.

With the example of PrivacyGrid we demonstrated, on the one hand, the difference between an LPPM and an LP3 and, on the other hand, the process of abstracting from an algorithmic description of a mechanisms to a formalization that uses high-level concepts such as k-anonymity or diversity. In Chapter 5, we will use the LP3 formalization of PrivacyGrid for demonstrating the protocol language of the σ-calculus.

2.6. Location Privacy Guarantees

In our survey of location privacy guarantees we observed that the authors pursue different protection goals, consider different attack scenarios, and have different understandings of location privacy in general. However, the papers share the general setting of LBS communication, where the users commonly share their identity, their location, and the service they want to request. All three of these attributes are personal data or can be linked to personal data. Hence, our taxonomy places all attack scenarios into three basic privacy violation categories: identity revealing, location tracking, and attribute disclosure. However, location tracking can be further subdivided to differentiate between leaking a single location once or continuously leaking spatio-temporal data, i.e., tracing a person's movement over the course of some time. With this distinction the taxonomy comprises four violations: *identity revealing*, *location leaking*, *location tracing*, and *attribute disclosure*. Table 2.2 shows the categories of violations, their brief description, and their relation to the four basic query parameters we derived from the LBS taxonomy: identity, location, service, and time.

Table 2.2.: Categories of privacy violation in LBS

Category	**Description**	**Data Types**
Identity Revealing	linking a query to an identifier	Identity
Location Leaking	linking a query to a location	Location
Location Tracing	linking multiple queries to locations	Location, Time
Attribute Disclosure	linking a query to a service request	Service

We propose the four basic data types as basis for "atomic protection goals." The four data types suffice to express all protection goals of the four categories, as can be seen from Table 2.2. We derived these from our privacy requirements taxonomy and will demonstrate that one can model and verify the most-cited LPPMs from the literature in Chapter 7 using them. Specific protection goals of LPPMs may cover more than just one of the four violation categories. Furthermore, as can be seen from Table 2.2, Location Tracing involves more than just one data type. In Subsection 5.2.1 we will give a formal language for describing privacy requirements. These are directly related to the four atomic data types.

Building on the atomic protection goals, we establish *ground requirements*, which comprise the essential location privacy properties that cover the four protection goals:

F1 *Identity*: a user's identity shall not be revealed to the LBS

F2 *Location*: a user's current location shall not be revealed to the LBS

F3 *Trace*: a user's trip of locations shall not be revealed to the LBS

F4 *Attribute*: a user's attribute, such as preferences, shall not be revealed to the LBS

The four ground requirements **F1**-**F4** directly relate to the four categories of privacy violation in LBS. These natural language requirements are formalized into properties that can be checked with the σ-calculus in Notation 6 (Chapter 5). In Chapter 7 we revisit the results of the survey described in this chapter and perform an evaluation of the σ-calculus based on a set of the most common LPPMs from the literature.

2.7. Summary

This chapter introduces the basic terminology of privacy preservation in location-based services. We perform a survey of LBSs and LPPMs to derive taxonomies of location-based services and protection mechanisms. We define the setting for this dissertation as online privacy protection in query-based LBS. The derived six categories of protection mechanisms are: Generalization, Perturbation, Mix Zones, Dummies, Cryptography, and Caches. We, furthermore, derive location privacy violations from the literature survey in four categories: identity revealing, location leaking, location tracing, and attribute disclosure. From these we formulate the four protection goals against violations as: Identity, Location, Trace, and Attribute. The taxonomies serve as basis for Chapter 5, where we introduce the σ-calculus, a framework that uses the derived categories and corresponding requirements for description languages of LPPMs and privacy properties of this chapter.

3. Theoretical Background

This chapter presents theoretical foundations of the dissertation. We introduce the state of the art of process calculi, modal logics, and model checking as these are the key formal methods necessary to accomplish the contributions of this dissertation. First, in Section 3.1 process calculi in general are introduced and then in Subsection 3.1.1 the focus is laid on the pi-calculus as our σ-calculus is a member of the family of pi-calculi. Next, in Section 3.2 we introduce the family of modal logics with their variety of modalities. Then, in Subsection 3.2.1 and Subsection 3.2.2 temporal and epistemic modalities are presented in more details, respectively. As our logic is a temporal-epistemic (although differing from standard epistemic modal logic) one, reasoning about development of knowledge over time, we introduce these two families of modal logics in more detail. In Section 3.3 the general idea of model checking is introduced as well as different approaches to make model checking of large systems feasible. As we present an explicit-state model checking tool in this dissertation, we focus on this category of model checking approaches but also present model checking approaches which avoid explicit state exploration. Finally, in Section 3.4 we give an introduction to satisfiability modulo theories and present basic approaches of solvers, as our model checking tool uses an SMT solver.

3.1. Process Calculi

Process calculi are a category of formal methods to model concurrent systems. Process calculi typically describe the behavior of such a system. Order of execution in concurrency and timing are of special interest where communication of different agents, components, or processes has to be considered [75].

Historically, process calculi were introduced as "process algebras" and started in the 1970s with the work of Hans Bekič, who aimed at developing an "algebra of processes" [75]. In 1974 he had already established parallel composition, sequential composition, and the null process – today standard building blocks of process calculi. Yet the first complete theory for process algebra goes back to Robin Milner who published a book called *A calculus on communicating systems* [76] in 1980 introducing the Calculus of Communicating Systems (CCS), which he developed and published papers about from 1973 to 1980 [75]. CCS is the first process calculus. In the beginning it is built on denotational semantics and the syntax includes a sequential composition operator $*$, an alternative composition operator ?, and a parallel composition operator $||$. In the final version as described in his book, Milner introduces the syntax that is today standard for most process calculi and includes $+$ as choice operator and $|$ as parallel operator. Sequential composition, however, was omitted in favor of action prefixing $a.P$, where an action is executed and then the process continues as P [77]. CCS still lacked communication, which is today a standard feature of process calculi. Table 3.1 shows the complete syntax of CCS as described in Milner's 1980 book [76].

Table 3.1.: Syntax of the calculus of communicating systems (CCS)

$$P ::= 0 \mid a.P_1 \mid A \mid P_1 + P_2 \mid P_1|P_2 \mid P_1[b/a] \mid P_1 \setminus a$$

The inactive process 0 is the null process as already introduced by Bekič, a denotes an action, the process identifier A allows the definition of processes and even enables recursion, + denotes the choice between two processes, | the parallel composition of two processes, $[a/b]$ denotes the renaming, where all actions a in the process are renamed as b, and finally $\backslash a$ denotes the restriction, where the process executes without the action a.

Another very influential process calculus was introduced by Tony Hoare in a 1978 paper [9]. The paper presents a process calculus named Communicating Sequential Processes (CSP) and introduced synchronous channel communication to the field of process algebras [75]. Generally, as the name suggests, CSP is focused on communication, which is visible from the syntax. Hoare introduced input (?) and output (!) operators to exchange local values between processes. The original syntax of CSP included repetition, choice, parallelism, guarded processes, and events including names and input/output operations. CSP was the first process calculus to incorporate communication into process calculi. However, it was still limited in its concurrency as guards could not occur in parallel processes. Table 3.2 shows a relevant excerpt of the syntax as proposed by Hoare in 1978.

Table 3.2.: Syntax excerpt of CSP [9]

$$\begin{aligned}
\textit{<command>} &::= \textit{<simple command>} \mid \textit{<structured command>} \\
\textit{<simple command>} &::= \textit{skip} \mid \textit{<process name>?<target variable>} \\
&\quad \mid \textit{<process name>!<expression>} \\
\textit{<structured command>} &::= \textit{<alternative command>} \mid * \textit{<alternative command>} \\
&\quad \mid \textit{<process>}||\textit{<process>} \\
\textit{<alternative command>} &::= \textit{<guarded command>}\square\textit{<guarded command>} \\
\textit{<guarded command>} &::= \textit{<guard>} \rightarrow \textit{<command list>}
\end{aligned}$$

The command *skip* denotes the terminating null process, ? and ! are the input and output commands as mentioned above, the operator $*$ denotes the repetition (not of parallel processes), || parallel processes, $\square$ the choice between two (guarded) commands, and finally $\rightarrow$ denotes the guarded commands, where a sequence of commands is only executed if the guard holds.

In 1984 Bergstra and Klop introduced the algebra of communicating processes

(ACP) [78]. As the name of ACP already suggests, Bergstra and Klop laid a focus on the aspect of algebras by formalizing processes via structured axioms of binary operators [75]. The syntax of their process calculus includes communication similar to CSP but also choice, sequencing, and concurrency with strict algebraic rules. The algebra is defined in an axiomatic way, i.e., axioms describe all operators. ACP was the first completely axiomatic process calculus but suffered from a focus on actions limiting the communication to sequential actions (possibly on shared variables). Table 3.3 shows all operators of ACP as introduced in the 1984 paper [78].

Table 3.3.: Syntax of the algebra of communicating processes (ACP)

$$x, y ::= x + y \mid x \cdot y \mid x||y \mid x|\lfloor y \mid x|y$$

The syntax of ACP allows choice ($+$) and sequencing ($\cdot$) of processes, where the original syntax did not use the infixed operator for sequencing. Furthermore, the concurrency operator $||$ called *merge* enables parallel composition of processes. The auxiliary operator $|\lfloor$ called *left-merge* is the deterministic parallel composition where the left process is chosen first. Communication is done in ACP with the interaction operator $|$ that synchronizes actions similar to input and output in CSP. The semantic definition of the communication can be seen in the axiom CM7 from the original paper: $(ax|by) = (a|b)(x||y)$ [78]. For two actions a, b and two processes x, y, this axiom shows that the communication between $a \cdot x$ and $b \cdot y$ is equal to the synchronization of the two actions followed by the merge of the two processes.

In summary, the three process calculi CCS, CSP, and ACP are considered the foundation of all process calculi. While Milner introduced with CCS the first process calculus, Hoare added communication to them with CSP, and Bergstra and Klop introduced an algebraic axiomatization of process calculi with their ACP [75]. Like most modern process calculi, our σ-calculus is a variation of the pi-calculus (introduced in the following subsection). The previous three calculi can be considered the origins of process calculi, while the pi-calculus is the standard basis for modern process calculi. The semantics of the pi-calculus and also of our σ-calculus make use of structural operational semantics (SOS).

To understand the general concept of semantics, we briefly introduce the terms *semantics* and *SOS*, which are used not only for process calculi but can be used for the definition of many types of languages. Semantics, in general, give meaning to syntax of languages and in computer science one differentiates between three types of semantics for formal languages: *denotational*, *operational*, and *axiomatic* semantics [79]. Informally, denotational semantics describe only the effect of executing syntactical constructs. Axiomatic semantics use assertions to describe specific properties of the effect of executing constructs. Operational semantics describe how the effect of an execution is achieved. Structural operational semantics (also called small-step semantics) are a form of opera-

tional semantics where the behavior of a bigger structure is described by the behavior of its parts [80]. For this dissertation we use SOS for defining the reduction of processes described as state transitions. This is a common application of SOS where a transition relation is specified in the form of inference rules.

3.1.1. Pi-Calculus

After the introduction of the origins of process calculi, we focus in this subsection on a (slightly) newer branch of process calculi, which is the state of the art on which most current research is based on. In 1992 Milner, Parrow, and Walker introduced the pi-calculus, which can be seen as an extension of Milner's process calculus CCS. The pi-calculus is a family of process calculi that deals with dynamic communication where the communication channels do not have to be fixed at communication start [81]. The pi-calculus introduced two major changes compared to previous process calculi: first it introduced names as identifiers for both, communication links (channels) and the communicated information, as opposed to differentiating between typed variables and constants as done in previous process calculi like CSP, and second it enabled the use of higher-order functions for communication of communication links. In this context higher-order communication means that a process P can share c, which is the communication channel with process Q, with another process R. In this case P shares its communication with Q with process R. Table 3.4 shows the syntax of the pi-calculus as described in the author's 1992 paper "A calculus of mobile processes, part I" [81].

Table 3.4.: Syntax of the pi-calculus

$$
\begin{aligned}
P ::= & \, 0 \\
& \mid P_1 + P_2 \\
& \mid \bar{y}x.P \\
& \mid y(x).P \\
& \mid \tau.P \\
& \mid P_1 | P_2 \\
& \mid (x)P \\
& \mid [x = y]P \\
& \mid A(y_1, ..., y_n)
\end{aligned}
$$

In more detail, the syntax of the pi-calculus as described in Table 3.4 includes the names x, y, the process P, and the agent A. The terminating null process 0 and the choice operator + are the same as in CCS. The communication is done via names acting as channels, where $\bar{y}$ is called negative prefix and can be considered an output, while $y()$ is called positive prefix and corresponds to an input. The syntax already shows the

mentioned higher order of communication, i.e., both, the channel and the content, are names, enabling the transmission of communication channels. This differentiates the communication of the pi-calculus from that of CSP and ACP, where only variables and actions could be transmitted, respectively. The silent prefix τ executes the silent action and then behaves like process P. Even though this action does nothing, the process $\tau.P$ differs from process P in its concurrency behavior when another process is synchronized with it as the silent action τ is executed. The parallel composition operator | looks similar to the parallel operator | in CCS but in the pi-calculus it enables synchronization of input and output of concurrent processes. The restriction $(x)P$ denotes the process P where all uses of x as a channel are prohibited. The match operator $[x = y]$ evaluates to true if the two names are identical and only then continues with process P and 0 otherwise. Finally, the pi-calculus allows the so called defined agents $A(y_1, ..., y_n)$ to give names to certain processes, very similar to the introduced process identifier A in CCS. The difference is that the syntax of the pi-calculus allows for a set of names, which are the only names the defined process may introduce, while in CCS no such restriction is possible. Like in CCS, the defined agents allow recursive definitions $A(y_1, ..., y_n) \stackrel{\text{def}}{=} P$, where P contains the agent identifier A.

The semantics of the pi-calculus are defined with a labeled transition system and structural inference rules. The reduction of processes uses substitutions $\{x/y\}$, which are not part of the syntax of processes as opposed to the notation $[a/b]$ in CCS. The substitution $\{x/y\}$ denotes the substitution of x for all free occurrences of y. Milner, Parrow, and Walker introduced the notion of *strong bisimulation* of processes and a congruence relation on processes called *strong equivalence* in their 1992 paper "A calculus of mobile processes, part II" to reason about properties of processes [82]. The authors define strong bisimilarity of processes via a strong simulation relation that simulates transitions from a process to another. Strong bisimilarity $\dot{\sim}$ is not preserved by substitution of free names and, therefore, the authors introduced the strong equivalence relation $\sim$ such that $P \sim Q$ if $P\sigma \dot{\sim} Q\sigma$ for all substitutions σ. The strong equivalence relation can be used to check whether a complex process P is equivalent to a simple process Q, which satisfies a certain property in an obvious way.

In summary, the pi-calculus enables the reasoning about concurrent processes with higher-order synchronized communication via dynamic channels. It combines the original idea of CCS with the communication aspect of CSP and ACP and adds the transmission of channel names. The pi-calculus is described by small-step operational semantics in form of structural inference rules for transitions (reduction of processes).

3.1.2. Applied Pi Calculus

The applied pi calculus is an extension of the pi-calculus that can be used to model security protocols. It adds a rich term algebra to allow the description of cryptographic primitives such as hashes or encryption and decryption [3]. The rich term algebra consists of a signature Σ, which is a set of function names, and an equational theory E, which is a set of equations describing the interpretation of the functions. The applied pi calculus removes the higher-order communication of the pi-calculus by introducing

dedicated channels, which cannot be communicated. The applied pi calculus is used to model a variety of real-world security protocols and the tool ProVerif [83] has been used to verify security properties such as secrecy but also forms of privacy [3]. The semantics of the pi-calculus generally use small-step operational semantics based on the reduction and structural equivalence of processes. Privacy-like properties are usually based on the epistemic principle of possible worlds, which describes the approach that knowledge is expressed as complete and consistent 'worlds' where each worlds corresponds to one possible, consistent truth evaluation. In this context possible worlds are processes that are equivalent modulo some relation that is based on the attacker's knowledge (observations).

In more detail, Table 3.5 shows the full syntax of the applied pi calculus. As mentioned before, the syntax shows that the applied pi calculus reintroduces a distinction between names and variables and, thus, does not support transmission of channels. Another difference compared to the pi-calculus is the addition of function applications in terms, or the term algebra in general. In comparison, the syntax of processes replaces choice and match of the pi-calculus with a conditional choice based on terms (if $M = N$ then P else Q). Input and output look similar but are defined via terms and variables, replacing the generic names of the pi-calculus. Name restriction ($\nu n.P$) introduces a new name in the scope of process P similar to the restriction ($(x)P$) of the pi-calculus, where the communication via the channel x is restricted. The applied pi calculus introduces replication ($!P$), which is not present in the original pi-calculus. The syntax of the pi-calculus can express similar processes with recursively defined agents, which were omitted in the applied pi calculus.

The applied pi calculus also uses active substitutions, like the pi-calculus, but introduces for these the *extended processes* A. The syntax of extended processes is depicted in Table 3.6. Extended processes capture the knowledge of an adversary, where the active substitutions model the attacker's intercepted terms.

As mentioned above, the applied pi calculus is used to model security protocols with message exchanges like handshake protocols. Figure 3.1 shows process P describing a handshake protocol formalized in the applied pi calculus.

$$
\begin{aligned}
P &\triangleq \nu sk_S.\nu sk_C.\nu s.\text{let } pk_S = \mathsf{pk}(sk_S) \text{ in let } pk_C = \mathsf{pk}(sk_C) \text{ in} \\
&\quad (\bar{c}\langle pk_S\rangle \mid \bar{c}\langle pk_C\rangle) \mid !P_S \mid !P_C) \\
P_S &\triangleq c(x\_pk).\nu k.\bar{c}\langle \mathsf{aenc}(x\_pk, \mathsf{sign}(sk_S, k))\rangle.c(z).\text{if } \mathsf{fst}(\mathsf{sdec}(k, z)) = tag \text{ then } Q \\
P_C &\triangleq c(y).\text{let } y' = \mathsf{adec}(sk_C, y) \text{ in let } y\_k = \mathsf{getmsg}(y') \text{ in} \\
&\quad \text{if } \mathsf{checksign}(pk_S, y') = \mathsf{true} \text{ then } \bar{c}\langle \mathsf{senc}(y\_k, \mathsf{pair}(tag, s))\rangle
\end{aligned}
$$

Figure 3.1.: Handshake protocol in applied pi calculus [3]

The handshake protocol, modeled as process P, describes the exchange of messages between a client, modeled as process P_C, and a server, modeled as process P_S, where a

Table 3.5.: Syntax of the applied pi calculus [3]

$M, N :=$	terms
n, u	name
x	variable
$g(M_1, ..., M_l)$	function application
$P, Q :=$	(plain) processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction
$\text{if } M = N \text{ then } P \text{ else } Q$	conditional
$u(x).P$	message input
$\bar{u}\langle M \rangle.P$	message output

Table 3.6.: Syntax of extended processes in the applied pi calculus [3]

$A, B :=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{M/x\}$	active substitution

secret s shall be securely transmitted from the client to the server. The handshake is based on a public key infrastructure, where asymmetric keys are used for the encryption and signing of messages. The protocol uses the construct "let $x = M$ in P", which is syntactic sugar for the extended process $\nu x.(\{M/x\}|P)$, where the variable x in P can be used for the term M.

In more detail, the process P starts with constructing two new private keys sk_C and sk_S for client and the server, respectively. Then the corresponding public keys $\mathsf{pk}(sk_C)$ and $\mathsf{pk}(sk_S)$ are output on a public communication channel c (via the output action $\bar{c}$). Finally, the process P replicates the processes P_S and P_C representing client and server sessions. The outputting and replication are executed in parallel to enable communication between a server and clients which know the server's public key. The server process P_S starts with receiving a client's public key (locally naming it x_pk). Then the server selects a key k and outputs an encrypted version of k, signed with its private key and encrypted with the just received public key of the client ($\mathsf{aenc}(x\_pk, \mathsf{sign}(sk_S, k))$). The server, then, awaits input from the client. The client process P_C does not receive the public key of the server (as it is assumed that a client knows the server and only handshakes with this specific server, while the sever initiates a handshake with every client). The client waits for the encrypted, signed key k and decrypts the ciphertext with its private key ($\mathsf{adec}(sk_C, y)$). The client, then, retrieves the key from the plaintext message ($\mathsf{getmsg}(y')$) and, if the signature was in fact performed using the server's private key ($\mathsf{checksign}(pk_S, y') = \mathsf{true}$), the client sends the secret s (paired with a tag for the server to verify integrity) encrypted with the just received symmetric session key ($\mathsf{senc}(y\_k, \mathsf{pair}(tag, s))$). Finally, the server receives this ciphertext and, after decrypting it with the session key, verifies the tag ($\mathsf{fst}(\mathsf{sdec}(k, z)) = tag$) and then concludes the handshake with an unspecified process Q.

$$
\begin{aligned}
\mathsf{fst}(\mathsf{pair}(x, y)) &= x \\
\mathsf{snd}(\mathsf{pair}(x, y)) &= y \\
\mathsf{sdec}(x, \mathsf{senc}(x, y)) &= y \\
\mathsf{adec}(x, \mathsf{aenc}(\mathsf{pk}(x), y)) &= y \\
\mathsf{getmsg}(\mathsf{sign}(x, y)) &= y \\
\mathsf{checksign}(\mathsf{pk}(x), \mathsf{sign}(x, y)) &= \mathsf{true}
\end{aligned}
$$

Figure 3.2.: Equational theory of the Handshake protocol [3]

Figure 3.2 shows the second part of the protocol as the applied pi calculus requires a term algebra to describe the functions used in the protocol. This term algebra is called an equational theory E based on the signature Σ, which is the set of functions used in the terms of the protocol. The signature of the handshake protocol is $\Sigma_H = \{\mathsf{pk}, \mathsf{aenc}, \mathsf{sign}, \mathsf{fst}, \mathsf{sdec}, \mathsf{adec}, \mathsf{getmsg}, \mathsf{checksign}, \mathsf{senc}, \mathsf{pair}, \mathsf{true}\}$. The corresponding equational theory E_H (depicted in Figure 3.2) is the smallest collection of equations

describing the functions of Σ_H. For instance, the equation $\mathsf{fst}(\mathsf{pair}(x, y)) = x$ uses the variables x and y to state that the first (fst) element of a pair is the first parameter of the function pair. Other functions like encryption and decryption are described in a similar way. For example, the equation $\mathsf{sdec}(x, \mathsf{senc}(x, y)) = y$ describes symmetric decryption of an encrypted message, while $\mathsf{adec}(x, \mathsf{aenc}(\mathsf{pk}(x), y)) = y$ describes asymmetric encryption, where a public key is derived from the private key ($\mathsf{pk}(x)$). Equational theories are needed for term evaluation, either in conditional processes or when reasoning about observational equivalence of processes.

The reasoning about secrecy properties in the applied pi calculus is done via adversarial processes. If one wants to verify that, for instance, the above handshake protocol satisfies secrecy, i.e., secret s is not leaked to an adversary, an adversarial process I has to be found such that $P|I$ can be reduced to a process that outputs s on a public channel. If this is the case, the protocol does not satisfy the secrecy property. With active modeling of adversary processes, not only passive eavesdropping attackers but also active man-in-the-middle adversaries can be modeled. The applied pi calculus is an extension of the pi-calculus that focuses on simple communication without higher-order functions but adds a term algebra to enable semantic relations of terms.

3.2. Modal Logics

Modal logic goes further than traditional predicate or propositional logic and differentiates between different modes of truth. Common examples of such modalities are "necessarily true", "known to be true", "believed to be true", "true in the future" [10]. Of the many different possible modalities the best known ones reason about necessity, possibility, belief, knowledge, or tense.

Historically, the idea of modes of truth goes back to ancient philosophers like Aristotle but formal modal axiomatic systems were introduced in the early 20th century. As most schools of logic, modal logic is a field shared between multiple disciplines, most prominently philosophy and computer science. In the late 1950s, Kripke, Prior, and Hintikka established relational semantics, which – today called Kripke Models – are the formal basis for all modal logics [84].

Before we introduce the semantics of modal logic, we first introduce the typical syntax of a (propositional) modal logic, which differs from a classical propositional logic in two extra operators: $\Box$ and $\Diamond$. The two unary operators correspond to a modality and, hence, express modes of truth, where one is defined to be the dual operator of the other [10]. Table 3.7 shows the syntax of basic modal logic in Backus Naur form (BNF).

Table 3.7.: Syntax of modal logic [10]

$$\phi := \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \longrightarrow \phi) \mid (\phi \longleftrightarrow \phi) \mid (\Box\phi) \mid (\Diamond\phi)$$

The symbols $\bot$ (bottom) and $\top$ (top) are boolean constants that are always false and true, respectively. The symbol p is an atomic propositional formula, the symbol $\neg$ denotes negation, $\wedge$ conjunction, and $\vee$ disjunction of boolean formulas. The binary operators $\longrightarrow$ and $\longleftrightarrow$ denote the implication and bi-implication, respectively. And finally, $\Box$ (box) and $\Diamond$ (diamond) are the two operators that are not part of standard propositional logic. As mentioned before, box and diamond are dual operators and represent the respective modality. For example, in a alethic logic, which studies the possibility of truth, $\Box$ can be read as "is necessarily true", while $\Diamond$ can be read as "is possibly true." The duality of the two operators is defined by the equivalence $\Box\phi \equiv \neg\Diamond\neg\phi$. In the logic of possibility this equivalence can be read in natural language as: "ϕ is necessarily true" is equivalent to "it is not possible that ϕ is not true."

As mentioned before, the semantics of modal logic are based on Kripke models. A Kripke model $\mathcal{M} = (W, R, L)$ is a triple consisting of a set W called universe with elements called worlds, a relation R on W ($R \subseteq W \times W$) called accessibility relation, and a labeling function $L : W \to \mathcal{P}(\mathsf{Atoms})$ [10]. In these possible worlds semantics, as they are often called, a world $w \in W$ represents a possible world, $R(w, w')$ – denoting that $(w, w') \in R$ – represents that world w' is accessible from world w, where the interpretation of worlds and accessibility depends on what is modeled, i.e., the modality. The satisfaction of a formula ϕ is defined via structural induction. Table 3.8 shows the satisfaction relation $x \Vdash \phi$ for a world $x \in W$ and a formula ϕ as defined by the syntax in Table 3.7.

Table 3.8.: Semantics of modal logic [10]

$x \Vdash \top$
$x \nVdash \bot$
$x \Vdash p$ iff $p \in L(x)$
$x \Vdash \neg\phi$ iff $x \nVdash \phi$
$x \Vdash \phi \wedge \psi$ iff $x \Vdash \phi$ and $x \Vdash \psi$
$x \Vdash \phi \vee \psi$ iff $x \Vdash \phi$ or $x \Vdash \psi$
$x \Vdash \phi \longrightarrow \psi$ iff $x \Vdash \psi$, whenever we have $x \Vdash \phi$
$x \Vdash \phi \longleftrightarrow \psi$ iff $(x \Vdash \phi$ iff $x \Vdash \psi)$
$x \Vdash \Box\psi$ iff, for each $y \in W$ with $R(x, y)$, we have $y \Vdash \psi$
$x \Vdash \Diamond\psi$ iff there is a $y \in W$ such that $R(x, y)$ and $y \Vdash \psi$

The semantics of the satisfaction relation $\Vdash$ defines that an atomic proposition p is satisfied in a world x only if it is contained in the label of the world $L(x)$. The boolean operators are satisfied in an intuitive way analog to classic propositional logic. The property $\Box\psi$ is satisfied only if all accessible worlds satisfy ψ, whereas a world satisfies $\Diamond\psi$ only if at least one accessible world satisfies ψ. If we stay with the interpretation

of possibility, $\Box\psi$, meaning ψ is necessarily true, is only satisfied if ψ holds in **all** possible worlds (reachable via relation R). In contrast, $\Diamond\psi$, meaning ψ is possibly true, is already satisfied if ψ holds in one possible world. While the interpretation of worlds, the accessibility relation R, and the operators $\Box$ and $\Diamond$ differ from modal logic to modal logic, the basic semantics do not change: $\Box$ is connected with $\forall_{y \in W}.R(x, y)$ and $\Diamond$ with $\exists_{y \in W}.R(x, y)$.

After introducing the basic syntax and semantics of modal logic, we present two sub-families of modal logics which are of importance for this dissertation. Of the above mentioned modal logic sub-families we show temporal modal logic in Subsection 3.2.1 and epistemic modal logic in Subsection 3.2.2. In the field of epistemic modal logic one is interested in knowledge about truth, while in temporal logic one reasons about the development of truth over time.

3.2.1. Temporal Logics

Temporal logic is a family of modal logics which was first proposed by Arthur Prior in the 1950s [85] and later refined by Hans Kamp in the late 1960s [86]. In temporal logic, truth is not static but dynamic. While "two times two is four" is a constant fact, the proposition "Donald Trump is the president of the USA" is not. This statement is true at time of writing this thesis, but was not true in the 1960s and will (probably) not be true in the near future. This example demonstrates the motivation behind a temporal modality, which describes the dynamic truth value of propositions.

In temporal modal logic, the operator $\Box$ (or G) is interpreted as "will globally be true in the future" and $\Diamond$ (or F) as "will eventually be true at some point." The duality of the two operators ($\Box\phi \equiv \neg\Diamond\neg\phi$) is defined as in basic epistemic logic and can be demonstrated in natural language: "p will always be true" is equivalent to "it is not the case that p will eventually be false at some point." A more intuitive formulation of this equivalence is $\Box\neg\phi \equiv \neg\Diamond\phi$, which can be expressed in natural language as "ϕ will always be false" is equivalent to "it is not the case that ϕ will eventually be true." Historically, Prior's tense logic (TL) also had a past modality to reason about dynamic truth development in the past. In most modern temporal logics this is omitted as reasoning about temporally related events can be done solely via future when shifting the (temporal) point of view [87].

In practice, temporal logic is a common choice for reasoning about the behavior of computer systems. In the application scenario of computer systems, one usually has a well-defined start time, where no past events have occurred. In the verification of requirements for computer systems, requirements like "a state where p holds will eventually be reached" is a classic liveness property that can be verified on formal models of a system.

The underlying semantics are typically labeled transition systems or automata that have local truth values as states and are based on Kripke models as introduced in the basic modal logic. The interpretation of possible worlds w are possible points in time and the accessibility relation $R(w, w')$ expresses that state w' can be reached after state w.

The two most popular temporal logics for formal verification of software systems are linear(-time) temporal logic (LTL) and computation tree logic (CTL) [10]. In the following, we will first introduce the syntax and semantics of LTL and then show the differences in CTL. Finally, we will briefly introduce CTL* – a combination of those two temporal logics and a common choice for reasoning about knowledge and time. For this dissertation, however, we will introduce a temporal logic that is based on the LTL modalities but omits the other temporal operators and adds a different temporal operator instead.

LTL

Linear-time temporal logic is a temporal modal logic that reasons about the future. The semantic basis is a state transition system, where sequences of states are called a path. The idea behind paths is that the future is linear and a path represents a possible future with sequences of time points (states). Table 3.9 shows the syntax of LTL as proposed by Amir Pnueli in 1977 (in its modern version in BNF) [88].

Table 3.9.: Syntax of LTL [10]

$$\begin{aligned}\phi := \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \longrightarrow \phi) \\ \mid (X\phi) \mid (F\phi) \mid (G\phi) \mid (\phi\, U\, \phi) \mid (\phi\, W\, \phi) \mid (\phi\, R\, \phi)\end{aligned}$$

As can be seen, the first line of Table 3.9 is, again, standard propositional logic syntax. The second line introduces the temporal operators X, U, W, R, while F and G are another name for $\Diamond$ and $\Box$, respectively. The interpretation of the next operator $X\phi$ is that ϕ should hold in a next state, the until operator $\phi\, U\, \psi$ expresses that ϕ should hold until a state is reached where ψ holds, the weak until operator $\phi\, W\, \psi$ expresses the same as the until but a state with ψ needs not necessarily be reached, finally, the release operator $\phi\, R\, \psi$ expresses that ϕ releases ψ, that is, ψ must hold until and including the moment when ϕ holds. The release R is the dual of the until U: $\phi\, R\, \psi \equiv \neg(\neg\phi\, U\, \neg\psi)$. In natural language, this equivalence can be expressed as "ϕ releases ψ" is equivalent to "it is not the case that ϕ is false until ψ is false."

The semantic interpretation of LTL formulas can be seen in Table 3.10. The Kripke model for LTL is $\mathcal{M} = (S, \rightarrow, L)$, where S is a set of states $s \in S$, $\rightarrow \subseteq S \times S$ is a transition relation between states, and L the usual labeling function. The satisfaction of a formula ϕ and a path π is defined via the satisfaction relation $\pi \vDash \phi$. A path is a sequence of states: $\pi = s_1 \rightarrow ...$ and the ith state of a path is denoted by $\pi^i := s_i$. The difference to the semantics of basic modal logic can be immediately seen by the left-hand side of the satisfaction relation: LTL reasons about paths and only indirectly about states. Even though we do not go over the semantics in detail, we point out the satisfaction of G and F, which are, as mentioned in the basic modal logic, also defined using universal and existential quantification, respectively.

Table 3.10.: Semantics of LTL [10]

$\pi \models \top$
$\pi \not\models \bot$
$\pi \models p$ iff $p \in L(s_1)$
$\pi \models \neg\phi$ iff $\pi \not\models \phi$
$\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$
$\pi \models \phi_1 \vee \phi_2$ iff $\pi \models \phi_1$ or $\pi \models \phi_2$
$\pi \models \phi_1 \longrightarrow \phi_2$ iff $\pi \models \phi_2$, whenever we have $\pi \models \phi_1$
$\pi \models X\phi$ iff $\pi^2 \models \phi$
$\pi \models G\phi$ iff, for all $i \geq 1, \pi^i \models \phi$
$\pi \models F\phi$ iff there is some $i \geq 1$ such that $\pi^i \models \phi$
$\pi \models \phi\, U\, \psi$ iff there is some $i \geq 1$ such that $\pi^i \models \psi$ and for all $j = 1, ..., i-1$ we have $\pi^j \models \phi$
$\pi \models \phi\, W\, \psi$ iff either there is some $i \geq 1$ such that $\pi^i \models \psi$ and for all $j = 1, ..., i-1$ we have $\pi^j \models \phi$; or for all $k \geq 1$ we have $\pi^k \models \phi$
$\pi \models \phi\, R\, \psi$ iff either there is some $i \geq 1$ such that $\pi^i \models \phi$ and for all $j = 1, ..., i$ we have $\pi^j \models \psi$, or for all $k \geq 1$ we have $\pi^k \models \psi$

As LTL formulas are evaluated on paths but in practice a requirement should be checked for the system as a whole, the satisfaction of a state is defined by quantifying over all paths starting in this state. For a model $\mathcal{M} = (S, \rightarrow, L)$, a state $s \in S$, and an LTL formula ϕ, $\mathcal{M}, s \models \phi$ if for every path π starting at s, we have $\pi \models \phi$.

CTL

Computation tree logic is a so-called branching time logic, which was introduced in 1981 by Edmund Clarke and Ernest Emerson [89]. Very different from the setting of LTL, in CTL the future is represented by a tree-like structure where each path models a possible future. The motivation for CTL is that in LTL the future is linear and one reasons about all possible paths starting at a state. In CTL the future is not linear but can branch into different futures. For certain types of behaviors it is necessary to express, for example, that it is possible (but not necessary) to eventually reach a specific state, which cannot be formulated in LTL syntax. CTL introduces path quantifiers to model this type of behavior. Table 3.11 shows the syntax of CTL.

The differences in the syntax of LTL and CTL formulas are the quantifiers A and E in front of temporal formulas. The operators known from LTL X, F, G, and U are

Table 3.11.: Syntax of CTL [10]

$$\phi := \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \longrightarrow \phi) \mid AX\phi \mid EX\phi$$
$$\mid AF\phi \mid EF\phi \mid AG\phi \mid EG\phi \mid A[\phi\, U\, \phi] \mid E[\phi\, U\, \phi]$$

preceded by either an A or an E, where A means 'along all paths' (or inevitably) and E means 'along at least one path' (possibly). LTL-like temporal formulas without path quantification are not allowed, e.g., $p\,U\,q$ and GFp are valid LTL syntax but no CTL formulas.

Table 3.12 shows the semantics of CTL via the satisfaction relation $\mathcal{M}, s \models \phi$, where just like in LTL $\mathcal{M} = (S, \rightarrow, L)$ is a model, $s \in S$ a state, and ϕ a CTL formula. The semantics define A and E also as dual operators as they use universal quantification and existential quantification and share the same duality as $\forall$ and $\exists$ in predicate logic, i.e., $\neg A\phi \equiv E\neg\phi$. CTL is called a branching time logic because it quantifies over states and paths as can be seen from Table 3.12, whereas LTL is called linear because it quantifies over states along a path.

We present two example graphs to demonstrate the semantic differences of LTL and CTL (and also show that neither is a superset of the other). Figure 3.3 shows a graphical representation of a model with nodes being states, edges being transitions, and labels being displayed in the nodes. The corresponding model $\mathcal{M}_1$ and the initial state s_0 satisfies the LTL formula $\phi_{LTL} = FG\,p$ but not the CTL formula $\phi_{CTL} = AFAG\,p$. The reason why the model does not satisfy the CTL formula ($\mathcal{M}_1, s_0 \not\models \phi_{CTL}$) is that the semantics of AF require all paths starting at s_0 to eventually satisfy $AG\,p$, yet the infinite path $\pi_1 := s_0 \rightarrow s_0 \rightarrow ...$, which never leaves state s_0, never satisfies $AG\,p$, hence π_1 serves as a counter example for $AFAG\,p$. The infinite path $\pi_2 := s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow ...$ obviously does not satisfy $AG\,p$ as it passes state s_1 where p does not hold. The LTL formula, however, holds for the model ($\mathcal{M}_1, s_0 \models \phi_{LTL}$). The reason for that is that the semantics of LTL are defined on linear paths and only the first temporal operator is quantified over all paths. In more detail, ϕ_{LTL} is satisfied by the model starting at s_0 if all paths starting at s_0 satisfy $FG\,p$. The two example paths π_1 and π_2 (and in fact all other infinite paths of $\mathcal{M}_1$) satisfy $FG\,p$ as all these paths have some index i where π^i is either s_0 or s_2 and the remaining sub-paths starting at π^i all satisfy $G\,p$.

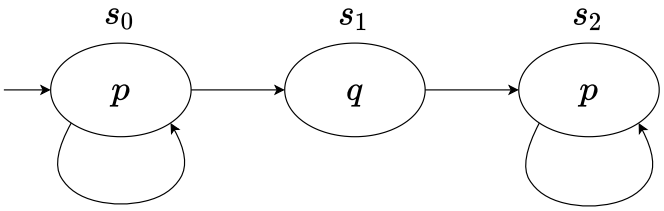


Figure 3.3.: Graph of a labeled transition system which satisfies $FG\,p$

Table 3.12.: Semantics of CTL [10]

$\mathcal{M}, s \models \top$
$\mathcal{M}, s \not\models \bot$
$\mathcal{M}, s \models p$ iff $p \in L(s)$
$\mathcal{M}, s \models \neg\phi$ iff $\mathcal{M}, s \not\models \phi$
$\mathcal{M}, s \models \phi_1 \wedge \phi_2$ iff $\mathcal{M}, s \models \phi_1$ and $\mathcal{M}, s \models \phi_2$
$\mathcal{M}, s \models \phi_1 \vee \phi_2$ iff $\mathcal{M}, s \models \phi_1$ or $\mathcal{M}, s \models \phi_2$
$\mathcal{M}, s \models \phi_1 \longrightarrow \phi_2$ iff $\mathcal{M}, s \models \phi_2$, whenever we have $\mathcal{M}, s \models \phi_1$
$\mathcal{M}, s \models AX\phi$ iff for all s_1 such that $s \to s1$ we have $\mathcal{M}, s_1 \models \phi$
$\mathcal{M}, s \models EX\phi$ iff for some s_1 such that $s \to s1$ we have $\mathcal{M}, s_1 \models \phi$
$\mathcal{M}, s \models AG\phi$ iff for all paths $s_1 \to s_2 \to s_3 \to ...$, where $s_1 = s$ and
for all s_i along the path, we have $\mathcal{M}, s_i \models \phi$
$\mathcal{M}, s \models EG\phi$ iff there is a path $s_1 \to s_2 \to s_3 \to ...$, where $s_1 = s$ and
for all s_i along the path, we have $\mathcal{M}, s_i \models \phi$
$\mathcal{M}, s \models AF\phi$ iff for all paths $s_1 \to s_2 \to s_3 \to ...$, where $s_1 = s$ and
for some s_i along the path we have $\mathcal{M}, s_i \models \phi$
$\mathcal{M}, s \models EF\phi$ iff there is a path $s_1 \to s_2 \to s_3 \to ...$, where $s_1 = s$ and
for some s_i along the path we have $\mathcal{M}, s_i \models \phi$
$\mathcal{M}, s \models A[\phi_1 \, U \, \phi_2]$ iff for all paths $s_1 \to s_2 \to s_3 \to ...$, where $s_1 = s$ and
there is some s_i along the path, such that $\mathcal{M}, s_i \models \phi_2$, and
for each $j < i$ we have $\mathcal{M}, s_j \models \phi_1$
$\mathcal{M}, s \models E[\phi_1 \, U \, \phi_2]$ iff there is a path $s_1 \to s_2 \to s_3 \to ...$, where $s_1 = s$ and
there is some s_i along the path, such that $\mathcal{M}, s_i \models \phi_2$, and
for each $j < i$ we have $\mathcal{M}, s_j \models \phi_1$

Figure 3.4 shows a graphical representation of a model which satisfies a property that can be expressed in CTL but not in LTL. The property $\psi_{CTL} = AGEF\, p$ is satisfied by the model $\mathcal{M}_2$ corresponding to the graph depicted in Figure 3.4. Semantically, $\mathcal{M}_2 \models \psi_{CTL}$ as all paths starting at s_0 can reach a state where p holds at all times. The similar LTL formula $\psi_{LTL} = GF\, p$, however, requires all paths starting at s_0 to eventually reach a state where p holds, for which the infinite path $\pi_3 := s_0 \to s_1 \to s_1 \to ...$ is a counterexample. Hence, the model $\mathcal{M}_2$ does not satisfy the LTL formula ($\mathcal{M}_2, s_0 \not\models \psi_{LTL}$). While simple existential CTL formulas of the form $EF\, p$ can be expressed in LTL via negation ($\neg G \neg p$), nested CTL formulas involving existential quantification cannot be expressed in the language LTL.

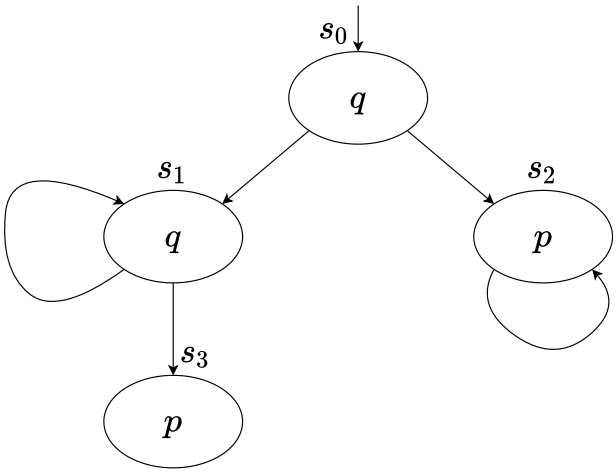


Figure 3.4.: Graph of a labeled transition system which satisfies $AGEF\,p$

CTL*

As we have shown with the above examples, LTL is not a subset of CTL (and vice versa). The temporal language CTL* is a superset of the two that can express all LTL and all CTL formulas and some formulas which cannot be modeled in either of the two languages. Syntactically, CTL* combines unquantified LTL formulas and quantified CTL formulas and thereby allows formulas like $AFG\,p \vee AGEF\,p$. It is an example of a formula, which is neither contained in LTL nor CTL as the formula is a disjunction of the two formulas ϕ_{LTL} (or rather the equivalent CTL* formula $AFG\,p$) and ψ_{CTL}, which we have shown to be each exclusive to only one of the languages. Hence, the disjunction cannot be expressed in either one of the two. The semantics split CTL* formulas into state and path formulas, which correspond to the respective semantics of LTL and CTL. CTL* is a common basis for temporal-epistemic logics as it is more expressive than the two individually and model checking is not more complex than LTL alone.

In summary, temporal modal logics have time-based modalities and interpret Kripke models according to states with a temporal relation. Temporal logics are commonly used for verification of safety or liveness requirements of (concurrent) software systems. Linear and branching interpretations of the future exist and allow for different formalized requirements. Temporal logic extends the syntax of basic modal logic with more than just the two modal operators $\Box$ and $\Diamond$ but the semantics are still based on Kripke models.

3.2.2. Epistemic Logics

Epistemic logic is a sub-family of modal logic that was introduced in 1962 by Hintikka [90]. Epistemic modal logic reasons about knowledge and is often applied in multi-agent systems. In epistemic logic, one does not make statements about truth but rather about what a specific actor thinks to be true. In this interpretation of modal logic, the $\Box$ operator is called K and is usually indexed with an identifier representing an agent:

$K_a p$ expresses that agent a knows the fact p. The dual operator $\Diamond$ of K does not have a standard notation, but $\neg K_a \neg p$ expresses that agent a does not know that p is false, which is equivalent with "p is consistent with a's knowledge" [10]. Sometimes the dual of K is called 'believe' and represented by B, however a belief-based family of modal logics called doxastic logic exist, which is solely dedicated to reasoning about beliefs.

Table 3.13 shows the syntax of epistemic logic. The syntax allows reasoning about knowledge from the individual perspective of all agents $a \in A$, where A is the set of all agents of a system. Furthermore, the syntax allows reasoning about common and distributed knowledge of groups $G \subseteq A$. The formula $E_G \phi$ expresses that every agent in group G knows ϕ, $C_G \phi$ expresses that ϕ is common knowledge to every agent in G, and finally $D_G \phi$ expresses that ϕ is distributed knowledge to every agent in G [11].

Table 3.13.: Syntax of epistemic logic [11]

$$\begin{aligned}\phi := \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \longrightarrow \phi) \\ \mid (K_a \phi) \mid (E_G \phi) \mid (C_G \phi) \mid (D_G \phi)\end{aligned}$$

The semantical differences of common and distributed knowledge can be seen in the satisfaction of formulas as depicted in Table 3.14. Standard Kripke models as described in Section 3.2 are not sufficient for reasoning about multi-agent systems. The Kripke model $\mathcal{M} = (S, L, \mathcal{K}_1, ..., \mathcal{K}_n)$ is an $(n+2)$-tuple, where S is the set of possible worlds (states), L is the label function, and $\mathcal{K}_i$ are the binary accessibility relations for all n agents. From Table 3.14 we can derive that an agent knows that a formula is true, if it is true in all worlds she views as possible (described by the accessibility relation $\mathcal{K}_i$). Furthermore, a formula is known to every agent of a group, if each agent knows the formula to be true individually. In contrast, common knowledge is nested knowledge, where the expression $E_G^k \varphi$ is an abbreviation for k nested E_G formulas, i.e., $E_G^0 \varphi = \varphi$ and $E_G^{k+1} \varphi = E_G E_G^k \varphi$. Common knowledge is, hence, knowledge that every agent knows, every agent knows that every agent knows, and so on. This is an interesting property for some logic puzzles like the muddy children problem [11]. Distributed knowledge is the combined knowledge of all agents of a group and, therefore, requires the intersection of the accessibility relations of all agents of the group. This intersection represents the possible worlds **all** agents agree on.

Table 3.14.: Semantics of epistemic logic [11]

$(\mathcal{M}, s) \models \top$
$(\mathcal{M}, s) \not\models \bot$
$(\mathcal{M}, s) \models p$ iff $p \in L(s)$
$(\mathcal{M}, s) \models \neg\psi$ iff $(\mathcal{M}, s) \not\models \psi$
$(\mathcal{M}, s) \models \psi \wedge \psi'$ iff $(\mathcal{M}, s) \models \psi$ and $(\mathcal{M}, s) \models \psi'$
$(\mathcal{M}, s) \models K_i\psi$ iff $(\mathcal{M}, t) \models \psi$ for all t such that $(s, t) \in \mathcal{K}_i$
$(\mathcal{M}, s) \models E_G\varphi$ iff $(\mathcal{M}, s) \models K_i\varphi$ for all $i \in G$
$(\mathcal{M}, s) \models C_G\varphi$ iff $(\mathcal{M}, s) \models E_G^k\varphi$ for $k = 1, 2, ...$
$(\mathcal{M}, s) \models D_G\varphi$ iff $(\mathcal{M}, t) \models \varphi$ for all t such that $(s, t) \in \bigcap_{i \in G} \mathcal{K}_i$

As epistemic modal logics reason about the knowledge of agents in a system and knowledge can satisfy different properties, categories of properties of knowledge were introduced to describe logics based on axioms [10]. A difference exists between knowing something for certain and only believing it to be true. Hence, the truth axiom, for instance, states the fact that if an agent knows a fact, this fact must be (globally) true. The standard set of axioms for an epistemic modal logic describing knowledge is called **S5**. The set **S5** contains the distribution of knowledge (**K**), the truth axiom (**T**), the positive and negative introspection (**4** and **5**, respectively), and the generalization of knowledge (**N**). The knowledge distribution axiom **K** $((K_i\phi \wedge K_i(\phi \implies \psi)) \implies K_i\psi)$ states the fact that an agent can derive implied knowledge. The truth axiom **T** $(K_i\phi \implies \phi)$ states the fact that if an agent knows a fact, this fact must be true. The positive introspective axiom **4** $(K_i\phi \implies K_iK_i\phi)$ states the fact that agents are aware of their knowledge. Similarly, the negative introspective axiom **5** $(\neg K\phi \implies K_i\neg K_i\phi)$ states the fact that agents are aware of their ignorance. The knowledge generalization axiom **N** (if $\models \phi$ then $M \models K_i\phi$) finally states the fact that if a fact is true in every *possible world* for the agent, then the agent must know this fact in all possible worlds.

In Chapter 5 we will describe the epistemic properties of our so-called "K-less epistemic modal logic" in regard to this set of axioms. This is relevant as we propose a temporal modal logic with dynamic epistemic formulas. Our logic is not a standard modal epistemic logic but still satisfies a subset of **S5**, which we will present in Subsection 5.2.2.

3.3. Model Checking

Model checking (MC) comprises formal verification methods to check whether a model of a system meets a given specification. Usually model checking requires three components:

first, a framework for modeling systems, second, a specification language, and third, a verification method [10]. The first component is typically a description language to formally model the behavior of a software or hardware system. The second component is some form of property language to describe the intended behavior of the system. In standard model checking, temporal logic is the common choice for this property language as one wants to describe a dynamic behavior of the system. The third component is a model checking algorithm that determines whether a modeled system meets the formalized property.

Historically, the idea of model checking was introduced in the 1980s and goes back to the work of Clarke, Emerson, and Sifakis and builds on previous work in the fields of temporal modal logic and formal transition systems and automata, e.g, by Pnueli, Kripke, Turing [91]. Whereas in the beginning the main question in model checking was, how to model systems and build a formal framework to verify properties, today the main research focus of model checking lies on the algorithmic challenge, how to design algorithms that scale to real-life problems. The prominent keyword to this regard is the "state explosion problem," which is the practical problem that Kripke models for real-world systems suffer from a combinatorial explosion of states, i.e., the number of states becomes untractable [91].

In the previous section we have already introduced temporal logic and the satisfaction of temporal logic formulas based on Kripke models. The original model checking problem is to find an algorithm that is, among other properties like efficiency, correct regarding the satisfaction relation, i.e., an algorithm that outputs 'yes' if $\mathcal{M} \models \phi$ and 'no' otherwise. Most model checkers – software tools that automatically verify a formula based on a model in a predefined description language – additionally provide a trace or witness that demonstrates a system behavior that leads to a violation of the specification if the output is 'no.'

As mentioned before, the more recent advances in model checking are often concerned with finding algorithms to deal with the state explosion problem, i.e., efficient algorithms that avoid explicit state space exploration. But also optimizations for explicit-state model checking are currently researched [92]. Several types of model checking algorithms have been proposed and can be placed in three categories: *structural methods*, *symbolic methods*, and *abstraction* [91]. Furthermore, one can also differentiate between *explicit-state model checking* and techniques which avoid explicit state exploration, like *bounded model checking* or *symbolic model checking*. In the following subsections, we first introduce explicit-state model checking approaches and then the remaining approaches. The sections give an overview of techniques to efficiently check models of (real-world) systems.

3.3.1. Explicit-State Model Checking

Explicit-state model checking is the first approach of model checking that was introduced in the 1970s and has its origins in reachability analysis techniques [92]. As the name suggests, explicit-state MC explores a complete state space, where the classical approach is a breadth-first or depth-first search. The explicit-state model checker SPIN [93], for

example, checks LTL properties by converting an LTL formula into a Büchi automaton that only accepts execution traces for which the LTL formula is satisfied [92]. A Büchi automaton is an infinite automaton $B = (S, s_0, L, T, F)$, where S is a finite set of states and $s_0 \in S$ the initial state, L is a label set, $T \subseteq S \times L \times S$ is a set of transitions, and $F \subseteq S$ is a set of accepting states. A Büchi automaton accepts an execution trace if it contains infinitely many accepting states from set F. The model checking approach based on Büchi automata, which is implemented in SPIN, translates a software system into one (or multiple) finite automaton, translates the LTL formula into a Büchi automaton, and then computes the product of the automata. If in the resulting Büchi automaton an accepting run exists, the LTL formula holds.

In recent years, explicit-state model checking is often used for the verification of software systems with asynchronous processes as these are suited for *partial-order reduction*, which limits the number of states to be explored. Partial-order reduction is a structural MC technique, which is one of the above mentioned categories of MC approaches.

Structural Model Checking

Structural model checking methods comprise all algorithms that exploit the structure of the syntactic definition of the system, e.g., modular structures like subroutines or parallel components or threads. The basic idea is to reduce the model by using this structure of the system to break the system down in parts or omit redundant parts. This makes structural approaches oftentimes application-specific but a number of known general techniques exist: symmetry reduction, partial-order reduction, on-the-fly state-space exploration, assume-guarantee reasoning, and parametric verification [91].

Symmetry reduction is an approach where the states of a model (e.g., Kripke structure) are partitioned into equivalence classes (called *orbits*) and then a reduced model containing representative states from each orbit can be used for checking properties with a much smaller state space that has to be explored explicitly [94]. Partial-order reduction exploits the property of concurrent models that the order of independent events does not matter and, similarly to symmetry reduction, reduces the state space by selecting a *persistent* subset of states and transitions from the original model sufficient for a given property [91]. On-the-fly state-space exploration is an approach where the temporal logic formula automaton is constructed simultaneously with – and guided by – the generation of the system model. This approach can be used to efficiently show that a property does not hold without constructing the whole model [95]. Assume-guarantee reasoning is a technique for model checking complex, modular systems. When using a divide-and-conquer approach to check correctness of smaller parts of the system, assume-guarantee reasoning uses correctness guarantees of already checked components to assume properties of other parts to reason about dependent components without modeling the whole system [91]. Parametric verification is a model checking approach where models are described via parameters. For instance, large concurrent systems consisting of an unknown number of identical concurrent processes can be checked without knowing the parameter that describes the number of processes. One possible parametric verification approach is *regular model checking* where configurations of the system are represented by words

(and each symbol corresponds to a state of the system) and sets of configurations can be represented by regular expressions. Transitions are then modeled by finite automata operating on regular sets [96].

3.3.2. Symbolic Model Checking

Symbolic model checking is a popular approach to deal with the state explosion problem. While explicit-state model checking approaches are based on explicit enumeration of states and transitions, symbolic model checking uses symbolic encodings. These encodings can be binary decision diagrams (BDDs), propositional formulas, or quantifier-free first-order constraints. BDDs were the standard approach for symbolic model checking since the 1990s but in more recent years other state encodings have surfaced [91]. In the following we introduce the basic principles of BDD-based model checking and SAT-based model checking as these two are the most common approaches today.

The idea of BDD-based model checking is to encode states as boolean vectors and transitions as boolean functions. This enables the modeling of Kripke models as ordered binary decision diagrams (OBDDs), which allow for cheap boolean operations, such as negation, conjunction, or disjunction, and fast traversal. Hence, OBDDs are a very efficient data structure for symbolically representing a huge state space [97]. Model checkers which use OBDDs, like *SMV* [98] (Symbolic Model Verifier) and its newer version *NuSMV* [99] (New Symbolic Model Verifier), usually provide a syntax for modeling the system in a modular way. NuSMV takes as input a program describing the system, and LTL formulas describing the specified behavior and returns the verification result, either 'true' or a trace as counterexample.

The motivation of SAT-based model checking is that complex real-world models have cycles and counterexamples for LTL formulas often contain cycles, making these traces so-called lasso-traces, i.e., an infinite sequence with a finite start-sequence ending in a cycle. The existence of these lasso-shaped counterexamples can be expressed in propositional logic as boolean constraints. With the help of a SAT solver the existence of such counterexamples can be disproved. The recent advances in SAT solving made this approach an alternative to BBD-based checking. Initially, SAT solvers could only help with so-called *bounded* model checking where model checking was only complete up to a certain number of steps.

Bounded model checking, while not a symbolic approach, can be combined with symbolic approaches and is one of the most popular model checking approaches in practice that can be combined with others [100]. The general idea is to find counterexamples ('falsification' instead of verification) in a bounded number of steps to avoid exploring the whole state space (or a symbolic representation of it). Additionally, by iteratively increasing this bound the check either terminates when an error is found or all states have been checked [100].

3.3.3. Abstraction-Based Model Checking

Abstraction-based model checking is based on reducing models by finding homomorphic models, which preserve desired properties of the original (but otherwise are overapproximations). The most prominent approach of this category is called *counterexample-guided abstraction refinement* (CEGAR), where, similar to bounded model checking, in an iterative process increasingly precise (abstracted) models are used to produce counterexamples [91].

As the example of symbolic model checking and bounded model checking shows, the model checking approaches are not completely disjoint. Symbolic model checker, for example, can benefit from many optimizations for explicit-state model checking. Especially modern symbolic model checking tools often combine approaches to make the verification more efficient. NuSMV, for instance, combines OBDD-based model checking with a bounded SAT-based approach. The on-the-fly model checker SPIN offers partial-order reduction to speed up the verification, combining two structural approaches.

3.4. Satisfiability Modulo Theories

Satisfiability (SAT) modulo theories (SMT) solvers make boolean decisions for first-order logical formulas. In contrast to classical SAT solvers, SMT includes background theories that allow satisfiability checks of formulas that quantify over variables other than boolean, e.g., integers, bit vectors, or lists, with respect to the corresponding background theories, like standard interpretation of '+' or '<' [101]. The SMT problem is the question whether (a series of) predicate formulas are satisfiable, i.e., whether a value assignment exists for which all formulas hold.

Two major implementation approaches for SMT solvers exist: *eager* and *lazy*. The *eager* approach for solving SMT problems is to translate them into SAT problems by converting the formulas to "equisatisfiable" propositional formulas, i.e., propositional formulas that are satisfiable if and only if the original formulas are satisfiable [101]. For example, integers can be represented by an appropriate number of individual bit variables, where the corresponding integer arithmetic is replaced by single-bit operators. While the *eager* approach enables the use of any already existing SAT solver, it can create inefficient SAT formulas as the high-level semantics of the underlying theories are lost when dividing variables into individual bits [101]. The category of *lazy* SMT approaches comprises SMT solvers that use background theory-specific inference rules and data structures specialized for the theory. *Lazy* approaches usually perform better than *eager* ones [101].

For our model checking we use a *lazy* SMT solver called CVC4 [102], which – as it is SMT standard – lets users select a subset of all supported theories when starting, as restricting the solver to theories which are necessary increases the performance. CVC4 supports the standard SMT input language format SMT-LIB [103]. SMT-LIB allows the declaration and definition of sorts (custom types derived from loaded background theories), functions, and variables. The actual SMT formulas are expressed as assertions and, finally, a SAT command can be used to check whether the assertions are satisfiable.

If so, a value assignment can be requested that satisfies all predicate formulas. If the set of assignments is not satisfiable, an unsatisfiability core can be requested which contains a subset of the formulas which are in conflict.

Figure 3.5 shows an example SMT program and its output as returned by CVC4. The program starts with the command *set-logic* to specify the subset of theories to use for this run. "QF_UFLIA" is the SMT_LIB identifier for the theory which is quantifier-free (QF), allows free sorts and function (UF), and includes linear integer arithmetic (LIA). The second line enables the command *get-value*, which is used in line 15. In lines 4 to 6 three variables x, y, t are declared. They are integer vectors, as they are declared as functions with one integer parameter and integer return type. In SMT all variables are functions and a function with no parameter is a constant. In the following, we use indexed variable notation x_i to refer to the ith position in vector x. Lines 8 to 10 define three assertions about the variables, which can be expressed as the three equations $t_0 = x_0$, $y_1 = t_0$, and $x_1 = y_1$. Line 12 is another assertion, which expresses the boolean property $\neg(x_1 = y_0 \wedge y_1 = x_0)$. In line 14 the command *check-sat* is used to check whether the four assertions are satisfiable modulo the selected theory. Finally, in line 15 the command *get-value* is used to get a valuation for the four variables x_0, y_0, x_1, and y_1 which satisfies the assertions. Lines 17 and 18 with a leading ';' show comments in SMT_LIB syntax and present the actual result for this program returned by CVC4. The format of value assignments is a list of tuples where each tuple represents a variable and its corresponding value. The output can be read as $x_0 = -1, y_0 = 2, x_1 = -1, y_1 = -1$, which is an assignment that satisfies the three equations in lines 8 to 10 and the negated conjunction in line 12. For variable t_0 no assignment is returned as line 15 does not explicitly request it (per assertion in line 8, $t_0 = -1$). The program actually demonstrates that the modeled swap method, where the values of x and y are to be swapped, is not correct as the negated property in line 12 checks the expected result after the swap $x_1 = y_0 \wedge y_1 = x_0$. The bug is in line 10, where "(y 0)" instead of "(y 1)" would be the expected assertion. However, it should be said that this very simple program is not sufficient for testing the swap method. The assignment $x_0 = y_0$ would still be a counter example for the bug-fixed version. An additional assertion of the form `(not (= (x 0) (y 0)))` would be necessary to make sure that x and y are not equal.

3.5. Summary

In this chapter, we introduced three areas of formal methods: process calculi, modal logics, and model checking. These three fields are the theoretical foundation of this dissertation. As described in Section 3.3, the process of model checking requires formal models, a specification language, and a model checking algorithm. In the introduction in Chapter 1, we motivated that the aim of this dissertation is a framework that enables the verification of location privacy of location-privacy preserving mechanisms. One can achieve this goal with a model checking approach. Over the course of this dissertation, especially in Chapter 5 and Chapter 6, we describe a model checking approach, where a

```
(set-logic QF_UFLIA)
(set-option :produce-models true)

(declare-fun x (Int) Int)
(declare-fun y (Int) Int)
(declare-fun t (Int) Int)

(assert (= (t 0) (x 0)))
(assert (= (y 1) (t 0)))
(assert (= (x 1) (y 1)))

(assert (not (and (= (x 1) (y 0)) (= (y 1) (x 0)))))

(check-sat)
(get-value ((x 0) (y 0) (x 1) (y 1)))

; valuation returned by CVC4:
; (((x 0) (- 1)) ((y 0) 2) ((x 1) (- 1)) ((y 1) (- 1)))
```

Figure 3.5.: SMT program in SMT-LIB syntax

process calculus related to the applied pi calculus is used to describe privacy preserving protocols, temporal modal logic formulas with epistemic propositions form the specification language, and an explicit-state algorithm with structural reduction performs the verification using an SMT solver to check knowledge bases for consistency. We introduced the concept of SMT solvers in this section to give the necessary background not only for our theory but also for the tool chain of our implementation. To this end, we also gave a small example of the SMT-LIB syntax and the output format of the solver CVC4.

4. Related Work

In the previous chapter, we have introduced the basic concepts of the respective fields of process calculi, modal logics, and model checking. This chapter builds on the introduced terminology and concepts and gives an overview of previous research related to this dissertation. In section 4.1, we present and discuss related work on research covering similar applications. As this dissertation presents the σ-calculus, a framework that enables the verification of location privacy, an important field are papers that propose formal methods to evaluate privacy in general where we lay a special focus on location privacy. In the following sections, Section 4.2 and Section 4.3, we present an overview of related work with comparable methods. Section 4.2 introduces and discusses research that uses process calculi to model protocol-like communication. Finally, Section 4.3 gives an overview of research in the field of modal logics, focusing on verification with temporal and epistemic logics.

4.1. Location Privacy Verification

The verification of location privacy is commonly statistics-based and usually one reasons about probabilistic properties. Statistics are often used for maps where regions and transitions have probabilities assigned. Probabilities can also be used to describe individual user profiles, i.e., regions and transitions have specific probabilities for a user. These statistics help modeling an adversary with background knowledge and enable calculating success rates for attacks on specific users. Most research done in the field of location privacy verification uses statistics [70, 104, 105]. In contrast, the σ-calculus which we present in this dissertation is not a statistics-based process calculus. Our proposed logic reasons about the binary decision whether data can leak for certain or not. The σ-calculus follows a worst-case assumption where a scenario is considered as a privacy violation if any user's privacy could be violated in the worst case. This is a more general evaluation of protocols, which does not rely on single users' history or specific knowledge of adversaries. In the following we will argue why we cannot simply adopt existing verification approaches, e.g., by mapping probabilities to 0 and 1. First we introduce the respective research, then compare their approach to our σ-calculus, and finally we explain why we cannot use the respective verification approach.

While many researchers propose location-privacy preserving mechanisms, only a few authors publish in the field of verifying location-privacy guarantees. The author with the most research published in this field is Reza Shokri who wrote his PhD thesis on "Quantifying and Protecting Location Privacy" in 2015 [106]. Shokri proposed several approaches for measuring location privacy [70, 107–109]. In their 2011 paper "Quantifying Location Privacy" [70] Shokri et al. propose a formal framework for modeling and verifying LPPMs. This is the first work that aims at modeling generic LPPMs, where previously authors usually evaluated their own LPPMs based on their own metrics. The paper describes an LBS model from the perspective of an attacker using Markov chains to propagate location estimations. The proposed location privacy metric is the adversary's

expected estimation error calculated based on the underlying obfuscation mechanisms, which are probability functions. In 2012 Chen et al. extended the approach to model more general LBS scenarios and an attack based on repeating behavior [104].

The attacker-oriented model of Shokri et al. has two major drawbacks. First, modeling LPPMs is complicated and, second, the privacy measure is comparable but not always meaningful. In more detail, the first drawback is that the LBS model is designed primarily for probabilistic LPPMs, e.g., perturbation mechanisms. Modeling other types of LPPMs requires abstraction and is not intuitive for non-experts in the field of probability functions. The second drawback lies in the underlying approach to measure location privacy via the attacker's estimation error itself. While this approach allows the comparison of LPPMs based on an attacker's estimated error, it does not specifically verify guarantees of the LPPMs. Shokri et al. model a set of specific types of attacks, which was extended by Chen et al., but this extended approach still only shows the robustness against certain types of attacks.

The approach by Shokri et al. quantifies location privacy via the adversary's estimated error while we aim at making a binary decision whether a mechanism preserves privacy or not. The naïve way to use the research of Shokri et al. would be to use their verification and simply map the estimated error to boolean decisions by choosing a threshold. This, however, does not change the problem that the underlying quantification is based on explicitly modeled attacks making the result reliant on a specific type of attack. Another way to use the work by Shokri et al. for our goal is to adopt and modify their proposed approach. Markov chains where all probabilities are set to 1 are either deterministic or invalid as the sum of probabilities of the outgoing edges of each node must equal 1. Markov decision processes (MDP) are Markov chains that allow non-deterministic edges, i.e., edges without probabilities. Replacing all probabilistic edges in a Markov chain with these non-deterministic edges yields a special type of MDP or a state transition system. However, the LPPMs are modeled as probability functions and the Markov chains are derived from these. Hence, replacing all probabilities destroys the models of mechanisms as the resulting models now only span all possible locations and the metrics calculate the same probability (1) for each leaf (terminal state). The resulting quantified values are meaningless and cannot be used for our approach.

In 2015 Zhang et al. proposed "A Framework for measuring query privacy in Location-based Service" [105]. Their framework allows the modeling of generic LPPMs and the quantification of query privacy. Their LBS model is very similar to that of Shokri et al. as it is based on Markov chains, however, Zhang et al. focus on cloaking-based LPPMs, i.e., generalization of locations. Furthermore, their LBS models propagate queries as opposed to just locations in the model of Shokri et al.. The approach by Zhang et al. measures location privacy by the attacker's success rate. Their framework suffers from the same drawbacks as the one by Shokri et al. and also just focuses on location, even though their query-oriented models would allow for different types of data protection properties. As the approach by Zhang et al. is very similar to the one by Shokri et al., the same reasons apply why we cannot use their verification framework to achieve our goal.

In 2019 Ding et al. proposed "A Calculus for Modeling and Quantifying Location Pri-

vacy" [110]. The paper describes an approach to quantify location privacy by modeling obfuscation-based LPPMs in a probabilistic process calculus and verify location privacy properties based on entropy. The authors introduce the δ-calculus – a calculus that reasons about unfolded traces of probabilistic automata as processes. The syntax of the δ-calculus allows the description of the communication of nodes, which have a location and a radius. The success of synchronized communication is based on the radius of the receiver and the locations of the two nodes. Users, LPPMs, and LBS can be modeled as nodes and exchange locations, while the obfuscation mechanisms are modeled via a probabilistic choice operator, i.e., either the real location is communicated or a fake one. An adversary is modeled explicitly, similar to the approach of the applied pi calculus, and the radius of the attacker node parametrizes the attacker type, i.e., a strong attacker has unlimited radius and thus can eavesdrop on all communication while weak attackers are limited to communication in their radius.

The approach by Ding et al. has some obvious limitations as not all types of LPPMs can be modeled with their δ-calculus. Furthermore, their entropy-based privacy measure can be used to analyze the impact of changing the probability in a specific mechanism but cannot be used to compare the privacy impact of completely different LPPM approaches like perturbation and generalization. Using their approach, again, the naïve way of avoiding probabilities by interpreting the results based on thresholds does not achieve our desired goal. The approach by Ding et al. returns probabilities for leaf nodes, representing a possible communication, and one can sum the probabilities of communications where the real location leaked to get a measure of the adversary's success. If we use the strong attacker with unlimited radius, the resulting probability is the input parameter p as the probability is just propagated when no communication can fail because of the strong attacker. A threshold of, for instance, 0.5 would always yield a successful verification when the mechanism is defined by a $p > 0.5$. Replacing the probabilities in the process calculus by 1 (and consequently changing the Markov Chains to MDPs), again, leads the modeling of mechanisms ad absurdum and results in the probabilities to always be $1/k$ where k is the number of locations (real and fake). In this case a threshold would just verify whether the mechanism's number of locations is below a certain number.

Privacy Verification

As location data can be used to infer other personal data, the field of location privacy verification overlaps with privacy verification. Hence, we also give an overview of the field of privacy verification in general. This overview is not complete but represents what we consider the most important work in the field. The formal verification of data protection properties is usually strongly connected to data flow. In the following we present important privacy verification frameworks and argue why we did not adopt these for verifying location privacy.

While formal privacy specification languages like EPAL [111] or P3P [112] already existed, Barth et al. were the first to propose a model for privacy policies that is formalized in an LTL-based temporal modal logic, which opened the field of verification of

privacy policies [113]. The authors used their first-order temporal language to express *positive* and *negative* formulas, representing what is allowed and denied, respectively. Based on these positive and negative policies the framework allows reasoning about consistency of policies as well as compliance with policies. While consistency with policies is not relevant for our research, the compliance checks serve the same goal as our privacy verification. Barth et al. model "communicating agents" with each agent having a set of attributes and "knowledge states" as triples (p, q, t) where p, q are agents and t is an attribute of agent q. Such a triple is interpreted as "Agent p knows the value of attribute t of agent q" [113]. This model can easily be adopted to an LBS setting where the "attributes" are the four data types we derived in our taxonomy in Section 2.6. The verification framework of Barth et al. uses temporal logic formulas for both, the model of a system and its policies. The compliance check of two propositional LTL formulas can be done via standard LTL model checking. If we adopted this approach, we could model an LBS setting where location (and other) attributes are communicated and policies could be modeled to express that the LBS (or a third party) cannot know the values. However, modeling the location privacy-preserving mechanisms would not work in their temporal logic as the inference of information is modeled explicitly using the predicate $\mathit{contains}(m, q, t)$, which holds if message m contains attribute t of agent q. Implicit knowledge gain based on dependencies of data using a term algebra is not in the scope of the paper as the focus is modeling policies.

In 2012 Tschantz built on the idea of formally modeling privacy policies by introducing the notion of purpose restriction to a similar model to that of Barth et al. In his doctoral thesis [114] Tschantz proposed the first formal privacy policy framework that incorporated purposes to policies as it is common in practice, e.g., in the medical field personal data is collected for a specific purpose and this purpose only. In his model, purposes are modeled as plans, where actions deviating from such a plan are considered violations of purpose restriction. To model realistic scenarios, the extended framework of Tschantz uses Partially Observable Markov Decision Processes, which, in contrast to the fully observable ones, allow agents only to observe a finite subset of performed actions. These models enable the reasoning about uncertainty and beliefs in the field of privacy policies and purpose restriction. The proposed framework, for instance, allows the modeling of a scenario where a physician makes observations and derives conclusions about the health state of a patient. In this scenario one can reason about whether the physician acts according to her plan (purpose) or whether personal information is processed in a way deviating from the purpose restriction.

Other research formalizes privacy not from a user-policy perspective but from a purely legal point of view. Popular applications in this field are the US American Health Insurance Portability and Accountability Act (HIPAA) and the European Union General Data Protection Regulation (GDPR), which are both laws involving strict privacy regulations. Compliance to these laws has been a well researched field, where formal verification methods of health data protection [115–117] and privacy-by-design oriented data protection in general [118–123] have been proposed.

Of these, the paper from 2015 by Antignac and Le Métayer serves a similar purpose as this dissertation because it presents a framework that enables modeling systems and

verifying privacy properties. The authors describe knowledge-based privacy properties and use a similar epistemic logic to the σ-calculus. The similarity lies in the K operator that both logics share, which is not the standard epistemic modality. They describe a privacy-by-design supporting approach to verify privacy in generic software architectures [121]. Similar to our logic, their proposed logic is based on a state transition system and uses an epistemic operator for some privacy properties. Their knowledge is modeled as first-order properties, while we use dynamically interpreted propositional knowledge. Furthermore, their logic is not a modal logic as the authors do not reason about temporal properties with a temporal modality. Another major difference in the approaches is that they model (static) software architectures with (almost) no order of execution where we follow a (dynamic) protocol-based order of process calculi. If we wanted to use the approach of Antignac and Le Métayer, we could model LBS protocols using their architecture models. However, even if we defined a fixed ordering on the execution, the architecture models do not allow non-deterministic events. Consequently, traces of events in the approach by Antignac and Le Métayer are linear and do not allow branching. Furthermore, the access control properties that the approach allows to verify can only reason about explicit data flow and explicitly modeled inference of variables. A term algebra to determine whether, for example, an area potentially leaks a location is missing. The only protocols we could verify with such modified architectures would be if an obvious privacy violation occurs when the LPPM does not perturb one of the data types.

4.2. Process Calculi

In this thesis we use a process calculus to model communication in location-based service systems. As introduced in Section 3.1, process calculi describe the behavior of concurrent systems. Other process calculi have been proposed to model systems with focus on communication. In the following we present related research where process calculi have been used to check privacy properties or to model locations and movement in computer systems. We introduce the important papers and how they are related to our approach or why we cannot use these approaches for the goal of this dissertation. Generally, most modern process calculi are related to the pi-calculus and process calculi for privacy fall in the category of the applied-pi-calculus because in this field privacy is often equivalent with secrecy, and cryptography is a common solution for achieving secrecy. On the other hand, process calculi used to reason about locations and movement usually fall in the category of ambient calculi, which add (location) context to processes.

We already introduced the applied pi calculus [3] in subsection 3.1.1. The applied pi calculus and especially the tool ProVerif [124] have been used to verify a variety of security protocols such as the TLS protocol or Diffie-Hellman [125–127]. As privacy and secrecy are closely related, one would expect this dissertation to use the applied pi calculus for privacy-related properties of protocol. The first privacy-related properties that were checked with process calculi are secrecy properties in the applied pi calculus [125, 128]. These secrecy properties are concerned with the question “does my secret

leak?" and are answered by finding a possible execution of the protocol in which an attacker obtains the secret by filling context holes with possible terms. In 2005 Kremer and Ryan used the applied pi calculus for verifying voting privacy in an electronic election protocol [129]. With the tool ProVerif this process has been automated [130, 131]. All these approaches use weak bisimilarity as observational equivalence to model the attacker's knowledge. The adversary has to be modeled explicitly as the applied pi calculus allows active attacks, where the adversary participates in the protocol in an unintended way, e.g., as *man in the middle*.

The approach of verifying voting privacy as proposed by Kremer and Ryan can be compared to secrecy where the secret is the voting option (as opposed to the voting event itself). In this setting, users want to cast a vote to the voting system without revealing which option they voted for. This is only partly comparable to our LBS setting where the users want to provide enough information to the LBS in exchange for service but not too much to reveal their exact position. The equational theory of the applied pi calculus can model cryptographic mechanisms as it is designed for modeling cryptographic primitives such as homomorphic encryption. However, modeling a generalization-based mechanism using the terms of the applied pi calculus is not trivial. The process calculus neither allows non-deterministic branching where the condition of the branch is not evaluated. Our σ-calculus uses this type of branching to model set properties of arbitrary sets. The applied pi calculus could be used to model a subset of LPPMs but it cannot be used to verify location privacy properties as we established in Section 2.6 because the term algebra does not include a set theory. For fixed-sized sets one could use equational rules such as $set(x, x, x) = set(x)$ and $loc(set(x)) = x$ to model the singleton set relations the σ-calculus uses. But even for cryptographic LPPMs, for which the applied pi calculus can be used to model them, verification of the desired privacy properties would be based on explicitly modeling attacks (similar to voting secrecy) and would not be a satisfying solution.

Chadha et al. observed in 2009 that the approach of finding bisimilar processes is comparable to the epistemic principle of possible worlds and finding compatible worlds. They proposed the first epistemic modal logic for better reasoning about knowledge-based properties in an applied pi calculus, combining a variant of the applied pi calculus with a temporal-epistemic modal logic [132]. The paper neither describes a model checking approach nor mentions an implementation but serves as an inspiration for our approach. We model our protocols with a different process language, specific for the LBS context, and our state transition system involves state keeping beyond processes. While the property logic of Chadha et al. is suited to express location privacy properties, their protocol models suffer from the same limitations as the applied pi calculus which Kremer and Ryan used for voting privacy. Their process and term language does not allow the modeling of LPPMs beyond cryptography. The paper by Chadha et al. describes a syntax and semantics of processes and a temporal-epistemic modal logic sufficient to prove secrecy properties like in the applied pi calculus. The σ-calculus is not just an adaptation of the work by Chadha et al. as we build a completely new process syntax and term algebra, formulate new semantics for process reduction and satisfaction of properties, and create a model checking algorithm to verify the properties as well as

implement a software. We only took from the paper the initial inspiration of "unrolling" processes and reasoning about the resulting traces with a temporal modal logic.

Security protocols are not the only communication systems that can be modeled with process calculi. Other prominent process calculi of the recent years are the join-calculus [133] and the ambient calculus [134]. The join-calculus can be used to model distributed and mobile programming and is related to the pi-calculus. Similar to the applied pi calculus, the join-calculus does not allow dynamic channel communication and is restricted to asynchronous communication; instead it offers multi-way joins, i.e., it matches inputs from multiple channels simultaneously [135].

The ambient calculus is a process calculus describing movement of processes and devices. It can be used to model interaction in dynamic concurrent systems and was designed to model communication in the internet. In 2000 Cardelli and Gordon proposed an ambient calculus to model spatial and temporal movement and distribution [136]. The paper proposes a modal temporal logic to reason about spatio-temporal movement using ambient processes as models, making it the first paper to propose a combination of process calculi and modal logics. Anderson and Pym proposed in 2016 a process calculus for modeling resources and processes in computer systems [137]. Their approach is based on the combination of a process calculus and a modal logic based on propositional formulae. The satisfaction relation of their modal logic is built on a labeled reduction relation and bisimulation. Anderson and Pym describe a calculus (similar to an ambient calculus) where processes have a context, which is a set of resources in this case, and hence the process reduction depends on its resources, i.e., instead of reducing a process $P \to P'$ in their calculus a process E and its resources R evolve based on actions a: $R, E \xrightarrow{a} R', E'$ described via operational semantics. The authors show that reasoning about locations as with an ambient calculus can be done when interpreting the resources as locations. From a process-reduction point of view, their resources can be compared to our state σ, which we also evolve together with the processes. However, the calculus of bunched resources uses its process model to reason about resource management in distributed systems and, hence, proves properties such as "a system can reach a certain state given a set of resources" or with a spatial interpretation "a system state can be reached with certain starting points." As the name suggests, locations are considered resources in their approach and it is neither intended nor possible to reason about the "leaking" of locations, which makes the calculus of bunched resources unsuitable for the verification of location privacy.

Other process calculi have been proposed to model dynamic communication with specific applications. Most of these can be compared to the pi-calculus with deltas in the syntax. Siewe et al. proposed in 2011 a calculus called CCA, which is a calculus of context-aware ambients [138]. This calculus can be seen as an extension of the ambient calculus with a focus on environment contexts. The motivation for the CCA are modern context-sensitive applications with dynamic behavior based on the context, e.g., differences between 'at home' and 'at work.' The syntax of CCA introduces context-guarded actions similar to the guarded commands in CSP.

Singh et al. proposed a process calculus specifically for mobile ad hoc (wireless) networks (MANETs) [139]. Their ω-calculus models communication in MANETs with a

syntax to describe unique communication characteristics in the setting of MANETs. The syntax includes so-called local broadcasts, which model the typical form of communication in ad-hoc networks. In contrast to the δ-calculus for LPPMs by Ding et al., in the ω-calculus the nodes representing devices are not parametrized via location and radius but have ad-hoc cliques modeling the relative position regarding other nodes. While the ω-calculus would be suited to model group-based privacy mechanisms with ad-hoc cliques representing k-anonymous query groups, the calculus misses location information as positions are modeled only relatively via the groups. The ω-calculus could be used for a setting with a malicious user compromising a group privacy mechanism but is not suited for generic LPPM modeling and cannot be used to verify privacy properties as one can only reason about group affiliations and transmission ranges.

4.3. Modal Logics

Modal logic has been used for a wide array of applications. In this dissertation we use a temporal-epistemic logic which is a temporal modal logic with K-less epistemic properties. In Subsection 4.3.1, we present research in the field of temporal-epistemic modal logics which have temporal and epistemic modalities. In Subsection 4.3.2, other research in the field of modal logics is presented. As this is a wide field, we focus on research using temporal or epistemic logic for model checking as this dissertation itself presents a model checking approach for a temporal logic.

4.3.1. Temporal-Epistemic Logics

In this thesis we use a temporal modal logic with epistemic propositions to reason about the development of knowledge over time. As introduced in Section 3.2, formal modal logics have been around since the 1950s. While epistemic modal logic and temporal modal logic developed independently until in 1977 Sato laid the basis for a temporal-epistemic logic [140], it was not until the 2000s that concrete axiomatizations of logics with both modalities combined were proposed [141, 142]. In 2003 van der Hoek and Wooldridge extended the alternating-time temporal logic (ATL) [143] by Alur et al., which is itself a multi-actor extension of the branching time logic CTL, by adding epistemic modalities. Their resulting logic called ATEL (alternating-time temporal epistemic logic) allows the reasoning about dynamic knowledge and group knowledge in multi-agent systems [141]. In 2004 Halpern et al. presented their temporal-epistemic modal logic $\mathrm{CKL_m}$, which is a propositional linear-time logic syntactically derived from LTL [142]. Furthermore, the authors provide a complete axiomatization of their logic and hence lay the basis for model checking of temporal-epistemic properties in multi-agent systems.

Just as classic temporal modal logics, temporal-epistemic modal logics suffer from the state explosion problem. Models for multi-agent systems grow very fast as the number of states scales with the number of agents. We have discussed model checking approaches for temporal logics that avoid checking the complete state space. Similar approaches have been proposed in the field of temporal-epistemic logics [144–146]. Su et al. proposed in 2007 a symbolic model checking approach using OBDDs [144]. Cohen et al. and

Lomuscio et al. proposed structural model checking approaches based on symmetry [145] and partial order [146], respectively. The symbolic model checking approach by Su et al. verifies temporal-epistemic formulas (Halpern et al.'s $\mathrm{CKL_m}$) with temporal syntax from LTL. The property language of our σ-calculus includes a temporal property that is very application-specific and semantically not expressible in LTL as it is not a temporal modality but directly related to epistemic properties instead. We want to express that the combination of multiple queries can lead to privacy violations where each single one individually is not a privacy violation. To achieve this expressiveness, we need to look at possible traces (= executions of the protocol) and check whether specific combinations of queries occur in a continuing manner, i.e., the violation may happen frequently and not only once. This property cannot be expressed via $GF\,violation$ or similar constructs as a violation is permanent and once one occurred, in all following states the violation will also hold. We are interested in violations from subsequent queries, which is why one cannot express the continuing violation as a modality of the form $M\,\phi$ where M is one of the common modalities of temporal logics. Therefore, unmodified model checking approaches for temporal-epistemic logics like CKL cannot be used to check our location privacy properties.

Cohen et al. model check KCTL formulas where the temporal part is CTL syntax and Lomuscio et al. check $\mathrm{CTL^*K_{-x}}$ formulas with CTL* syntax without the "next" operator. Just as the previous approach, these two model checking approaches cannot be used for our location privacy properties. Furthermore, the three proposed model checking approaches are not useful optimizations for model checking location privacy in the σ-calculus. Su et al. use fixed-point iteration to efficiently check $\mathrm{CKL_n}$ formulas. Su et al. compute the set of reachable states, local epistemic truth, and temporal properties using fixed-point iteration on boolean formulas ordered by implication. Cohen et al. use symmetries of agents to reduce the number of checks of "concurrent" agents with similar knowledge. Lomuscio et al. use stuttering equivalence of traces to reduce the state space and perform a depth-first search on the reduced model. As we do not reason about multiple actors, our temporal formulas are not complex, and our state space is small, neither of these approaches are worth adopting for model checking location privacy properties of the σ-calculus.

The most prominent model checkers for epistemic modal logic in multi-agent systems (MAS) are MCK [147] introduced by Gammie and van der Meyden in 2004 and MCMAS [148] introduced by Lomuscio and Raimondi in 2006. Both tools have been used to model-check probabilistic knowledge. Huang et al. were the first to model-check perfect-recall (knowledge is preserved indefinitely) probabilistic knowledge in MAS using MCK [149]. In 2011 Wan et al. used MCMAS to check probabilistic temporal-epistemic properties using discrete-time Markov chains (DTMC) [150]. They propose the logic PCTLK (probabilistic CTL with knowledge), which is a probabilistic extension of CTL with added epistemic modality. The specification language of the model checker MCK is CTL* syntax with the epistemic modalities K and CK for common knowledge (denoted as C_G in Subsection 3.2.2). Using MCK as model checking back end is not an option as we cannot express our desired location privacy properties. While one can reduce the epistemic formulas of MCK to a single agent and only use the modality K

with propositions to match the desired epistemic formulas of the σ-calculus, MCK lacks the temporal operator to express the continuing violation property. MCMAS, on the other hand, provides a specification language called ATLK, which is the alternating-time temporal logic (ATL) with epistemic modalities for knowledge and common knowledge of agents. Instead of quantifying over paths (as in CTL), ATLK allows quantification over sets of agents to express which coalitions of agents can achieve certain knowledge. Our privacy properties cannot be expressed in ATLK either (for the same reason), hence using MCMAS is not an option either.

Uncertainty is an important aspect for the modeling of knowledge in distributed systems. Uncertainty in knowledge usually comes with observing sequences of non-deterministic events, e.g., when observing a coin toss the results change the observer's certainty about it being a fair coin. This type of uncertain knowledge (technically not knowledge in the sense of the truth axiom **T** as introduced in Subsection 3.2.2) is usually modeled with probabilistic transitions in underlying models like Markov chains. In 2009 Delgado and Benevides proposed the first temporal-epistemic logic for probabilistic multi-agent systems based on the branching time logic CTL [151]. This approach of modeling knowledge in an uncertain system is based on Markov Decision Processes (MDP), which are Markov chains that also allow deterministic transitions, and uses an accessibility relation for the epistemic modality. The authors use non-deterministic transitions to model "freedom of choice" of agents when it comes to order of execution.

4.3.2. Other Modal Logics

The field of linear or branching time logics derived from LTL and CTL is wide. Probabilistic extensions like PCTL [152] can be used to model probabilistic properties. The symbolic probabilistic model checker PRISM [153] has been used to verify real-life systems with uncertainty [154–156]. In applications where timing plays an essential role, reasoning about real time can be used to verify safety-critical properties. Real-time extensions of temporal logics like TCTL [157] or TLTL [158] can be used to model real-time specifications of systems. The model checking tool UPPAAL [159] allows the modeling of real-time systems as timed automata and the verification of properties using a subset of TCTL. UPPAAL has been used to verify time-critical real-world applications [160–162].

In 1994 Lamport proposed the temporal logic of actions, which is a non-modal temporal logic focused on actions where one reasons about actions and variables with two states: "now" and "next" [163]. TLA has been used to model concurrent systems [164] and serves as the basis for following logics. Model checking and temporal logic can be used to verify more than just the usual safety or fairness properties. Security properties have been proven using model checkers. Syverson and Meadows proposed in 1996 the first temporal language to express protocol security properties, called NPATRL, which is inspired by the temporal logic of actions (TLA) [165]. The authors used the NRL (Naval Research Laboratory) Protocol Analyzer (NPA) [166], which can be seen as a prototypical model checker for cryptographic security properties written in the declarative programming language Prolog, which is based on Horn clauses.

4.4. Summary

This chapter presents related research in the fields of privacy verification, process calculi, and modal logic and model checking. The research area of verifying location-privacy preservation is quite small and we presented the relevant work. Some authors have already proposed formalisms to quantify location privacy and even complete frameworks to model LBSs and verify their privacy guarantees. However, these frameworks are limited to certain types of LPPMs and purely location-oriented properties, i.e., previous work was not concerned with other personal data leakage, like the identity of users. While most location privacy models are based on statistics, the σ-calculus checks boolean properties verifying high-level protection concepts.

In the field of process calculi most modern calculi are modifications of the pi-calculus, such as the applied pi calculus, but also, for instance, the join calculus and the ambient calculus. These calculi have been used in the past two decades to model communication and also mobility. Hence, our σ-calculus is not the first calculus to model communication in spatially-aware systems. However, reasoning about locations in messages as opposed to reasoning about positions as communication ranges is not an established concept for process calculi. Compared to previous work our calculus is simple yet effective in its modeling of purely query-based communication, i.e., a single, directed communication channel with fixed message types. We took inspirations from the ambient calculus, and the σ-calculus is in its term algebra comparable to the applied pi calculus but we consider the σ-calculus to be a new calculus in the family of pi-calculi.

The field of modal logic is very wide and we focus on the selection of temporal and epistemic modal logics that have been used to model-check system behavior. The combination of temporal and epistemic modalities has been proposed and applied in practical model checking of multi-agent systems to reason about dynamic knowledge. Compared to related research our temporal modal logic with K-less epistemic properties is lightweight and reduced to a limited syntax as the σ-calculus reasons only about the knowledge of one agent: the LBS.

5. The σ-Calculus

Huge parts of this chapter were previously submitted in a journal article to the Journal of Logical and Algebraic Methods in Programming which is currently in review. In Chapter 2 we already set up requirements for a formal description language for LPPM protocols and location privacy properties. In this chapter we build on this basis and introduce the σ-calculus, which implements a set-based term language, queries, and obfuscation computation. Our location privacy framework consists of two components: a process calculus (Subsection 5.1.1 and Subsection 5.1.2) and a temporal-epistemic logic (Subsection 5.2.1 and Subsection 5.2.2). The former allows the description of Location Privacy Preserving Protocols (LP3) and the latter allows reasoning about specific privacy guarantees these protocols give. Although we do not focus on concurrent or parallel processes, we took our initial inspiration from the applied pi calculus as it is a modern process calculus with a term algebra and a specific application focus. For the scope of this chapter (and the following), we often use the term temporal-epistemic (or just epistemic) logic when we refer to our logic which is not a standard epistemic modal logic. Only when the distinction is of importance, we use the term "K-less" to denote our logic as opposed to standard epistemic modal logic. In Lemma 4, we compare our K-less epistemic modal logic to standard epistemic modal logic in terms of epistemic axioms.

5.1. Location Privacy-Preserving Protocols

Before we can formally describe what an LP3 is, we first must define LBS in this context. Generally, an LBS is any service that requires location data in exchange for the service. We make five assumptions that characterize our system model, LBS model, and attacker model.

Assumption 1 (LBS)**.** For the scope of this paper we define LBS systems to be limited to systems of the form described in Section 2.1 with users, the LBS, and an optional anonymization server as involved parties. In these systems users interact (directly or indirectly) with the LBS via queries that contain information about the user, current location and time, and the requested service. A typical application of these kinds of queries is "find nearest X." We explicitly exclude LBS where only user-to-user communication happens and no server is involved. It should be noted that peer-to-peer anonymization before querying an LBS server is still in the scope of our definition. Realistic LBS models involve more than one user query and also possible service sessions, where a user sends multiple queries in a period of time. The models of the σ-calculus represent this behavior in form of replication of processes (more in Subsection 5.1.1).

Assumption 2 (Attacker)**.** As opposed to the applied pi calculus, which is the most common process calculus for security protocols, we do not consider the attacker to be "on the wire" or "in the middle" as an unexpected participant of the communication. Instead, we consider an attack model in which the LBS as the communication endpoint is

the attacker. This is also in some cases referred to as a "strong passive attacker" because the LBS learns all sent queries and does not rely on interfered communication (while not actively sabotaging the communication). Because of this attacker model we do not deem it necessary to model any communication prior to the actual query that reaches the LBS, hence making channel communication obsolete. The syntax of the σ-calculus, thus, contains no channels, which is unusual for a process calculus.

Assumption 3 (User-Centricity)**.** In contrast to the usual model checking perspective, the σ-calculus follows a user-centric approach as opposed to a system-centric model. The σ-calculus considers one "representative" user with their respective personal data and other users who only contribute for group-based privacy protection. This user-centricity can be seen in the syntax of processes and terms in Subsection 5.1.1, e.g., loc is the location of the representative user, while loc_1, loc_2 represent the location of other users.

Assumption 4 (Worst-Case Scenario)**.** The σ-calculus is based on a worst-case assumption, i.e., processes are modeled in a way that the question "is it possible that privacy is violated" can be answered binary with yes or no. The worst-case scenarios include settings where multiple users are in the same location or request the same service. To find these worst cases the logic of the σ-calculus relies on the question whether or not a set can be a singleton set (more on this in Subsection 5.2.2).

Assumption 5 (Location)**.** We assume a discrete model of both location and time. The motivation behind this assumption is that our attacker model is based on the boolean decision of knowing exact data (cf. Assumption 4). The user's location is a coordinate in a discrete coordinate system, and the point in time of a query is a discrete time stamp, such that two time stamps can build a time window with finite time stamps in between.

An LP3 is a protocol that protects the privacy of users in the context of such an LBS system. We differentiate between location privacy preserving protocols and location privacy preserving mechanisms/algorithms (LPPA) to emphasize the difference between an algorithm which describes how to obfuscate personal data and a protocol which describes what pieces of data the user sends to the LBS. Many papers in the literature describe algorithms to obscure locations without giving the context of how to apply the mechanism in a protocol. This leaves questions open, for example, whether pseudonyms should be re-used for consecutive queries or whether accurate time stamps should be included? For us an LP3 is the description of an LBS system that utilizes an LPPA to preserve location privacy.

As motivated above, our framework aims at enabling the description of these LP3s in a comparable way as well as verifying privacy requirements that need to be met by such a system. For the formal description language of the protocols we decided on a process calculus as process calculi are a common way to model protocol-like communication and data exchange. Even though we do not use channels, the σ-calculus still uses one uni-directed form of communication: a query with fixed data types and synchronized communication between user and LBS. Furthermore, LP3 models utilize the order-of-execution aspect of sequential composition. In the following subsection we will introduce

the syntax of our new process calculus and explain our reasoning behind the decisions made.

5.1.1. Process Syntax

In this subsection, we introduce the syntax of processes of the σ-calculus. The syntax of processes to describe LP3s encompasses plain processes and a term language. First we introduce the syntax of plain processes, which contains sequential composition of processes but no concurrent parallel processes. Afterwards, we show the syntax of terms and functions and discuss their intended semantics.

Plain Processes

As motivated in Section 2.2 the process language should be able to describe queries containing instances or sets of four data types (identity, location, service, and time), obfuscation computations that generalize or perturb the data, and a term language (will be properly introduced in Table 5.2) that allows for set-specific and location-specific operations. Table 5.1 presents the syntax of a process calculus that satisfies the mentioned requirements. While the *null process*, *replication*, *conditional*, and *while* are common components for a process language, the *computation* and *query/multi query* are process components specific for LP3s. The former model the only form of communication (directed at the LBS) and the latter model the obfuscation, e.g., location generalization.

In the following we describe what is shown in Table 5.1 in more detail. The *null process* (0) is the usual terminating symbol. $!P$ is the replication of process P to show that a process does not terminate but repeats instead. The conditional process type (if p then P else Q) branches on the boolean predicate p to continue with either process P or Q. The computation process type (Compute$(n = T).P$) assigns a term T to a name n, either a new name or an already introduced name, and continues with process P. A term T can either be a name itself, a reserved name (variables for identity, location, and time), or the result of a function application on a term. Syntax of terms and functions is presented in Table 5.2 and discussed in the following. The single query process type (Query$(E).P$) represents a query that is sent to the LBS. The request quadruple E contains information about the query issuer, the issuer location, the requested service, and the time of the query (as motivated in Section 2.2). The multiple queries process type (kQuery$(M).P$) represents the sending of k different queries to the LBS. The multi-request set M is a request quadruple that contains indexed names representing sets (more on the special names in Notation 3).

Remark 1. The syntax as presented in Table 5.1 does not allow for sequenced replication of the form $!P.Q$ as replication models the infinite repetition of a process and, hence, process Q would never be reached. However, the syntax allows nested replication of the form $!(!P)$. It is obvious that this process describes the same protocol as $!P$ as in both cases process P is repeated infinitely. The syntax also allows more complex nested replication, e.g., $!(Compute(n = T).!Q$ where Q is a plain process that itself could also contain replication. While processes with nested replication are possible, they behave like

Table 5.1.: Syntax of the σ-calculus

plain processes $P, Q :=$	
0	null process
$!P$	replication
$\text{if}\, p \,\text{then}\, P \,\text{else}\, Q$	conditional
$\text{Compute}(n = T).P$	computation
$\text{Query}(E).P$	query
$\text{kQuery}(M).P$	multi query

where p, n, T, E, M follow the syntax of Table 5.2

the corresponding process with only the innermost replication as the outer replication would in practice never be reached. In Table 5.4 the corresponding equivalence rule will be introduced.

Notation 1 (Syntactic sugar)**.** We omit the .0 in a composition $P.Q$ where Q is the null process. We also define the shortened process "if p then P" to denote "if p then P else Q", where Q is the null process.

Terms and Functions

The syntax of plain processes contains multiple instances of terms and variables such as p in the condition of *if*, or n and T in a *Compute* process. Table 5.2 shows the syntax of these terms. The term language serves the purpose of modeling the obfuscation of the four basic data types as described in Section 2.6. The definition of terms reflects this. A term is either a name, a reserved name, a function application on a list of terms, or an indexed term. The reserved names are the variables necessary for most protocols, reflecting the four basic data types of the representative user and other users. Indexed terms correspond to other users (cf. Assumption 3) and can only be used in kQuery multi-request quadruple terms M.

In more detail, Table 5.2 presents the syntax of terms T, propositional variables p, request quadruples E, and functions f. As terms were already introduced above, we explain the possible forms of terms here. Names can be any string of characters that is not a reserved keyword (reserved names, function names, process names, etc.). A reserved name rn is one of the explicit eight options *pid*, *pids*, *loc*, *locs*, *serv*, *servs*, t, and ts. These reserved names are the singular and plural form of the four basic data types introduced in Section 2.6. While *pid* denotes the identity of the representative user (cf. Assumption 3, *pids* denotes the set of identities of all gathered users (either real users or dummies). Similarly, the other reserved names represent the location, requested

Table 5.2.: Syntax of terms and propositions

$$
\begin{aligned}
\text{term } T &:= \\
&\quad n \mid rn \mid f(T,..,T) \mid iT \\
\text{reserved name } rn &:= \\
&\quad pid \mid pids \mid loc \mid locs \mid serv \mid servs \mid t \\
&\quad \mid ts \\
\text{indexed term } iT &:= \\
&\quad irn \mid n_i \\
\text{indexed reserved name } irn &:= \\
&\quad pid_i \mid loc_i \mid serv_i \mid t_i \\
\text{request quadruple } E &:= \\
&\quad (G,R,S,F) \\
\text{multi-request quadruple } M &:= \\
&\quad \{E_1,..,E_m\} \mid (G_i,R_i,S_i,F_i) \\
\text{function } f &:= \\
&\quad f_S \mid f_L \mid f_I \\
\text{set function } f_S &:= \\
&\quad \cap \mid \cup \mid card \\
\text{location function } f_L &:= \\
&\quad MBB \mid dist \mid move \mid noise \mid redund \\
\text{integer function } f_I &:= \\
&\quad hash \mid rand \\
\text{boolean proposition } p &:= \\
&\quad k\_users \mid dummies \mid l\_diverse \mid s\_diverse \\
&\quad \mid T \mathcal{R} T \\
\text{binary relation } \mathcal{R} &:= \\
&\quad = \mid \mathcal{R}_\mathcal{I} \mid \mathcal{R}_\mathcal{S} \\
\text{integer relation } \mathcal{R}_\mathcal{I} &:= \\
&\quad < \mid > \mid \leq \mid \geq \\
\text{set relation } \mathcal{R}_\mathcal{S} &:= \\
&\quad \subseteq \mid \supseteq
\end{aligned}
$$

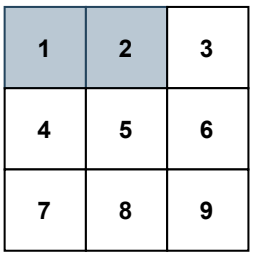
service, and time stamp of the request of the representative user and the group of users, respectively. Indexed terms iT are variable terms that do not specifically correspond to the representative user but instead refer to any of the gathered users individually. These terms are used for the multi-query process kQuery (more on the multi-request quadruple M later). Indexed terms are either indexed reserved names irn or names with an index n_i. An indexed reserved name is one of the four explicit options pid_i, loc_i, $serv_i$, or t_i. An indexed name n_i is the name bound to a term which may contain an indexed reserved name. Examples of such indexed names can be seen in the multi-request quadruple M, which is defined as (G_i, R_i, S_i, F_i), where the four names are indexed names representing terms with the four indexed reserved names. Such a multi-request quadruple requires compute processes with name assignments like $G_i = hash(pid_i)$. $R_i = MBB(locs)$ is also a possible assignment for the indexed name Ri. With this example we show that indexed names need not be bound to terms that are different for each gathered user. In the case of the two previous computations, all users would report the same generalized area but with individually hashed identifiers. A request quadruple E consists of four terms G, R, S, and F, which are either the reserved names pid, loc, $serv$, and t, respectively, names, or function applications on terms containing the respective reserved name. We use the naming convention of calling these individual terms G for group, R for region, S for service set, and F for time frame.

The two examples already show terms with function applications. Functions f in Table 5.2 have fixed arity and expect certain input types. The syntax only allows a fixed set of functions and these can be categorized into three groups: set-specific functions f_S, location-specific functions f_L, and integer-specific functions f_I. The set-specific functions include the intersection and union of sets, and also the cardinality to measure the number of elements in a set. The functions have the expected arity of two for $\cap$ and $\cup$, and one for $card$. All function in f_S take sets as input. The location-specific functions f_L include the minimal bounding box of a set of locations MBB, the distance of two locations $dist$, the move function $move$, a function to add noise $noise$, and a redundancy function $redund$. In more detail, the minimum bounding box computes the minimal rectangular area that contains all locations of a set. The distance function is a simple Manhattan distance on the discrete grid, where only locations to the left, right, top, and bottom have distance 1. The move function takes a set of locations and an integer and returns the set of locations that are in distance of the integer of the input set. Example 1 demonstrates an application of the function $move$. The noise function takes a location as input and returns a set of possible locations, which could contain the original location before the addition of noise. The intended semantics of the function $noise$ reflect what we discussed about perturbation mechanisms in Section 2.2. We use sets of locations to model the potential choices an attacker has when observing a noise-obfuscated location with knowledge of the underlying mechanism. The redundancy function takes a location as input and returns a set of locations, similar to $noise$. The difference in the intended semantics of $redund$ is that this function returns a set of distributed locations, which are not adjacent (as opposed to the neighboring locations in the return set of $noise$). The identity-specific functions f_I contain a hash function $hash$ and a random number generating function $rand$. The function $hash$ takes an identity as input and returns a

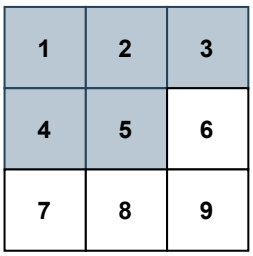
set of identities, which represents the potential users who could have been the issuer from the perspective of the attacker. The function *rand* has an arity of zero and also returns a set of identities with the same interpretation. The difference between the intended semantics of these two anonymization functions is that *hash* will stay the same for subsequent queries, while *rand* will return a different pseudonym for each new query.

A propositional variable p, as seen in Table 5.2, is either a relation of two terms or one of the four explicit boolean flags: *k_users*, *dummies*, *l_diverse*, and *s_diverse*. The intended semantics of the flag *k_users* is to indicate that the representative user has gathered other real users to perform group-based obfuscation. The flag *dummies* indicates that the representative user forms a group with fake dummy users. The flags *l_diverse* and *s_diverse*, on the other hand, signal that the formed group's location and service sets satisfy diversity, respectively. Location diversity means that the group of users are not all in the same location and service diversity means that not all users request the same type of service. The set of binary relations $\mathcal{R}$ contains equality, integer-specific relations, and set-specific relations. All relations expect the two terms to be of the same type, i.e., integers can only be compared to integer terms and sets to set terms. Equality, integer relations, and set relations follow the straight forward notation and their intended semantics are defined in the usual way. Relations can be negated, just as the four boolean flags, with the obvious meaning that the relation does not hold.

Example 1. In this example we want to demonstrate an application of the function *move* as it is an important function that is used for the satisfaction of properties later in Subsection 5.2.2. Let locations be integers representing positions on a 3×3 grid such that 1 is the top left tile, 2 the top middle, and 9 the bottom right. Such a grid is sketched in Figure 5.1. Then the equation $move(\{1,2\},1) = \{1,2,3,4,5\}$ holds. This equation states that the set $\{1,2,3,4,5\}$ contains all the locations which can be reached from the location set $\{1,2\}$ with distance 1. Figure 5.1a shows the grid before moving and Figure 5.1b after moving. As mentioned above, we use the discrete integer Manhattan metric as distance measure, therefore diagonal tiles are not adjacent (distance of 1). The fact that $move(\{1,2\},1)$ does not contain location 6 (middle right tile) demonstrates the use of such a distance metric.



(a) Grid before moving



(b) Grid after moving

Figure 5.1.: 3×3 grid where positions in the location set are highlighted

Notation 2. The reserved names *pids*, *locs*, *servs*, and *ts* are sets that contain the (variable) reserved names of the corresponding name, e.g., *locs* contains *loc* and the indexed name loc_i (here i is a variable: loc_1, loc_2,... are the locations of unnamed other users whose privacy is not considered). The design decision to have one (representative) user-related reserved name per data type comes from the user-centricity from Assumption 3.

The boolean propositional flags p represent the branching points for modeling the execution of protocols. A shortened "if p then P" as introduced in Notation 1 indicates that the protocol does not give a choice, i.e., the protocol only continues if p holds. It is also possible to branch on term relations, both, integer and set relations. The relations can be necessary to express more complex obfuscation mechanisms that aim for non-trivial guarantees, e.g., $card(region) > threshold$. The fixed set of non-relational boolean flags contains *k_users*, *dummies*, *l_diverse*, and *s_diverse*. *k_users* and *dummies* are key boolean flags because they allow modeling the principle of k-anonymity on which generalization and other mechanisms rely. Diversity of sets is another important aspect of protection mechanisms that are based on hiding in numbers (as discussed in Section 2.2 this is true for most mechanisms). As branching is such a key aspect of the modeling of protocols, the syntax of boolean flags also allows for term relations. As terms are limited to integers and sets, the term relations are standard integer and set relations. The syntax of terms and propositions suffices to model the common mechanisms' modifications of the four base data types as sets. In Section 2.3 we have already shown six examples of common mechanisms from the literature modeled using the process and term syntax of the σ-calculus, albeit in .lp3 syntax, which will be introduced in Section 6.2.

Remark 2. The syntax of processes and terms include the letter k in kQuery and *k_users*, which might create the expectation of symbolic interpretation or quantification. However, k is not a parameter or variable but just a name representing a value greater than 1. For instance, no process called "3Query" exists in the syntax nor a boolean flag "2_*users*." The naming comes from the common concept of k-anonymity and is used in analogy to *l_diverse* and *s_diverse*. Similarly, l and s represent an arbitrary but fixed number greater than 1.

Remark 3. Note that the term language is not typed and syntactically a name can be of any type. However, the reserved names are expected to have certain types and the functions expect certain types, e.g., *move* takes two input parameters: a region (set) and a distance (integer). The expected types are mostly intuitive but we discussed all functions' expected semantics.

Notation 3. The multi-request quadruple M from Table 5.1 shows a special type of term: indexed terms like G_i represent terms that are related to multiple users (real or fake). These terms can be composed with the four basic indexed reserved names: pid_i, loc_i, $serv_i$, and t_i. The indexed term G_i, however, cannot contain loc_i, for instance, as groups can contain only the indexed reserved name pid_i.

Example 2. Let $P_1 = \text{Query}((\{pid\}, \{loc\}, \{serv\}, \{t\}))$ be a plain process that represents an LBS without any privacy preservation. The natural language description of this LP3 is that a user sends a query to the LBS without any anonymization or location obfuscation. The query contains an identifier of the user, the location of the user (modeled as a discrete coordinate), the requested service, and the time stamp of the request. Let $P_2 = \text{if}\, k\_users\, \text{then}\, \text{Compute}(R = MBB(locs)).\text{Query}((\{pid\}, R, \{serv\}, \{t\}))$ be another plain process utilizing a k-anonymity LPPA that performs a region query with multiple user locations. This LP3 describes the protocol PrivacyGrid (our running example from Section 2.5) without replication and service diversity. In more detail, in the protocol other users are gathered (k_users in if) and the minimal bounding box (MBB) of all locations ($R = MBB(locs)$ in Compute) is calculated and then a query to the LBS (Query) is issued. The query is similar to the one in P_1 with the exception of reporting the region (R instead of singleton set $\{loc\}$). This is a simple generalization approach where (as opposed to PrivacyGrid) only the location is protected.

As can be seen from the syntax in Table 5.1, processes in the σ-calculus are built by sequencing composition ($P.Q$). We introduce *process components* to distinguish between a process as a whole and its building blocks.

Definition 5.1.1 (Process component)**.** A **process component** of a plain process P is a (partial) process Q that is contained in the composed process P and not a null process. We define the size of a plain process, denoted by $size(P)$, to be the number of process components in P.

Example 3. Let us consider the plain processes $P_1 = \text{Query}((\{pid\}, \{loc\}, \{serv\}, \{t\}))$ and $P_2 = \text{if}\, k\_users\, \text{then}\, \text{Compute}(R = MBB(locs)).\text{Query}((\{pid\}, R, \{serv\}, \{t\}))$ from the previous example. $size(P_1) = 1$ as P_1 only contains the one process component $\text{Query}(pid, loc, serv, t)$. $size(P_2) = 3$ because process P_2 contains three process components (if then else, Compute, Query).

The syntax of Table 5.1 allows for arbitrary terms and names. On the other hand, the names from Table 5.2 can only be bound to terms by a Compute. The syntax of processes is not very restricting and allows nearly arbitrary terms. To talk about meaningful processes, we first define unbound names and then closed processes.

Definition 5.1.2 (Bound names)**.** A **bound name** n in a process P is a term that occurs in a process component and is bound by a previous compute process component, i.e., $P = P_0.P_1.\ ...\ .P_m$ is a finite sequence of process components, where n is part of a term in a process component P_i and a process component $P_j, j \in [0, i-1]$ of the form $\text{Compute}(n = T)$ exists. **Unbound** names in a process are all names that are not bound.

Definition 5.1.3 (Closed processes)**.** A plain process is called **closed** if it is composed in a way that all names n in any process component are either reserved names or bound names.

Closed plain processes have a finite number of different terms as they are limited by the finite number of computation process components in a plain process. In the following we always refer to closed processes, if we write "plain process."

5.1.2. Process Semantics

Location privacy preserving protocols can be described with plain processes as we have introduced them in the previous subsection. The attacker's knowledge also has to be kept track of, to be able to reason about location privacy as motivated in Section 2.6. To that end, we introduce the *location state* of a process.

Definition 5.1.4 (Location state)**.** A **location state** σ is a quintuple: $Int \times \{(E, C)\} \times \{n = T\} \times Bool^4 \times \{T \mathcal{R} T\}$. $\sigma := (\sigma_{Ind}, \sigma_{Qry}, \sigma_{Eq}, \sigma_{Prop}, \sigma_{Rel})$, where σ_{Ind} is an integer, σ_{Qry} is a set of tuples consisting of a request quadruple E and a constraint set $C = \{p\}$, where p are propositions as defined in Table 5.2, σ_{Eq} is a set of equations of form $n = T$, σ_{Prop} is a boolean vector $(k\_users, dummies, l\_diverse, s\_diverse)$, and σ_{Rel} is a set of relations of terms of the form $T_1 \mathcal{R} T_2$ where $\mathcal{R}$ is a relation as defined in Table 5.2. The **initial state** $\sigma^{init} := (0, \emptyset, \emptyset, empty, \emptyset)$ is the initial location state, where *empty* denotes the tuple with all propositions set to *false*.

The location state σ is defined as a quintuple comprised of the following sub-states: the index state σ_{Ind}, the query state σ_{Qry}, the equation state σ_{Eq}, the property state σ_{Prop}, and the relation state σ_{Rel}. The index state is an integer that represents the index of a query. The index is initialized as 0 and incremented after each sent query, hence σ_{Ind} shows the number of queries that have been sent. The query state stores the request quadruples of the sent queries and all additional constraints, i.e., the propositions and relations. The constraint set C stores the combined information of the temporary property state and relation state into the permanent query state to capture the current constraints. This constraint set C is particularly important because it reflects the knowledge the LBS has of the underlying protection mechanism used for a specific query. While σ_{Prop} and σ_{Rel} are only internal states which are reset after an issued query, the query state preserves the properties and relations of the time of the query in the constraint set C. The equation state contains a mapping of all (bound) names and terms, while the property state contains all propositional facts, and the relation state contains all term relations (cf. Table 5.2). Informally, the location state represents the current internal knowledge of the LP3, which is not necessarily the knowledge of the attacker as the following example shows.

Example 4. Let us consider the location state $\sigma_1 = (0, \{\}, \{R = MBB(locs)\}, (true, false, false, false), \{\})$. In natural language the state can be interpreted as follows. No query has been issued yet (0 in σ_{Ind} and empty σ_{Qry}), a generalization computation has occurred (σ_{Eq} contains a minimal bounding box computation), and multiple users have been gathered for anonymization (k_users flag set). At this point the attacker has no knowledge about the user's data (empty σ_{Qry}). The state σ_1 represents the internal knowledge about the system. The state σ_1 can correspond to an execution of

Table 5.3.: Extended syntax of processes

extended processes $A :=$		
	$P\|\sigma$	process with location state
	$A.A$	finite replication

PrivacyGrid where the spatial generalization has already been performed but the service set has not been diversified yet (after line 4 and before line 5 in Figure 2.2). Now let us consider $\sigma_2 = (1, \{(\{pid\}, \{loc\}, \{serv\}, \{t\})\}, \{\}, empty, \{\})$. This state represents a point in the execution of an LP3 where one query has been sent to the LBS (1 in index state). The attacker knows the identity, location, service, and time of the query as no protection has been performed (σ_{Qry} contains request quadruple of singleton sets, e.g., the representative user's location $\{loc\}$ instead of, e.g., a generalized term $MBB(locs)$). As mentioned in Assumption 4 singleton sets in queries represent unprotected personal data. The state σ_2 can correspond to an execution of protocol P_1 from Example 2 which does not modify user queries.

Extended Processes

Definition 5.1.5 (Extended processes)**.** To model the protocol executions and the corresponding location states, we introduce extended processes. Extended processes are used to connect location states and plain processes. An extended process A can either be of the form $P|\sigma$ describing a plain process with a location state σ or of the form $A.A$ as the finite replication of an extended process A. Table 5.3 shows the syntax of extended processes. Furthermore, we define $size(A)$ to denote the size of the underlying plain process, i.e., $size(A) = size(P)$, where $A = P|\sigma$ for any location state σ.

We introduce the notation of frames to relate extended processes and their states without the need of plain processes.

Notation 4 (Frames)**.** The *frame fr* of an extended process $A = P|\sigma$ is the location state σ: $fr(A) = \sigma$.

Equivalence of processes is used to reason about "minimal" processes, i.e., processes without redundant replication. We define the process equivalence relation $\cong$ to be the smallest equivalence relation on extended processes such that the rules in Table 5.4 hold.

The first rule simply states that infinite replication subsumes finite replication, i.e., repeating a replicated process does not change the extended process' state. The second rule resolves nested replication, i.e., double replication can be reduced to simple replication without altering the state. The third rule expresses that the null process is equivalent with its frame.

Table 5.4.: Equivalence of processes

$P.(!P)\|\sigma \cong !P\|\sigma$	$REPL1$
$!(!P)\|\sigma \cong !P\|\sigma$	$REPL2$
$0\|\sigma \cong \sigma$	$NULL$

Notation 5 (Equivalent processes)**.** We define $\Omega(A)$ to denote the set of all extended processes A' that are equivalent to A, i.e., $A' \cong A$.

Process Reduction

As introduced in Section 3.1, the semantics of process calculi are oftentimes based on the reduction of processes. Usually step-wise reduction models the execution steps of a protocol, where the choice and order of reduction rules resolve the nondeterminism of the concurrent systems. In the σ-calculus we also reduce processes using a set of reduction rules. As the syntax does not include parallel processes, the σ-calculus does not have rules for a non-deterministic order of execution. However, one key feature of the σ-calculus is the branching behavior. We do not evaluate the term algebra during the reduction of processes but instead consider both cases: either a proposition or relation is true, or it is false. The following reduction rules reflect this non-deterministic branching by having two rules for *condition* processes.

We define the labeled reduction relation $\xrightarrow{l}$ with label l to be the smallest relation of extended processes (closed under process equivalence) such that the rules in Table 5.5 hold. $\xrightarrow{*}$ denotes the reflexive transitive closure of $\xrightarrow{l}$.

The function *set* takes the substate σ_{Prop} and returns it with the specified flag set to the given value if p is a non-relational proposition as described in Table 5.2. *update* is an overloaded function that either updates the equation state σ_{Eq} by replacing an equation $n = T$ by a new equation $n = T'$ possibly re-evaluating a recursive call to n, or it updates the query state σ_{Qry} by adding one or more request quadruples E to the query set after using the equations in σ_{Eq} to replace all unknown names by their respective terms and also adding the true propositions of σ_{Prop} and all relations in σ_{Rel} in the constraints set of the query state. The function *add* checks if a proposition p is a relation and only then adds it to the relation state. $\neg p$ is a negated relation (cf. Table 5.2) and not the negation of a boolean value. Table 5.5 uses the state modification notation $\sigma(\sigma_{Substate} = x)$ where the location state σ stays the same with only the explicitly assigned sub-states changing.

Table 5.5 first shows the two structural rules REPL and PARA for internal reduction. These two rules resolve the replication of plain and extended processes. The REPL rule shows how replication is unrolled in our calculus. As can be seen, infinite replication is reduced to finite replication, that is, double repetition of the process. The rationale for this rule is the observation that infinite process replication does not alter the state of

Table 5.5.: Reduction rules

$REPL$	$!P\|\sigma \xrightarrow{REPL} P\|\sigma.P\|\sigma$
$PARA$	$\dfrac{P\|\sigma \xrightarrow{*} 0\|\sigma'}{P\|\sigma.Q\|\sigma \xrightarrow{PARA} Q\|\sigma(\sigma_{Ind} = \sigma'_{Ind}, \sigma_{Qry} = \sigma'_{Qry})}$
$THEN$	$[\text{if}\, p\, \text{then}\, P\, \text{else}\, Q]\|\sigma \xrightarrow{THEN} P\|\sigma(\sigma_{Prop} = set(\sigma_{Prop}, p, 1),$ $\sigma_{Rel} = add(\sigma_{Rel}, p))$
$ELSE$	$[\text{if}\, p\, \text{then}\, P\, \text{else}\, Q]\|\sigma \xrightarrow{ELSE} Q\|\sigma(\sigma_{Prop} = set(\sigma_{Prop}, p, 0),$ $\sigma_{Rel} = add(\sigma_{Rel}, \neg p))$
$COMPn$	$[\text{Compute}(n = T).P]\|\sigma = (\sigma_{Ind}, \sigma_{Qry}, \sigma_{Eq}, \sigma_{Prop}, \sigma_{Rel}) \xrightarrow{COMP}$ $P\|\sigma(\sigma_{Eq} = \sigma_{Eq} \cup \{n = T\}),$ where n is an unbound name
$COMPu$	$[\text{Compute}(n = T').P]\|\sigma = (\sigma_{Ind}, \sigma_{Qry}, \sigma_{Eq}, \sigma_{Prop}, \sigma_{Rel}) \xrightarrow{COMP}$ $P\|\sigma(\sigma_{Eq} = update(\sigma_{Eq}, n = T')),$ where n is a bound name
$QURY$	$[\text{Query}(E).P]\|\sigma \xrightarrow{QURY} P\|\sigma(\sigma_{Ind} = \sigma_{Ind} + 1,$ $\sigma_{Qry} = update(\sigma_{Qry}, E, \sigma_{Eq}, \sigma_{Prop}, \sigma_{Rel}), \sigma_{Eq} = \emptyset,$ $\sigma_{Prop} = empty, \sigma_{Rel} = \emptyset)$
$KQURY$	$[\text{kQuery}(M).P]\|\sigma \xrightarrow{QURY} P\|\sigma(\sigma_{Ind} = \sigma_{Ind} + 1,$ $\sigma_{Qry} = update(\sigma_{Qry}, M, \sigma_{Eq}, \sigma_{Prop}, \sigma_{Rel}), \sigma_{Eq} = \emptyset,$ $\sigma_{Prop} = empty, \sigma_{Rel} = \emptyset)$

an extended process except for computation and query processes. While the former can only make a difference when used recursively, i.e., Compute($n = f(n)$), the latter only increases the number of sent queries. Both state changes do not impact the privacy state of protocols, which we will show Subsection 5.2.2. The PARA rule unrolls serialization of processes with their respective states. Only the location, time, and query state are carried over from the first reduced process due to the local scope of the computations, predicates and relations. These states are reset, while the queries are global knowledge for the LBS. The other eight rules can be used to reduce conditionals, computations, and queries. The labels are important for the reduction because the conditional process can be reduced in two different ways. Our calculus does not include an equational theory that evaluates the conditions of an "if" but instead we branch on these processes and consider both the *true* and *false* case. The THEN and ELSE rules are used to branch on a conditional process. THEN continues with the process P and sets the proposition of the condition to true if p is one of the non-relational propositions or add p to the set of relations otherwise. ELSE continues with the process Q and sets p to false in

the proposition state (if applicable) or adds the negated relation to the relation state. The rules COMPn and COMPu model the impact of the Compute process on the state. COMPn is the rule for the case where a new name is assigned a term and alters the state by adding the equation to the equation state. COMPu, however, models the case where n is a bound name and will be reassigned. In both cases the reduction continues with process P. The rules QURY and KQURY model the effect of the two different Query processes on the state. QURY continues with process P and increments the index state to indicate that a query has been received by the LBS. Furthermore, the query state is updated by considering the equation, proposition, and relation states. These states are reset to their initial state because they only represent the knowledge of one query and can change for a potential subsequent query. KQURY works in the same fashion as QURY with the only difference that the query state is not updated by one request quadruple E but a multi-request quadruple M.

Example 5. Let us consider the plain process $P_2 = \text{if}\, k\_users\, \text{then}\, \text{Compute}(R = MBB(locs)).\text{Query}((\{pid\}, R, \{serv\}, \{t\}))$ from Example 2 (which is a reduced version of our running example protocol PrivacyGrid) and define the extended process $A_1 = P_2|\sigma^{init}$ using the initial location state σ^{init} from Definition 5.1.4. $\text{if}\, k\_users\, \text{then}\, \text{Compute}(R = MBB(locs)).\text{Query}((\{pid\}, \{R\}, \{serv\}, \{t\}))|\sigma^{init}$ can be reduced to the process $A_1' = 0|\sigma'$ where $\sigma' = (1, \{((\{pid\}, MBB(locs), \{serv\}, \{t\}), \{k\_users\})\}, \emptyset, empty, \emptyset)$ by application of the internal reduction rules THEN, COMPn, and QURY in that order.

Extended processes combine plain processes and location states. However, the syntax of extended processes allows arbitrary combinations. To describe the set of states that correspond to the internal knowledge of a plain process, we introduce consistent states.

Definition 5.1.6 (Consistent states)**.** A location state σ is called **consistent** if it can be reached via the reduction rules starting from the initial state σ^{init}. That is, a plain process P exists such that $P|\sigma^{init} \xrightarrow{*} P'|\sigma$.

A consistent location state allows only for a finite set of queries because infinite replication is resolved into finite replication (REPL rule). For collecting all equivalent states (later in Subsection 5.2.2) we need to quantify over (a finite set of) potential states. To enable this quantification, we introduce **similar distinctive** states. These represent all consistent states that are similar to each other but do not belong to the same process.

Definition 5.1.7 (Similar distinctive states)**.** Two consistent location states σ and σ' are called **distinctive** if they differ in the query state (cf. Definition 5.1.4), i.e., $\sigma_{Qry} \neq \sigma'_{Qry}$. Two location states are called **similar distinctive** if they are distinctive and all their other sub-states are equal, i.e., $\sigma_{Ind} = \sigma'_{Ind}$, $\sigma_{Eq} = \sigma'_{Eq}$, $\sigma_{Prop} = \sigma'_{Prop}$, and $\sigma_{Rel} = \sigma'_{Rel}$. We define $\Delta(\sigma)$ to be the set that contains σ and all location states σ' that are similar distinctive from σ.

Although the set of all possible location states is infinite as σ is a quintuple containing potentially infinite sets, similar distinctive location states are more restricted.

Lemma 1. *The set $\Delta(\sigma)$ of all distinctive location states of σ is finite.*

Proof. By definition, $\Delta(\sigma)$ contains the state σ and only similar distinctive states σ'. Similar distinctive states only differ from σ in the query state. The query state consists of a set containing all queries sent to the LBS and a constraint set C. By Definition 5.1.6 the query set only contains a finite number of queries in a consistent state as the reduction rules in Table 5.5 reduce replication to finite repetition. Furthermore, for two consistent location states with the same (finite) equation state only a finite number of different queries exists as the number of terms is limited. The constraint set, similarly, is finite as only a finite number of propositions (4) and relations are possible, limited by the number of process components. □

Lemma 2. *The frames of two equivalent, extended processes A and A' are equal. That is if $A \cong A'$, then $fr(A) = fr(A')$.*

Proof. The equivalence of processes is defined via three rules. All three rules preserve the location state σ as it stays the same on the left-hand side and the right-hand side. As, by definition, the frame of an extended process is its location state, the frames of two equivalent processes are equal. □

The purpose of our introduction of the equivalence relation of extended processes is describing different plain processes with the same state. Hence, Lemma 2 is not a surprising property.

5.2. Location Privacy Properties

The σ-calculus entails not only a process language but also a property language and logic to describe and verify location privacy guarantees of processes. In the following subsection, we introduce a formula language to express temporal-epistemic properties. In Subsection 5.2.2, we introduce the semantics of properties culminating in the satisfaction relation of formulas and processes.

5.2.1. Property Syntax

As introduced in Section 2.6 location privacy can be reduced to the four atomic protection goals: protecting identity, location, service, and time. As violations against these protection goals can be expressed as a party learning about personal data of a subject, privacy can be expressed in terms of knowledge. Hence, leaking of data is related to the gain of knowledge about said data. Therefore, to describe privacy violations we use an epistemic knowledge notation K to express that the LBS knows a fact about a representative user. Due to our worst-case assumption, showing that the data of one user can leak is sufficient to derive a violation of privacy.

Table 5.6 shows the complete syntax of location privacy property formulas, divided into static and temporal formulas. Static formulas δ express requirements that should hold locally in a location state σ of a process. A static formula makes epistemic statements

Table 5.6.: Properties

Static Epistemic Formulas

$\delta ::= \neg\delta \mid \delta \vee \delta \mid \delta \wedge \delta \mid K_{Id} \mid K_{Loc} \mid K_{Serv} \mid K_T$

Temporal Properties

$\phi ::= \delta \mid Cont\,\delta \mid \neg_T\phi \mid \phi \vee_T \phi \mid \phi \wedge_T \phi \mid \Box\phi \mid \Diamond\phi$

about the knowledge the LBS has about the user. K_{Id}, K_{Loc}, K_{Serv}, and K_T are propositional variables which express whether the LBS knows the identity of the user, location of the user, requested service, and time of a query, respectively. The four properties are not parameterized but are propositional properties that relate to the respective base data type. Negations, disjunctions, and conjunctions are also possible with the usual interpretation. Temporal formulas ϕ make statements about a process as a whole (protocol), that is, how static properties behave over the course of multiple or even all states. A temporal formula can be a static formula δ, a continuous evaluation of a static formula $Cont\,\delta$, the LTL[1]- like global modality of $\Box\phi$ and future modality of $\Diamond\phi$, and also the negation, disjunction, and conjunction of temporal properties. The symbols for temporal negation, conjunction, and disjunction are indexed versions of the respective standard symbols for the boolean operations to distinguish them from the static negation, disjunction, and conjunction. We will give more details on the interpretation of these formulas in Subsection 5.2.2.

At this point we want to discuss the syntax of Table 5.6 in more detail. In our logic, K is not a modal operator. Instead, the four K formulas are simple propositional variables because we do not reason about different actors or nested knowledge (as already mentioned in Subsection 4.3.1). In contrast, our logic has a temporal modality with the temporal operator $\Box$. The reason why our property syntax contains a temporal modality lies in the execution of the protocols. At execution time t_0, i.e., before the protocol starts, no privacy violation has happened for any protocol. This changes for execution points t_i during the protocol execution. When the first query was sent to the LBS the protocols may differ in the potential privacy violations. For a software developer it is also of interest to know at what point the violation happened. Maybe only multiple queries lead to leakage due to correlated knowledge of the LBS. That is, two potentially consecutive queries (cf. Assumption 5 and Assumption 1) with perturbed or generalized regions of origin might be in such a distance that only one location for either of the queries is a possible candidate considering a maximum velocity that a user can move with. In a worst-case scenario as described in Assumption 4 it is sufficient to check whether it is possible that two areas intersect in only one location, where the first area is the region, where the user could have moved to in one time step and the second area is the region of the second query. Next to the common temporal modality $\Box$, we also have

[1]Linear Temporal Logic

the temporal operator *Cont* to denote continuing violation of privacy. This operator can be compared to the temporal modality "Past" as it describes that the property was true before. However, it differs from the standard past modality ($\Diamond^{-1}$) as $Cont\,\delta$ states the fact that property δ holds for the current query and has been true for a previous query. It should also be noted that *Cont* is not a classical temporal operator because it is only defined for static properties δ and hence cannot be nested with other temporal operators (for instance, unlike $\Box\Diamond\delta$, $Cont\,\Diamond\delta$ is not a well-formed formula in Table 5.6). The reason why the *Cont* property is of relevance is that some protocols do not allow the linking of two leaks, for instance due to the exchange of identities, and therefore a second query might not result in a second violation. With *Cont* we can express that long-term tracing should not be possible.

The syntax of Table 5.6 includes two negations, disjunctions, and conjunctions, each. The semantics in Subsection 5.2.2 will show that $\neg\delta \equiv \neg_T\delta$ holds and similarly $\delta \vee \delta \equiv \delta \vee_T \delta$ and $\delta \wedge \delta \equiv \delta \wedge_T \delta$. While the three operators $\neg$, $\wedge$, and $\vee$ have standard precedence over each other (negation highest) and the same holds for $\neg_T$, $\wedge_T$, and $\vee_T$, the precedence in mixed formulas is not obvious. We define static operators to have a higher precedence than temporal ones, such that $\wedge$ takes precedence over $\wedge_T$ and $\vee$ over $\vee_T$. However, to increase human readability of the syntax, temporal negation takes precedence over static conjunction and disjunction, e.g., the brackets in $\neg_T(\delta \vee \delta)$ are not optional.

Example 6. Let us consider the reduced PrivacyGrid process P_2 from Example 2 and suppose we want to express the privacy requirement that the LBS should not know where the user issued the query. This can be expressed as the static formula $\delta_1 = \neg K_{Loc}$. To express that δ_1 should hold at all times, we use the temporal formula $\phi_1 = \Box\neg K_{Loc}$.

The interesting question is now whether property ϕ_1 holds for the protocol P_2 or formally: does $P_2 \models \phi_1$ hold? Subsection 5.2.2 will introduce the necessary semantics for the satisfaction of properties and Section 6.1 will give a model checking algorithm to automatically verify properties using an SMT solver.

Notation 6. The formulas $F_1 = \Box\neg K_{Id}$, $F_2 = \Box\neg(K_{Loc} \wedge K_T)$, $F_3 = \Box\neg_T Cont\,(K_{Loc} \wedge K_{Id})$, and $F_4 = \Box\neg K_{Serv}$ are the syntactic formalization of the four ground requirements **F1-F4** (as introduced in Section 2.6) in the σ-calculus. F_1 states that at no point of the protocol the LBS should know the identity of the user (*Identity*), F_2 states that at no point of the protocol the LBS should know the location and the time of the user's query (*Location*), F_3 states that at no point of the protocol the LBS should have continuing knowledge of the location and identity of the user (*Tracing*), and finally F_4 states that at no point the LBS should know the service request of the user (*Attribute*).

5.2.2. Property Semantics

The semantics of the σ-calculus are based on a labeled transition system (LTS). We chose an LTS as the semantic basis to reason about the temporal development of knowledge because transition systems or automata are a common choice for temporal logics.

Formally an LTS is defined as: $LTS = (S, S_0, L, TR)$, where S is a set of states, $S_0 \subset S$ is the set of initial states, L is a set of labels, and TR is a transition relation $S \times L \times S$ relating a state and a label to a state. For the labeled transition system of the σ-calculus, S is the set of all extended processes A, S_0 is the subset of all extended processes with $fr(A) = \sigma^{init}$, L is the set of all labels in Table 5.5, and TR is the labeled reduction relation $\xrightarrow{l}$. As "states" would be an overloaded term in the scope of this dissertation, we define **traces** as follows:

Definition 5.2.1 (Traces)**.** A **trace** tr is a finite derivation of extended processes and labeled relations starting and ending with an extended process ($tr = A_0 \xrightarrow{l_1} A_1 ... \xrightarrow{l_n} A_n$), where each A_i is an extended process and each l_i is a label such that each triple $A_i \xrightarrow{l_{i+1}} A_{i+1}$ is a valid reduction relation. We say a trace is **maximal** if A_n is a reduced process, i.e., $A_n = 0|\sigma$ for some location state σ. We define $tr[i]$ to denote the extended process A_i and $tr[i, j]$ the sub-trace of tr starting from A_i and ending with A_j. Furthermore, we define $tr(A)$ to be the set of all maximal traces tr that start at the extended process A, i.e., $\forall_{tr \in tr(A)}.tr[0] = A$.

All traces are finite because the reduction rules always lead to a reduced process (cf. Lemma 3).

Definition 5.2.2 (Size of traces)**.** The **size of a trace**, denoted by $|tr|$, is the number of extended processes it contains: $|tr| = n + 1$ for $tr = A_0 \xrightarrow{l_1} A_1 ... \xrightarrow{l_n} A_n$. We define the threshold $max_{size}(A)$ to denote the maximum size any trace $tr \in tr(A)$ can possibly reach. Furthermore, the maximum number of possible traces a process can have is denoted by $max_{tr}(A)$.

Recall that $size(A)$ denotes the size of an extended process, which is given by the number of process components of its plain process P (cf. Definition 5.1.1).

Lemma 3. *The size of all traces based on an extended process A that describes an LP3 is limited by $max_{size}(A) = max(2 * (size(A) - 1), 1)$. The number of traces of such an extended process A are limited by $max_{tr}(A) = 2^{2^{size(A)-2}}$.*

Proof. We need to show that for all processes A (1) for all traces $tr \in tr(A)$, $|tr| \leq max_{size}(A)$ and (2) $|tr(A)| \leq max_{tr}(A)$.
(1) As defined a trace represents a sequence of reductions of an extended process. All reduction rules reduce the number of process components (cf. Definition 5.1.1) effectively by at least one, except for the REPL rule. This rule changes the number of process components from n to $2 * (n - 1)$, where n is the size of the original process, i.e., before reduction. Assuming a replication is the first process component and only that one replication occurs, the size of the longest trace cannot exceed $2 * (size(A) - 1)$. As explained in Remark 1 at most one replication process component is necessary to model a protocol.
(2) By definition, different traces only occur when the process contains conditional process components like "if then else." At these process components the possible traces fork

into two. The worst case is a process with only an outermost replication, a query, and conditional processes components. In this case $2^{2^{\#\text{ process components}-2}}$ different traces can be produced. This is equal to $2^{2^{size(A)-2}}$. □

Example 7. One example of a trace is the reduction from Example 5: $tr_1 = A_1 \xrightarrow{THEN} A_2 \xrightarrow{COMPn} A_3 \xrightarrow{QRY} A_1'$, where $A_2 = \text{Compute}(R = MBB(locs)).\text{Query}((\{pid\}, \{R\}, \{serv\}, \{t\}))|\sigma_2$ and $\sigma_2 = \sigma^{init}(\sigma_{Prop} = (true, false, false, false))$, $A_3 = \text{Query}((\{pid\}, \{R\}, \{serv\}, \{t\}))|\sigma_3$ and $\sigma_3 = \sigma_2(\sigma_{Eq} = \{R = MBB(locs)\})$.

As stated previously, the motivation behind our temporal-epistemic logic is to reason about the (location privacy-based) knowledge of the LBS and the development of this knowledge over time. Traces represent the temporal order of the knowledge. But we still need to relate the epistemic properties from Table 5.6 to the traces. Before we describe the satisfaction of epistemic formulas, we need to define what terms constitute privacy leakage in the σ-calculus. As visible from the definition of location states, our calculus relies on sets for locations, groups, services, and time frames, and the main part of the logic is the relation of these sets to singleton sets, as mentioned in Assumption 4. Singleton sets represent unprotected personal data, which implies privacy violation in our attack model (cf. Assumption 2). To formally describe the potential relation of terms and singleton sets, we first define the singleton set equivalence relation $\simeq_S$ to be the smallest relation of terms such that

$RULE1$	$T \simeq_S T$, where T is a term that evaluates to a singleton set
$RULE2$	$\{t\} \simeq_S T$, where $T' = replace(T, t)$ and $\{t\} = T'$

where $replace(T, t)$ replaces all unbound names in a term T by t.

Example 8. Let us consider the term $T = \{loc\} \cup \{n_1, n_2\}$, where n_1 and n_2 are unbound names. Then $T \simeq_S \{loc\}$ because $replace(T, loc)$ returns $\{loc\} \cup \{loc\} = \{loc\}$.

Definition 5.2.3 (State equivalence)**.** We define the state equivalence relation $\simeq$ to be the smallest relation of location states σ (cf. Definition 5.1.4) which are identical in the knowledge of the attacker. That is, both states, if they leak, leak the same basic data type(s).

The relation $\simeq$ can be compared to observational equivalence, which is a common way to model knowledge of attackers in process calculi, e.g., for security protocols. The following rules describe the relation:

ID $(\_, \{((\{pid\}, R, S, F), \_)\}, \_) \simeq (\_, \{((G_1, R, S, F), C), ((G_2, R', S', F'), C')\}, \_)$
where $G_1 \cap G_2 = \{pid\}$ and $dummies \notin C \cup C'$

$LOC1$ $(\_, \{((G, \{loc\}, S, F), \_)\}, \_) \simeq (\_, \{((G, R, S, F), C)\}, \_)$
where $R \simeq_S \{loc\}$ and $l\_diverse \notin C$

$LOC2$ $(\_, \{((G, \{loc\}, S, F), \_\}, \_) \simeq (\_, \{((G, R_1, S, F), \_), ((G', R_2, S', F'), \_)\}, \_)$
where $R_1 \cap move(R_2, 1) \simeq_S \{loc\}$

$SERV$ $(\_, \{((G, R, \{serv\}, F), \_\}, \_) \simeq (\_, \{((G, R, S, F), C)\}, \_)$
where $S \simeq_S \{serv\}$ and $s\_diverse \notin C$

where the underscore _ is a wildcard for any number of sub-states that do not matter in the corresponding state tuples.

Notation 7. G, R, S, T are names for terms that are variable and only used to match with the corresponding other query. The notation is G for a set of identities (group), R for a set of locations (region), S for a set of services, F for a set of time stamps (time frame), and C for a set of constraints.

Remark 4. Although wildcards match any number of sub-states, a location state σ will always consist of the five sub-states as defined in Definition 5.1.4. In the state expression $(\_, \{(\{pid\}, R, S, F)\}, \_)$ the first wildcard matches an integer for the index state σ_{Ind} and the second wildcard matches $\sigma_{Eq}, \sigma_{Prop}, \sigma_{Rel}$ for any equation state, property state, and relation state.

The equivalence relation of location states relies on sets. As we have motivated in Section 2.3, our models and logic use set operations. The relation $\simeq$ reflects this because it shows that semantically the attacker's knowledge is equal to the case where data is leaked if the corresponding set can be reduced to a singleton set. In more detail, rule ID states that two group queries, i.e., queries that hide the identity of a user in a group of pseudonyms, can leak the identity of the user, if the two group sets are disjoint except for the user. Rule LOC1 uses the singleton set equivalence relation to state that the LBS knows the location of a user if the region of the query can be related to a singleton location and the region is not location diverse (indicated by the absence of the $l\_diverse$ proposition in the constraint set C). LOC2 shows the impact of continuing queries by expressing that the LBS knows the location of a user if the intersection of two consecutive query regions, shifted by the possible movement, can be a singleton set. The rule SERV works in a fashion similar to the rule LOC1, relating a non-diverse service set to a singleton set (again, indicated by the absence of the $s\_diverse$ proposition in the constraint set C).

Remark 5. The ID rule's restriction of constraint sets is based on the assumption that two generated dummy user groups of the same user will not be disjoint. If a mechanism

uses a completely random dummy generation, the *dummies* flag should be replaced by k_users.

Now that we have an equivalence relation on location states, we can define the satisfaction of static formulas based on location states. Table 5.7 shows when a state σ satisfies a static epistemic formula δ. The semantics of the static properties are based on the state equivalence relation $\simeq$ and the decision whether an equivalent state exists that contains a query with a singleton set of the corresponding type (group, location, services, time frame). For equivalent states we quantify only over similar distinctive states $\Delta(\sigma)$ (cf. Definition 5.1.7). Now that we have an equivalence relation on location states, we can define the satisfaction of static formulas based on location states. Table 5.7 shows when a state σ satisfies a static epistemic formula δ. While we call formulas δ "static" properties (in contrast to temporal formulas), the static properties are evaluated on dynamic processes. To emphasize the dynamic nature of the semantics, we use notation based on Kripke models. Given a labeled transition system $M = \langle S, V, R\rangle$, where S is the set of states such that $\sigma, \sigma' \in S$ and $\Delta(\sigma) \subset S$, $V : S \to \mathcal{P}(\{id, loc, serv, t\})$ is the valuation function that assigns the holding base properties $p \in \{Id, Loc, Serv, T\}$ to a state, and R is a set of transitions on states in S.

Definition 5.2.4. The relation R on two states σ and σ' is defined by the state equivalence relation: $(\sigma, \sigma') \in R$ iff $\sigma \simeq \sigma'$. The valuation V of a state σ is defined as follows:

$$
\begin{array}{ll}
Id \in V(\sigma) & \text{iff } \sigma_{Qry} \textit{ contains } (\{pid\}, R, S, F) \\
Loc \in V(\sigma) & \text{iff } \sigma_{Qry} \textit{ contains } (G, \{loc\}, S, F) \\
Serv \in V(\sigma) & \text{iff } \sigma_{Qry} \textit{ contains } (G, R, \{serv\}, F) \\
T \in V(\sigma) & \text{iff } \sigma_{Qry} \textit{ contains } (G, R, S, \{t\})
\end{array}
$$

Remark 6. As mentioned before, the term "static" properties may be misleading as σ_{Qry} *contains* x is not a look-up in a static object but changes during the execution of the protocol.

The semantics of the static properties are based on the state equivalence relation $\simeq$ and the decision whether an equivalent state exists that contains a query with a singleton set of the corresponding type (group, location, services, time frame). For equivalent states we quantify only over similar distinctive states $\Delta(\sigma)$ (cf. Definition 5.1.7).

Table 5.7.: Satisfaction of static formulas δ

$\sigma \models p$	*iff* $p \in V(\sigma)$
$\sigma \models K_p$	*iff for some* $\sigma' \in \Delta(\sigma), (\sigma, \sigma') \in R$ *and* $\sigma' \models p$
$\sigma \models \neg\delta$	*iff* $\sigma \not\models \delta$
$\sigma \models \delta_1 \vee \delta_2$	*iff* $\sigma \models \delta_1$ *or* $\sigma \models \delta_2$
$\sigma \models \delta_1 \wedge \delta_2$	*iff* $\sigma \models \delta_1$ *and* $\sigma \models \delta_2$

Example 9. Let us now consider the state $\sigma' = (1, \{((\{pid\}, MBB(locs), \{serv\}, \{t\}), \{k\_users\})\}, \emptyset, empty, \emptyset)$ from Example 5 and the static formula $\delta_1 = \neg K_{Loc}$ from Example 6. We now want to check whether the final state of our protocol satisfies our property $\sigma' \models \delta_1$. As an intermediate step we have to check whether $\sigma' \models K_{Loc}$. σ' does not contain a query with the singleton set $\{loc\}$, it is however equivalent (by application of LOC1 and RULE2) to such a state if we replace in $MBB(locs)$ (recall $locs = \{loc, loc_1, .., loc_n\}$ cf. Notation 2) all other user locations loc_i by loc (by our worst-case assumption (Assumption 4) we can assume all users to be in the same location). Hence, $\sigma' \models K_{Loc}$ does hold and consequently its negation $\sigma' \not\models \delta_1$ does not hold.

As mentioned, the knowledge K is not an epistemic modality in our logic. It still represents the knowledge of a single actor (the attacker) and it is of interest to describe the properties of this knowledge by categorizing the logic of the σ-calculus. As introduced in Subsection 3.2.2, properties of an epistemic logic can be described by a set of axioms **S5**.

Lemma 4. *The (K-less) epistemic property logic of the σ-calculus satisfies the axiom set $\{\boldsymbol{T}, \boldsymbol{N}\} \subset \boldsymbol{S5}$. Contrary to other epistemic logics, the axioms **K**, **4**, and **5** do not hold in the σ-calculus.*

Proof. Our propositional epistemic formulas (K_{Id}, K_{Loc}, K_{Serv}, K_T) do not allow expressing knowledge about implication or causality, making the distribution of knowledge impossible. Thus, axiom **K** does not hold for our logic. The dynamic properties K in our syntax model the certain knowledge (as opposed to "human knowledge" which oftentimes should be categorized as belief) of the LBS about the user's data. The LBS cannot learn a false fact. Hence the axiom **T** holds true for our logic. Positive and negative introspective describe meta-knowledge, that is, an agent knows about her knowledge. This type of property is not possible in our syntax and hence our logic does not satisfy axioms **4** and **5**. The knowledge generalization axiom states that if a fact is true in every *possible world* for the agent, then the agent must know this fact in all possible worlds. In our logic possible worlds for the attacker are based on the four atomic base facts (as introduced in Section 2.6) about the representative user, e.g., the real location. Hence, the distinction of possible worlds coincides with the only fact the only agent reasons

about. Therefore, if for the attacker only worlds are possible in which an atomic fact holds, then the attacker will know it in all of these worlds. This means the axiom **N** is satisfied in our logic. □

After the satisfaction of static formulas, we also define the satisfaction of temporal formulas. Process replication (via the ! operator or explicit repetition) leads to multiple queries in a trace of a protocol. In fact, realistic protocols should include replication to model a realistic LBS application where the users issue more than one query to the same LBS. Temporal formulas ϕ allow statements about sequences of queries. For the satisfaction of the temporal formula $Cont\,\delta$ it is important to describe the state of a process where the LBS' knowledge is synchronized with the system's internal knowledge. This state is the point directly after a query. A stable frame describes this state.

Definition 5.2.5 (Stable frames)**.** A frame (cf. Notation 4) is **stable** if its index state σ_{Ind} equals the cardinality of the request quadruple set of the query state σ_{Qry}.

Table 5.8 shows the satisfaction of temporal formulas based on extended processes and traces. A process A, a trace tr, and an index of this trace i satisfy a formula ϕ according to these rules. For a static formula δ we make use of the satisfaction relation defined above. For the continuing static formula $Cont\,\delta$ it is sufficient to check whether δ is satisfied twice in the course of the given trace. Checking for two satisfying states suffices because the reduction of processes reduces all potentially infinite protocols to finite process traces and a second query in the finite trace represents infinite repetition. The motivation behind our decision that it is sufficient to check two queries, is that we are only interested in the worst-case scenario. That is, we want to show that a protocol cannot leak the location information to the LBS. If one possible scenario exists in which the LBS learns the exact location of a user, we already consider this a violation of our privacy requirement. This worst-case assumption (cf. Assumption 4) combined with the equivalence relation of processes that reduces multiple replications (cf. Remark 1) supports our intuition that finding two different violations of a static property defines a continuing violation for us. The satisfaction of the global and future modality formulas $\Box\phi$ and $\Diamond\phi$ are defined in a standard way such that all (or at least one, respectively) following extended process(es) in the trace satisfy the formula ϕ. The negation, disjunction, and conjunction are defined in the usual way. The satisfaction relation is defined in a recursive way such that all temporal formulas will eventually be evaluated using the static case $A, tr, i \models \delta$. This rule quantifies over the similar distinctive processes $\Omega(tr[i])$ to capture processes with different query states but comparable observable knowledge. In the case of $A, tr, i \models Cont\,\delta$ the rule requires not only two different index states but also the frames to be stable to guarantee that two states are considered that both contain sent queries. An unstable frame is a state with "unsynchronized" knowledge where the LBS has not yet received a query.

Table 5.8.: Satisfaction of properties

$A, tr, i \models \delta$	*iff* $\exists_{A' \in \Omega(tr[i])}.fr(A') \models \delta$
$A, tr, i \models Cont\,\delta$	*iff* $\exists_{0 \leq j < i}.\sigma = fr(tr[i])$ and $\sigma' = fr(tr[j])$ are stable frames and $\sigma_{Ind} > \sigma'_{Ind}$ and $A, tr, j \models \delta$ and $A, tr, i \models \delta$
$A, tr, i \models \Box\phi$	*iff* $\forall_{i \leq j \leq \lvert tr \rvert}.A, tr, j \models \phi$
$A, tr, i \models \Diamond\phi$	*iff* $\exists_{i \leq j \leq \lvert tr \rvert}.A, tr, j \models \phi$
$A, tr, i \models \neg_T\phi$	*iff* $A, tr, i \not\models \phi$
$A, tr, i \models \phi_1 \vee_T \phi_2$	*iff* $A, tr, i \models \phi_1$ or $A, tr, i \models \phi_2$
$A, tr, i \models \phi_1 \wedge_T \phi_2$	*iff* $A, tr, i \models \phi_1$ and $A, tr, i \models \phi_2$

Remark 7. From the semantic definitions one can see that De Morgan's law $\neg_T(\phi_1 \vee_T \phi_2) = \neg_T\phi_1 \wedge_T \neg_T\phi_2$ holds and also the standard equivalence between the $\Box$ and $\Diamond$ modalities $\neg_T\Box\phi \equiv \Diamond\neg_T\phi$.

Remark 8. As can already be seen from Table 5.6, the property syntax does not include implications. The reason why we did not include an implication in static formulas is the ground requirement nature of the four K propositions, which we formulated in such a way that no dependencies exist. As we do not quantify over predicates and our properties are static, the only static properties are combinations of the four K propositions using negation, conjunction, and disjunction. Obvious implications of predicates like $p \vee q \implies p$ hold in our semantics (for the standard definition $p \implies q \equiv \neg p \vee q$) but do not increase the expressiveness of the static formulas.
For temporal formulas one could introduce implication $\phi \implies \phi$ as syntactic sugar for $\neg_T\phi \vee_T \phi$.

Example 10. Let us revisit the extended process based on a reduced version of PrivacyGrid $A_1 = P_2|\sigma^{init}$ from Example 5 and property $\phi_1 = \Box\neg K_{Loc}$ from Example 6 and also consider the trace tr_1 from Example 7 and the state σ' from Example 5. We now want to check whether our trace tr_1 satisfies our global property ϕ_1. For this we have to check $A_1, tr_1, 0 \models \phi_1$ (more on the choice of index 0 in Section 6.1). We have already found in Example 9 that $\sigma' \not\models \delta_1$. Since $\sigma' = fr(tr_1[3])$ and $\phi_1 = \Box\delta_1$, we know that ϕ_1 cannot be satisfied by trace tr_1 as is contains at least one index $i = 3$ where δ_1 is not satisfied.

5.3. Discussion

In the designing of the σ-calculus we made a few decisions that we want to discuss in this section. In Chapter 3 we described the state of the art of process calculi and modal logics. In this chapter we presented a process calculus which is comparable to the applied pi

calculus in its process and term language. The syntax of processes does differ (no parallel processes, no channels, queries instead) but the term language looks similar. Both calculi have some form of equational theory to evaluate terms and functions, even though the σ-calculus does not evaluate terms in the reduction process, e.g., in conditions. A big difference in the two term languages is our focus on sets. The σ-calculus uses sets as an essential part of its language and logic. Sets are used to represent areas, uncertainty, and redundancy. Furthermore, violation of privacy is checked via the potential reduction of sets to singleton sets. The decision to heavily rely on sets is not an obvious one and hence deserves some discussion.

The challenge of the σ-calculus is to model protocols which use protection mechanisms with completely different approaches. While the applied pi calculus (mostly) models protocols with cryptographic primitives and thus has an equational theory that reflects this, our calculus aims at modeling protection mechanisms ranging from generalization and perturbation to cryptography and dummies. Finding a modeling language that is able to express all of these protection mechanisms is not trivial as the underlying logic should neither be too complex (or even undecidable) nor too abstract. When we looked for common ground of all mechanisms, we found the possibility to express all of them with the help of sets. An alternative to our set-based approach is to find a suitable abstraction for each protection mechanism individually and then combine the languages. This approach can easily lead to an undecidable logic and generally to a confusingly complex syntax. Another alternative is to use a more location-centered language with focus on (global) positions and regions, maybe more similar to the ambient calculus (mentioned in Chapter 4). The downside of such an approach is the focus on location itself. Our view on privacy in LBS is that not only the user's location is sensitive data and should be protected. In LBS the users also share possibly identifying data like the type of service they request. A set-oriented logic is capable of treating locations, identities, attributes, and time in a similar way and is, therefore, our choice for the σ-calculus considering our understanding of LBS communication and location privacy.

Another decision of the σ-calculus that deserves discussion is the temporal modal logic without an epistemic modality. In the state of the art (Chapter 3) we described modal logics and also epistemic modal logics. The epistemic modality is used to reason about knowledge in multi-actor systems. Privacy properties can be seen as making statements about "who shall (not) know what." Therefore, epistemic logics are a popular choice for reasoning about privacy as they enable expressing that certain actors should not know certain facts about users. We already briefly motivated that our attack scenario involves the LBS actor itself as the attacker. This scenario represents the strongest passive attacker equivalent to an eavesdropper that can listen to the whole unencrypted communication of all users. Considering the LBS as the only attacker combined with the user-centricity assumption of the σ-calculus, which makes one representative user the target of attacks (and thus also the target to protect), makes it sufficient to only reason about the data of one user and the knowledge of one actor (the LBS). Quantifying over actors as it is possible in modal epistemic logics is, therefore, unnecessary because one only needs to reason about the knowledge of the LBS about the user's data (for example $K_{LBS}loc(user)$). In our attack scenario one is neither interested in other users' data

($K_{LBS}loc(user2)$) nor in other users' knowledge ($K_{user2}loc(user)$). Nested knowledge, as one of the more interesting possibilities of modal epistemic logics, is also of no interest in our scenario because one does not need to reason about the knowledge of the user about the knowledge of the LBS about their data ($K_{user}K_{LBS}loc(user)$). While these types of formulas represent interesting properties, from the perspective of privacy it does not matter whether the user knows that someone compromised their personal data; as long as their data is compromised a privacy violation occurred. An epistemic modal predicate logic as we have sketched above with formulas like $K_{LBS}loc(user)$ can express the same (and many more) properties as our simple K-less logic but is more complex. One needs neither an epistemic modality nor predicates for users. To avoid unnecessary complexity we decided on the four simple propositions as described in the previous sections.

5.4. Summary

In summary, in this chapter we introduced the σ-calculus, a process calculus specifically for modeling location privacy-preserving protocols and verifying location privacy properties. The σ-calculus includes a process and term language which reflect our taxonomy of protection mechanisms by using set operations to obfuscate the personal data. Our calculus makes assumptions about the general LBS setting and the attack scenario, of which the most notable are the worst-case assumption and the representative user. Placing a single user in the focus is an unusual choice for the model of a multi-actor system but intuitive for a privacy setting. The σ-calculus uses operational small-steps semantics for reduction of processes and state transition. While process reduction via small-step rules is common for process calculi, keeping an additional state is not. The verification of privacy properties is done with a temporal modal logic on the semantic construct of process-reduction traces, which represent executions of a protocol. The combination of modal logics and process calculi is not common, whereas temporal and epistemic modal logics are a popular method for verifying privacy. The σ-calculus enables the checking of the important requirements in the field of location privacy, including the often overlooked protection against attribute disclosure.

6. LP3Verif

Parts of this chapter were previously submitted in a journal article to the Journal of Logical and Algebraic Methods in Programming which is currently in review. This chapter builds on the previous one, where we introduced a new process calculus, the σ-calculus. We presented syntax and semantics of LP3s and privacy properties. In this chapter, we first describe model checking algorithms in pseudo-code, which show how privacy properties of LP3 models can be verified. We then show that our model checking is in fact sound and complete. Finally, we present our implementation of a model checker, we call LP3Verif. Section 6.2 gives a detailed description of the implementation and discusses underlying design decisions.

6.1. Model Checking

The model checking question in the σ-calculus is the following: "Does a given LP3 satisfy a certain privacy property?" Above we have shown the satisfaction relation of traces and properties. If we want to check a protocol for privacy violations, we have to consider all possible executions of the protocol. The traces represent these protocol executions. Formally, to check a property ϕ for the complete protocol A, we have to verify $\forall_{tr \in tr(A)}.A, tr, 0 \models \phi$. The index is 0 because we start with the initial protocol A and $tr[0] = A$ for all traces $tr \in tr(A)$. A static property $\phi = \delta$ is, thus, only checked for the initial state of the protocol. $\Box$, $\Diamond$, and $Cont$ are used to reason about whole traces.

Example 11. An example of a protocol that demonstrates that it is not sufficient to consider one trace is the extended process A_1 from Example 5. The shortest possible trace $tr_2 = A_1 \xrightarrow{ELSE} 0|\sigma^{init}$ does satisfy all privacy requirements because no query is sent to the LBS. Based on this trace alone the protocol would look safe. Keeping this example in mind, it is not sufficient to find one trace that satisfies the property, but all possible traces have to be considered.

Definition 6.1.1 (Model checking relation)**.** We define the model checking relation $\vdash$ to be the verification of a property ϕ of a protocol P, i.e., $P \vdash \phi$ if the model checking algorithm $MC(P, \phi)$ returns true. Furthermore, we define the state-based static model checking relation $\sigma \vdash \delta$ to be the successful verification of a static property, i.e., $\sigma \vdash \delta$, if the algorithm $checkStatic(\sigma, \delta)$ returns true.

The algorithm MC (Algorithm 2) shows a pseudo-code model checking algorithm that takes a process P and a privacy property ϕ as input and returns *true* if $P \vdash \phi$ and a witness otherwise. The sub-algorithm *checkStatic* (Algorithm 6) describes the checking of static properties, that is, for a local state. It uses the helper algorithm $checkKB$ described in Algorithm 7 that checks the consistency of a knowledge base. Algorithm 8 shows the sub-algorithm *checkTemporal* for temporal-epistemic properties that follows the satisfaction relation very closely. Section 8.3 will present a complete example that demonstrates the verification process.

Algorithm 2 Model Checking Algorithm

Require: P process, ϕ property
Ensure: $MC(P, \phi)$ returns *true* if $P \vdash \phi$, a witness otherwise

```
1:  procedure MC(P, φ)
2:      T ← buildTraces(P, [])                          ▷ set of traces
3:      normalize φ                                     ▷ bring φ in normal form
4:      F ← getStatic(φ)                                ▷ set of static formulas
5:      S ← getStates(T)                                ▷ set of all states σ
6:      B ← [][]                                        ▷ array of size |S| × |F|
7:      for all s ∈ S do
8:          for all f ∈ F do
9:              if (s, f) not in cache then             ▷ look up state,property in cache
10:                 B[s][f] ← checkStatic(s, f)
11:                 add (s, f), B[s][f] to cache        ▷ add check result to cache
12:             end if
13:         end for
14:     end for
15:     for all t ∈ T do
16:         if ¬ checkTemporal(T, B, φ, t, 0) then
17:             return witness                          ▷ execution trace with value assignment
18:         end if
19:     end for
20:     return true
21: end procedure
```

The MC algorithm is based on an explicit trace building. In line 2, the algorithm recursively applies the reduction rules until the traces have the form of a tree with the process P as root and null processes as leaves. The *buildTraces* algorithm is depicted in Algorithm 3 and discussed below in detail. At this point, we only remark that the number of traces and their size is bounded, as shown in Lemma 3. Even though the bound of the number of traces is double exponential, from our observations in practice a process' number of traces rarely exceeds 10. Explicitly building all possible traces is therefore no performance issue. Line 4 shows a call to the algorithm *getStatic*, which is described in Algorithm 4 and discussed in detail below. In short, the method extracts all static properties δ from a temporal formula ϕ. Line 5 shows a call to the algorithm *getSates*, which is also discussed later in more detail and depicted in Algorithm 5. It gathers all different states σ from the traces. Lines 7-14 show that all combinations of states and static properties contained in the formula are checked. This requires potentially more calls of the *checkStatic* method than a top-down approach where one starts with a temporal formula and only checks the necessary combinations of states and static properties. However, the number of states and static properties is usually quite small and caching the results speeds up the verification of multiple temporal formulas with the same static properties. As motivated in Section 2.6 the four static K properties can be seen as atomic building blocks for more complex location privacy properties, hence most temporal formulas share static properties like $\neg K_{Loc}$. Therefore, the model checking algorithm is based on the assumption that most of the combinations of states and static properties will be used for at least one property anyway. This makes the algorithm simpler while having relatively small overhead (more on that in Chapter 7). While the boolean look-up table B contains the already verified atomic static properties of this algorithm's execution, the cache in lines 9 and 11 represents a global structure, which caches the *checkStatic* results of all temporal properties, making use of the assumption that different temporal properties will be built from the same atomic static properties and different protocols will share most of the states. Lines 15-19 show that all traces are checked for temporal properties, but the verification is stopped when the first violation is found. In line 17 a witness is returned. This witness was already generated in the algorithm *checkStatic*. More on witnesses follows in the discussion of the algorithm *checkTemporal*.

An alternative model checking approach would be to start with a temporal property ϕ and only call the *checkStatic* method for a location state if the temporal property requires it (top-down). However, the privacy properties one usually wants to express are properties with the global modality at the beginning, which implies that whole traces have to be checked. Consequently, this approach does not improve the performance for common temporal properties, where usually all traces and all states along the traces have to be checked. This top-down approach starting with temporal properties would decrease the number of calls to *checkStatic* for properties that are violated right away after checking of one state. However, as we argued above, for the verification of multiple temporal properties (like **F1-F4**), it is highly likely that more than one state has to be checked. Therefore, our bottom-up approach does not perform worse in practice while being simpler as it utilizes a look-up table for temporal properties.

Algorithm 3 Algorithm for generating all traces

Require: P process, tr trace
Ensure: $buildTraces(P, tr)$ returns all traces starting with process P

```
1: procedure BUILDTRACES(P, tr)
2:     add P to tr
3:     if P is 0 process then
4:         return {tr}
5:     else
6:         L ← ∅                                   ▷ empty set of traces
7:         for each process P' such that P ⟶ P' do
8:             L ← L ∪ BUILDTRACES(P', tr)
9:         end for
10:        return L
11:    end if
12: end procedure
```

The algorithm *buildTraces* as depicted in Algorithm 3 is a recursive approach to generating a set of all possible traces. As can be seen in line 2 of Algorithm 2, we call *buildTraces* with an empty trace in the model checking algorithm. The idea behind this algorithm is to recursively search for successor processes, i.e., processes that are related via the reduction relation $\xrightarrow{l}$. The base case of the recursion can be seen in line 3: the reduction stops at the null process. In this case, a singleton set with just the current trace, ending with a 0 process, is returned (line 4). If P is not a terminating process, in lines 7-9 all traces starting at successor processes are added to the set of traces L.

The algorithm *getStatic* as depicted in Algorithm 4 is a method to extract an essential set of static properties from a temporal property ϕ. This algorithm assumes the formula ϕ to be in the normal form. One can see in line 3 of Algorithm 2 that ϕ will be in normal form. This normal form is based on two simple principles. The normal form requires, fist, that negations are in the innermost position and, second, that conjunctions and disjunctions should be static formulas if possible. Recall that the syntax of both, static formulas and temporal formulas, allows negations, conjunctions, and disjunctions. This is necessary because of the temporal property $Cont$, which does not preserve boolean operations, e.g., $Cont\,(\neg\delta) \not\equiv \neg_T Cont\,\delta$. The first principle means that negations should be static negations $\neg$ if possible and as close to the four ground properties (K_{Id}, K_{Loc}, K_{Serv}, and K_T) as possible. The second principle means that conjunctions and disjunctions should be static formulas if possible, i.e., avoiding $\wedge_T$ and $\vee_T$ in temporal properties if not surrounding $Cont$. Example 12 demonstrates how to transform a formula into the normal form.

Example 12. Let us consider the temporal property $F_2' = \Box\neg_T(K_{Loc} \wedge_T K_T)$ which is equivalent to the formula F_2 as defined in Notation 6. This formula is not in normal form as it violates both principles. F_2' uses temporal conjunction where static conjunction would be possible and it has a negation outside the conjunction. To transform this

Algorithm 4 Algorithm for extracting all static properties

Require: ϕ temporal property in normal form
Ensure: *getStatic*(ϕ) returns the set of static properties F

```
1:  procedure GETSTATIC(φ)
2:      F ← ∅                                   ▷ empty set of static formulas
3:      switch φ do
4:          case δ
5:              F ← F ∪ δ
6:          end case
7:          case Cont δ
8:              F ← F ∪ δ
9:          end case
10:         case ¬_T φ_1
11:             F ← F ∪ GETSTATIC(φ_1)
12:         end case
13:         case φ_1 ∨_T φ_2
14:             F ← F ∪ GETSTATIC(φ_1)
15:             F ← F ∪ GETSTATIC(φ_2)
16:         end case
17:         case φ_1 ∧_T φ_2
18:             F ← F ∪ GETSTATIC(φ_1)
19:             F ← F ∪ GETSTATIC(φ_2)
20:         end case
21:         case □φ
22:             F ← F ∪ GETSTATIC(φ_1)
23:         end case
24:         case ◇φ
25:             F ← F ∪ GETSTATIC(φ_1)
26:         end case
27:     end switch
28:     return F
29: end procedure
```

formula into the normal form, one can first easily replace the temporal negation and conjunction with their static equivalents. This is a big change in the composition of the formula as F_2' is of the form $\Box\neg_T(\phi \wedge_T \phi)$ where both ϕ are static formulas δ. The new formula with static operators is of the form $\Box\phi$ where ϕ is a static formula of the form $\neg(\delta \wedge \delta)$. The second and last step to transform F_2' into normal form is to "pull" the negation to the innermost position. Using De Morgan's law, the transformation results in the normal form $F_2'' = \Box\neg K_{Loc} \vee \neg K_T$, which is equivalent to formula F_2.

Lines 4-9 of Algorithm 4 show that in both cases, static and continuing property, only δ itself is added to the set of static properties F. In all other cases of temporal properties, recursive calls to the algorithm *getStatic* are made. For all finite properties ϕ the algorithm terminates with either δ or $Cont\,\delta$. An alternative approach would be to always use the set $\{K_{Id}, K_{Loc}, K_{Serv}, K_T\}$, which could be used to verify all temporal properties. However, for the generation of witnesses this is not the case. While knowing the boolean value for the four ground properties suffices to also know the boolean value for $K_{Loc} \vee K_T$ or $\neg K_{Id}$, deductions from witnesses are more complicated. If K_{Loc} is not satisfied and a witness w_1 demonstrates this and we also know that K_T does not hold from a witness w_2, these two witnesses alone are not sufficient to show that $K_{Loc} \vee K_T$ does not hold. Combining these witnesses (which is not trivial!) would solve this problem but the model checking algorithms rely on a SAT solver to generate witnesses. To avoid the need for merging witnesses, the *getStatic* algorithm requires the formula to be in normal form and adds all static sub-formulas to the set. The normal form guarantees that witnesses can be generated by the SAT solver.

Example 13. Let us consider the temporal formula ϕ' from the previous example. The call of *getStatic*(ϕ') returns the set $\{\neg K_{Loc} \vee \neg K_T\}$ as ϕ' is of the form $\Box\phi$ (case in line 21) where ϕ is a static formula (case in line 4). A more complex example of a formula in normal form is $\phi_1 = (\Box\delta_1) \wedge_T \Diamond\delta_2 \vee_T Cont\,\delta_3$. The algorithm call *getStatic*(ϕ_1) returns the set $\{\delta_1, \delta_2, \delta_3\}$ as the conjunction and disjunction are temporal operators, hence the cases in lines 17, 13, and 7 are matched, respectively.

The algorithm *getStates* as depicted in Algorithm 5 is a method to extract from a set of traces all states σ that have different query states. The algorithm iterates in a straightforward way over all traces and all extended processes of these traces. Lines 5 and 6 show that only states with new query states are added to the set. At the end the set S represents states, in which the LBS has received different queries.

Remark 9. Not distinctive states should not be confused with equivalent states as introduced in Definition 5.2.3. Equivalent states are identical in their knowledge of the LBS about the representative user's data. All states with the same query state are equivalent states, but not all equivalent states have identical query states σ_{Qry}.

Example 14. Let us consider the extended process A_1 from example Example 5. The two traces tr_1 (Example 7) and tr_2 (Example 11) form the set $T = \{tr_1, tr_2\}$. The algorithm call *getStates*(T) returns a set $S = \{\sigma^{init}, \sigma'\}$, where $\sigma' = (1, \{((\{pid\}, MBB(locs), \{serv\}, \{t\}), \{k\_users\})\}, \emptyset, empty, \emptyset)$ is from Example 5. Depending on the order of

Algorithm 5 Algorithm for extracting all distinctive states (cf. Definition 5.1.7)

Require: T set of traces
Ensure: S is the set of states contained in T with distinctive query states

```
1: procedure getStates(T)
2:     S ← ∅                                              ▷ empty set of states
3:     for each trace tr ∈ T do
4:         for each process A ∈ tr do
5:             if ¬∃_{σ'∈S}.A.σ.σ_Qry = σ'.σ_Qry then     ▷ σ_Qry is new
6:                 S ← S ∪ A.σ
7:             end if
8:         end for
9:     end for
10:    return S
11: end procedure
```

the traces and states, a different set could be returned. However, the set S will always contain two states where one contains an empty query state and the other the query state $(\{pid\}, MBB(locs), \{serv\}, \{t\})$.

6.1.1. Static Epistemic Properties

The *checkStatic* algorithm (Algorithm 6) and the *checkKB* algorithm (Algorithm 7) are closely connected as the two algorithms together make up the model checking of static properties. While *checkStatic* does the preparation for the SMT solver, *checkKB* performs the actual call to the solver. In more detail, Algorithm 6 creates a new knowledge base in line 4. This knowledge base can be filled with asserted facts and an SMT solver can then check this knowledge base for consistency, i.e., whether any facts contradict each other. These facts or statements can be equations, inequations, boolean functions, or variable assignments. The *checkStatic* algorithm proceeds by adding facts derived from the query state σ_{Qry}. Furthermore, in line 6, a FunctionTheory is asserted. This line represents the facts about the functions from Table 5.2 (*MBB, move, dist, noise, redund, hash, and random*). In illustration, we explicitly describe the variable assignments of the function *MBB*. The assignments describe the semantics of the function based on discrete, grid-like integer locations. The minimal bounding box function MBB takes as input a set of locations and returns the minimal rectangle region that contains all locations in the set. For a hypothetical 2 by 2 grid with indexes 1 to 4 (top left is 1, top right 2, etc.), the facts $MBB(\{1,2\}) = \{1,2\}$ and $MBB(\{1,4\}) = \{1,2,3,4\}$ are 2 of the 16 (for each element of the power set $\mathcal{P}(\{1,2,3,4\})$) equations to describe the function. Similarly explicitly defined are the other functions like *dist*, which returns the integer distance between two locations, or *move*, which takes a set of locations and an integer velocity as input and returns the set of locations that are reachable with a number of steps equal to the velocity value. In lines 7-19 of *checkStatic* the properties of the constraint set C are asserted as facts and in lines 20-22 the relations in C. Finally,

Algorithm 6 checkStatic Algorithm

Require: σ location state, δ static property
Ensure: $checkStatic(\sigma, \delta)$ returns true if the state σ satisfies the property δ, produces a witness otherwise

```
1:  procedure CHECKSTATIC(σ, δ)
2:      KB ← new Knowledge Base
3:      for each query ∈ σ_Qry do
4:          assert G_i = query.G ∧ R_i = query.R        ▷ assert adds fact to KB
5:          assert S_i = query.S ∧ F_i = query.F
6:          assert FunctionTheory                       ▷ e.g. MBB({x}) = {x}
7:          for each prop ∈ query.P do
8:              switch prop do
9:                  case k_dummies
10:                     assert |G_i| > 1
11:                 end case
12:                 case l_diverse
13:                     assert |R_i| > 1
14:                 end case
15:                 case s_diverse
16:                     assert |S_i| > 1
17:                 end case
18:             end switch
19:         end for
20:         for each rel ∈ query.Rel do
21:             assert rel
22:         end for
23:     end for
24:     return CHECKKB(KB, δ)
25: end procedure
```

after all σ-related facts have been asserted, the algorithm calls *checkKB* in line 24 with the filled knowledge base and the unchanged static property δ.

The algorithm *checkKB* (Algorithm 7) first, in lines 2-21, adds asserted requirements derived from the static formula δ to the knowledge base (already filled with the facts from Algorithm 6). Lines 8 and 16 show stack-like syntax to remove the last added assertion from the knowledge base, e.g., line 8 removes the assertion from line 4. This popping process is only done for K_{Id} and K_{Loc} because these two are checked with the rule ID and LOC2, which not only rely on the singleton set equivalence relation $\simeq_S$. The SAT method is the call to the SMT solver to do a satisfiability check of the knowledge base and it returns true if the asserted facts do not contradict each other, and false otherwise. After a false satisfiability check the method *produceWitness* generates a variable assignment which demonstrates the contradiction. Line 35 shows that the *checkKB* algorithm returns true if the knowledge base was consistent.

Example 15. Let us consider a simple example of a call to the algorithm *checkStatic* and its corresponding call of *checkKB*. The call $checkStatic(\sigma', \neg K_{Loc})$ returns *true*. The state $\sigma' = (1, \{((\{pid\}, MBB(locs), \{serv\}, \{t\}), \{k\_users\})\}, \emptyset, empty, \emptyset)$ contains one query such that lines 4 and 5 add $G_1 = \{pid\}$, $R_1 = MBB(locs)$, $S_1 = \{serv\}$, and $F_1 = \{t\}$ to the knowledge base KB. Then $checkKB(KB, \neg K_{Loc})$ is called. Line 25 leads to a recursive call negating the result of $checkKB(KB, K_{Loc})$. Line 12 adds $|R_1| = 1$ to the knowledge base. The satisfiability check in line 13 returns *true* as $R_1 = MBB(locs) = \{loc\}$ a possible assignment and does not contradict with $|R_1| = 1$. The assertion in line 17 will also not change the satisfaction of the knowledge base as R_2 is not defined and, hence, not restricted in any way. Thus, $checkKB(KB, K_{Loc})$ returns *true* and consequently $checkKB(KB, \neg K_{Loc})$ returns *false* and produces a witness (more on witnesses in subsection 6.2.4).

Lemma 5. *The algorithms checkStatic and checkKB (Algorithm 6 and Algorithm 7) are sound and complete in regard to the satisfaction relation of states and static properties. That is, checkStatic*(σ, δ) *returns true* $\iff \sigma \models \delta$.

Proof. The bi-implication can be proven via case distinction on the seven types of static properties.
The first case is $\delta = K_{Id}$. The proof of this bi-implication can be split into two implications.

$$checkStatic(\sigma, K_{Id}) \iff |G_i| = 1 \text{ or } |G_1 \cup G_2| = 1 \tag{1}$$

implies

$$\sigma \models K_{Id} \iff \text{for some } \sigma' \in \Delta(\sigma), (\sigma, \sigma') \in R \text{ and } \sigma' \models K_{Id} \tag{2}$$

This implication can be shown by splitting (1) into two equations.

$$G_i| = 1 \tag{1a}$$

$$|G_1 \cup G_2| = 1 \tag{1b}$$

(1a) implies (2): $|G_i| = 1$ is equivalent with σ_{Qry} contains a query with a singleton group. Hence, (1a) implies $Id \in V(\sigma)$, which implies that $\sigma \models Id$. As $(\sigma, \sigma) \in R$ holds,

Algorithm 7 checkKB Algorithm

Require: KB knowledge base, δ static property

Ensure: $checkKB(KB, \delta)$ returns true if the knowledge base KB is consistent with the property δ, returns false and produces a witness otherwise

```
1: procedure CHECKKB(KB, δ)
2:     switch δ do
3:         case K_Id
4:             assert |G_i| = 1                                   ▷ for i ∈ {1, 2}
5:             if ¬SAT(KB) then          ▷ check knowledge base for satisfiability
6:                 return produceWitness(KB)
7:             end if
8:             pop                                          ▷ removes last assertion
9:             assert |G_1 ∩ G_2| = 1
10:        end case
11:        case K_Loc
12:            assert |R_i| = 1
13:            if ¬SAT(KB) then
14:                return produceWitness(KB)
15:            end if
16:            pop
17:            assert |R_1 ∩ move(R_2, v_max)| = 1
18:        end case
19:        case K_Serv
20:            assert |S| = 1
21:        end case
22:        case K_T
23:            assert |T| = 1
24:        end case
25:        case ¬δ_1
26:            return ¬ CHECKKB(KB,δ_1)    ▷ also produce a witness if result is true
27:        end case
28:        case δ_1 ∨ δ_2
29:            return CHECKKB(KB,δ_1) ∨ CHECKKB(KB,δ_2)          ▷ same as above
30:        end case
31:    end switch
32:    if ¬SAT(KB) then
33:        return produceWitness(KB)
34:    end if
35:    return true
36: end procedure
```

because the identity $\sigma \simeq \sigma$ is contained in the state equivalence relation, $\sigma \models K_{Id}$ also holds.
(1b) implies (2): $|G_1 \cup G_2| = 1$ is equivalent with σ_{Qry} contains two queries and their respective groups have exactly one element in common. If any of the two groups is a singleton set, the implication is the same as in case (1a) and thus proven. Otherwise, by application of rule ID $\sigma \simeq \sigma'$ holds where σ' contains a singleton group. Hence, $\sigma \models K_{Id}$ holds.
(2) implies (1): If for some $\sigma' \in \Delta(\sigma), (\sigma, \sigma') \in R$ and $\sigma' \models K_{Id}$ holds, only two possible cases can be true. Either $\sigma \models K_{Id}$ or $\sigma \simeq \sigma'$ and $\sigma' \models K_{Id}$. The first case is trivial as it can only hold if σ contains a query with a singleton set which is equivalent with (1a). The second implication also holds as the only (non-identity) rule of the relation $\simeq$ that relates to a singleton group is ID and this rule implies that $G_1 \cup G_2 = \{pid\}$ which is equivalent to (1b).
The other three proofs for (K_{Loc}, K_{Serv}, and K_T) follow a similar pattern where the identity of the relation $\simeq$ is equivalent to the first case of the corresponding property in Algorithm 7 and the four explicit rules of $\simeq$ are equivalent to potential second cases.
checkStatic$(\sigma, \neg\delta_1)$ returns true only if the negated property holds. Line 26 of Algorithm 7 is equivalent to the satisfaction of static formulas for negations.
The disjunction of static properties in line 29 of Algorithm 7 similarly is equivalent to the semantics definition.
checkStatic$(\sigma, \delta_1 \wedge \delta_2)$ follows implicitly as this static formula is not defined in the grammar of the model checker and must be derived from the negation and disjunction (cf. Remark 7). □

6.1.2. Temporal Properties

The *checkTemporal* algorithm (Algorithm 8) uses the look-up table B with the static truth values of the four static K properties of all different σ. For each type of temporal property the algorithm either uses the content of B or a recursive call for another index as defined in the satisfaction of temporal formulas (cf. Table 5.8). While line 4 is straightforward, lines 7-13 deserve a brief discussion. Recall, the continuing property $Cont\,\delta$ is satisfied if the static property δ holds at the current state and has held for a previous query as well. The *for* loop in line 8 ranging from 0 to $i-1$ and the disjunction in line 10 show that the algorithm searches for at least one previous state which satisfies δ and line 13 shows that the current state must also satisfy δ.

Theorem 6. *The model checking algorithm MC is sound and complete. That is* $P \vdash \phi \iff \forall_{tr \in tr(A)}.A, tr, 0 \models \phi$, *where* $A = P|\sigma^{init}$.

Proof. The proof can be split into two implications for soundness and completeness.

$$P \vdash \phi \implies \forall_{tr \in tr(A)}.A, tr, 0 \models \phi \qquad (\Rightarrow)$$

$$\forall_{tr \in tr(A)}.A, tr, 0 \models \phi \implies P \vdash \phi \qquad (\Leftarrow)$$

($\Rightarrow$) can be proven via case distinction on the different types of properties ϕ: δ, $Cont\,\delta$, $\Box\phi$, $\Diamond\phi$, $\neg_T\phi$, $\phi_1 \vee_T \phi_2$, and $\phi_1 \wedge_T \phi_2$.

Algorithm 8 checkTemporal Algorithm

Require: T set of traces, B boolean array (as defined in Algorithm 2), ϕ normalized property, t trace, i index

Ensure: $checkTemporal(T, B, \phi, t, i)$ returns true if the sub-trace of t starting at $t[i]$ satisfies the property ϕ

```
1: procedure CHECKTEMPORAL(T, B, φ, t, i)
2:     switch φ do
3:         case δ
4:             return B[t[i]][δ]
5:         end case
6:         case cont δ
7:             bool ← false
8:             for j ← 0, i − 1 do
9:                 if |t[i].σ_Qry| > |t[j].σ_Qry| then
10:                    bool ← bool ∨ B[t[j]][δ]
11:                end if
12:            end for
13:            return bool ∧ B[t[i]][δ]
14:        end case
15:        case ¬_T φ1
16:            return ¬ CHECKTEMPORAL(T, B, φ1, t, i)
17:        end case
18:        case φ1 ∨_T φ2
19:            return CHECKTEMPORAL(T, B, φ1, t, i) ∨
                       CHECKTEMPORAL(T, B, φ2, t, i)
20:        end case
21:        case □φ
22:            bool ← true
23:            for j ← i, |t| − 1 do
24:                bool ← bool ∧ CHECKTEMPORAL(T, B, φ, t, j)
25:            end for
26:            return bool
27:        end case
28:    end switch
29: end procedure
```

The static case $P \vdash \delta$ holds only if the *checkStatic* algorithm returns true as can be seen from line 4 in Algorithm 8 ($P \vdash \delta \iff checkStatic(\sigma^P, \delta)$). Via Lemma 5 $checkStatic(\sigma, \delta) \iff \sigma \models \delta$ holds and $A, tr, 0 \models \delta$ per definition ($\exists_{A' \in \Omega(tr[i])}.fr(A') \models \delta$) is true if an equivalent process is found such that its frame satisfies δ. The implication is proven for the static case because $A \cong A$ holds and, hence, whenever $P \vdash \delta$, $\exists_{A' \in \Omega(tr[i])}.fr(A') \models \delta$ must hold for $A' = A$.
A similar argument can be made for the other cases of temporal properties as the *checkStatic* algorithm's definition of temporal properties is equivalent to the satisfaction relation, we have proven Lemma 5, and, thus, we have proven ($\Rightarrow$).

($\Leftarrow$) can also be proven via case distinction on the different types of properties of ϕ.
$A, tr, 0 \models \delta$ holds only if an equivalent process with the current index of the trace exists and its frame satisfies the static property ($\exists_{A' \in \Omega(tr[0])}.fr(A') \models \delta$). The model checking algorithm makes use of the property that equivalent processes have equal frames (see Lemma 2). Following this, one can derive that if $A, tr, 0 \models \delta$ holds, then also $P \vdash \delta$ holds (Lemma 5).
$A, tr, 0 \models Cont\,\delta$ holds only if an index exists such that the corresponding frame is stable and its index state less than the current one's and the two states corresponding to both indexes satisfy the static property, i.e., $\exists_{0 \leq j < i}.\sigma = fr(tr[i])$ and $\sigma' = fr(tr[j])$ are stable frames and $\sigma_{Ind} > \sigma'_{Ind}$ and $A, tr, j \models \delta$ and $A, tr, i \models \delta$. Two stable frames with different location states can only happen if their query states have different cardinality (Notation 4), hence, line 9 in the algorithm *checkTemporal* is implied by the satisfaction relation.
$A, tr, 0 \models \Box\phi$ holds only if all indexes of the trace satisfy the property. Line 26 of *checkTemporal* returns only true if all indexes of the trace return true and, thus, the implication holds.
The other cases of temporal properties follow the same argument as $\Box\phi$. Again, the conjunction follows from the negation and the disjunction. Additionally, $\Diamond\phi$, follows from the negation and the global modality (cf. Remark 7). □

6.2. Implementation

Based on the σ-calculus we implemented LP3Verif, a tool that automatically verifies location privacy properties for LP3 processes following the model checking algorithms starting from Algorithm 2. The tool is implemented in Java and takes as input text files that model an LP3 of the σ-calculus and a list of properties following the described syntax of the formulas.

Figure 6.1 shows a sketch of the modeling and verification process. The user models an LP3 in the σ-calculus process language from a pseudo-code algorithm. The user also formulates their location privacy properties and writes both into an `.lp3` file. LP3Verif takes a path to this file as command line input and the result for each property: either *true* or a witness otherwise. The graph shows a theoretical procedure where a single static property is checked for the first trace. As can be seen from Figure 6.1, the

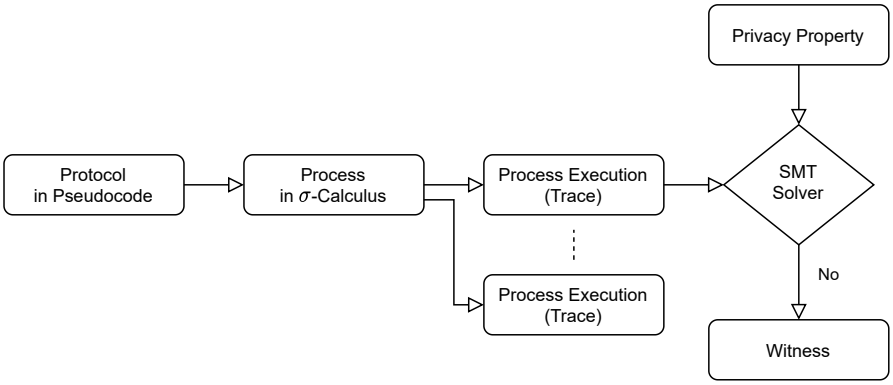


Figure 6.1.: Verification process from pseudocode to witness

checking of static properties is done with an SMT solver. The choice for the SMT solver CVC4 is based on its support of finite set theory, which is an important part of the σ-calculus. LP3Verif uses CVC4 as back-end solver, which is called for the *SAT(KB)* check in the sub-algorithm CHECKKB. Only one witness per property is generated as a single violation of a requirement is enough proof that the property does not hold. Therefore the actual workflow consists of calling the SMT solver for all static properties and only generating a witness for the first violating process state. The graph does not cover the process of caching, either, as it is described in Algorithm 2. The data structure B caches the results coming from the SMT solver and consequently only new properties follow the depicted process flow. We use the *get-value* command provided by CVC4 to generate a minimal value assignment for the locations of other users that satisfies both, the protocol description as well as the negated property, as our witnesses represent counter examples. Chapter 8 presents a case study where Figure 8.4 shows such a witness. The source code of the tool can be accessed in the following repository: `https://www.tuhh.de/sts/research/data-protection-machine-learning/lp3verif.html`.

The grammar of the file encoding differs from that of the protocol language we presented above as it follows a line-by-line syntax with reserved keywords for process components and the closing of a construct via `end`. Figure 6.2 shows an exemplary snippet of an `.lp3` file.

In `.lp3` files the null process can be omitted and an `if` construct does not necessarily need the else. An `end` closes the last opened construct, i.e., process, replication, if, or while. Properties are just one line following the `property` keyword and the name and hence do not need to be closed. To denote the box (and also the derived diamond) we make use of the standard notation in LTL where G represents box and F diamond. We implemented a standard precedence for the operators such that `not` has the highest and `G/F` the lowest precedence. Parentheses can be used to deviate from this order.

```
process Alice_simp
    !
        if k_users
            Compute(R=MBB(locs))
            Compute(h=hash(pid))
            Query(h,R,serv,t)
        end
    end
end

property p
    G not (K_loc and K_t)
```

Figure 6.2.: Alice's simple protocol in `.lp3` format

6.2.1. Grammar

In this subsection, we discuss the grammar of `.lp3` files in detail. As explained above, the syntax of processes in *lp3* format differs from the syntax as introduced in Subsection 5.1.1. An `.lp3` file contains a protocol following the grammar, we will introduce in the next paragraph, and a number of properties following a syntax very similar to the one introduced in Subsection 5.2.1. Table 6.1 shows the token types used in the *lp3* format. In the following we use a syntax for regular expressions, where `*`, `+`, and `?` denote the repetition of an expression zero or more times, one or more times, and zero or one times, respectively.

Integers: An integer literal is a sequence of digits, where no leading zero is allowed, e.g., `01` is not a valid integer.

Symbols: Symbols are any sequence of characters used for names, terms, or functions. A symbol must not start with a leading digit and must not be a reserved word.

Reserved words: Some character sequences are not symbols because they belong to the set of reserved words, which may only be used in specific contexts. The full set of reserved words encompasses all process keywords, all formula keywords, and the two declarations: *process* and *property*. The process keywords are: `!`, `if`, `else`, `end`, `while`, `Compute`, `Query`, and `kQuery`. The formula keywords are: `G`, `F`, `not`, `and`, `or`, and `Cont`.

Remark 10. As Table 6.3 will show, our grammar does not distinguish between static and temporal negations, disjunctions, and conjunctions. As from a semantic point of view these boolean operations are equivalent, we can assume the formulas to be of a form close to the normal form where all negations, disjunctions, and conjunctions are assumed to be static if possible, e.g., only negations in front of a *Cont* or a $\Box$ are temporal formulas and all other occurrences are static. The resulting syntax allows users to formalize properties without having to think about the composition of the formula and whether an operator is static or temporal.

Table 6.1.: Token types for `.lp3` format

Token Type	Format	RegEx	Examples
<integer>	sequence of digits (without leading zero)	`0|([1-9][0-9]*)`	`0 2 10`
<symbol>	sequence of characters (not beginning with a digit or a reserved word)	`[a-zA-Z][0-9a-zA-Z_]*`	`a t1`
Reserved words	character sequences, which are reserved for special keywords and may not be symbols	`(process|if|else|end|while|!|Compute|QuerykQuery|property|G|F|not|and|or)`	`! if end`

The grammar of the *lp3* format allows two types of complex expressions: processes and properties. Both have their own sub-grammar. The definitions of processes and properties are given in Table 6.2 and Table 6.3, respectively. Note that, while most function names are self-explanatory, the function `noiset` is a function for perturbing temporal data, whereas `noise` can be used to perturb location data. Both, processes and properties use the following set of expressions `<expr>`.

an integer literal	`<integer>`
a function `<func>`	`MBB|move|hash|rand|noise|noiset|redund|swap`
a `<term>`, which is one of:	
a symbol	`<symbol>`
a function application	`<func>(<term>+)`
a relation `<rel>`	`<term>(=|<|>|subset|supset)<term>`
a proposition `<prop>`	`k_users|dummies|l_diverse|s_diverse|<rel>`

Table 6.2.: Grammar of processes

a `<component>`, which is one of:	
a compute	`Compute(<symbol>=<term>)`
a query	`Query(<symbol>,<symbol>,<symbol>,<symbol>)`
a multi query	`kQuery(<symbol>,<symbol>,<symbol>,<symbol>)`

a replication	`!`
	`  <component>+`
	`end`

a condition	`if <prop>`
	`  <component>+`
	`(else`
	`  <component>+)?`
	`end`

a while	`while <prop>`
	`  <component>+`
	`end`

a `<process>`	`process <symbol>`
	`  <component>+`
	`end`

Table 6.3.: Grammar of properties

a `<static>` property, which is one of:	
an epistemic	`K_id|K_loc|K_serv|K_t`
a negation	`not <static>`
a conjunction	`<static> and <static>`
a disjunction	`<static> or <static>`
a `<temporal>` property, which is one of:	
a static	`<static>`
a continuing	`Cont <static>`
a global	`G <temporal>`
a future	`F <temporal>`
a negation	`not <temporal>`
a conjunction	`<temporal> and <temporal>`
a disjunction	`<temporal> or <temporal>`
a `<property>`	`property <symbol>` `<temporal>`

Table 6.4.: Grammar of `.lp3` files

an lp3 file	`<process>` `<property>+`

Table 6.4 shows the simple grammar of an `.lp3` file built from a process and possibly multiple properties. This grammar disregards empty lines. Our implementation of a parser processing `.lp3` files filters empty lines and also comments indicated by `#`.

6.2.2. Program Structure

The implementation of LP3Verif follows the object-oriented programming paradigm. Protocols and properties are instances of class objects, which are built via nesting, e.g., a replication process has another plain process as class variable which describes the process to be repeated. Overall the program entails 30 files (4251 LOC) in 6 packages: *processCalculus*, *terms*, *properties*, *modelChecker*, *utils*, and *main*. The package *processCalculus* contains all classes that describe processes, states, and traces. The package

terms contains classes for all terms, equations, and relations. The package *properties* encompasses all classes that describe epistemic or temporal formulas. The package *modelChecker* encompasses all classes and methods to perform the verification and witness generation. While *utils* contains all necessary helper structures for file reading, parsing, and solver calling, the package *main* only contains the main function.

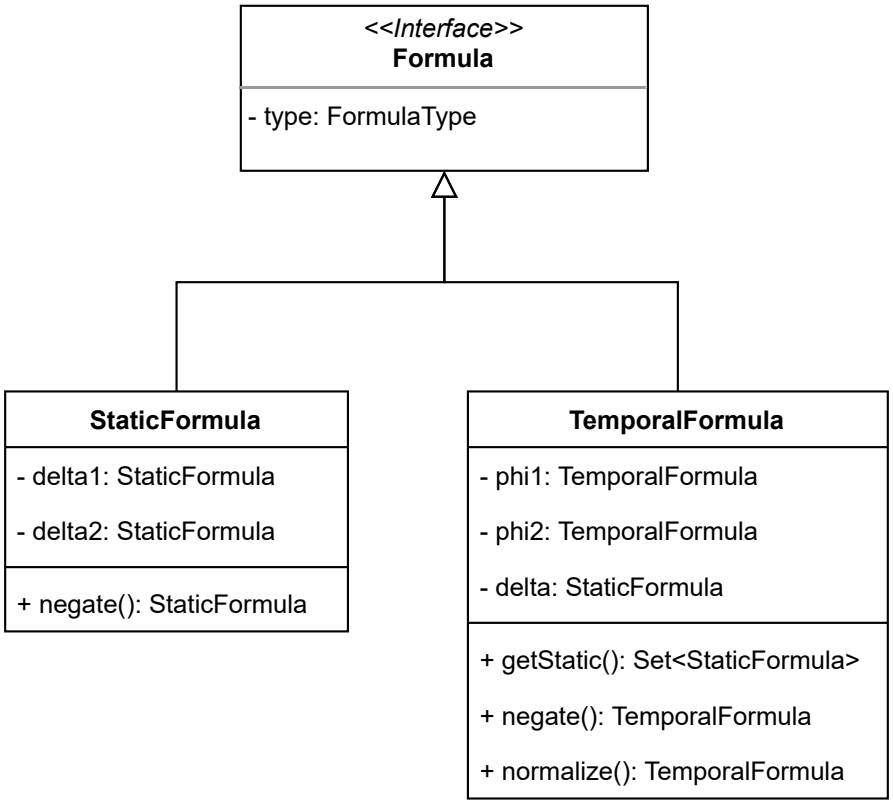


Figure 6.3.: UML class diagram of formulas

Figure 6.3 shows a UML class diagram depicting classes and methods of formulas and Figure 6.4 shows the classes and methods of processes. The diagrams are structured in a similar way as both, formulas and processes, are interfaces implemented by two different classes. A formula is either a static formula or a temporal formula and a process is either a static process or an extended process. The diagrams omit the obvious "getter" and "setter" methods, constructors, as well as all methods linked to checking for equality of objects.

The Java code snippet in Figure 6.5 shows how a protocol is constructed using the *processCalculus* and *terms* classes. The code snippet in Figure 6.6 shows how a privacy property is constructed using the *formula* classes. The two snippets show how the process and formula classes, as shown in the class diagrams, are used to construct a complete protocol and a verifiable property.

As LP3Verif utilizes an external solver for the verification of static properties, i.e.,

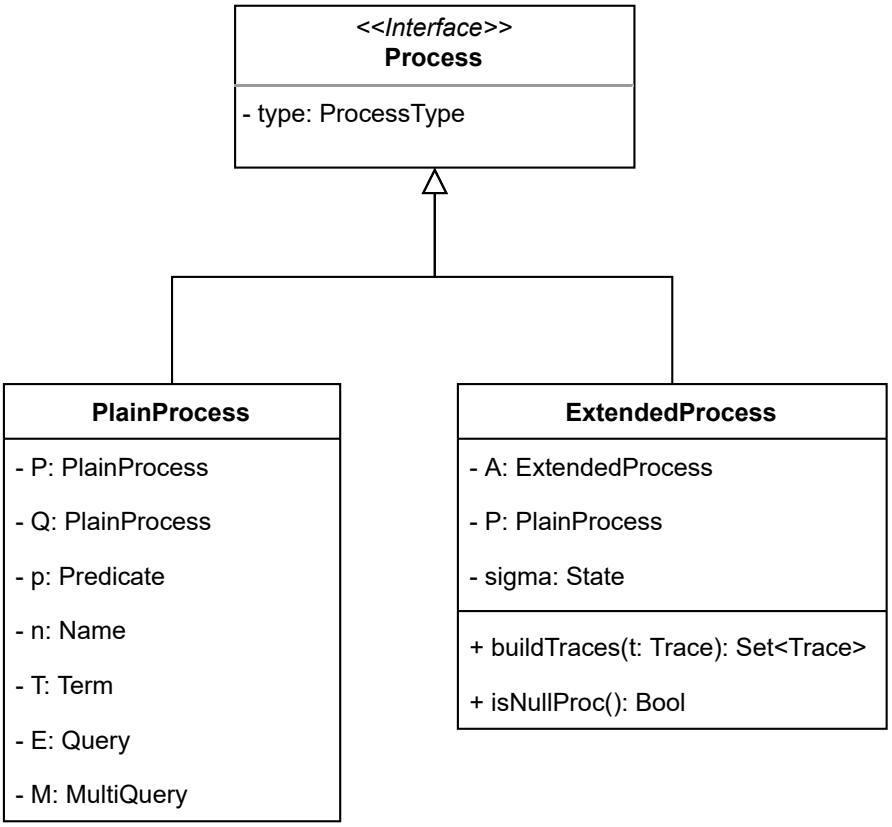


Figure 6.4.: UML class diagram of processes

the checking of data leakage in process states, the more interesting decisions of data structures are the ones for the parser, the reduction of processes, and the verification of temporal properties. The parser that translates `.lp3` files to process calculi, as depicted in Figure 6.5, simply checks whether a file satisfies the grammar, as described in Subsection 6.2.1. The non-empty (and non-comment) lines are checked for keywords and each line creates a temporary process (of the corresponding type), which is put on a stack. The keyword `end` triggers a popping of the stack to close the last opened process. Finally, when the last `end` is reached the complete process is built by iteratively appending the closed processes on the stack. A similar approach is used to parse properties where ambiguities (negation, disjunction, and conjunction can be either temporal or static) are resolved by assuming temporal formulas until an epistemic operator is reached (cf. Remark 10).

As the algorithm *buildTraces* (cf. Algorithm 3) suggests, LP3Verif recursively applies all possible reductions until a null process is reached. This recursion performs sufficiently well as Chapter 7 will show. The verification of temporal properties also uses recursion to resolve temporal operators like $\Box$. However, the recursion level is only as deep as the number of nested operators and also shows a good performance as Chapter 7 will

```
Predicate kusers = new Predicate(PredicateType.K_USERS);
Name R = new Name("R");
Function MBB = new Function(FunctionType.LOC,LocFunction.MBB);
List<Term> locs = Collections.singletonList(new Term(TermType.RESNAME,new
    ReservedName(ResNames.LOCS)));
Term Term1 = new Term(TermType.FUNC,MBB,locs);
Name h = new Name("h");
Function hash = new Function(FunctionType.INT,IntFunction.HASH);
List<Term> pid = Collections.singletonList(new Term(TermType.RESNAME,new
    ReservedName(ResNames.PID)));
Term Term2 = new Term(TermType.FUNC,hash,pid);
Term serv = new Term(TermType.RESNAME,new ReservedName(ResNames.SERV));
Term t = new Term(TermType.RESNAME,new ReservedName(ResNames.T));
Term region = new Term(TermType.NAME,R);
Term id = new Term(TermType.NAME,h);
Query query = new Query(id,region,serv,t);
PlainProcess P0 = new PlainProcess(ProcessType.NULL);
PlainProcess Pquery = new PlainProcess(ProcessType.QUERY,query,P0);
PlainProcess Pcomp2 = new PlainProcess(ProcessType.COMP,h,Term2,Pquery);
PlainProcess Pcomp1 = new PlainProcess(ProcessType.COMP,R,Term1,Pcomp2);
PlainProcess Pif = new PlainProcess(ProcessType.CONDITION,kusers,Pcomp1,P0);
PlainProcess Prepl = new PlainProcess(ProcessType.REPL,Pif);
State init = new State();
ExtendedProcess A = new ExtendedProcess(ExtendedProcess.ProcessType.PLAINSTATE,
    Prepl,init);
```

Figure 6.5.: Alice's simple protocol in Java code

show. Both implementations, reduction and temporal verification, could also be done without recursion by isolating rule applications and temporal checks, respectively. This would allow for a performance-oriented multi-threading approach. However, Chapter 7 will show that the two methods *buildTraces* and *checkTemporal* present no bottleneck of Lp3Verif.

6.2.3. Back-End Solver

In this subsection, we give an overview of the SMT solver integration into LP3Verif. We already mentioned in Subsection 6.1.1 what part of the verification and witness generation is done by the SMT solver. LP3Verif translates the query state σ_{Qry} into asserted statements readable for the SMT solver CVC4. As discussed in Subsection 6.2.3, the theories are the difference between a SAT solver and an SMT solver. The theories are sorted, such that variables and functions have types and the types have values to quantify over. Accordingly, the preprocessing step of LP3Verif before asserting statements is declaring and defining the necessary sorts, functions, and variables. From the predefined set of available theories CVC4 provides, LP3Verif uses the integer theory and the finite set theory. The only types necessary for LP3Verif are integers and sets of integers. The

```
StaticFormula Kloc = new StaticFormula(StaticFormula.FormulaType.KLOC);
StaticFormula Kt = new StaticFormula(StaticFormula.FormulaType.KT);
StaticFormula andStatic = new StaticFormula(StaticFormula.FormulaType.
    CONJUNCTION,Kloc,Kt);
TemporalFormula andTemporal = new TemporalFormula(FormulaType.STATIC,andStatic)
    ;
TemporalFormula not = new TemporalFormula(FormulaType.NEGATION,andTemporal);
TemporalFormula phi = new TemporalFormula(FormulaType.GLOBAL,not);
```

Figure 6.6.: Privacy property of Alice's simple protocol in Java code

snippet in Figure 6.7 shows the declaration and definition of the sorts, variables, and functions using only integers and sets.

```
( define-sort Location () Int )
( define-sort Region () (Set Location) )
( define-sort ID () Int )
( define-sort Group () (Set ID) )
( define-sort Time () Int )
( define-sort Frame () (Set Time) )
( define-sort Service () Int )
( define-sort Services () (Set Service) )
( define-sort Speed () Int )
( declare-fun MBB (Region) Region )
( declare-fun move (Region) Region )
( declare-fun hash (Group) Group )
( declare-fun rand (Group) Group )
( declare-fun noise (Location) Region )
( declare-fun redund (Location) Region )
( declare-fun swap (Group) Group )
```

Figure 6.7.: Definition of types and declaration of functions

In a similar fashion all variables have to be declared and, as explained in Subsection 6.2.3, the functions have to be explicitly defined. Lines like
`( assert (!  (= (MBB (singleton 1)) (singleton 1)) :named MBB1) )`
define the results of functions based on the inputs. The expression `(!  :named [Name])` is SMT syntax for naming assertions which enables the solver to return the names of contradicting assertions. The middle part of the line `(= (MBB (singleton 1)) (singleton 1))` is an equation stating that the function call of *MBB* with input {1} is equal to the set {1}.

After all types, functions, and variables have been defined, the actual query state is asserted. For this, all queries are asserted in lines like this:

`( assert (! (and (= G1 pids) (= R1 (MBB locs)) (= S1 (singleton serv)) (= F1 (singleton t))) :named QUERY1) )`. The keyword `=`, again, denotes equality of integers or sets and the keyword `and` denotes boolean conjunction (of equations). The constraints from constraint set C are also added to the facts via asserted statements as shown in Algorithm 6. Finally, one property at a time is asserted, for instance: `( assert (! (not (not (= 1 (card S1)))) :named PROPERTY) )`. The keyword `not` denotes boolean negation. The SAT command from Algorithm 7 is done in CVC4 via the line `( check-sat )`. If this satisfiability check is successful, the next command can be:
`( get-value (pid G1 R1) )`
to get value assignments for the specified variables, in this case a pseudonym, a group (pseudonym set), and a region (location set). The resulting output then looks like `(pid 2) (G1 (singleton 2)) (R1 (union (singleton 0) (singleton 1)))`. This output can be interpreted as individual assignments, where the variable *pid* has the value 2, the variable $G1$ the value $\{2\}$, and the variable $R1$ the value $\{0, 1\}$.

6.2.4. Output

This subsection describes the output format of LP3Verif and how the returned strings should be interpreted. The general form of output strings is one line, for each property, determining whether a property is satisfied by the protocol or not. This first line is either "The protocol [name] satisfies the property [formula]" or "The protocol [name] does not satisfy the property [formula]." If the protocol does not satisfy the property, a witness follows the first line. Such a witness spans multiple lines and contains information about the representative user's personal data (identity, location, requested service, and timestamp), other users' location, queries issued during the execution, and the complete execution trace leading to the violating state. While the first three lines use the output from the SMT solver CVC4, the trace is a generated string with processes in syntax of the σ-calculus and arrows indicating a reduction. A witness, as described, can take the following form:

```
                                Witness:
          Representative User:  (pid 1) (loc 2) (serv 0) (t 1)
                  Other User Location:  (loc_1 1) ...
Query:  ((G1 (singleton 1)) (R1 (union (singleton 1) (singleton 2))) ...
       Trace:  !Compute().Query()|σ -> Compute().Query()|σ' -> ...
```

Ellipses indicate the omission of more information like additional query parameters and longer traces. For a better understanding of witnesses for location-related properties LP3Verif also outputs a grid that explains the meaning of the intergers of locations. As described in Example 1, location values can be interpreted on a grid. The lines following a witness show a three by three grid with numbers from 1 (top left) to 9 (bottom right). This grid is useful for more complicated witnesses, where possibly movement is involved. For instance, such a grid could explain why a witness including two region queries with

intersecting areas demonstrates a violation of location privacy. Why $R1 = \{1, 2\}$ and $R2 = \{5, 6, 8, 9\}$ is a valid witness, is maybe not immediately obvious without said grid.

6.3. Discussion

This chapter describes our model checking approach for the σ-calculus as well as our implementation of the model checking algorithms. In Chapter 3 we introduced the state of the art of model checking including symbolic model checking and bounded model checking, which are standard approaches to deal with the state explosion problem. Due to the standard notation of model checking and also the term "state explosion problem," for the scope of this section, we divert from our terminology and talk about "states" as locations in traces (as opposed to "processes" cf. definition 5.2.1). Our model checking procedure is neither symbolic nor bounded. This has several reasons and the main one is that the model resulting from LP3 protocols in the σ-calculus does not suffer from state explosion as these models in practice stay small (< 20 states) and sparse (number of states roughly equals the number of transitions). Bounding the number of steps is neither necessary nor efficient on LP3 reduction models as the interesting states are usually at the end of a trace (protocols usually end with the issued query). Symbolic representation of the process reduction with OBDDs is an alternative approach that could improve the performance of the model checking process. However, symbolic model checking is not guaranteed to improve performance (compared to explicit model checking) on small models. The translation of the states and transitions into an OBDD are for small models more expensive than the benefit of efficiency for checking on OBDDs.

While our model checking approach uses explicit state transition systems, we use the principle of structural model checking to reduce the number of checks that have to be performed. Instead of checking the value of the static propositions for all states, we check only the subset of states with different query states σ_{Qry} (partial order reduction). This approach uses the structural property of the model that all states with the same epistemic state from the perspective of the LBS can be treated the same. For instance, all states that correspond to an execution of a protocol before a query has been issued are merged into one state. This reduction reduces the number of static checks that have to be done but not the number of look-ups when traversing the traces for temporal properties. An alternative approach is to abstract the model, hence also reducing the number of states to be traversed for the checking of temporal properties. While this approach slightly increases performance, it destroys the link between states and protocol execution steps. This link makes witnesses more helpful as these can be accompanied by an execution trace leading to the violation. Due to the application-specific nature of our properties (including a non-standard temporal operator) and the focus on witness generation, we decided to implement our own model checking procedure rather than build on an existing model checker like NuSMV.

We do, however, use CVC4 as back-end solver because we reduce the model checking of static properties to an instance of SAT, where we use integer and finite set theories. A SAT solver cannot provide satisfiability results quantifying over integers and sets.

SMT solvers supply theories like integers to be quantified over and usually also allow commands that return a value assignment that satisfies a set of asserted statements (or a counterexample). Popular SMT solvers are Z3, MathSAT, or STP. While all of these include the theory of integers and linear arithmetic, none of the three includes a theory for finite sets at the time of writing this dissertation. Finite set theory is not a standard theory and hence not supported by most SMT solvers. CVC4 is one of the few that comes with support for finite sets, including operations for union, intersection, membership, and cardinality.

One limitation of the described approach of this dissertation in general is that assumption that the modeling process starts at a pseudo-code algorithm. This is a fair assumption when a new location-privacy preserving mechanism is designed. However, in practice already implemented mechanisms exist and one might argue that it would also be of interest to evaluate these mechanisms. In this dissertation we do not describe how to abstract σ-calculus protocols from source code. The reason for that is two-fold. First, we assume that implementations of LPPMs follow one of the presented approaches or are based on a pseudo-code algorithm that can be used to model the protocol in the description language of the σ-calculus. Second, the idea of the σ-calculus is to verify location privacy of LPPM abstractions and verifying implementations is not in the scope of this dissertation as in the process of implementing a pseudo-code algorithm new bugs can be introduced that were not present in the algorithm.

6.4. Summary

In summary, this chapter builds on the previous chapter, which introduced the σ-calculus, by describing a model checking approach for the verification of privacy properties. Our finite-state model checking is based on the explicit building of all traces and uses an a priori checking of all static properties of a selection of states based on a partial-order reduction approach. Our model checking algorithms uses a cache to avoid redundant verifications. Furthermore, we introduce LP3Verif in this chapter, which is a tool that implements the model checking algorithms in Java. We introduce `.lp3` files, which describe a protocol and a list of properties. LP3Verif can be called with a path to such a file and returns the verification results of the properties. The result is either *true* or a witness, which is a variable assignment that demonstrates how the property could be violated by an execution of the protocol. LP3Verif does not use an established model checker, instead we only call the SMT solver *CVC4* as back end for satisfiability checks.

7. Evaluation

In the motivation of this dissertation, we identified three unsolved issues in the field of LPPMs: a missing unified description of LPPMs and location privacy, missing formal guarantees, and the derived lack of comparability of LPPMs. In this chapter we evaluate how the σ-calculus and LP3Verif help solving these issues. This chapter gives an evaluation of the expressiveness of the σ-calculus, the performance of our implementation LP3Verif, and the general decisions and assumptions we made. In Section 7.1 we come back to the literature surveys from Chapter 2 and use the most cited papers to evaluate the σ-calculus. In Section 7.3 we evaluate our tool LP3Verif to assess the general performance but also to show the impact of certain design decisions on the performance. In Section 7.4, finally, we discuss threats to validity of our framework and the general approach of this dissertation.

7.1. σ-Calculus

In this section we evaluate our process calculus. We first demonstrate the expressiveness of our modeling language by modeling a set of most-cited LPPM protocols from our literature review and then show verification results by presenting the respective privacy guarantees of these protocols based on our ground requirements (introduced in Section 7.3 and formally defined in Notation 6).

7.1.1. Expressiveness of Protocol Language

Parts of this subsection and especially the following table were previously submitted in a journal article to the Journal of Logical and Algebraic Methods in Programming which is currently in review. During our search key-based literature review we found 38 papers that met the search key "LPPM" and proposed a new algorithm for privacy preservation in an online LBS setting. Of these 38 papers 9 described a purely probabilistic mechanism that gives probabilistic guarantees. We exclude these papers because our σ-calculus only supports binary properties without probabilities (cf. Assumption 4). The remaining 29 protocols form our evaluation set. We modeled these protocols in the LP3 syntax (all protocol models are available at `https://www.tuhh.de/sts/research/data-protection-machine-learning/lp3verif.html`). Table 7.1 shows the results of our study. The different protocols are listed to the left and the four ground requirement properties **F1**-**F4** (as defined in Notation 6) on the top. As a brief recapture, $F_1 = \Box\neg K_{Id}$ is the Identity requirement, $F_2 = \Box\neg(K_{Loc} \wedge K_T)$ the Location requirement, $F_3 = \Box\neg_T Cont\, K_{Loc} \wedge K_{Id}$ the Trace requirement, and $F_4 = \Box\neg K_{Serv}$ is the Attribute requirement. The checkmarks indicate that the protocol satisfied the corresponding property of the column. The checkmarks in parentheses describe the case where the identity is hidden via hashing which, depending on the background knowledge, is sufficient to re-identify sequential queries with the same hash. Several papers only describe an algorithm to anonymize locations but do not provide an LP3 as defined in Section 5.1. In these cases, we assumed no

further protection, i.e., the corresponding data type is the user's data in a singleton set. This assumption implies that protocols which do not specify protection of a certain data type are assumed to leak this data.

Table 7.1.: Verification results of 29 LP3 models

Paper	Protocol	Category	F1	F2	F3	F4
[18]	Beresford et al.	Mix Zones	✗	✗	✓	✗
[19]	Freudiger et al.	Mix Zones	✗	✗	✓	✗
[22]	Gong et al.	Mix Zones	✗	✗	✓	✗
[20]	Xinxin et al.	Mix Zones	✗	✗	✓	✗
[21]	MobiMix	Mix Zones	✗	✗	✓	✗
[2]	Privacygrid	Generalization	✗	✗	✗	✓
[26]	CliqueCloak	Generalization	✓	✗	✓	✗
[24]	PRIVE	Generalization	(✓)	✗	✗	✗
[4]	Lee et al.	Generalization	✗	✗	✗	✗
[32]	ReverseCloak	Generalization	✗	✗	✗	✗
[27]	Casper	Generalization	✗	✓	✓	✗
[23]	L2P2	Generalization	(✓)	✗	✓	✓
[34]	Xu et al.	Generalization	✗	✓	✓	✗
[29]	feeling-based	Generalization	✗	✓	✓	✗
[35]	Location Diversity	Generalization	✗	✓	✓	✗
[42]	Kato et al.	Dummies	✗	✓	✓	✗
[37]	Kido et al.	Dummies	✗	✓	✓	✗
[36]	SpotME	Dummies	✗	✓	✓	✗
[43]	SybilQuery	Dummies	(✓)	✓	✓	✗
[41]	MobiPriv	Dummies	(✓)	✗	✗	✗
[6]	Assam et al.	Perturbation	✗	✓	✗	✗
[44]	Hoh et al.	Perturbation	✗	✗	✗	✗
[5]	CAP	Perturbation	✗	✓	✓	✗
[63]	CaDSA	Cache	✗	✗	✓	✗
[56]	MobiCrowd	Cache	✗	✗	✗	✗
[58]	Ghinita et al.	Crypto	(✓)	✓	✓	✗
[8]	Trust no one	Crypto	(✓)	✓	✓	✓
[7]	MaPIR	Crypto	(✓)	✓	✓	✗
[62]	SpaceTwist	Crypto	(✓)	✓	✓	✗

As Table 7.1 indicates, the σ-calculus is able to model all of the 29 protocols. The size of the protocols is between 6 and 13 lines. The σ-calculus can also verify (or falsify) the four ground requirement properties for all protocols, showing the differences in their guarantees. Only one protocol, namely "Trust no one," satisfies all four properties. One can also see from the table that similar approaches get similar results in the four property columns. This is because we model very high-level protocols based on their core ideas how to handle the personal data. We conclude the σ-calculus offers a possibility

to formally model LPPM protocols, verify location privacy requirements, and compare protocols based on their privacy guarantees.

It should be noted that some categories of mechanisms do not score well in our table due to our worst-case assumption. For instance, Mix Zones serve a specific goal of privacy in limited spaces but do not tackle the location privacy in other places. Other protocols suffer from our attacker model and the worst-case assumption. Reducing the number of queries by caching reduces in practice the probability of leaks but is not modeled in our worst-case scenarios. It is noteworthy mentioning that the modeling process itself is not straightforward and involves thinking about the specific privacy mechanism and how it should be applied. We see this "phase of deliberation" as a benefit because it steers software developers towards thinking about their approach before implementing it.

In Section 2.2 we already showed formalizations of LPPMs in the syntax of LP3Verif. Here, we repeat these protocols and show similar LPPMs from the same category to demonstrate the small differences and discuss their impact. The full list of the 29 formalized LPPMs is found in Appendix A. In the field of Generalization mechanisms, we compare the protocol PrivacyGrid [2] with a protocol proposed by Lee et al. in 2010 [4]. Figure 7.1 shows the LP3 of PrivacyGrid and Figure 7.2 shows the LP3 with the mechanism by Lee et al.

```
process PrivacyGrid
2    !
        if k_users
4           Compute(R=MBB(locs))
            if s_diverse
6               Query(pid,R,servs,t)
            end
8       end
    end
10  end
```

Figure 7.1.: Generalization protocol with mechanism PrivacyGrid [2]

Both protocols make a region query as can be seen from the minimum bounding box computation of the locations *locs* in lines 4 and the query having the computed region R as location parameter in lines 6 and 5, respectively. The difference of the two protocols, apart from the obvious additional if-statement, is just one letter: while in PrivacyGrid in line 6 the service set *servs* is requested, in the protocol by Lee et al. in line 5 the singleton service set *serv* with just the representative user's service request is in the query. With the additional service diversity check, these additions make the difference between satisfying property **F4** (attribute protection), in the case of PrivacyGrid, and not, in the case of Lee et al. (cf. Table 7.1).

Next, let us compare CAP [5] and a mechanism proposed by Assam and Seidl [6] both

```
process Lee
    !
        if k_users
            Compute(R=MBB(locs))
            Query(pid,R,serv,t)
        end
    end
end
```

Figure 7.2.: Generalization protocol with mechanism by Lee et al. [4]

from the category of Perturbation mechanisms. Figure 7.3 shows the LP3 of CAP and Figure 7.4 shows the LP3 with the mechanism by Assam and Seidl.

```
process CAP
    !
        Compute(R=noise(loc))
        Query(pid,R,serv,t)
    end
end
```

Figure 7.3.: Perturbation protocol with mechanism CAP [5]

```
process Assam
    !
        if k_user
            Compute(T=noise(ts))
            Query(pids,locs,servs,T)
        end
    end
end
```

Figure 7.4.: Perturbation protocol with mechanism by Assam and Seidl [6]

The obvious difference is again an additional if-statement but apart from that both protocols look similar. However, CAP perturbs the location of the user (line 3) and the approach by Assam and Seidl perturbs the time stamp (line 4). Both are valid approaches to obfuscate spatio-temporal data. Assam and Seidl choose a perturbation approach that relies on group privacy, which is usually not the case as Perturbation mechanisms were introduced as an alternative to trusted third party approaches. The mechanism by Assam and Seidl makes a group query where the time stamp is the same

noisy time window for all users. From Table 7.1 one can see that this approach, in contrast to the mechanism CAP, does not prevent long-term tracking (**F3**). This is due to our decision that for long-term tracking of users the correct time stamps are not of relevance. Hence, temporal perturbation is not effective.

Finally, let us compare MaPIR [7] and a mechanism proposed by Jaiswal and Nandi called TrustNoOne [8] both from the category of Crypto mechanisms. Figure 7.5 shows the LP3 of MaPIR and Figure 7.6 shows the LP3 of TrustNoOne.

```
process MaPIR
    !
        Compute(h=hash(pid))
        Compute(R=redund(loc))
        Query(h,R,serv,t)
    end
end
```

Figure 7.5.: Crypto protocol with mechanism MaPIR [7]

```
1 process TrustNoOne
    !
3       Compute(h=hash(pid))
        Compute(R=redund(loc))
5       Compute(S=redund(serv))
        Query(h,R,S,t)
7   end
end
```

Figure 7.6.: Crypto protocol with mechanism TrustNoOne [8]

The difference between the two cryptography-based protocols is line 5 of TrustNoOne and the consecutive query with the redundant service set S instead of the explicit service *serv* in MaPIR. This shows that in TrustNoOne attribute disclosure is also considered a privacy risk while in MaPIR it is not. This difference makes TrustNoOne the only protocol in Table 7.1 to satisfy all four properties while MaPIR does not satisfy the attribute protection **F4**.

7.1.2. Expressiveness of Properties

In Subsection 7.1.1 we have shown that the syntax of the σ-calculus can model the most common non-probabilistic LPPMs. In this subsection we show that the property language of the σ-calculus suffices to express the common location privacy guarantees in the LBS context. We analyzed the 29 papers from our literature survey for their

promised privacy guarantees and mentioned privacy definitions. Table 7.2 shows the 29 papers, the name of their respective privacy guarantee(s), and the formalization in the σ-calculus.

Table 7.2.: Location privacy protection goals and σ-calculus syntax

Protection Goals	**Paper**	**Formalization**
Prevent long-term tracking	[18–22, 29, 42]	$\Box\neg_T Cont\,(K_{Id} \wedge K_{Loc})$
Protect user anonymity	[2, 4, 23, 24, 26, 27, 34, 35] [5, 8, 41, 43, 44, 58]	$\Box\neg K_{Id}$
Protect user location	[2, 23, 32, 35–37, 41, 43] [7, 8, 62, 63]	$\Box\neg K_{Loc}$
Protect user preference	[23, 35, 41, 43, 58]	$\Box\neg K_{Serv}$
Protect spatio-temporal data	[6, 29, 56]	$\Box\neg(K_{Loc} \wedge K_T)$

The privacy guarantees of the 29 papers can be reduced to achieving the five protection goals: 1) prevent long-term tracking, 2) protect user anonymity, 3) protect user location, 4) protect user preference, and 5) protect spatio-temporal data. These five privacy properties can be formalized in the syntax of the σ-calculus as can be seen in Table 7.2. Of the five formulas four correspond directly to our four ground requirements **F1-F4**. In detail, 1) corresponds to formula F_3, 2) to F_1, 3) has no correspondence, 4) corresponds to F_4, and 5) to F_2. From Table 7.1, one can see that protection goal 2) (anonymity) is stated by 14 papers and is therein the most frequent protection goal. Protection goal 3) (location privacy) is the second most frequent goal with 12 papers. The other three are not so common among the 29 papers. Goal 1) (tracking) is found in 7 papers, goal 4) (preference privacy) in 5 papers, and protection goal 5) (spatio-temporal privacy) is found only in 3 papers. As can be seen from the numbers and listed papers in Table 7.2, some papers named more than one protection goal. The table list all the goals a paper stated for their proposed mechanism. We did not infer any related protection goals, e.g., the three papers which named protection goal 5) are not listed for goal 3) even though an obvious implication exists.

7.2. Scalability and Runtime

In this section, we evaluate the runtime complexity of the model checking algorithms presented in Section 6.1. As our model checking makes use of SAT solving, we also have to include the complexity of solving the satisfiability problem. We evaluate the overall complexity of our model checking procedure based on all its algorithms afterwards.

Table 7.3 shows the upper bound estimations of the runtime complexity of each algorithm and the overall estimation at the bottom. The algorithm *buildTraces* recursively iterates over all process components to build the trace tree. In the (theoretical) worst case, *buildTraces* has exponential complexity when each process component consists of replication. Hence, *buildTraces* has an exponential runtime of $\mathcal{O}(2^n)$ with the number

of process components as the parameter n. However, in the average case *buildTraces* has a linear runtime of $\mathcal{O}(n)$ when only one replication and one condition occurs and each subsequent process has to be visited at most four times. The algorithm *getStatic* recursively searches temporal formulas for static sub-formulas, which in theory has linear runtime based on the depth of the nested temporal formula. However, the algorithm is only called for formulas in normal form, which requires the formulas to have the static formulas as outermost as possible. As infinite nesting of temporal operators does not model realistic properties, we can assume temporal formulas to have a fixed depth of at most c, where c is a constant. Hence, based on this assumption, the runtime of *getStatic* is constant ($\mathcal{O}(1)$). The algorithm *getStates* iterates over all traces once to gather all unique query states. Hence, *getStates* has an exponential runtime of $\mathcal{O}(2^n)$ with n, again, being the number of process components. Similarly to the runtime of *buildTraces*, the average case of *getStates* has linear complexity as in practice only $c\,n$ number of traces exist. The algorithm *checkStatic* in the worst case performs a SAT check for each property. Hence, *checkStatic* has an exponential runtime of $\mathcal{O}(2^m)$, where m represents the number of boolean variables that need to be considered for the SAT check. The algorithm *checkTemporal*, in the worst case, iterates exponentially many states of the trace. Hence, *checkTemporal* has an exponential runtime of $\mathcal{O}(2^n)$ as the number of states of a trace is limited by the number of process components. The overall runtime complexity of the model checking algorithm *MC*, as some algorithms are executed multiple times but none of them are nested, is the maximum of the above runtimes where the complexity of *checkStatic* and *checkTemporal* have to be adjusted to reflect multiple callings. *MC* calls *checkStatic* for each unique state and static formula. As the number of static formulas is constant, we can derive a (worst-case) exponential complexity of $\mathcal{O}(2^{n+m})$ where in the average case the number of unique states is less than the number of process components n, leading to a runtime of $\mathcal{O}(n\,2^m)$ for the average case of calling *checkStatic*. *MC* calls *checkTemporal* in the worst case for each trace. As the number of traces can be 2^n, we can derive an exponential complexity of $\mathcal{O}(2^{2n})$ in the worst case. In the average case the number of traces is a constant multiple of the number of process components and *checkTemporal* is not called for each trace. Hence, the average runtime is $\mathcal{O}(n\,2^n)$ for the calls of *checkTemporal*. Overall the resulting runtime is $\mathcal{O}(2^{2n})$ for the worst case as we can assume that, when scaling n, the number of process components, linearly, the number of boolean variables m for the SAT solver scales only logarithmically due to efficient encoding. These boolean variables represent all variables used in assertions (where integers and sets are encoded).

We can derive from this approximation that our model checking algorithm has exponential runtime, which means it does not scale well. However, while the average case is still exponential, the exponent n will in practice never grow too large. From our experience an n greater than ten is already uncommon and we cannot envision a protocol so complex to utilize more than 20 process components.

Table 7.3.: Runtime complexity of all algorithms (worst case)

Algorithm	**Runtime**
buildTraces	$\mathcal{O}(2^n)$
getStatic	$\mathcal{O}(1)$
getStates	$\mathcal{O}(2^n)$
checkStatic	$\mathcal{O}(2^m)$
checkTemporal	$\mathcal{O}(2^l)$
MC	$\mathcal{O}(2^{n+l})$

7.3. LP3Verif

This section presents a performance evaluation of LP3Verif. We first show the general performance of LP3Verif as is. Then we show how our design decisions impact the performance of LP3Verif. In more detail, we first show how the decision of "pre-checking" static properties performs in comparison to classical path traversal. Then we compare a purely explicit model checking approach to LP3Verif with the partial-order reduction optimization. Finally, we show the impact of caching by comparing LP3Verif to a version without cached look-ups. The experiments were conducted one a Windows 10 commodity notebook with a $2.8\,GHz$ processor. All executions were averaged over 10 runs where the coefficient of variation (standard deviation divided by mean) was always below 1 %.

Table 7.4 shows the average execution time of LP3Verif when verifying the four properties **F1** to **F4**. Two interesting observations can be made from the table. First, the execution time of LP3Verif varies strongly for the four properties and, second, the execution time also varies for the different protocols. The first fact can be observed from the difference in the average execution time of each property **F1** to **F4**. While the Attribute requirement **F4** takes 0.34 seconds on average, the Trace requirement **F3** takes 10.74 seconds on average. This drastic difference can be explained when looking at the corresponding formulas $F_3 = \Box\neg_T Cont\, K_{Loc} \wedge K_{Id}$ and $F_4 = \Box\neg K_{Serv}$. The formula F_3 contains two static formulas (K_{Loc} and K_{Id}) while F_4 only contains one static formula K_{Serv}. This is a significant difference as will be later discussed in the analysis of the execution time of the different parts of the verification. Furthermore, the formula F_3 consists of a nested temporal formula ($\Box$ and *Cont* inside) while F_4 only consists of one temporal formula ($\Box$). The global future modality $\Box$ is more easily verified than *Cont* because it is sufficient to find one state that violates the static property as a counter example for F_4 whereas one has to iterate over whole traces for F_3 as *Cont* reasons about the past and requires checking all states because the static property could eventually be true (more than once).

The second fact, that execution times strongly vary between the 29 protocols, can be seen from the difference in the overall execution time for protocols. The verification of the protocol L2P2 takes 33.40 seconds while, in contrast, the verification of the protocol by Hoh et al. takes only 2.25 seconds. This observed difference in execution time can be explained by two major aspects of the protocols. First, L2P2 is with 14 lines one

of the longest protocols and the one by Hoh et al. only has 8 lines. Additionally, the former produces 8 possible traces while the latter produces only half the number of that. Thus, the numbers and sizes of traces, that have to be considered during the verification, differ. Another aspect that has an impact on the execution time, is the number of satisfied privacy properties the protocols have. The protocol by Hoh et al. does not satisfy any of the four properties whereas the protocol L2P2 satisfies two of the four privacy properties (cf. Table 7.1). Generating a witness for an obvious violation is faster than successfully verifying the absence of such a violation.

Table 7.4.: Execution times of LP3Verif (in s)

Protocol	**F1**	**F2**	**F3**	**F4**	**Overall**
Beresford et al.	1.22	1.65	1.81	0.59	5.43
Freudiger et al.	0.86	1.83	2.02	0.69	5.49
Gong et al.	0.74	1.66	1.79	0.57	4.79
Xinxin et al.	0.73	1.61	1.79	0.57	4.74
MobiMix	0.69	1.61	2.22	0.57	5.16
Privacygrid	0.40	1.89	13.31	0.14	15.78
CliqueCloak	0.35	1.80	12.66	0.26	15.10
PRIVE	0.50	1.92	15.01	0.27	17.74
Lee et al.	0.46	2.14	16.20	0.32	19.16
ReverseCloak	0.46	2.44	16.72	0.29	19.96
Casper	0.42	6.31	18.01	0.26	25.05
L2P2	0.92	4.00	28.06	0.38	33.40
Xu et al.	0.42	6.39	18.27	0.29	25.40
feeling-based	0.53	7.26	22.45	0.29	30.60
Location Diversity	0.84	6.28	17.85	0.26	25.41
Kato et al.	0.42	6.01	17.21	0.26	23.94
Kido et al.	0.41	6.26	17.87	0.26	24.90
SpotME	0.41	6.25	17.82	0.26	24.78
SybilQuery	0.46	6.52	18.00	0.26	25.29
MobiPriv	0.41	1.80	14.35	0.27	16.87
Assam et al.	0.47	0.80	1.74	0.30	3.33
Hoh et al.	0.42	0.71	0.83	0.26	2.25
CAP	0.40	2.39	3.92	0.25	7.02
CaDSA	0.57	1.70	1.93	0.29	4.53
MobiCrowd	0.43	0.81	1.02	0.31	2.59
Ghinita et al.	1.69	3.15	7.03	0.33	12.23
Trust no one	1.74	3.18	7.72	0.41	13.08
MaPIR	1.64	3.11	7.03	0.32	12.13
SpaceTwist	1.70	2.99	6.71	0.34	11.77
Average	0.71	3.26	10.74	0.34	15.10

Figure 7.7 shows on a logarithmic scale the percentages of the execution time of

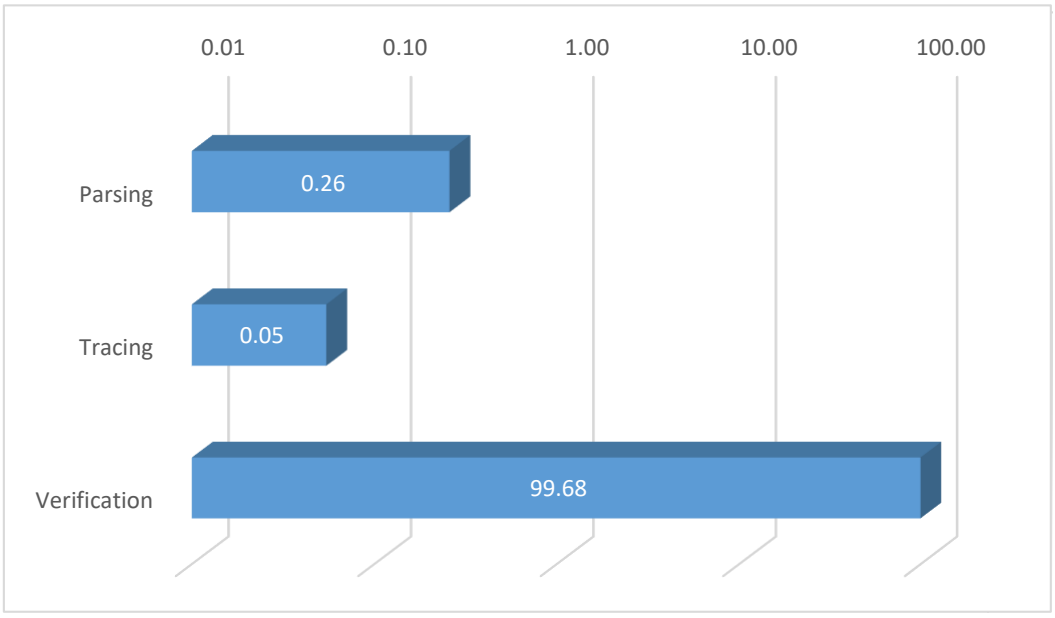


Figure 7.7.: Percentage of execution of an average LP3Verif call

LP3Verif that each of the three main parts (parsing the *.lp3* file, building the trace tree, and performing the verification) takes on average. It is obvious that the verification takes the vast majority of the execution time (99.68 %), whereas the time of the parsing (0.26 %) and the tracing (0.05 %) are neglectable. When taking a closer look at the verification process, it can be further sub-divided into three parts: the preparation (including the translation to normal form and extraction of unique query states), the checking of static properties, and the checking of temporal properties. Figure 7.8 shows the percentages of the execution time of the verification part on a logarithmic scale. The graph shows that the verification is dominated by the checking of static properties, which takes 99.988 % of the execution time of the verification process. The preprocessing (0.0008 %) and the checking of temporal properties (0.0037 %) have an insignificant impact on the overall execution time.

As mentioned above, we want to evaluate the three improvements that are intended for decreasing the execution time: normal form of formulas, partial-order reduction of state space, and caching of static verification results. We compare the LP3Verif implementation, as presented in Chapter 6, with versions without one of the three improvements. Figure 7.9 shows the execution of all 29 protocols from Table 7.4 in sequence. The impact of the improvements can be seen in the comparison of fully optimized version (LP3Verif), a version without transforming the four properties into normal form (no Normal Form), a version without reduction of the state space static checks (no P-O Reduction), and finally a version without caching of static verification results (no Cache). The graph shows that each of the three optimizations have a huge impact on the execution time. The numbers represent the average time it takes to verify all four properties of all 29

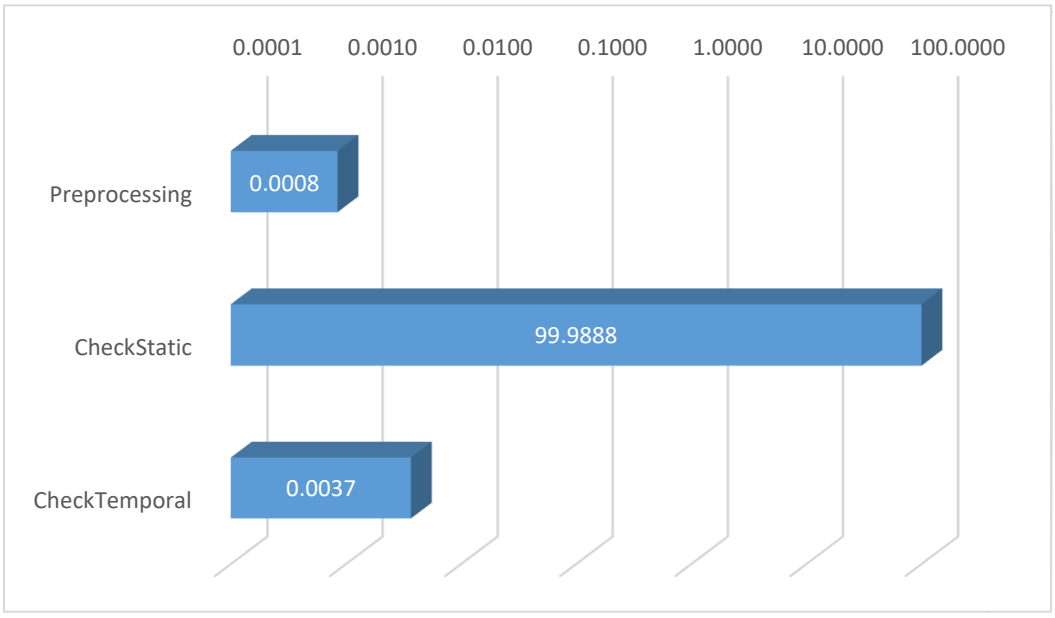


Figure 7.8.: Percentage of execution of an average LP3Verif verification process

protocols in sequence. This can be seen as our benchmark test. LP3Verif takes approximately 2 minutes, while the version without normalized formulas already takes nearly 5 minutes. The version without partial-order reduction takes almost 14 minutes, whereas the version without caching takes more than 10 minutes. The huge impact of the optimizations can be explained when looking back at Figure 7.8. The checking of static properties is the part of the verification that has to be improved to reduce execution time. Each of the three optimizations improve the checking of the static properties in their own way. The normal form reduces the number of static properties as it minimizes the number of temporal operations yielding bigger, and thus fewer, static formulas. The partial-order reduction, on the other hand, reduces the number of process states that have to be checked. Finally, caching obviously reduces the number of times the *checkStatic* method has to be called. Figure 7.9, furthermore, shows that the partial-order reduction has the highest impact on the execution time of the benchmark test, while the normal form has the lowest.

7.4. Threats to Validity

We identify two categories of possible limitations to the σ-calculus or threats to validity: First the assumptions we make, which define our models, and second the evaluation of the σ-calculus. In Assumption 2 we characterize the attack scenario/attacker model. We assume the attacker to be the LBS itself. Another possible attacker model would be to assume the attacker "on-the-wire" as an eavesdropper who intercepts messages. Having only the LBS as potential attacker could mean that the σ-calculus misses po-

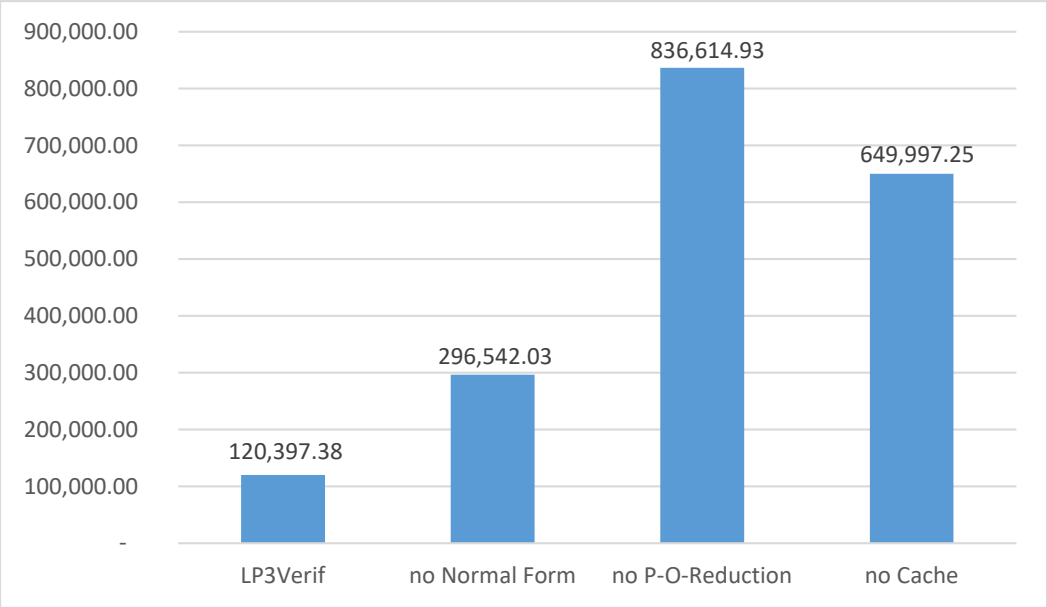


Figure 7.9.: Execution times of different optimizations (in ms)

tential attacks and consequently privacy violation, which would make our verification incomplete. However, we argue that in times of big data the users are more concerned with what the big tech companies do with their personal data, which they provide in exchange for a service, than with malicious third parties. Furthermore, we classify the LBS as a stronger attacker, who receives all queries with complete certainty as opposed to an eavesdropper who might intercept a few messages. Another assumption we make is the user-centric model, with one representative user. This approach is not standard and might contrast with system-based epistemic models, which are usually used to express knowledge in a system with multiple parties. A problem with this non-standard approach could be that the language and logic are not useful or not intuitive enough for easy modeling. However, for us it is intuitive to view location privacy from the perspective of one single user, who does not want to compromise their personal data. Combined with our worst-case assumption we ask the question "what is the worst that could happen to me?"

For our evaluation we performed a literature review to find a representative set of LPPMs to model and compare. In our final selection we excluded 9 papers due to their probabilistic protection mechanisms and protection goals. The σ-calculus is based on a worst-case assumption and offers binary decisions. This is an intended limitation which coincides with the scope of the σ-calculus as we currently want to make binary decisions only. However, one could claim that a potential probability-motivated extension of the σ-calculus would impact the semantics of the non-probabilistic processes. Fortunately, this is not the case as the σ-calculus uses small-step operational semantics, which can be extended with new reduction rules specifically dedicated to probabilistic processes. The

current syntax, similarly, is not impacted by the addition of new process components due to the compositional character of processes.

7.5. Summary

In summary, in this chapter we evaluated the σ-calculus and our implementation LP3-Verif. The evaluation of the σ-calculus is done in two parts, based on its modeling capability and verification capability. From our literature survey of LPPMs we identify the set of most-cited protocols (which do not purely rely on probabilistic guarantees) that serves as the basis for our evaluation. We show that this set of protocols can be modeled in the language of the σ-calculus and so can their respective privacy requirements. Furthermore, we check all the protocols with the same location privacy benchmark set of four properties to compare the protocols based on their verified guarantees. We also evaluate our tool LP3Verif for performance and scalability to show that LP3Verif returns results in a reasonable amount of time. Additionally, we evaluate the impact of caching, partial-order reduction, and our formula normal form on the performance of our implementation. Finally, we discuss threats to validity of the σ-calculus and our model checking approach in general, including points already made throughout the previous chapters.

8. Case Studies

In this chapter, we present a case study of a real-world location-privacy preserving mechanism and an extended example of a software developer designing a new protocol. In Section 8.1, we given an overview of location privacy-preserving mechanisms in practice where we introduce implementations of LPPMs which can be used with real LBS applications (as opposed to proof-of-concept implementations which only work with modified LBS applications). Then in Section 8.2, we present *Location Guard*, one of these LPPM applications, in more detail. The section introduces the underlying LPPM and shows how to translate the pseudo-code algorithm into the σ-calculus. Then in Subsection 8.2.3, we discuss the specific privacy threats to LBS applications in this context and derive appropriate location privacy requirements for the protocol. We then show that our tool LP3Verif can verify the claimed privacy guarantee. In Section 8.3, a second case study demonstrates how an LPPM design process can be guided by LP3Verif and the σ-calculus. The section also presents a verification simulation where a property is verified step-by-step.

8.1. LPPMs in Practice

In this section, we present an overview of implementations of LPPMs used in practice and briefly introduce *Location Guard*, before we focus on *Location Guard* in more detail and use it as basis for the case study in the following section. While in academia many LPPMs have been proposed in the past decade, fewer applications exist in practice. According to its number of downloads, *Location Guard* is currently the most popular LPPM application. Its underlying LPPM is described in a paper by Chatzikokolakis et al. [167]. *Location Guard* is a browser extension for Chrome, Firefox, Opera, and also for the Firefox browser on Android. It uses a Perturbation mechanism to add noise to the locations a user requests from the geolocation API.

Another browser extension called *TrackMeNot* chooses a completely different approach by hiding real location queries in a vast amount of dummy queries with fake locations. Technically, TrackMeNot is not a pure LPPM application to protect location privacy in LBS queries but generally protects the "user profile" by issuing search queries to random search engines. However, as it also serves as an LPPM by our definition we include this app in our list. The application was proposed by Nissenbaum and Daniel in a 2015 paper [168].

Browsers and mobile applications are the most common platforms to access LBS. Consequently, mobile applications are another option for LPPM implementations. However, to work with arbitrary LBS applications, such an LPPM would have to intercept system API calls and modify their results. Mobile operating system platforms such as iOS or Android manage these API calls via their own access control management (mostly) under user control. Users can either allow individual applications access to location data[1],

[1] An issue is that it is not always clear what location data is. For example, WiFi can be used to localize persons [169, 170]

or not. A few years back, Android had a granularity option where users could specify two levels of location data: "high accuracy" and normal. But location management is a binary decision to either allow or deny access. However, we know of two apps, which were at least for a brief time on the Android app store that pursued a policy approach to protect location privacy of Android users: *ipShield* [64] and *LP-Guardian* [65]. Both are now discontinued and no longer available as they only worked with specific Android versions. We choose *Location Guard* for the case study in Section 8.2 as it is of all applications the most suitable one being a "pure" LPPM application as opposed to TrackMeNot, which is an application for general user profile confusion, and it is the most commonly used LPPM browser extension.

As categorized in Section 2.1 and Section 2.2, location privacy-preserving mechanisms in this dissertation are online mechanisms for query-based LBS communication, i.e., the LPPMs modify user queries such that an LBS cannot infer personal data. As already mentioned in Section 2.1, we consider online LPPMs to be the most challenging and therefore interesting scenario because of the limited information compared to offline mechanisms that anonymize a complete data table. Another aspect, which makes implementations of online LPPMs even more challenging, is the already existing LBS infrastructure. Users want to use services, like Google Places, in a privacy-preserving way. As the LBS backends already exist and cannot be altered, mechanisms have to be adopted to work with real LBS queries. For instance, a query which requires a position parameter cannot be issued with a region. As possible solution, a generalization-based LPPM could use a random point in this region or the center to solve this problem.

As motivated above, implementations that do not need additional infrastructure or changes to the LBS queries are more likely to be used in practice. Applications on the user's device fall into this category as they do not require an additional server but directly change the location at the user's side. As browsers and smart phones are the primary environments where users access location-based services, we differentiate between two potential types of applications in this category: browser extensions and mobile applications. We searched the Chrome web store, the Firefox browser add-ons, the Google play store, and the Apple app store for applications that employ LPPMs to protect the user's location. We found many applications that protect the privacy of users at the cost of service, e.g., applications that replace real positions with completely fake ones. This type of protection does not satisfy our informal definition in Section 2.2, therefore, we excluded all applications that reduce the quality of service in such a drastic way. With this restriction we exclude VPN and location spoofing apps as well as policy and complete location blocking apps as they do not serve the purpose of sharing (honest) location data with an LBS.

Apart from proof-of-concept implementations, only very few real-world implementations are used in practice. Our application survey only resulted in four applications worth mentioning. Considering the large number of LPPM papers in the literature, this result is surprising. This discrepancy could be explained by a missing demand by users for privacy preservation in practice. The use of such applications can reduce the quality of service (QoS) by reducing the accuracy of potential query results or simply by demanding additional input (like privacy preference). The so-called *privacy paradox*

describes the phenomenon that users, while valuing their privacy, willingly share their personal data with services [171]. This very common paradox, described in a 2006 study on social media, can be observed not only on social media but also in the usage of many data-based services (LBS, IoT, e-commerce, etc.). More recent studies on the privacy paradox suggest, however, that the behavior of privacy-aware persons is moving towards consistent control over personal data [172]. This trend could mean an increasing interest in LPPM applications.

Another reason why LPPM applications, as we view them, are not widely employed is the dominance of LBS providers. The category of online LPPMs that protect against adversaries potentially at the LBS itself requires an obvious separation between LPPM infrastructure and the LBS infrastructure. Privacy protection at the LBS itself does not fall in our category of LPPM scenarios as these protection mechanisms can only protect against offline privacy attacks, i.e., privacy leaks via published (or otherwise leaked) data tables. Yet, the most common location privacy preservation performed in practice is done by the LBS themselves. In May 2018 the EU GDPR became enforceable in all countries of the European Union. The implementation as national laws brought a new spike of privacy-awareness in the EU and lead not only to an increase of popups asking for cookie consent but generally to more refined privacy policies. LBS providers have set up privacy policies about processing of personal data. Google, for instance, states that personal data (by the GDPR, location data falls into this category) is protected using two techniques: generalization and perturbation[2]. While the privacy policy does not go into details how the algorithms work, they promise to achieve k-anonymity and l-diversity (via generalization) and differential privacy (via perturbation). The parameters of these guarantees are not public either. These are reasonable privacy mechanisms considering the amount of data that Google processes and the sensitivity of these data. Other LBS (like Apple Maps, Pokémon Go, or Yelp) providers give even more vague privacy guarantees for their users, such as "aggregation" (Yelp[3]), "technical safeguards" (Apple[4]), or "appropriate security measures" (Niantic[5]). These forms of location privacy protection fall in the category of offline mechanism as the anonymization is performed on complete data sets or subsets (views) and only protects users from third parties violating their data privacy. As this dissertation proposes an approach for online LPPM scenarios, we focused on the very few implementations of LPPMs in practice that perform obfuscation in between the user and the LBS.

8.2. Location Guard

In this section we first describe the browser extension *Location Guard* in more detail, then introduce the underlying mechanism in Subsection 8.2.1, and derive the corresponding LP3 model in Subsection 8.2.2.

[2]"How Google anonymizes data – Privacy & Terms – Google," `https://policies.google.com/technologies/anonymization?hl=en-US`

[3]"Yelp Privacy Policy," `https://terms.yelp.com/privacy/en_us/20200101_en_-us/#third-parties`

[4]"Legal - Apple Privacy Policy - Apple," `https://www.apple.com/legal/privacy/en-ww/`

[5]"Niantic Privacy Policy," `https://nianticlabs.com/privacy/en/`

Location Guard is a web browser extension that lets users select between three privacy levels: *high*, *medium*, and *low*. Additionally, the user has the option to use the real location or use a static fake location. Furthermore, the user can set fine-grained privacy levels by selecting a radius in meters, where the level *low* corresponds to $200\,m$, *medium* to $500\,m$, and *high* to $2000\,m$. The impact of this radius is that whenever a website, like Google Maps, accesses the location API of the browser, *Location Guard* intercepts the real location and perturbs it with random noise such that the resulting location is approximately within radius distance of the real location. A cache duration can be set, which determines how long the fake perturbed location is cached for a subsequent LBS query and at what point a new random noisy location is generated. For the sake of this case study we focus only on the Perturbation aspect of *Location Guard* and hence omit the possibility to use static fake locations (or the real one).

8.2.1. LPPM

In 2013 Andrés et al. proposed the concept of *geo-indistinguishability* (based on differential privacy as introduced in 2006 by Cynthia Dwork [173]) defining the level of privacy within a radius [12]. Geo-indistinguishability guarantees that if two locations are within a radius r, the distance between the two corresponding distributions (from the perturbation mechanism) is at most l, where l represents a privacy level with a smaller l denoting higher privacy. Formally, a mechanism K satisfies ϵ-geo-indistinguishability iff for all locations x, x': $d_p(\mathrm{K}(x), \mathrm{K}(x')) \leq \epsilon d(x, x')$, where d denotes the Euclidean metric, d_p the distance of two probability distributions, and $l := \epsilon d(x, x')$ is the privacy level.

The authors propose the Planar Laplace mechanism PL_ϵ that uses a Laplace probability density function to draw a random radius. The algorithm, as depicted in Algorithm 9, draws for the maximum privacy level uniformly a random angle and radius to calculate the sanitized location, which is then set to the closest location in the set of acceptable locations $\mathcal{A}$. In the last step, discrete coordinate locations (instead of real-valued numbers) are calculated and $\mathcal{A}$ can be seen as a fine-grained network of coordinates determining the precision of the returned coordinates. The inverse cumulative distribution function $C_{\epsilon'}^{-1}(p)$ calculates the radius r where a random point falls into with probability p. The inverse cumulative distribution function uses the Lambert function W to calculate the radius but we do not go into more detail as the stochastic details of the mechanism are not relevant for our model.

8.2.2. LP3

In this subsection we abstract and formalize *Location Guard* and its underlying LPPM as described above. First, from the description of the mechanism and the algorithm itself, one can deduce that the mechanism falls into the category of Perturbation mechanisms. The mechanism does not require other users and, hence, can be applied on the user's side. On the abstraction level of the σ-calculus, the mechanism can be described as adding random noise to the real location. The obfuscation of *Location Guard* can be described by a computation $R = noise(loc)$. The queries of *Location Guard* are unaltered

Algorithm 9 Planar Laplace mechanism LP_ϵ (shortened) from [12]

Require: x point to sanitize, ϵ privacy parameter, $\mathcal{A}$ acceptable locations
Ensure: z sanitized version of input x

1: **procedure** $LP_\epsilon(x, \epsilon, \mathcal{A})$
2: $\epsilon' \leftarrow \max \epsilon'$ satisfying precision for $r_{max} = diam(\mathcal{A})$
3: draw θ uniform. in $[0, 2\pi)$ ▷ draw angle
4: draw p uniform. in $[0, 1)$, set $r \leftarrow C_{\epsilon'}^{-1}(p)$ ▷ draw radius
5: $z \leftarrow x + \langle r\cos(\theta), r\sin(\theta)\rangle$ ▷ to cartesian, add vectors
6: $z \leftarrow closest(z, \mathcal{A})$ ▷ truncation
7: **end procedure**

save for the location as the extension only intercepts the location API call and does not modify any communication with an LBS. Therefore, queries to an LBS can be described by $Query(pid, R, serv, t)$ where only the location is an obfuscated term and all other parameters are the user's data.

```
process LocationGuard
2    !
         Compute(R=noise(loc))
4        Query(pid,R,serv,t)
     end
6 end
```

Figure 8.1.: *Location Guard* as LP3

The LP3 model of *Location Guard* can be seen in Figure 8.1. The LP3 shows that *Location Guard* is in its core a very basic perturbation approach that really just protects the location of users. This model of *Location Guard* assumes that the cache duration is set to 0. If one wanted to model a non-zero cache duration, lines 2 and 3 could be swapped to express the situation where a fixed noisy location is reported. However, this change does not have an impact on privacy in the σ-calculus because if region R leaked the location of the user (in the worst case), it would always leak it in other (random) computations as the σ-calculus considers the worst-case scenario. As the region caching of *Location Guard* is a feature for QoS rather than privacy, we omit in our model this aspect of the protocol.

8.2.3. Privacy Properties

Location Guard is a browser extension that only claims to protect the actual location of its users. Of our set of ground requirements **F1-F4**, the location requirement **F2** matches this claim. Even though *Location Guard* does not claim to protect the identity of its users, we also verify the identity requirement **F1**. We expect the protocol to satisfy formula $F_2 = \Box\neg(K_{Loc} \wedge K_T)$ and to violate the property $F_1 = \Box\neg K_{Id}$.

Figure 8.2 confirms that the protocol does not satisfy the identity property F_1 but satisfies the location property F_2. The result of the identity property's verification is not surprising, as the screenshot depicts a witness that demonstrates an example of a violating execution of the protocol. In Subsection 6.2.4, we described how to interpret the output of LP3Verif in general and particularly how to read witnesses. In this case, the witness shows an execution where the representative user has the id 8 and issues the query "Query1" with the group parameter being the singleton set $\{8\}$. The rest of the query and the second query are not relevant for this property. The trace shows the process reduction leading to this particular query. As the LocationGuard LP3 does not branch, the trace represents the only possible execution of the protocol. First, the replication is resolved via the reduction rule REPL (cf. Table 5.5) and then the rule PARA requires the reduction rules COMP and QURY to reduce the first part of the sequential replication to 0. These reductions lead to the process Compute$(R = noise(loc))$.Query$(pid, R, serv, t)$.0 with a query state $\sigma_{Qry} = (\{pid, noise(loc), serv, t\})$. This state already violates F_1, hence, not further reduction is necessary to demonstrate the violation.

```
The protocol LocationGuard: !Compute(R = noise(loc)).Query((pid, R, serv, t)).0|([], [], [])

The property F1: G ¬ K_ID
The property F2: G ¬ K_LOC ^ K_T
The protocol LocationGuard does not satisfy the property G ¬ K_ID
Witness:
  Representative User: (pid 8) (loc 1) (serv 1) (t 0)
  Other User Location: (loc_1 2)
  Query1: (G1 (singleton 8)) (R1 (union (union (union (singleton 1) (singleton 2)) (singleto
n 4)) (singleton 5))) (S1 (singleton 1)) (F1 (singleton 0))
  Query2: (G2 (union (union (union (union (union (union (union (union (singleton 1) (singlet
on 2)) (singleton 3)) (singleton 4)) (singleton 5)) (singleton 6)) (singleton 7)) (singleton
 8)) (singleton 9))) (R2 (union (union (union (union (union (union (union (union (singleton
1) (singleton 2)) (singleton 3)) (singleton 4)) (singleton 5)) (singleton 6)) (singleton 7))
 (singleton 8)) (singleton 9))) (S2 (union (union (singleton 1) (singleton 2)) (singleton 3)
)) (F2 (union (union (singleton 1) (singleton 2)) (singleton 3)))
  Trace: !Compute(R = noise(loc)).Query((pid, R, serv, t)).0|([], [], []) --> Compute(R = no
ise(loc)).Query((pid, R, serv, t)).0|([], [], []).Compute(R = noise(loc)).Query((pid, R, ser
v, t)).0|([], [], []) --> Compute(R = noise(loc)).Query((pid, R, serv, t)).0|([(pid, noise(l
oc), serv, t)], [], [])

Grid:
+-----+
|1 2 3|
|4 5 6|
|7 8 9|
+-----+
The protocol LocationGuard satisfies the property G ¬ K_LOC ^ K_T
```

Figure 8.2.: Screenshot of LP3Verif showing two verification results

8.3. A Software Developer

Parts of this section and especially the following example protocol were previously submitted in a journal article to the Journal of Logical and Algebraic Methods in Programming which is currently in review. This section presents an extended example featuring software developer Alice. Suppose Alice wants to use the Google Places API for nearest restaurant queries. She values her privacy and does not want to reveal her location data to the LBS. As she is a software architect, she wants to design a privacy-preserving system to make these kind of queries. She starts with a very simple approach and sketches an algorithm in pseudo-code.

1: User sends query to Anonymization Server (AS)
2: AS gathers locations from $k-1$ other users
3: AS computes the region of all locations and its center c
4: AS sends a pseudonymous query with location c

The algorithm is based on the principle of k-anonymity and uses a trusted intermediate server that queries for the users. Because she wants to check whether her idea does in fact preserve user privacy, Alice can use the proposed σ-calculus to formalize her algorithm and verify her privacy requirement. In the process of modeling the pseudo-code algorithm in the protocol language she realizes that she did not think of all the details regarding her queries. She decides to model her protocol as follows: P_{simp} = if k_users then Compute($R = MBB(locs)$).Compute($h = hash(pid)$).Query($h, R, serv, t$). As P_{simp} shows, she decided to protect neither her service request nor her time stamp, but she uses the k_users flag to model the gathering of other users, she generalizes the location of the user to the minimum bounding box of all gathered user locations, and uses a single hash as her pseudonym to hide her identity. Now Alice wants to formalize her privacy requirement to check her protocol for potential privacy leaks. She chooses the ground requirement **F2** (location requirement) to be the property matching her protection goal. Subsequently, she wants to verify the property $F_2 = \Box\neg(K_{Loc} \wedge K_T)$. Alice uses LP3Verif to check the privacy requirement and consequently translates the protocol and property to LP3 syntax. Figure 8.3 shows Alice's first protocol sketch and the corresponding privacy property.

Alice now wants to check whether her modeled protocol does meet her privacy requirement. Therefore, she formally checks whether the protocol $A_{simp} = P_{simp}|\sigma^{init}$ satisfies the property F_2. The tool LP3Verif tells her that $A_{simp} \not\models F_2$. Furthermore, a witness tells her in what scenario her current location is in fact revealed. Such a witness is depicted as a screenshot of LP3Verif in Figure 8.4. The representative user and the nearest users are all in the same location (9) and the query region R1, consequently, is the singleton set $\{9\}$. This can happen in crowded places like a train station or a mall. In this case the minimal bounding box is just the region with the user's location itself, i.e., $MBB(\{loc\} \cup \{loc_1, loc_2\}) \simeq_S loc$.

Alice knows that her protocol falls into the category of Generalization approaches. She takes a look at Table 7.1 and finds with Location Diversity a protocol from this category which satisfies **F2**. She looks up its LP3 formalization and identifies the

```
process Alice_simp
    !
        if k_users
            Compute(R=MBB(locs))
            Compute(h=hash(pid))
            Query(h,R,serv,t)
        end
    end
end

property p
    G not (K_loc and K_t)
```

Figure 8.3.: Alice's simple protocol in `.lp3` format

lack of location diversity as the problem. Thus she alters her algorithm such that the AS gathers users with at least 2 different locations. She models this protocol as P_{alt} = !(if k_users then if $l\_diverse$ then Compute($R = MBB(locs)$).Compute($h = hash(pid)$).Query($h, c, serv, t$)). She models the protocol in LP3 syntax (depicted in Figure 8.5) and uses LP3Verif to verify the privacy property **F2**, again. She checks whether her new, extended process $A_{alt} = P_{alt}|\sigma^{init}$ satisfies the property F_2 and the result is positive: $A_{alt} \models F_2$. She can now proceed to implement her algorithm feeling more secure about location privacy.

Verification

We demonstrate the verification process of LP3Verif by giving a step-by-step simulation of the model checking of the σ-calculus. We demonstrate the verification of the property $F_2 = \Box\neg(K_{Loc} \wedge K_T)$ of the first protocol P_{simp} (Equation (8.1)) by focusing on the important results of sub-algorithms or interesting steps.

$$P_{simp} = \text{ !(if } k\_users \text{ then Compute}(R = MBB(locs)).\text{Compute}(h = hash(pid)).\text{Query}(h, R, serv, t) \text{)} \quad (8.1)$$

The verification is model-checked via the algorithm call $MC(P_{simp}, F_2)$. The method *buildTraces* returns a set of four traces $T = \{tr_1, tr_2, tr_3, tr_4\}$ where tr_1 is the resulting trace when applying the reduction rules REPL, PARA (THEN, COMPn, COMPn, QURY), THEN, COMPn, COMPn, QURY in that order. The trace tr_1 has a size of 7 and its final extended process is P_{fin} (Equation (8.2)).

$$P_{fin} = 0|\sigma^{simp}, \text{ where } \sigma^{simp} = (2, \{((hash(pid), MBB(locs), serv, t), \{k\_users\})\}, \emptyset, empty, \emptyset) \quad (8.2)$$

```
The protocol Alice_simp does not satisfy the property G ¬ K_LOC ^ K_T
Witness:
  Representative User: (pid 1) (loc 9) (serv 9) (t 0)
  Other User Location: (loc_1 9)
  Query1: (G1 (union (union (union (union (union (union (union (union (singleton 1) (s
ingleton 2)) (singleton 3)) (singleton 4)) (singleton 5)) (singleton 6)) (singleton 7)
) (singleton 8)) (singleton 9))) (R1 (singleton 9)) (S1 (singleton 9)) (F1 (singleton
0))
  Query2: (G2 (union (union (union (union (union (union (union (union (singleton 1) (s
ingleton 2)) (singleton 3)) (singleton 4)) (singleton 5)) (singleton 6)) (singleton 7)
) (singleton 8)) (singleton 9))) (R2 (union (union (union (union (union (union (union
(union (singleton 1) (singleton 2)) (singleton 3)) (singleton 4)) (singleton 5)) (sing
leton 6)) (singleton 7)) (singleton 8)) (singleton 9))) (S2 (union (union (singleton 1
) (singleton 2)) (singleton 3))) (F2 (union (union (singleton 1) (singleton 2)) (singl
eton 3)))
  Trace: !if K_USERS then Compute(R = MBB(locs)).Compute(h = hash(pid)).Query((h, R, s
erv, t)).0 else 0|([], [], []) --> if K_USERS then Compute(R = MBB(locs)).Compute(h =
hash(pid)).Query((h, R, serv, t)).0 else 0|([], [], []).if K_USERS then Compute(R = MB
B(locs)).Compute(h = hash(pid)).Query((h, R, serv, t)).0 else 0|([], [], []) --> if K_
USERS then Compute(R = MBB(locs)).Compute(h = hash(pid)).Query((h, R, serv, t)).0 else
 0|([(hash(pid), MBB(locs), serv, t)], [], [])
```

Figure 8.4.: Screenshot of a console witness

The state σ^{simp} indicates that two queries have been issued (2 in index state) and they are the same query (only one query in query set of query state). In fact, all states starting at the third (where the first query was sent) extended process ($tr_1[2]$) have the same query state in this trace, as trace tr_1 represents the execution of the protocol where the if-branch is taken twice and, hence, the same query is issued, adding no new query to the query state. Table 8.1 shows the complete trace tr_1 and Table 8.2 shows the extra reduction steps of the rule PARA. These rules only reduce the first half of the parallel processes and are internal reductions, which are not part of the actual trace and are only shown to demonstrate the internal reductions of LP3Verif.

Next, F_2 is normalized to $\Box(\neg K_{Loc} \vee \neg K_T)$. The method *getStatic* returns a set of all static formulas contained in the normalized form of F_2, which is $F = \{\neg K_{Loc} \vee \neg K_T\}$. The method *getStates* is the last preparation step where a set of all states with different query states is returned. The set $S = \{\sigma^{init}, \sigma^{simp}\}$ is a set containing two states (the initial and the final one after sending the second query). As trace tr_1 represents the execution where the same query is issued twice, only two different query states exist: the empty one and the one with the region query.

The verification algorithms calls the method *checkStatic* for all combinations of the sets F and S. The important call we want to highlight here, is for the combination of the state σ^{simp} and the static formula $\neg K_{Loc} \vee \neg K_T$. This call checks whether the states after the first query violate the location property, i.e., the location of the user and the time of the query are leaked. The call $checkStatic(\sigma^{simp}, \neg K_{Loc} \vee \neg K_T)$ returns *false* because the region of the query R_1 can be a singleton region in the worst case ($R_1 = MBB(locs) \simeq_S \{loc\}$). The reason why R_1 can be a singleton region is that it is the minimum bounding box of the grouped users' locations. If all users are in the same

```
process Alice_alt
    !
        if k_users
            if l_diverse
                Compute(R=MBB(locs))
                Compute(h=hash(pid))
                Query(h,R,serv,t)
            end
        end
    end
end

property p
    G not (K_loc and K_t)
```

Figure 8.5.: Alice's alternative protocol in `.lp3` format

location, the minimum bounding box is this single location. This worst case situation could not happen if the set of locations were further restricted, e.g., via location diversity. The boolean matrix B has, consequently, stored *false* for σ^{simp} and $\neg K_{Loc} \vee \neg K_T$ as the state σ^{simp} allows the leaking of location data. The disjunction also violates the $\neg K_T$ part but we omit this aspect as the disjunction is already *false* from the first part.

The algorithm *checkTemporal* returns *false* for trace t_1 at index $i = 3$, which is the first extended process with the same query state as σ^{simp}. The state σ^{simp} is the final state whose query state was used in the evaluation of *checkStatic*. The algorithm *checkTemporal* returns false based on B, which has stored the information that σ^{simp} does not satisfy $\neg K_{Loc} \vee \neg K_T$. As index $i = 3$ satisfies neither part of the disjunction, the whole property is not satisfied. As this one index suffices to show that the global modality $\Box$ does not hold, we can deduce that P_{simp} does not satisfy F_2.

8.4. Summary

In summary, in this chapter, we briefly introduced LPPMs in practice and discussed why currently LPPM applications compatible with arbitrary LBS are still rare; a fact we hope to change with this dissertation. We chose to take a closer look at the web browser extension *Location Guard* for our case study and described its features and underlying mechanism. As it uses a Perturbation mechanism solely based on adding random noise to locations, the browser extension works as a stand-alone application without the need for a trusted third party. We then modeled the corresponding protocol in LP3 syntax and chose the two location privacy requirements **F1** and **F2** the protocol should fulfill. Next, we demonstrated the modeling and verification process in the σ-calculus with our tool LP3Verif and showed that the protocol, as expected, satisfies the location requirement

Table 8.1.: Reduction of trace tr_1

$$
\begin{aligned}
&!(\ \text{if } k\_users \text{ then Compute}(R = MBB(locs)).\text{Compute}(h = hash(pid)).\\
&\quad \text{Query}(h, R, serv, t)\ )|\sigma^{init} = (0, \emptyset, \emptyset, empty, \emptyset) && \xrightarrow{REPL}\\
&\text{if } k\_users \text{ then Compute}(R = MBB(locs)).\text{Compute}(h = hash(pid)).\\
&\quad \text{Query}(h, R, serv, t)|\sigma^{init}.\text{if } k\_users \text{ then Compute}(R = MBB(locs)).\\
&\quad \text{Compute}(h = hash(pid)).\text{Query}(h, R, serv, t)|\sigma^{init} && \xrightarrow{PARA}\\
&\text{if } k\_users \text{ then Compute}(R = MBB(locs)).\text{Compute}(h = hash(pid)).\\
&\quad \text{Query}(h, R, serv, t)|(1, \{((hash(pid), MBB(locs), serv, t), \{k\_users\})\},\\
&\quad \emptyset, empty, \emptyset) && \xrightarrow{THEN}\\
&\text{Compute}(R = MBB(locs)).\text{Compute}(h = hash(pid)).\text{Query}(h, R, serv, t)|\\
&\quad (1, \{((hash(pid), MBB(locs), serv, t), \{k\_users\})\}, \emptyset,\\
&\quad (true, false, false, false), \emptyset) && \xrightarrow{COMPn}\\
&\text{Compute}(h = hash(pid)).\text{Query}(h, R, serv, t)|(1,\\
&\quad \{((hash(pid), MBB(locs), serv, t), \{k\_users\})\}, \{R = MBB(locs)\},\\
&\quad (true, false, false, false), \emptyset) && \xrightarrow{COMPn}\\
&\text{Query}(h, R, serv, t)|(1, \{((hash(pid), MBB(locs), serv, t), \{k\_users\})\},\\
&\quad \{R = MBB(locs), h = hash(pid)\}, (true, false, false, false), \emptyset) && \xrightarrow{QURY}\\
&0|(2, \{((hash(pid), MBB(locs), serv, t), \{k\_users\})\}, \emptyset, empty, \emptyset)
\end{aligned}
$$

but does not protect the identity of the user. Finally, we gave another demonstration of how to use LP3Verif based on an example of a software developer designing a new LPPM mechanism. The new mechanism belongs to the category of Generalization approaches and after a revision inspired by our look-up table in Table 7.1 satisfies the location requirement **F2**.

Table 8.2.: Internal reduction of the rule PARA in Table 8.1

$$\text{if } k\_users \text{ then Compute}(R = MBB(locs)).\text{Compute}(h = hash(pid)).$$
$$\text{Query}(h, R, serv, t)|\sigma^{init} \xrightarrow{THEN}$$
$$\text{Compute}(R = MBB(locs)).\text{Compute}(h = hash(pid)).\text{Query}(h, R, serv, t)|$$
$$(0, \emptyset, \emptyset, (true, false, false, false), \emptyset) \xrightarrow{COMPn}$$
$$\text{Compute}(h = hash(pid)).\text{Query}(h, R, serv, t)|(0, \emptyset, \{R = MBB(locs)\},$$
$$(true, false, false, false), \emptyset) \xrightarrow{COMPn}$$
$$\text{Query}(h, R, serv, t)|(0, \emptyset, \{R = MBB(locs), h = hash(pid)\},$$
$$(true, false, false, false), \emptyset) \xrightarrow{QURY}$$
$$0|(1, \{((hash(pid), MBB(locs), serv, t), \{k\_users\})\}, \emptyset, empty, \emptyset)$$

9. Summary and Future Work

This chapter summarizes the contributions and findings of this dissertation. In Section 9.1, we repeat the problems motivated in Chapter 1, summarize our proposed solutions, and also briefly recap how our approach differs from previous related work. In Section 9.2, we give an outlook on related research in this field that could build on this dissertation. The section focuses on possible extensions of our σ-calculus and our tool LP3Verif but also suggests research topics worth exploring in the adjacency of privacy verification of LPPMs.

9.1. Summary

In this dissertation, we presented a formal framework for evaluating and comparing location privacy-preserving mechanisms based on their privacy guarantees. We performed literature surveys to create taxonomies of LBS, LPPMs, and their privacy protection goals. From these we derived the core protection concepts of mechanisms and language features needed for modeling LPPMs, LBS systems, and privacy requirements.

The derived six categories of LPPMs are: Generalization, Perturbation, Mix Zones, Dummies, Cryptography, and Caches. Important language features for modeling location privacy-preserving protocols are the obfuscation mechanisms themselves, which serve the purpose of modifying the data itself or the links between them. We aim at modeling existing mechanisms, and creating new ones is not in the scope of this dissertation. Another important language feature is modeling queries to the LBS, which are the main communication in LBS systems where the users issue queries containing (among others) location data. We found that *sets* – the mathematical collection of elements – are an efficient base for modeling the different protection approaches and the query communication containing personal data. We found that simple set operations suffice to model LPPMs of all six categories and also that a logic based on reduction of sets to singletons suffices to reason about location privacy in these models.

With a concept for modeling data obfuscation for LPPMs, next, we identified process calculi to be fitting formal methods to model location-based service systems including query-based communication needed for LP3 models. The family of pi-calculi is suited for modeling data exchange between distributed parties. As LP3 models involve no dynamic concurrency in their communication, we decided for a simplified pi-calculus (similar to the applied pi calculus used for security protocols) as the initial inspiration of the calculus. Our calculus needs a uni-directed channel (to model queries directed at LBS) with a fixed quadruple of expected terms (personal data as query parameters). Furthermore, the calculus needs a term algebra to evaluate set-based operations to model knowledge gain of an adversary. The resulting σ-calculus differs from other process calculi, like the applied pi calculus, in its reduced, application-focused syntax but also in its ability to model mechanisms using complex regions as well as identity swaps with a light-weight term algebra. The σ-calculus allows the modeling of LP3s and also comprises a property language to express location privacy requirements of LP3s as well

as a verification logic to show that a protocol violates given properties.

We proposed a model checking approach that enables the automatic verification of privacy properties of LP3 models. The model checking algorithms follow an explicit state space exploration approach with a partial-order reduction in the checking of static properties where we traverse all states (a small number) for cheap temporal look-up checks and perform the more expensive static checks only on the subset of states which are epistemically different. We propose the tool LP3Verif, which implements the described algorithms and uses caching as well as a translation to a self-defined normal form to improve the performance of the verification as formulas in the normal form minimize the number of necessary static checks. LP3Verif is written in Java and is a command line-based application that can be called with an .lp3 file to return the verification results: either true or a witness that demonstrates how the respective privacy property can be violated in the protocol. LP3Verif uses CVC4 as back-end SMT solver to perform singleton checks using integer and fine-set theory. Furthermore, CVC4 supports the generation of witnesses by providing value assignments.

We evaluated the σ-calculus and LP3Verif on the basis of the most cited LPPM protocols from the literature. We modeled 29 LPPM protocols and 4 properties and verified the properties of all protocols. The resulting table demonstrates that LP3Verif can be used to model, verify, and compare location privacy of LPPMs. We, furthermore, showed how different types of mechanism can be expressed in σ-calculus syntax. Finally, we showed the performance of LP3Verif by presenting graphs with timing behavior of the tool and also comparisons of our implementation with version without the three major optimizations: caching, partial-order reduction, and normal form. The evaluation showed that LP3Verif verifies privacy properties of real-world protocols in a reasonable amount of time ($< 4\,s$ on average). Also while the performance of LP3Verif could be sped up with more efficient algorithms, e.g., to enable multi-threading, it heavily ($> 99\%$) relies on the underlying SMT solver. Hence, the performance of LP3Verif benefits from research that improves SMT solvers. The scalability analysis showed that LP3Verif scales not too well with increasing protocol lengths (exponential on protocol size). However, in practice LP3 protocols are not large enough for the scaling to have a huge impact.

Finally, we presented two case studies, a real-world example application and a hypothetical example of a software developer designing a new LPPM. *Location Guard* is an internet browser extension that perturbs locations that websites request from the browser by adding random noise. We modeled *Location Guard* as an LP3 and verified the claimed location privacy property. An example of software developer Alice shows how LP3Verif can be used in the design process of a new LPPM and also demonstrates the verification process of the σ-calculus.

In summary, we observed a lack of formal tools to support software developers by making LPPMs comparable and their privacy claims provable. Our proposed solution, LP3Verif and the σ-calculus, is a starting point to close a gap between a large number of LPPMs in academia and the very few implementations in practice.

9.2. Future Work

Building on top of this dissertation, we see three possible directions for future research. The first direction is a probabilistic extension of the σ-calculus which enables the verification of additional privacy properties, the second direction is a qualitative extension of the σ-calculus that allows making more precise statements to evaluate the severity of privacy leakage, and the third direction is an alternative process calculus, which one could develop when assuming different attacker models and, hence, laying a different modeling focus.

Probabilistic Extension

With LP3Verif we have presented a tool for modeling LPPMs and evaluating them on the basis of boolean guarantees, where a protocol either satisfies a location privacy property or it does not. We decided on non-probabilistic models and guarantees because we aimed for simple yet expressive properties that evaluate the general approach of the protection mechanism on a design level. Other approaches to verifying location privacy exist. Of these, quantifying privacy using statistics is a popular choice. Most quantifications of location privacy can only be calculated for specific scenarios, i.e., probability distributions and function for users, maps, and the attacker's background knowledge. These models can quantify location privacy for such a specific attack scenarios only. Even though this type of LPPM evaluation is not the goal of the σ-calculus, one could extend it to achieve similar probabilistic properties. Hence, a potential direction for future work is a probabilistic extension of the σ-calculus that could take inspiration from the work of Ding et al., who proposed a probabilistic process calculus (cf. Section 4.1). However, their approach is too simple to model attack scenarios because their δ-calculus can only express how a priori known probabilities behave in the long run, comparable to temporal model checking on Markov chains. In order to add property expressiveness to the σ-calculus, a quantified extension would serve the purpose of evaluating the robustness of different LPPM approaches against specific attacks. This extension would allow an additional level of privacy verification, where on top of the current properties, one could parametrize properties to ask questions like "how robust is the mechanism against a background knowledge attack?" or "how likely is an attacker to correctly guess my location?" and similar questions that help with comparing LPPMs that, for instance, both satisfy the location privacy property but might still give different levels of guarantees: "is noise more effective than dummies?" To formalize a verification framework for such types of questions, the σ-calculus would have to be modified such that the process language allows probabilistic processes and terms and the general worst-case assumption would not be valid for these properties. Instead, one could ask for the worst case (but also an average or best case probability) without an initial assumption.

Qualitative Statements

As mentioned before, LP3Verif gives boolean guarantees of either privacy compliance or privacy violation. A way to increase the expressiveness of these guarantees would be a quantitative extension of the violation cases. With such an extension one could differentiate between scenarios where no protection happens at all and those where only a rare event could trigger privacy leakage. On first glance, this seems like an easy add-on to LP3Verif as described in Chapter 6. However, such an extension is not that trivial as it requires significant changes of the underlying semantics of the static properties. The first naïve way to make a qualitative distinction of property violations would be to use the information of the generated witnesses to extract the likelihood of the scenario, e.g., in terms of low, medium, high. Witnesses include a trace that leads to the violating state. However, these traces only demonstrate an execution of the protocol, yet do not make the likelihood of the execution obvious. For instance, the location of a user can leak even in region queries if the collected group of users are in the same location. The relative rareness of this scenario happening cannot be extracted from the trace of a witness, which would only show that a region query was issued but could not distinguish the scenario from one where a region by fault of design leaks information, e.g., via intersection of moving regions. A second naïve approach to adding qualitative statements to LP3Verif would be to use the information at the level of static property verification. The verification of static formulas, indeed, includes more information regarding the different scenarios that lead to a privacy violation. However, the verification of static properties in the σ-calculus uses sets as core concept to model protection mechanisms. To check a privacy violation, the σ-calculus checks whether a certain set can be related to a singleton set. LP3Verif utilizes the SMT solver CVC4 to perform the necessary set operations. This means that neither the current logic nor the implementation allows a distinction between a singleton set or a set that can be reduced to a singleton set via observational equivalence. Thus, qualitative statements cannot be derived from the current version of LP3Verif. However, as the distinction between unlikely and more common violation scenarios is an interesting feature, we view a qualitative extension of the σ-calculus as a promising future research topic. The challenging part would be to find a way to incorporate such a distinction into the σ-calculus without losing the simplicity of the set-based approach and on the other hand giving a real benefit to the expressiveness of the tool. A good qualitative weighting of the likelihood of events is also not trivial once one has established a way to differentiate between the scenarios. A discretization of quantitative events, e.g., bounds like “more than 3 assumptions necessary” or “less than 2 deduction steps”, would be a solution which only requires the selection of meaningful threshold values. Alternative approaches could utilize empirical information to derive meaningful likelihood categories similar to risk analysis approaches.

Alternative Calculus

Another assumption of the σ-calculus is the attacker model itself. We consider the adversary to be the LBS or an actor with access to the queries issued to the LBS. Others have made different choices, like Ding et al. who considered a weaker attacker with limited communication range. However, all the related research has focused on scenarios where the adversary is either the LBS or an observer with LBS-like knowledge. Malicious users are a concept very common in the field of security but not established in location privacy. For future research, we see an interesting area in "attacks on collaborative privacy" where one evaluates how much damage a malicious user can inflict, especially when protection mechanisms rely on group anonymity or peer-to-peer communication. A formal modeling language for such protocols would need a focus on distributed communication (as opposed to the single-channel queries of our σ-calculus). Such a process calculus would model users in a different way compared to the σ-calculus as one representative user would not suffice to model peer-to-peer message passing. While the σ-calculus lays focus on data perturbation, a process calculus for modeling malicious users in LBS systems would focus on the message exchanges of users (with each other and the LBS). Such behavior could be modeled with a syntax similar to the applied pi calculus. However, this new calculus should answer questions like "how many malicious users does it take to cancel privacy in groups?" or similar quantifiable properties that evaluate the robustness of mechanism regarding attack from users. From a theoretic point of view this direction of research would require major modifications to the logic of the σ-calculus as the basic assumption of the representative user and actor-specific properties would no longer be valid. Apart from the alterations to the process language, the epistemic logic would have to be changed at least to a first-order logic to reason about the knowledge of multiple potential attackers and probably also a full modal epistemic logic with group and distributed knowledge would be necessary to express how malicious users can use group knowledge (such as a shared privacy parameter or type of service) to increase their knowledge. In summary, a framework for reasoning about the impact of malicious users in location privacy would require a new combination of process calculus and modal logic where most of the assumptions of the σ-calculus do not apply.

As the verification of location privacy in LBS systems is still a new field, research in different directions can be of interest. We provided three challenging topics for future research in this area but of course many more interesting questions are still open and, as the different taxonomies of location privacy we presented in Section 2.4, indicate, LPPMs can be viewed from multiple perspectives and also evaluated with different measures. The σ-calculus helps to evaluate location privacy properties of LPPMs based on a specific set of decisions and assumption, which we think represent the most relevant scenario. However, other aspects of location privacy in LBS systems can be interesting. Additionally, new technology may arrive that enables new privacy mechanisms or new attack vectors. As location-based services evolve, so does the field of its formal privacy verification.

A. Protocols

```
process Assam
    !
        if k_users
            Compute(T=noiset(t))
            Query(pids,locs,servs,T)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.1.: Perturbation protocol based on LPPM by Assam et al.

```
process Beresford
    Query(pid,loc,serv,t)
    !
        Compute(P=swap(pid))
        Query(P,loc,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.2.: Mix Zone protocol based on LPPM by Beresford et al.

```
process CaDSA
    !
        if dummies
            Query(pids,locs,servs,ts)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.3.: Cache protocol based on LPPM called *CaDSA*

```
process CAP
    !
        Compute(R=noise(loc))
        Query(pid,R,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.4.: Perturbation protocol based on LPPM called *CAP*

```
process Casper
    !
        if k_users
            Compute(R=MBB(locs))
            if l_diverse
                Query(pid,R,serv,t)
            end
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.5.: Generalization protocol based on LPPM called *Casper*

```
process CliqueCloak
    !
        if k_users
            Compute(R=MBB(locs))
            Compute(id=rand(pid))
            Query(id,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.6.: Generalization protocol based on LPPM called *CliqueCloak*

```
process Feeling-Based
    !
        if k_users
            Compute(R=MBB(locs))
            Compute(T=MBB(ts))
            if l_diverse
                Query(pid,R,serv,T)
            end
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.7.: Generalization protocol based on LPPM called *feeling-based*

```
process Freudiger
    Query(pid,loc,serv,t)
    !
        Compute(P=swap(pid))
        Query(P,loc,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.8.: Mix Zone protocol based on LPPM by Freudiger et al.

```
process Ghinita
    !
        Compute(h=hash(pid))
        Compute(R=redund(loc))
        Query(h,R,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.9.: Cryptography protocol based on LPPM by Ghinita et al.

```
process Gong
    Query(pid,loc,serv,t)
    !
        Compute(P=swap(pid))
        Query(P,loc,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.10.: Mix Zone protocol based on LPPM by Gong et al.

```
process Hoh
    !
        if k_users
            Compute(T=MBB(ts))
            Query(pid,loc,serv,T)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.11.: Perturbation protocol based on LPPM by Hoh et al.

```
process Kato
    !
        if dummies
            Compute(R=MBB(locs))
            Query(pids,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.12.: Dummies protocol based on LPPM by Kato et al.

```
process Kido
    !
        if dummies
            Compute(R=MBB(locs))
            Query(pid,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.13.: Dummies protocol based on LPPM by Kido et al.

```
process L2P2
    !
        if k_users
            Compute(R=MBB(locs))
            Compute(h=hash(pid))
            if s_diverse
                Query(h,R,servs,t)
                Compute(R=MBB(locs))
                Compute(P=swap(pids))
                Query(P,R,servs,t)
            end
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.14.: Generalization protocol based on LPPM called *L2P2*

```
process Lee
    !
        if k_users
            Compute(R=MBB(locs))
            Query(pid,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.15.: Generalization protocol based on LPPM by Lee et al.

```
process LocationDiversity
    !
        if k_users
            if l_diverse
                Compute(R=MBB(locs))
                Query(pid,R,serv,t)
            end
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.16.: Generalization protocol based on LPPM called *Location Diversity*

```
process LocationGuard
    !
        Compute(R=noise(loc))
        Query(pid,R,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
```

Figure A.17.: Perturbation protocol based on browser extension *LocationGuard*

```
process MaPIR
    !
        Compute(h=hash(pid))
        Compute(R=redund(loc))
        Query(h,R,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.18.: Cryptography protocol based on LPPM called *MaPIR*

```
process MobiCrowd
    !
        Query(pid,loc,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.19.: Cache protocol based on LPPM called *MobiCrowd*

```
process MobiMix
    Query(pid,loc,serv,t)
    !
        if k_users
            Compute(P=swap(pid))
            Query(P,loc,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.20.: Mix Zones protocol based on LPPM called *MobiMix*

```
process MobiPriv
    !
        if k_users
            Compute(R=MBB(locs))
            Query(pids,R,serv,t)
        else
            if dummies
                Compute(R=MBB(locs))
                Query(pids,R,serv,t)
            end
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.21.: Dummies protocol based on LPPM called *MobiPriv*

```
process PrivacyGrid
    !
        if k_users
            Compute(R=MBB(locs))
            if s_diverse
                Query(pid,R,servs,t)
            end
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.22.: Generalization protocol based on LPPM called *PrivacyGrid*

```
process PRIVE
    !
        if k_users
            Compute(h=hash(pid))
            Compute(R=MBB(locs))
            Query(h,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.23.: Generalization protocol based on LPPM called *PRIVE*

```
process ReverseCloak
    !
        if k_users
            Compute(R=MBB(locs))
            Query(pid,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.24.: Generalization protocol based on LPPM called *ReverseCloak*

```
process SpaceTwist
    !
        Compute(h=hash(pid))
        Compute(R=noise(loc))
        Query(h,R,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.25.: Cryptography protocol based on LPPM called *SpaceTwist*

```
process SpotME
    !
        if dummies
            Compute(R=MBB(locs))
            Query(pid,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.26.: Dummies protocol based on LPPM called *SpotME*

```
process SybilQuery
    !
        if dummies
            Compute(h=hash(pid))
            Compute(R=MBB(locs))
            Query(h,R,serv,t)
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.27.: Dummies protocol based on LPPM called *SybilQuery*

```
process TrustNoOne
    !
        Compute(h=hash(pid))
        Compute(R=redund(loc))
        Compute(S=redund(serv))
        Query(h,R,S,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.28.: Cryptography protocol based on LPPM called *Trust no one*

```
process Xinxin
    Query(pid,loc,serv,t)
    !
        Compute(P=swap(pid))
        Query(P,loc,serv,t)
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.29.: Mix Zone protocol based on LPPM by Xinxin et al.

```
process Xu
    if k_users
        if l_diverse
            !
                Compute(R=MBB(locs))
                Query(pid,R,serv,t)
            end
        end
    end
end

property F1
    G not K_id
property F2
    G not (K_loc and K_t)
property F3
    G not Cont (K_loc and K_id)
property F4
    G not K_serv
```

Figure A.30.: Generalization protocol based on LPPM by Xu et al.

B. LP3Verif User Manual

Steps for installation and setup:

1. Install required SMT solver
   - Download CVC4 (`https://cvc4.github.io/downloads.html`) and if necessary follow the installation instructions from the user manual (`http://cvc4.cs.stanford.edu/wiki/User_Manual`).
2. Download LP3Verif binaries
   - Download the zip file from the latest release of the repository linked in `https://www.tuhh.de/sts/research/data-protection-machine-learning/lp3verif.html` and unzip in target directory.
3. Configure LP3Verif
   - In the unzipped LP3Verif folder edit the `tool.config` file with the editor of your choice.
   - Edit the solver path to the absolute path (using / as separator) where your CVC4 binaries are located. If necessary edit the name of the solver, e.g., `cvc4.exe` for Windows.
   - Then edit the smt2 path and name to a file of your choice. This file will be used by the SMT solver and can contain additional tracing/debug information for LP3Verif.
4. Setup execution
   - Depending on your operating system you can now create a shell or batch script to start the executable `.jar` file.
   - An example of a Windows batch file can be seen in Figure B.1.
5. Start LP3Verif
   - The command `java -jar "LP3Verif.jar"` starts LP3Verif in interactive mode where a command line prompt will ask for a path to an `.lp3` file.
   - Calling LP3Verif with the additional argument `-lp3 path/file.lp3` directly starts the verification of the specified `.lp3` file.

```
1 @echo off
  java -jar "LP3Verif.jar"
3 @pause
```

Figure B.1.: Batch file to start LP3Verif on Windows

Bibliography

[1] V. Primault, A. Boutet, S. B. Mokhtar, and L. Brunie, "The Long Road to Computational Location Privacy: A Survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2772–2793, 2019.

[2] B. Bamba, L. Liu, P. Pesti, and T. Wang, "Supporting anonymous location queries in mobile environments with PrivacyGrid," in *Proceeding of the 17th International Conference on World Wide Web - WWW '08*. Beijing, China: ACM Press, 2008, pp. 237–246.

[3] M. D. Ryan and B. Smyth, "Applied pi calculus." *Formal Models and Techniques for Analyzing Security Protocols*, vol. 5, pp. 112–142, 2011.

[4] H.-J. Lee, S.-T. Hong, M. Yoon, J.-H. Um, and J.-W. Chang, "A new cloaking algorithm using Hilbert curves for privacy protection," in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, ser. SPRINGL '10. San Jose, California: Association for Computing Machinery, Nov. 2010, pp. 42–46.

[5] A. Pingley, W. Yu, N. Zhang, X. Fu, and W. Zhao, "CAP: A Context-Aware Privacy Protection System for Location-Based Services," in *2009 29th IEEE International Conference on Distributed Computing Systems*, Jun. 2009, pp. 49–57.

[6] R. Assam and T. Seidl, "Preserving privacy of moving objects via temporal clustering of spatio-temporal data streams," in *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, ser. SPRINGL '11. Chicago, Illinois: Association for Computing Machinery, Nov. 2011, pp. 9–16.

[7] P. M. Wightman, M. Zurbarán, M. Rodríguez, and M. A. Labrador, "MaPIR: Mapping-based private information retrieval for location privacy in LBISs," in *38th Annual IEEE Conference on Local Computer Networks - Workshops*, Oct. 2013, pp. 964–971.

[8] S. Jaiswal and A. Nandi, "Trust no one: A decentralized matching service for privacy in location based services," in *Proceedings of the Second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds*, ser. MobiHeld '10. New Delhi, India: Association for Computing Machinery, Aug. 2010, pp. 51–56.

[9] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, Aug. 1978.

[10] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Aug. 2004.

[11] R. Fagin, Y. Moses, J. Y. Halpern, and M. Y. Vardi, *Reasoning About Knowledge*. MIT Press, 2003.

[12] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. Berlin, Germany: Association for Computing Machinery, Nov. 2013, pp. 901–914.

[13] European Union, "General Data Protection Regulation 2016/679," http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf, 2016.

[14] J. Krumm, "Inference Attacks on Location Tracks," in *Pervasive Computing*, ser. Lecture Notes in Computer Science, A. LaMarca, M. Langheinrich, and K. N. Truong, Eds. Berlin, Heidelberg: Springer, 2007, pp. 127–143.

[15] ——, "A survey of computational location privacy," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 391–399, Aug. 2009.

[16] S. Spiekermann, "General Aspects of Location-Based Services," in *Location-Based Services*. Morgan Kaufmann Publishers, 2004, vol. 9, pp. 14–33.

[17] K. Gratsias, E. Frentzos, V. Delis, and Y. Theodoridis, "Towards a Taxonomy of Location Based Services," in *Web and Wireless Geographical Information Systems*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, K.-J. Li, and C. Vangenot, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3833, pp. 19–30.

[18] A. Beresford and F. Stajano, "Mix zones: User privacy in location-aware services," in *IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Mar. 2004, pp. 127–131.

[19] J. Freudiger, R. Shokri, and J.-P. Hubaux, "On the Optimal Placement of Mix Zones," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, I. Goldberg and M. J. Atallah, Eds. Berlin, Heidelberg: Springer, 2009, pp. 216–234.

[20] X. Liu, H. Zhao, M. Pan, H. Yue, X. Li, and Y. Fang, "Traffic-aware multiple mix zone placement for protecting location privacy," in *2012 Proceedings IEEE INFOCOM*, Mar. 2012, pp. 972–980.

[21] B. Palanisamy and L. Liu, "MobiMix: Protecting location privacy with mix-zones over road networks," in *2011 IEEE 27th International Conference on Data Engineering*, Apr. 2011, pp. 494–505.

[22] X. Gong, X. Chen, K. Xing, D.-H. Shin, M. Zhang, and J. Zhang, "From Social Group Utility Maximization to Personalized Location Privacy in Mobile Networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1703–1716, Jun. 2017.

[23] G. Sun, D. Liao, H. Li, H. Yu, and V. Chang, "L2P2: A location-label based approach for privacy preserving in LBS," *Future Generation Computer Systems*, vol. 74, pp. 375–384, Sep. 2017.

[24] G. Ghinita, P. Kalnis, and S. Skiadopoulos, "PRIVE: Anonymous location-based queries in distributed mobile systems," in *Proceedings of the 16th International Conference on World Wide Web - WWW '07*. Banff, Alberta, Canada: ACM Press, 2007, pp. 371–380.

[25] K. Huguenin, I. Bilogrevic, J. S. Machado, S. Mihaila, R. Shokri, I. Dacosta, and J.-P. Hubaux, "A Predictive Model for User Motivation and Utility Implications of Privacy-Protection Mechanisms in Location Check-Ins," *IEEE Transactions on Mobile Computing*, vol. 17, no. 4, pp. 760–774, Apr. 2018.

[26] B. Gedik and L. Liu, "Location Privacy in Mobile Systems: A Personalized Anonymization Model," in *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, Jun. 2005, pp. 620–629.

[27] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The New Casper: Query Processing for Location Services without Compromising Privacy," in *Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006, pp. 763–774.

[28] C.-Y. Chow, M. F. Mokbel, and X. Liu, "A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-based Services," in *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, 2006, pp. 171–178.

[29] T. Xu and Y. Cai, "Feeling-based location privacy protection for location-based services," in *Proceedings of the 16th ACM Conference on Computer and Communications Security - CCS '09*. Chicago, Illinois, USA: ACM Press, 2009, p. 348.

[30] B. Agir, T. G. Papaioannou, R. Narendula, K. Aberer, and J.-P. Hubaux, "User-side adaptive protection of location privacy in participatory sensing," *GeoInformatica*, vol. 18, no. 1, pp. 165–191, Jan. 2014.

[31] H. Ngo and J. Kim, "Location Privacy via Differential Private Perturbation of Cloaking Area," in *2015 IEEE 28th Computer Security Foundations Symposium*, Jul. 2015, pp. 63–74.

[32] C. Li and B. Palanisamy, "ReverseCloak: Protecting Multi-level Location Privacy over Road Networks," in *Proceedings of the 24th ACM International on Conference*

on Information and Knowledge Management - CIKM '15. Melbourne, Australia: ACM Press, 2015, pp. 673–682.

[33] C.-Y. Chow, M. F. Mokbel, and W. G. Aref, "Casper*: Query processing for location services without compromising privacy," *ACM Transactions on Database Systems*, vol. 34, no. 4, pp. 24:1–24:48, Dec. 2009.

[34] T. Xu and Y. Cai, "Location anonymity in continuous location-based services," in *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*, ser. GIS '07. Seattle, Washington: Association for Computing Machinery, Nov. 2007, pp. 1–8.

[35] M. Xue, P. Kalnis, and H. K. Pung, "Location Diversity: Enhanced Privacy Protection in Location Based Services," in *Location and Context Awareness*, ser. Lecture Notes in Computer Science, T. Choudhury, A. Quigley, T. Strang, and K. Suginuma, Eds. Berlin, Heidelberg: Springer, 2009, pp. 70–87.

[36] D. Quercia, I. Leontiadis, L. McNamara, C. Mascolo, and J. Crowcroft, "SpotME If You Can: Randomized Responses for Location Obfuscation on Mobile Phones," in *2011 31st International Conference on Distributed Computing Systems*, Jun. 2011, pp. 363–372.

[37] H. Kido, Y. Yanagisawa, and T. Satoh, "Protection of Location Privacy using Dummies for Location-based Services," in *21st International Conference on Data Engineering Workshops (ICDEW'05)*, Apr. 2005, p. 1248.

[38] J. Krumm, "Realistic Driving Trips For Location Privacy," in *Pervasive Computing*, ser. Lecture Notes in Computer Science, H. Tokuda, M. Beigl, A. Friday, A. J. B. Brush, and Y. Tobe, Eds. Berlin, Heidelberg: Springer, 2009, pp. 25–41.

[39] V. Bindschaedler and R. Shokri, "Synthesizing Plausible Privacy-Preserving Location Traces," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 546–563.

[40] T.-H. You, W.-C. Peng, and W.-C. Lee, "Protecting Moving Trajectories with Dummies," in *2007 International Conference on Mobile Data Management*, May 2007, pp. 278–282.

[41] L. Stenneth, P. S. Yu, and O. Wolfson, "Mobile systems location privacy: "MobiPriv" a robust k anonymous system," in *2010 IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications*, Oct. 2010, pp. 54–63.

[42] R. Kato, M. Iwata, T. Hara, A. Suzuki, X. Xie, Y. Arase, and S. Nishio, "A dummy-based anonymization method based on user trajectory with pauses," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '12. Redondo Beach, California: Association for Computing Machinery, Nov. 2012, pp. 249–258.

[43] P. Shankar, V. Ganapathy, and L. Iftode, "Privately querying location-based services with SybilQuery," in *Proceedings of the 11th International Conference on Ubiquitous Computing*, ser. UbiComp '09. Orlando, Florida, USA: Association for Computing Machinery, Sep. 2009, pp. 31–40.

[44] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Preserving privacy in GPS traces via uncertainty-aware path cloaking," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. Alexandria, Virginia, USA: Association for Computing Machinery, Oct. 2007, pp. 161–171.

[45] K. Micinski, P. Phelps, and J. S. Foster, "An empirical study of location truncation on Android," in *Mobile SecurityTechnologies (MoST '13)*, San Francisco, CA, USA, May 2013.

[46] N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Optimal Geo-Indistinguishable Mechanisms for Location Privacy," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. Scottsdale, Arizona, USA: Association for Computing Machinery, Nov. 2014, pp. 251–262.

[47] S. Oya, C. Troncoso, and F. Pérez-González, "Back to the Drawing Board: Revisiting the Design of Optimal Location Privacy-preserving Mechanisms," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, Oct. 2017, pp. 1959–1972.

[48] K. Chatzikokolakis, C. Palamidessi, and M. Stronati, "A Predictive Differentially-Private Mechanism for Mobility Traces," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, E. De Cristofaro and S. J. Murdoch, Eds. Cham: Springer International Publishing, 2014, pp. 21–41.

[49] Y. Xiao, L. Xiong, S. Zhang, and Y. Cao, "LocLok: Location cloaking with differential privacy via hidden markov model," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1901–1904, Aug. 2017.

[50] L. Yu, L. Liu, and C. Pu, "Dynamic Differential Location Privacy with Personalized Error Bounds," in *Proceedings 2017 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2017, pp. 1–15.

[51] K. Chatzikokolakis, C. Palamidessi, and M. Stronati, "Constructing elastic distinguishability metrics for location privacy," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 156–170, Jun. 2015.

[52] G. Zhong, I. Goldberg, and U. Hengartner, "Louis, Lester and Pierre: Three Protocols for Location Privacy," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, N. Borisov and P. Golle, Eds. Berlin, Heidelberg: Springer, 2007, pp. 62–76.

[53] S. Mascetti, D. Freni, C. Bettini, X. S. Wang, and S. Jajodia, "Privacy in geo-social networks: Proximity notification with untrusted service providers and curious buddies," *The VLDB Journal*, vol. 20, no. 4, pp. 541–566, Aug. 2011.

[54] R. A. Popa, A. J. Blumberg, H. Balakrishnan, and F. H. Li, "Privacy and accountability for location-based aggregate statistics," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. Chicago, Illinois, USA: Association for Computing Machinery, Oct. 2011, pp. 653–666.

[55] S. Guha, M. Jain, and V. N. Padmanabhan, "Koi: A Location-Privacy Platform for Smartphone Apps," in *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012, pp. 183–196.

[56] R. Shokri, P. Papadimitratos, G. Theodorakopoulos, and J.-P. Hubaux, "Collaborative Location Privacy," in *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, Oct. 2011, pp. 500–509.

[57] U. M. Aïvodji, K. Huguenin, M.-J. Huguet, and M.-O. Killijian, "SRide: A Privacy-Preserving Ridesharing System," in *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '18. Stockholm, Sweden: Association for Computing Machinery, Jun. 2018, pp. 40–50.

[58] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: Anonymizers are not necessary," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. Vancouver, Canada: Association for Computing Machinery, Jun. 2008, pp. 121–132.

[59] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, D. Boneh *et al.*, "Location privacy via private proximity testing." in *Networkand Distributed Security Symposium (NDSS'11)*. The Internet Society, 2011.

[60] S. Pidcock and U. Hengartner, "Zerosquare: A privacy-friendly location hub for geosocial applications," in *Proceedings of IEEE Mobile Security Technologies Workshop (MoST'13)*. IEEE Press, 2013, pp. 1–10.

[61] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," *Journal of Computer Security*, vol. 24, no. 2, pp. 137–180, Apr. 2016.

[62] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu, "SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services," in *2008 IEEE 24th International Conference on Data Engineering*, Apr. 2008, pp. 366–375.

[63] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li, "Enhancing privacy through caching in location-based services," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015, pp. 1017–1025.

[64] S. Chakraborty, C. Shen, K. R. Raghavan, Y. Shoukry, M. Millar, and M. Srivastava, "ipshield: A framework for enforcing context-aware privacy," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 2014, pp. 143–156.

[65] K. Fawaz and K. G. Shin, "Location Privacy Protection for Smartphone Users," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. Scottsdale, Arizona, USA: Association for Computing Machinery, Nov. 2014, pp. 239–250.

[66] J. Domingo-Ferrer and R. Trujillo-Rasua, "Microaggregation- and permutation-based anonymization of movement data," *Information Sciences*, vol. 208, pp. 55–80, Nov. 2012.

[67] S. Martínez-Bea and V. Torra, "Trajectory anonymization from a time series perspective," in *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, Jun. 2011, pp. 401–408.

[68] S. Mascetti, C. Bettini, D. Freni, and X. S. Wang, "Spatial generalisation algorithms for LBS privacy preservation," *Journal of Location Based Services*, vol. 1, no. 3, pp. 179–207, Sep. 2007.

[69] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li, "Achieving k-anonymity in privacy-aware location-based services," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, Apr. 2014, pp. 754–762.

[70] R. Shokri, G. Theodorakopoulos, J. L. Boudec, and J. Hubaux, "Quantifying Location Privacy," in *2011 IEEE Symposium on Security and Privacy*, May 2011, pp. 247–262.

[71] S. Zeng, Y. Mu, M. He, and Y. Chen, "New Approach for Privacy-Aware Location-Based Service Communications," *Wireless Personal Communications*, vol. 101, no. 2, pp. 1057–1073, Jul. 2018.

[72] P. Samarati and L. Sweeney, "Protecting Privacy when Disclosing Information: K-Anonymity and its Enforcement through Generalization and Suppression," Computer Science Laboratory, SRI International, Tech. Rep., 1998.

[73] C.-Y. Chow and M. F. Mokbel, "Trajectory privacy in location-based services and data publication," *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 1, pp. 19–29, Aug. 2011.

[74] M. Wernke, P. Skvortsov, F. Dürr, and K. Rothermel, "A classification of location privacy attacks and approaches," *Personal and Ubiquitous Computing*, vol. 18, no. 1, pp. 163–175, Jan. 2014.

[75] J. C. Baeten, "A brief history of process algebra," *Theoretical Computer Science*, vol. 335, no. 2-3, pp. 131–146, May 2005.

[76] R. Milner, *A Calculus of Communicating Systems*, ser. Lecture Notes in Computer Science. Berlin Heidelberg: Springer-Verlag, 1980.

[77] J. C. M. Baeten and J. A. Bergstra, *On Sequential Composition, Action Prefixes and Process Prefix*. Springer, 1994.

[78] J. A. Bergstra and J. W. Klop, "Process algebra for synchronous communication," *Information and Control*, vol. 60, no. 1, pp. 109–137, Jan. 1984.

[79] H. R. Nielson and F. Nielson, *Semantics with Applications*. Chichester: Wiley, 1992, vol. 104.

[80] G. D. Plotkin, "The origins of structural operational semantics," *The Journal of Logic and Algebraic Programming*, vol. 60-61, pp. 3–15, Jul. 2004.

[81] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, I," *Information and Computation*, vol. 100, no. 1, pp. 1–40, Sep. 1992.

[82] ——, "A calculus of mobile processes, II," *Information and Computation*, vol. 100, no. 1, pp. 41–77, Sep. 1992.

[83] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *The Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, Feb. 2008.

[84] van Benthem, J., *Modal Logic for Open Minds*. Stanford, CA, USA: Center for the Study of Language and Information (CSLI) Publications, 2010, no. 199.

[85] A. N. Prior, "Diodoran Modalities," *The Philosophical Quarterly (1950-)*, vol. 5, no. 20, pp. 205–213, 1955.

[86] H. Kamp, "Tense Logic and the Theory if linear order," PhD Thesis, University of California, Los Angeles, 1968.

[87] P. Øhrstrøm and P. Hasle, *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Springer Science & Business Media, Aug. 2007.

[88] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (Sfcs 1977)*, Oct. 1977, pp. 46–57.

[89] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Logics of Programs*, ser. Lecture Notes in Computer Science, D. Kozen, Ed. Berlin, Heidelberg: Springer, 1982, pp. 52–71.

[90] J. Hintikka, "Knowledge and Belief: An Introduction to the Logic of the Two Notions," *Studia Logica*, vol. 16, pp. 119–122, 1962.

[91] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*. Cham, Switzerland: Springer, 2018.

[92] G. J. Holzmann, "Explicit-State Model Checking," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Cham: Springer International Publishing, 2018, pp. 153–171.

[93] ——, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[94] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla, "Symmetry reductions in model checking," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, A. J. Hu and M. Y. Vardi, Eds. Berlin, Heidelberg: Springer, 1998, pp. 147–158.

[95] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple On-the-fly Automatic Verification of Linear Temporal Logic," in *Protocol Specification, Testing and Verification XV: Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995*, ser. IFIP Advances in Information and Communication Technology, P. Dembiński and M. Średniawa, Eds. Boston, MA: Springer US, 1996, pp. 3–18.

[96] P. A. Abdulla, A. P. Sistla, and M. Talupur, "Model Checking Parameterized Systems," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Cham: Springer International Publishing, 2018, pp. 685–725.

[97] S. Chaki and A. Gurfinkel, "BDD-Based Symbolic Model Checking," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Cham: Springer International Publishing, 2018, pp. 219–245.

[98] K. L. McMillan, "Symbolic Model Checking," in *Symbolic Model Checking*. Boston, MA: Springer US, 1993, pp. 25–60.

[99] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A New Symbolic Model Verifier," in *Computer Aided Verification*, N. Halbwachs and D. Peled, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, vol. 1633, pp. 495–499.

[100] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded Model Checking," in *Advances in Computers*. Elsevier, 2003, vol. 58, pp. 117–148.

[101] C. Barrett and C. Tinelli, "Satisfiability Modulo Theories," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Cham: Springer International Publishing, 2018, pp. 305–343.

[102] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds. Berlin, Heidelberg: Springer, 2011, pp. 171–177.

[103] C. Barrett, A. Stump, C. Tinelli *et al.*, "The SMT-Lib standard: Version 2.0," in *Proceedings of the 8th International Workshop on Satisfiability modulo Theories (Edinburgh, England)*, 2010.

[104] X. Chen, A. Mizera, and J. Pang, "Quantifying location privacy revisited: Preliminary report," University of Luxembourg, Tech. Report, 2014.

[105] X. Zhang, X. Gui, and F. Tian, "A Framework for measuring query privacy in Location-based Service," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 9, no. 5, pp. 1717–1732, 2015.

[106] R. Shokri, "Quantifying and protecting location privacy," *it - Information Technology*, vol. 57, no. 4, Jan. 2015.

[107] R. Shokri, J. Freudiger, M. Jadliwala, and J.-P. Hubaux, "A distortion-based metric for location privacy," in *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society - WPES '09*. Chicago, Illinois, USA: ACM Press, 2009, pp. 21–30.

[108] J. Freudiger, R. Shokri, and J.-P. Hubaux, "Evaluating the Privacy Risk of Location-Based Services," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, G. Danezis, Ed. Berlin, Heidelberg: Springer, 2012, pp. 31–46.

[109] R. Shokri, G. Theodorakopoulos, and C. Troncoso, "Privacy Games Along Location Traces: A Game-Theoretic Framework for Optimizing Location Privacy," *ACM Transactions on Privacy and Security*, vol. 19, no. 4, pp. 1–31, Dec. 2016.

[110] J. Ding, X. Li, Y. Guo, L. Yin, and H. Zhang, "Process Calculus for Modeling and Quantifying Location Privacy," *Procedia Computer Science*, vol. 147, pp. 407–415, Jan. 2019.

[111] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, "Enterprise privacy authorization language (EPAL)," IBM Research, Research Report, 2003.

[112] J. Reagle and L. F. Cranor, "The Platform for Privacy Preferences," *Communications of the ACM*, vol. 42, no. 2, pp. 48–49, Feb. 1999.

[113] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum, "Privacy and Contextual Integrity: Framework and Applications," in *2006 IEEE symposium on security and privacy*, ser. Symposium on Security and Privacy. USA: IEEE Computer Society, 2006, pp. 184–198.

[114] M. C. Tschantz, "Formalizing and Enforcing Purpose Restrictions," Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Tech. Rep., May 2012.

[115] A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D. Kaynar, and A. Sinha, "Understanding and Protecting Privacy: Formal Semantics and Principled Audit Mechanisms," in *International Conference on Information Systems Security*, S. Jajodia and C. Mazumdar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–27.

[116] N. Dong, H. Jonker, and J. Pang, "Formal Analysis of Privacy in an eHealth Protocol," in *Computer Security – ESORICS 2012*, ser. Lecture Notes in Computer Science, S. Foresti, M. Yung, and F. Martinelli, Eds. Berlin, Heidelberg: Springer, 2012, pp. 325–342.

[117] F. Amato and F. Moscato, "A model driven approach to data privacy verification in E-Health systems," *Transactions on Data Privacy*, vol. 8, no. 3, pp. 273–296, 2015.

[118] M. Maffei, K. Pecina, and M. Reinert, "Security and Privacy by Declarative Design," in *2013 IEEE 26th Computer Security Foundations Symposium*, Jun. 2013, pp. 81–96.

[119] D. Le Métayer, "Privacy by Design: A formal framework for the analysis of architectural choices," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, 2013, pp. 95–104.

[120] T. Antignac and D. Le Métayer, "Privacy Architectures: Reasoning about Data Minimisation and Integrity," in *Security and Trust Management*, ser. Lecture Notes in Computer Science, S. Mauw and C. D. Jensen, Eds. Cham: Springer International Publishing, 2014, pp. 17–32.

[121] ——, "Trust Driven Strategies for Privacy by Design," in *Trust Management IX*, ser. IFIP Advances in Information and Communication Technology, C. Damsgaard Jensen, S. Marsh, T. Dimitrakos, and Y. Murayama, Eds. Cham: Springer International Publishing, 2015, pp. 60–75.

[122] T. Antignac, D. Sands, and G. Schneider, "Data Minimisation: A Language-Based Approach," in *ICT Systems Security and Privacy Protection*, S. De Capitani di Vimercati and F. Martinelli, Eds. Cham: Springer International Publishing, 2017, vol. 502, pp. 442–456.

[123] K. Bavendiek, R. Adams, and S. Schupp, "Privacy-Preserving Architectures with Probabilistic Guaranties," in *Privacy, Security and Trust*, 2018, pp. 1–10.

[124] M. Abadi and B. Blanchet, "Computer-assisted verification of a protocol for certified email," *Science of Computer Programming*, vol. 58, no. 1, pp. 3–27, Oct. 2005.

[125] B. Blanchet, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[126] K. Bhargavan, C. Fournet, A. D. Gordon, and S. Tse, "Verified interoperable implementations of security protocols," *ACM Transactions on Programming Languages and Systems*, vol. 31, no. 1, pp. 5:1–5:61, Dec. 2008.

[127] R. Küsters and T. Truderung, "Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation," in *2009 22nd IEEE Computer Security Foundations Symposium*, Jul. 2009, pp. 157–171.

[128] C. Fournet and M. Abadi, "Hiding Names: Private Authentication in the Applied Pi Calculus," in *Software Security — Theories and Systems*, G. Goos, J. Hartmanis, J. van Leeuwen, M. Okada, B. C. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, vol. 2609, pp. 317–338.

[129] S. Kremer and M. Ryan, "Analysis of an Electronic Voting Protocol in the Applied Pi Calculus," in *European Symposium on Programming*, ser. Lecture Notes in Computer Science, M. Sagiv, Ed. Berlin, Heidelberg: Springer, 2005, pp. 186–200.

[130] S. Delaune, M. Ryan, and B. Smyth, "Automatic Verification of Privacy Properties in the Applied pi Calculus," in *Trust Management II*, Y. Karabulut, J. Mitchell, P. Herrmann, and C. D. Jensen, Eds. Boston, MA: Springer US, 2008, vol. 263, pp. 263–278.

[131] M. Backes, C. Hritcu, and M. Maffei, "Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-Calculus," in *2008 21st IEEE Computer Security Foundations Symposium*. IEEE, 2008, pp. 195–209.

[132] R. Chadha, S. Delaune, and S. Kremer, "Epistemic Logic for the Applied Pi Calculus," in *Formal Techniques for Distributed Systems*, D. Lee, A. Lopes, and A. Poetzsch-Heffter, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5522, pp. 182–197.

[133] C. Fournet and G. Gonthier, "The reflexive CHAM and the join-calculus," in *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '96. New York, NY, USA: Association for Computing Machinery, Jan. 1996, pp. 372–385.

[134] L. Cardelli and A. D. Gordon, "Mobile ambients," in *International Conference on Foundations of Software Science and Computation Structure*. Springer, 1998, pp. 140–155.

[135] C. Fournet and G. Gonthier, "The Join Calculus: A Language for Distributed Mobile Programming," in *Applied Semantics*, ser. Lecture Notes in Computer Science, G. Barthe, P. Dybjer, L. Pinto, and J. Saraiva, Eds. Berlin, Heidelberg: Springer, 2002, pp. 268–332.

[136] L. Cardelli and A. D. Gordon, "Anytime, anywhere: Modal logics for mobile ambients," in *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '00*. Boston, MA, USA: ACM Press, 2000, pp. 365–377.

[137] G. Anderson and D. Pym, "A calculus and logic of bunched resources and processes," *Theoretical Computer Science*, vol. 614, pp. 63–96, Feb. 2016.

[138] F. Siewe, H. Zedan, and A. Cau, "The Calculus of Context-aware Ambients," *Journal of Computer and System Sciences*, vol. 77, no. 4, pp. 597–620, Jul. 2011.

[139] A. Singh, C. R. Ramakrishnan, and S. A. Smolka, "A process calculus for Mobile Ad Hoc Networks," *Science of Computer Programming*, vol. 75, no. 6, pp. 440–469, Jun. 2010.

[140] M. Sato, "A Study of Kripke-type Models for Some Modal Logics by Gentzen's Sequential Method," *Publications of the Research Institute for Mathematical Sciences*, vol. 13, no. 2, pp. 381–468, 1977.

[141] W. van der Hoek and M. Wooldridge, "Cooperation, Knowledge, and Time: Alternating-time Temporal Epistemic Logic and its Applications," *Studia Logica*, vol. 75, no. 1, pp. 125–157, Oct. 2003.

[142] J. Y. Halpern, R. van der Meyden, and M. Y. Vardi, "Complete Axiomatizations for Reasoning about Knowledge and Time," *SIAM Journal on Computing*, vol. 33, no. 3, pp. 674–703, 2004.

[143] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time temporal logic," in *Proceedings 38th Annual Symposium on Foundations of Computer Science*, Oct. 1997, pp. 100–109.

[144] K. Su, A. Sattar, and X. Luo, "Model Checking Temporal Logics of Knowledge Via OBDDs," *The Computer Journal*, vol. 50, no. 4, pp. 403–420, Jul. 2007.

[145] M. Cohen, M. Dam, A. Lomuscio, and H. Qu, "A symmetry reduction technique for model checking temporal-epistemic logic," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jul. 2009, pp. 721–726.

[146] A. Lomuscio, W. Penczek, and H. Qu, "Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems," *Fundamenta Informaticae*, vol. 101, no. 1-2, pp. 71–90, Jan. 2010.

[147] P. Gammie and R. van der Meyden, "MCK: Model Checking the Logic of Knowledge," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, R. Alur and D. A. Peled, Eds. Berlin, Heidelberg: Springer, 2004, pp. 479–483.

[148] A. Lomuscio and F. Raimondi, "MCMAS: A Model Checker for Multi-agent Systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, H. Hermanns and J. Palsberg, Eds. Berlin, Heidelberg: Springer, 2006, pp. 450–454.

[149] X. Huang, C. Luo, and R. van der Meyden, "Symbolic model checking of probabilistic knowledge," in *Proceedings of Hte 13th Conference on Theoretical Aspects of Rationality and Knowledge - TARK XIII*. Groningen, Netherlands: ACM Press, 2011, pp. 177–186.

[150] W. Wan, J. Bentahar, and A. Ben Hamza, "Model Checking Epistemic and Probabilistic Properties of Multi-agent Systems," in *Modern Approaches in Applied Intelligence*, ser. Lecture Notes in Computer Science, K. G. Mehrotra, C. K. Mohan, J. C. Oh, P. K. Varshney, and M. Ali, Eds. Berlin, Heidelberg: Springer, 2011, pp. 68–78.

[151] C. Delgado and M. Benevides, "Verification of Epistemic Properties in Probabilistic Multi-Agent Systems," in *Multiagent System Technologies*, ser. Lecture Notes in Computer Science, L. Braubach, W. van der Hoek, P. Petta, and A. Pokahr, Eds. Berlin, Heidelberg: Springer, 2009, pp. 16–28.

[152] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, Sep. 1994.

[153] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala, "Symbolic Model Checking of Probabilistic Processes Using MTBDDs and the Kronecker Representation," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, S. Graf and M. Schwartzbach, Eds. Berlin, Heidelberg: Springer, 2000, pp. 395–410.

[154] M. R. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips, "Design and analysis of DNA strand displacement devices using probabilistic model checking," *Journal of The Royal Society Interface*, vol. 9, no. 72, pp. 1470–1485, Jul. 2012.

[155] T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre, "Quantitative Verification of Implantable Cardiac Pacemakers," in *2012 IEEE 33rd Real-Time Systems Symposium*, Dec. 2012, pp. 263–272.

[156] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic model checking in practice: Case studies with PRISM," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 4, pp. 16–21, Mar. 2005.

[157] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking for real-time systems," in *Fifth Annual IEEE Symposium on Logic in Computer Science*, Jun. 1990, pp. 414–425.

[158] J.-F. Raskin and P.-Y. Schobbens, "State clock logic: A decidable real-time logic," in *Hybrid and Real-Time Systems*, ser. Lecture Notes in Computer Science, O. Maler, Ed. Berlin, Heidelberg: Springer, 1997, pp. 33–47.

[159] K. G. Larsen, P. Pettersson, and W. Yi, "Model-checking for real-time systems," in *Fundamentals of Computation Theory*, ser. Lecture Notes in Computer Science, H. Reichel, Ed. Berlin, Heidelberg: Springer, 1995, pp. 62–88.

[160] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Verification of an Audio Protocol with bus collision using Uppaal," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, R. Alur and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer, 1996, pp. 244–256.

[161] H. Lonn and P. Pettersson, "Formal verification of a TDMA protocol start-up mechanism," in *Proceedings Pacific Rim International Symposium on Fault-Tolerant Systems*, Dec. 1997, pp. 235–242.

[162] A. P. Ravn, J. Srba, and S. Vighio, "A Formal Analysis of the Web Services Atomic Transaction Protocol with UPPAAL," in *Leveraging Applications of Formal Methods, Verification, and Validation*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds. Berlin, Heidelberg: Springer, 2010, pp. 579–593.

[163] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, pp. 872–923, May 1994.

[164] H. Wang, Q. Zhou, and Y. Shi, "Describing and Verifying Web Service Composition Using TLA Reasoning," in *2010 IEEE International Conference on Services Computing*, Jul. 2010, pp. 234–241.

[165] P. Syverson and C. Meadows, "A Formal Language for Cryptographic Protocol Requirements," *Designs, Codes and Cryptography*, vol. 7, no. 1, pp. 27–59, Jan. 1996.

[166] C. Meadows, "The NRL Protocol Analyzer: An Overview," *The Journal of Logic Programming*, vol. 26, no. 2, pp. 113–131, Feb. 1996.

[167] K. Chatzikokolakis, C. Palamidessi, and M. Stronati, "Location Guard: Location privacy for the rest of us," in *8th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2015)*, Philadelphia, PA, USA, 2015.

[168] H. Nissenbaum and H. Daniel, "Trackmenot: Resisting Surveillance in Web Search," Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 2567412, Feb. 2015.

[169] W. Schulz, F. Wittner, K. Bavendiek, and S. Schupp, "Modeling and verification in GDPR's data protection impact assessment," in *Data Protection and Privacy: Data Protection and Democracy*, 2019, pp. 145–172.

[170] C. Kurtz, F. Wittner, M. Semmann, W. Schulz, and T. Böhmann, "The Unlikely Siblings in the GDPR Family: A Techno-Legal Analysis of Major Platforms in the Diffusion of Personal Data in Service Ecosystems," in *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*, Jan. 2019, pp. 1–10.

[171] S. B. Barnes, "A privacy paradox: Social networking in the United States," *First Monday*, vol. 11, no. 9, Sep. 2006.

[172] T. Dienlin and S. Trepte, "Is the privacy paradox a relic of the past? An in-depth analysis of privacy attitudes and privacy behaviors," *European Journal of Social Psychology*, vol. 45, no. 3, pp. 285–297, 2015.

[173] C. Dwork, "Differential Privacy," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Berlin, Heidelberg: Springer, 2006, pp. 1–12.