Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

BACHELOR'S THESIS IN INFORMATICS

Privacy-preserving VoIP Signaling with Secure Multiparty Computation

Christian Kilb

Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

Privacy-preserving VoIP Signaling with Secure Multiparty Computation

Privatheitsschützende VoIP-Signalisierung mittels Secure Multiparty Computation

Author:	Christian Kilb
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Dr. Marcel von Maltitz
	Richard von Seck, M. Sc.
Date:	September 15, 2019

I confirm that this Bachelor's Thesis is my own work and I have documented all sources and material used.

Garching, September 15, 2019

Location, Date

Signature

Abstract

Voice over IP (VoIP) providers require access to user data in order to fulfill their VoIP signaling tasks. They currently know the associations between user identities, phone numbers and IP addresses, which is a significant amount of privacy-critical data. Privacy of user data on provider-side can be improved with Secure Multiparty Computation (SMC). SMC is an emerging cryptographic technique which enables multiple parties to jointly perform computations on private data in a secure manner. This thesis applies SMC to VoIP signaling. SMC with secret sharing is used to split and distribute the private metadata among multiple providers. Single providers then no longer have access to the user metadata and its associations in plain text. Still, the VoIP functionality can be preserved if multiple providers cooperate and perform SMC on the secret shared data. Due to legal reasons, absolute privacy in VoIP provider systems is not attainable. The presented SMC approach manages to harmonize privacy by default with controlled Lawful Interception. This thesis contributes a design of a VoIP signaling system with an SMC lookup table at its core which protects the user metadata of phone numbers and IP addresses. A corresponding provider application is implemented as a prototype in Java using the FRESCO SMC framework. Performance measurements show that the application of SMC is most realistic in low-latency environments, since the execution time scales linearly with latency. The evaluation results also suggest that a hashing approach is best suitable for the realization of an SMC lookup table, matching the scaling behavior of classic map data structures. Overall, the developed SMC VoIP signaling approach with improved privacy is able to preserve functionality and achieve realistic applicability if providers are willing to cooperate and the latency between providers is sufficiently low.

Zusammenfassung

Voice over IP (VoIP) Provider benötigen zur Erfüllung der VoIP-Signalisierungsfunktionalität Zugriff auf Nutzerdaten. Die Provider kennen momentan die Zusammenhänge zwischen Nutzeridentitäten, Telefonnummern und IP-Adressen, was eine signifikante Menge privatheitskritischer Daten darstellt. Die Privatheit der Nutzerdaten auf Providerseite lässt sich mit Hilfe von Secure Multiparty Computation (SMC) verbessern. SMC ist eine aufkommende kryptographische Technik, die es mehreren Parteien ermöglicht gemeinsam sichere Berechnungen auf privaten Daten durchzuführen. Diese Arbeit wendet SMC auf VoIP-Signalisierung an. SMC mit Secret-Sharing wird verwendet um private Metadaten aufzuteilen und auf mehrere Provider zu verteilen. Einzelne Provider haben dann keinen Zugriff mehr auf die Nutzermetadaten und deren Assoziationen im Klartext. Dennoch kann die VoIP-Funktionalität erhalten werden, indem mehrere Provider kooperieren und gemeinsam SMC auf den geteilten Daten ausführen. Aus rechtlichen Gründen ist eine absolute Privatheit in VoIP-Providersystemen nicht erreichbar. Der präsentierte SMC-Ansatz schafft es, Privatheit als Standard mit den rechtlichen Regeln zur Herausgabe von Nutzerdaten in Einklang zu bringen. Diese Arbeit trägt einen Entwurf für ein VoIP-Signalisierungssystem basierend auf einer SMC-Auflösungstabelle bei, das die Nutzermetadaten Telefonnummer und IP-Adresse schützt. Eine entsprechende Prototyp-Providerapplikation wurde in Java unter Verwendung des SMC-Frameworks FRESCO implementiert. Performanzmessungen zeigen, dass die Anwendung von SMC am realistischsten in Umgebungen mit geringer Latenz ist, da die Ausführungszeit linear mit der Latenz skaliert. Die Auswertung legt auch nahe, dass ein Hashing-Ansatz für die Realisierung einer SMC-Auflösungstabelle am besten geeignet ist, in Übereinstimmung mit dem Skalierungsverhalten klassischer assoziativer Datenstrukturen. Insgesamt ist der entwickelte SMC-VoIP-Signalisierungsansatz mit verbesserter Privatheit in der Lage, die Funktionalität zu erhalten und realistische Anwendbarkeit zu erreichen, falls die Provider gewillt sind zu kooperieren und die Latenz zwischen den Providern gering genug ist.

Acknowledgments

I would like to thank Prof. Dr.-Ing. Georg Carle for giving me the opportunity to write my thesis at the network chair and for providing me with an SMC topic. My gratitude also goes to my advisors Dr. Marcel von Maltitz and Richard von Seck, who always gave helpful suggestions and detailed feedback. Thanks to Marcel for introducing me to SMC and for continued inspiration and guidance even after finishing your doctorate. Also thanks to Richard for providing an additional perspective and for helping with some technicalities. I appreciate their efforts and their positive influence. I learned a lot with this thesis.

Contents

Chapter 1

Introduction

Voice over IP (VoIP) telephony providers are the central party of VoIP signaling. Providers act as mediators for clients and help them establish connections to other clients when they make phone calls. To perform this task, they need to know where the other clients are reachable. The providers hold these mappings between phone numbers and IP addresses in their location databases, which are queried each time a client requests the resolution of a target phone number to the corresponding IP address. The issue with this situation is that the providers have access to a significant amount of client metadata. A solution is desirable where provider knowledge of privacy-critical metadata is minimized while preserving the VoIP functionality at the same time.

The goal of this thesis is to improve the privacy of telephony client metadata by applying Secure Multiparty Computation (SMC) to VoIP signaling. SMC is an uprising and promising cryptographic technique suitable for building privacy-preserving services. It enables several parties to jointly perform computations on private data while keeping the privacy of the data intact and without having to depend on a trusted third party. Applied to an existing multiparty system with a desire for better privacy, this essentially allows to preserve the functionality of the system while improving privacy at the same time. Such an upgrade comes in general with the drawback of increased computation times due to the SMC overhead. The objective of this thesis is to protect the client metadata within the VoIP provider location database, consisting of phone numbers and IP addresses, with SMC.

The research questions to be addressed with this thesis are:

1. How can SMC be applied to VoIP signaling in order to improve metadata privacy?

2. How can Lawful Interception still be realized if data is privacy-protected?
3. What are the performance and scalability properties of such a VoIP solution?

In the context of this thesis, a fitting SMC solution is to be developed by designing an SMC VoIP provider system, implementing a prototype provider SMC application and evaluating its feasibility, performance and scalability.

The thesis is structured in the following way: First, background information is given in Chapter 2 about VoIP, SMC, a state-of-the-art SMC framework and Lawful Interception. Then, Chapter 3 analyzes the problem domain and presents main use cases and a list of functional and nonfunctional requirements. The analysis is followed by a design of a solution to the described problem with Chapter 4. The main architecture of the designed system is explained there as well the components and dynamic behavior. This design is then realized with a prototype SMC provider implementation, which is the topic of Chapter 5. Performance measurements were conducted with the prototype provider application and a test client in a testbed environment. This evaluation and its results are presented in Chapter 6. At the end of the thesis, related work and future work is outlined in Chapter 7 and 8 respectively before the conclusion is drawn in the last Chapter 9.

Chapter 2

Background

The next sections give an overview of the background on Voice over IP, Secure Multi-party Computation, the used SMC framework Fresco and Lawful Interception.

2.1 Voice over IP telephony

Voice over IP (VoIP) [10] is a technology stack for voice communication over the Internet. It features many extensions and related protocols. In the context of this thesis, the focus is only on the relevant central parts of VoIP.

The main actors of managed VoIP are clients and providers. The former group wants to use VoIP for telephony. The latter group is responsible for providing the VoIP telephony services. One of their tasks for example is to help clients at establishing connections to others.

Important VoIP protocols are the Session Initiation Protocol (SIP) [16] and the Real-time Transport Protocol (RTP) [17]. The signaling protocol SIP is used as control layer when making phone calls. RTP is the protocol for transmission of the actual audio data between clients.

Main SIP components are user agents, proxy servers and location and registrar services. User agents are the clients. The SIP proxies are the front-end servers of VoIP providers. Each provider manages a database containing associations between SIP URIs and IP addresses. This database can be queried with the location or registrar service for read or write requests respectively.

When initiating a call, the user agent of the caller sends an INVITE signal to the user agent of the callee via the SIP proxy. The provider then performs a lookup in the location database to find the target IP address, before forwarding the INVITE signal there. The client at the other end receives this call initiation message and can accept it to complete the telephony session establishment. Voice data is now transmitted directly between the clients using RTP.

Both SIP and RTP messages can be transmitted securely by using their encrypted protocol variants [20]. This protects the transfer of voice data for example. However, the metadata on provider-side is currently not protected: The location databases of providers contain the associations between phone numbers and IP addresses in plain text.

2.2 Secure Multiparty Computation

Secure Multiparty Computation (SMC) is a cryptographic technique which enables multiple parties to jointly compute a function in a secure way without needing to depend on a trusted third party.

The classic SMC example is the Millionaire's Problem of Yao [21]. Two millionaires want to find out who is richer without disclosing their actual wealth. In a mathematical sense, their goal is to calculate the inequality comparison function $\leq$. At the end of the function evaluation, both millionaires should only have learned the result of the comparison as a boolean value, but no other additional information.

More formally, a secure computation of a function $y = f(x_1, ..., x_n)$ with n parties, each inputting their secret x_i, has to fulfill two conditions [7]:

1. **Correctness:** The correct value of y is computed.
2. **Privacy:** y is the only new information that is released.

Figure 2.1 visualizes this concept. SMC does not require a trusted third party for the function computation. The computing parties also do not need to trust each other when using SMC and can keep their function inputs secret. In the SMC model of active security against a dishonest majority of malicious adversaries, which is used in this thesis, the security and privacy properties are still achieved if up to $n - 1$ parties are corrupted.

SMC can be realized with secret sharing [7]. Using the concept of m-out-of-m secret sharing, a secret value can be split up into m shares. Any strict subset of these m shares

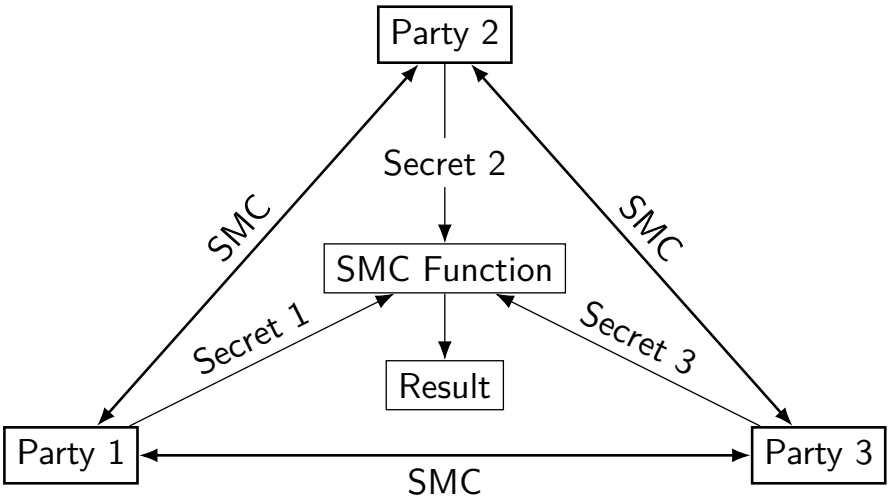

FIGURE 2.1: SMC concept

does not allow the reconstruction of the secret value. Only if all shares are available, this value can be revealed again by combining all m shares:

$$\text{SecretValue} = \text{Share } 1 \circ \text{Share } 2 \circ \ldots \circ \text{Share } m$$

Using SMC with secret sharing, the secure function computation is performed with the input values in a secret shared form. The secret inputs in Figure 2.1 would therefore first be split into shares, before they are used for the function evaluation.

To summarize, SMC in general addresses the security of multiparty computations and the privacy of input data. SMC fills the gap in the data security triangle [19]: It is already possible to secure data in transport, e.g. with TLS, and data at rest, e.g. with hard disk encryption. Now, SMC allows to secure data also during computation.

2.3 FRESCO

The *Framework for Efficient Secure Computation* ("FRESCO") [2] is an actively developed, open source state-of-the-art SMC framework. Implemented in Java and licensed under the MIT license, it can be used to realize SMC applications. FRESCO allows the specification of such an application independent of the used SMC protocol suite. This abstraction provides flexibility in the choice of SMC primitives. Protocol suites consist of a set of basic SMC operations like addition and multiplication. The framework user can then combine these operations to build more complex methods as FRESCO SMC applications. Currently, SPDZ [8] can be seen as the main protocol suite. FRESCO

not only offers the basic SPDZ operations such as addition or multiplication, but also more complex derived ones, e.g. modulo computation. Secret values within Fresco are represented by `SInt` objects, which contain an SMC secret share. Normal unprotected integer values can be *closed* and converted into `SInt` objects. Computations can then use the input secret shares to create intermediate ones and finally a result. At the end of a computation, this result value is then made visible to the parties by *opening* it.

2.4 Lawful Interception

In the context of VoIP Lawful Interception (LI), providers can be legally requested by Law Enforcement Agencies (LEAs) to selectively intercept communication and hand over user data and metadata.

As an example of concrete LI requirements, this section examines EU law and German law regarding requirements concerning the interception of user metadata.

On EU level, the "Council Resolution of 17 January 1995 on the lawful interception of telecommunications" [6] serves as main LI guideline document. These guidelines are standardized on a European level in the technical standard "ETSI Standard TS 101 331" [9]. National governments put the EU LI requirements into force by creating corresponding local laws, for example [4] in Germany.

As central requirement, LEAs can request VoIP providers to hand over metadata, also referred to as "call-associated data" or "intercept related information", of specific targeted identities. Such metadata includes identifiers, phone numbers, IP addresses, user action timestamps and location information, if available. Providers are also asked to intercept communication and associated data in a transparent way, which results in no visible service changes for the interception target. Additionally, standardized LI interfaces are required, which enable automatic LI request processing and serve as a gateway for the intercepted data.

The EU justifies the existence of LI because it is seen as "an important tool for the protection of national interest, in particular national security and the investigation of serious crime" [6].

Chapter 3

Analysis

The analysis chapter describes the setting, the attacker model and a solution sketch for an SMC VoIP signaling system, followed by a list of matching main use cases and requirements.

3.1 Setting

Starting point of the analysis is an abstracted state of the art of VoIP. The main actors are clients, providers and LEAs. These parties can interact with each other. The focus is on the signaling functionality and user metadata. The primary interaction is that clients can call other clients using the providers as mediators. In this VoIP signaling use case, the task of the provider is to resolve the phone number of the callee to the corresponding IP address by consulting the location database. In this lookup table data structure, providers save the associations between phone numbers and IP addresses of their users. The secondary interaction is LI, which can be requested by LEAs from providers, who then would have to hand over intercepted client data.

3.2 Attacker model

With this setting described in Section 3.1, the provider lookup table contains the associations between phone numbers and IP addresses in plain text. The provider therefore knows who owns which phone number and under what IP address it is reachable, as well as who calls whom. This metadata on provider-side is not privacy-protected, but

is privacy-critical. A passive attacker observing the provider system, and not the network, could therefore monitor and comprehend the links between user identities, phone numbers and IP addresses. The security goal of confidentiality is violated. Given an IP address for example, it is possible to find out the associated phone number(s), if any, and the owner's identity. Connections can also be made back in the other directions. Observing interactions between users and providers from within the provider system for a prolonged period of time would allow the passive attacker to build a social graph. This can be done by combining the monitored call metadata, more specifically phone numbers and identities of caller and callee. Additionally, the links between identities, phone numbers and IP addresses can be used to derive the users' geolocations. Such a malicious actor could abuse this discovered metadata. Because of that, it is desirable to improve the metadata privacy while preserving the basic VoIP functionality. As a side effect of privacy-protected user data, transparency and accountability of LI would be enhanced, if state actors would be more constrained on a technical level to use official LI interfaces instead of directly monitoring the provider system in a possibly unauthorized or non-transparent way.

Furthermore, phone numbers are currently owned by users for a relatively long time. This too can be seen as problematic, since the same personal phone number is often used and distributed to others for many different purposes and could be linked back accordingly. This leads to another desirable feature, which is to increase the purpose binding of phone numbers, fulfilling the privacy goal of unlinkability.

3.3 Solution sketch

The goal of this thesis is to protect the metadata regarding phone numbers and IP addresses from a passive attacker on provider-side. This could be achieved if the providers themselves have no direct knowledge of or access to this metadata. Here, SMC presents itself as a possible solution to this challenge. SMC allows multiple parties to securely operate on private data. Computations would be performed on secret shares instead of plain text data. Providers could therefore work together and replace the plain text lookup table data with secret shared data. Essentially, the data structure of the provider location database would become an SMC lookup table. One assumption SMC requires is that at most $n - 1$ out of n providers working together are compromised by the attacker. With such an approach, the metadata privacy of phone numbers and IP addresses in the lookup table can be improved. LI can still be supported, if LEAs request from every provider to hand over specific intercepted user metadata in a secret shared

form instead of in plain text. They can then combine the shares themselves to discover their real values.

Purpose binding of phone numbers can be increased, if they are treated as pseudonyms with user chosen lifetime and usage. Users should have the ability to create and manage multiple of such lifetime restricted pseudonyms. They can then use and distribute each of the pseudonyms context-dependently to others, providing flexibility and further improving the privacy.

3.4 Use cases

Use cases can be derived from the interactions between users, providers and LEAs. The primary use case is the signaling functionality. Users want to call other users. Providers help them at establishing a connection by resolving the target pseudonym, which represents the phone number, to the associated IP address as endpoint. On this metadata in the provider lookup table, SMC is applied to protect that data. Users should be able to manage their own pseudonyms in that SMC lookup table including the creation of new ones and updates to or deletions of existing ones. The secondary use case is LI. LEAs still require the feasibility of interceptions, even though the metadata is protected with SMC.

3.4.1 SMC Lookup

The providers jointly operate a distributed SMC lookup table. This is the central data structure of the provider location service. It contains the mappings between pseudonyms and IP addresses. The goal of a lookup is to find the table entry with the endpoint that corresponds to a particular pseudonym.

3.4.2 Pseudonym registration

Users can generate and register new pseudonyms. A registration of a pseudonym creates an entry in the lookup table of the providers, which associated the new pseudonym with an endpoint, a lifetime and extra data. All of these values are chosen by the user. A pseudonym can only be registered if it does not exist yet.

3.4.3 Pseudonym resolution

Users can request the resolution of pseudonyms to endpoints. When providers receive such a request, they resolve the pseudonym to the corresponding endpoint by consulting the lookup table. If a matching table entry has been found, the providers check that the lifetime of the pseudonym is not exceeded yet and then they return the endpoint to the user. A table entry with an expired pseudonym would be removed instead.

3.4.4 Pseudonym update or deletion

Users can update their pseudonyms, e.g. if the endpoint has changed, and can also delete their pseudonyms together with associated endpoints. Similar to pseudonym registrations or resolutions, this request is also processed with the help of a table lookup. Before corresponding entries in the lookup table are updated or deleted, the providers have to check if the user has the necessary authorization in order to assure that only the user who created a pseudonym can update or delete it.

3.4.5 Lawful Interception

LEAs can demand from the providers to hand over user metadata, which might be saved on provider-side in the lookup table or some log files. In addition to retroactive LI requests, where metadata about past user actions is queried, proactive LI requests are a second type of possible requests by LEAs, in which providers are asked to intercept specific user requests in real time and deal with them accordingly.

3.5 Requirements

The requirements of an SMC VoIP signaling system for improved privacy are described in the next subsections and are divided into functional, security and privacy, performance, and legal requirements.

3.5.1 Functional requirements

Requirements of Users

Users can register new pseudonyms, resolve any pseudonym, and update and delete their own pseudonyms. Every pseudonym is associated with an endpoint, a lifetime and some arbitrary extra data for extensibility.

Requirements of Providers

Providers can verify requests and messages either received from users, other providers or LEAs. Such a verification step can be used to analyze the validity of a request and to check if the entity the request was received from has the necessary authorization.

Providers can rate limit users and deny to process their requests. This might be necessary if a user makes too many requests per time in general, which could negatively impact provider availability, or if the providers detect spam calls and want to block them.

Providers can log user requests. Log entries can contain a timestamp, the request type and the user id, as well as the pseudonym and endpoint in some form. The log can then later be consulted in case of LI.

Requirements of Law Enforcement Agencies

LEAs can instruct providers to perform LI. This can include querying information from the provider log files and requesting a live interception of certain users.

3.5.2 Security and privacy requirements

The baseline security and privacy requirements are those of VoIP SIP. Some central requirements are now improved with an increased demand in privacy.

Confidentiality: Single providers have no knowledge of the actual values of the pseudonyms and endpoints that they process in their system, e.g. in their SMC lookup table or log files.

Integrity and Authenticity: All communication between provider and user, other providers or LEA is authenticated.

Controlled Access: Requests by users, providers and LEAs to a provider must be authorized. Users have to be signed in for pseudonym resolution or management requests. Pseudonym updates or deletions can only be done by the pseudonym owner, i.e. the user who registered it. Only providers are allowed to participate in SMC lookups. Providers have to check the authenticity and validity of LEA requests before answering to it.

Accountability: Users initiate requests to providers using their private sign in identifiers. These are then included in the log files of the providers, making requests traceable. The user's interactions with the providers can be traced back, by LEAs

for example, if the connection between user and identifier is uncovered and combined with information from the provider logs.

Availability: Providers are able to apply rate limiting to decrease negative availability impacts by single users. Additionally, redundant providers can be added for robustness.

Unlinkability: Links between identities, pseudonyms and endpoints should be hidden from single providers. Connections between two pseudonyms or two endpoints should also be unknown to single providers.

Transparency: The transparency requirements are similar to VoIP SIP. Users send their data to the providers and have no direct knowledge of how the providers use it. Transparent data usage insights are therefore not given from the user perspective. Another aspect of transparency is LI. Providers do know when and to what extend LEAs request interceptions.

Intervenability: Users have the ability to manage their own pseudonyms and have control over who can contact them by distributing the pseudonyms accordingly and by controlling the pseudonym lifetimes. Context dependent pseudonym usage also gives them control over how their different pseudonyms can be linked together.

Data Minimization: Single providers no longer have access to their users' pseudonyms and endpoints in plain text, while still being able to provide the required basic VoIP telephony functionality.

3.5.3 Performance requirements

In general, the performance compared to VoIP SIP is expected to be lower due to the performance overhead introduced by SMC.

Response time: For usability, user requests, especially lookup requests, should have a low response time. This is ideally less than a second, but a few seconds could still be tolerable. LEA requests on the other hand are allowed to have a longer response time. This however depends on the legal requirements, so that a few hours response time could be acceptable in some cases.

Scalability: The systems of the providers should scale sufficiently in terms of request handling throughput, total user count and total number of lookup table entries.

3.5.4 Legal requirements

Providers are legally required, to the extent of applicable law, to take measures regarding LI. Interception can be demanded by LEAs in a retroactive or proactive way. In the retroactive case, providers are asked to hand over certain user metadata which they keep in their databases or log files. In the proactive case, providers are given certain search criteria, that they apply on live requests in real time, in order to notify the LEA about these particular active user requests and eventually to take additional interception measures.

The LI search criteria and query types that the providers are required to support are as follows:

- **Timestamp range query:** Return all log entries or active events within a given time frame.
- **User request origin query:** Return a list of requests which have been or are being initiated by a given user.
- **Pseudonym query:** Return all log entries or active events associated with a given pseudonym.
- **Endpoint query:** Return all log entries or active events associated with a given endpoint.
- **Combined query:** Combination of any query type from above for a more narrow entry selection.

Chapter 4

Design

The objective of this chapter is to design an abstracted VoIP system using SMC. Similar to classic VoIP, the primary provider task is to manage a lookup table containing associations between pseudonyms and IP addresses, which are important for the signaling functionality. The second provider task is to enable LI. The design applies SMC on the lookup table in order to protect its content. Users should still be able to query the providers with pseudonym related requests such as resolutions. To realize this SMC lookup table, multiple providers have to work together. In classic VoIP, users would only directly communicate with their own provider. With the SMC provider system, users would instead interact with multiple providers simultaneously.

The next sections first give an overview over the architecture of the designed SMC VoIP system with the lookup table at its core, then describe important components and at the end explain the interactions and dynamic of the system.

4.1 Architecture

The actors of the designed system are clients, n providers and LEAs. In the context of this thesis, n is chosen to be three. This number could however also be selected higher in order to have more providers participate in the designed SMC system. Providers have two main components and offer three interfaces. The two components are the lookup table and the log. Providers interact with other actors via interfaces for provider-provider, provider-client and provider-LEA communication. Interactions of the first type between providers are necessary for SMC. The second kind of interaction between clients and providers are for pseudonym resolution or management requests which are

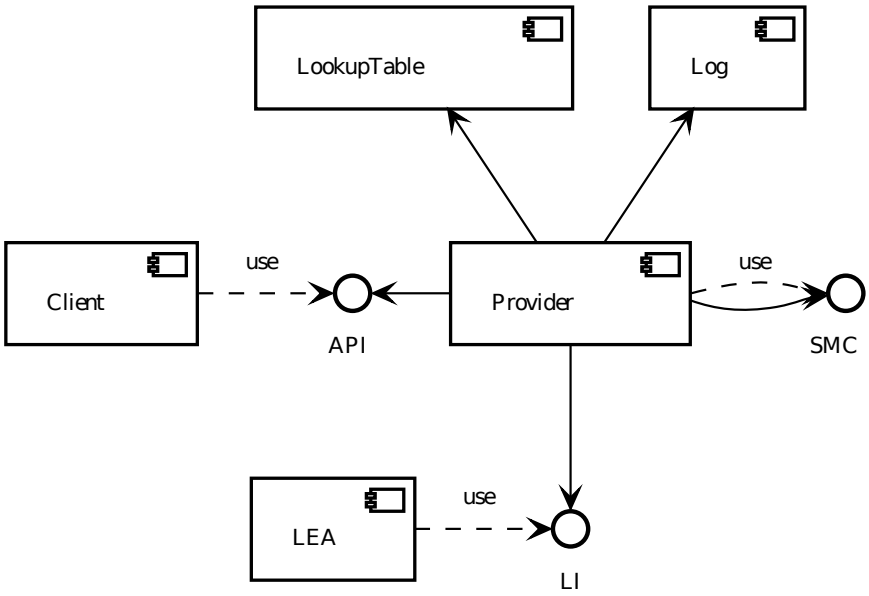

Figure 4.1: Provider components and interfaces

received by the providers via a dedicated API. The last of the three interfaces is the LI interface, where LEAs can request interceptions. Figure 4.1 contains a UML component diagram depicting the described actors, components and interfaces. Communication channels themselves are already assumed to be sufficiently secured, e.g. with TLS.

Each provider holds and manages a matching instance of the lookup table. This component is the central one of the whole system. It contains the mapping of pseudonyms to endpoints. The table is consulted by the providers if a corresponding request is received from a user. A lookup is performed in order to find the correct table entry, on which providers can follow up with read or write operations. If the request has been processed, a response is returned to the user indicating the success of the request and delivering requested data or other information.

Since pseudonyms and endpoints within the table should be protected from the providers for privacy reasons, they are not saved as plain text. Instead, the values in the lookup table that are to be kept private are secret shared among the providers. SMC can be jointly performed by the providers to still work with the secret values in useful ways. This is necessary for example in case of pseudonym user requests. Such a scenario is displayed in Figure 4.2: A user makes a pseudonym request to all three providers at the same time. Now the goal of the providers is to find the corresponding lookup table entry. This requires them to compare the pseudonym key of the request with the pseudonyms in the table. SMC between the providers is necessary at this point to search in the table while each provider only has access to their own secret share of each

pseudonym. The detailed notation of the lookup table entries in Figure 4.2 is as follows: Pseudonyms are denoted by $\Psi_{i,j}$ where i is owner of the pseudonym and j indicates the j-th pseudonym of this user. Similarly, endpoints are denoted by $E_{i,j}$. Values in square brackets represent SMC shares, e.g. $[\Psi_{A,1}]_k$, with k indicating the k-th share part.

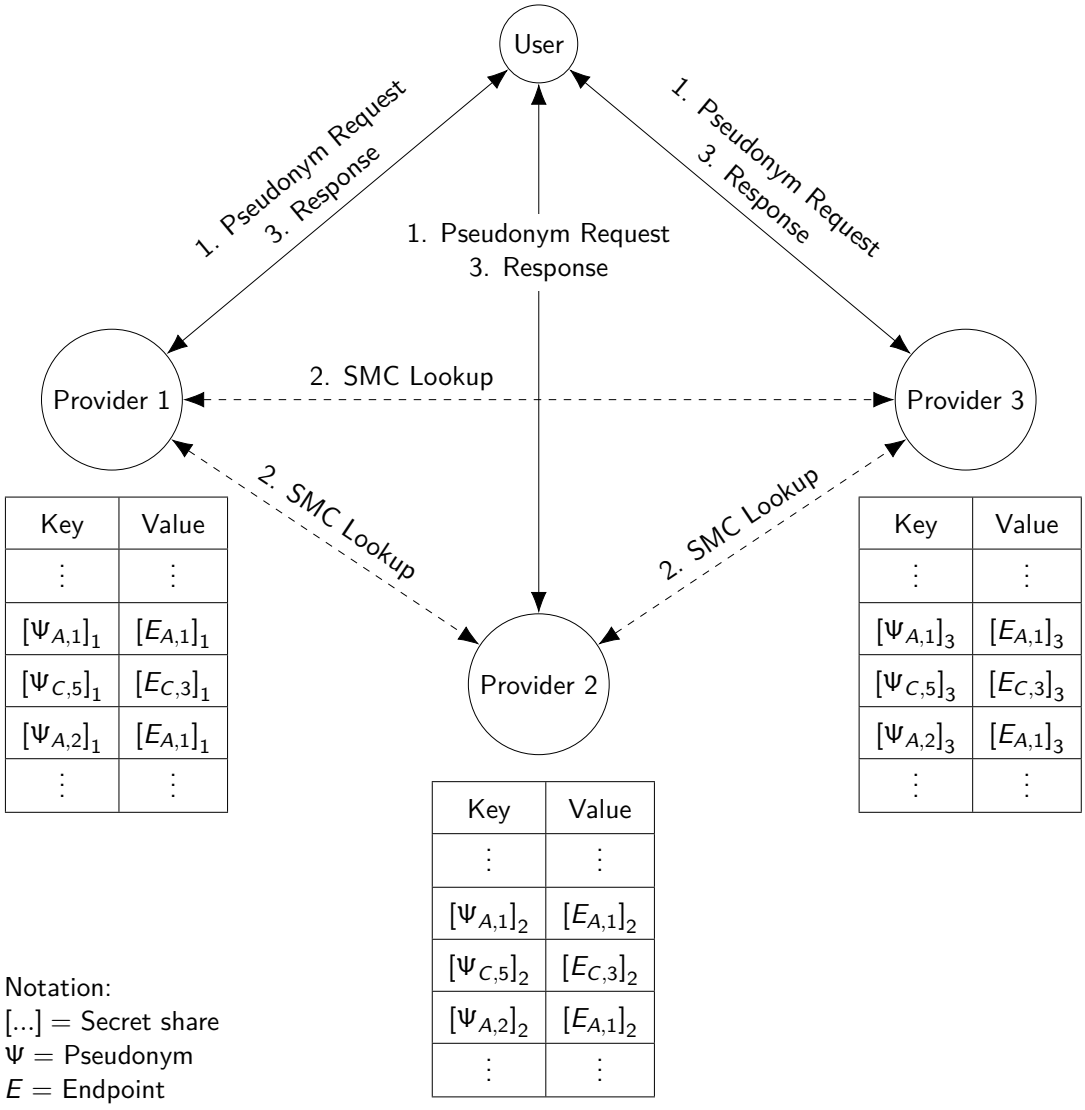

Figure 4.2: Central architecture

4.2 Lookup table data structure

The main data structure of the whole system is the SMC lookup table. It is a map data structure, where pseudonyms act as keys and values are a tuple of endpoint, lifetime

and extra data. The pseudonyms and endpoints are saved in the lookup table as secret shares. These shares are distributed among the providers. Every provider holds and manages a matching instance of the lookup table. Their internal structure is equal, but the secret shares as table contents are different for each provider. SMC comes into play when providers want to perform operations on the lookup table, such as insertions or lookups. An additional map property is support for concurrency. This is relevant for the scalability of the lookup table regarding request processing throughput. In general the table should be optimized for read operations as these are expected to occur more frequently than write operations.

Map data structures are usually realized with hashing for best performance. Whether this argument also holds when applying SMC is initially not clear, because SMC operations do not follow the execution time rules of normal operations. That is why for the design of the lookup table data structure three variants are proposed: Realizing the map as list, tree-like and with hashing. Each approach varies in performance, scalability and requirements regarding necessary SMC operations.

4.2.1 Map as list

Lists are rather simple data structures and generally offer linear time complexity $\mathcal{O}(n)$, since the list is traversed linearly. Yet, they can be used to realize a map if list entries are seen as key-value pairs. Lookups are then performed by searching the list front to back, making a key equality check at each entry. With the keys being pseudonyms and saved as shares, this equality check has to be an SMC equality check. Listing 4.1 shows this linear SMC search logic as pseudocode.

Listing 4.1: SMC Lookup (linear search)

```
# Input 1: pseudonym of type PseudonymShare
# Input 2: listMap of type List[Tuple[PseudonymShare, MapValue]]
# Output: Reference to the matching list entry
#         or None if the pseudonym is not in the list

for entry in listMap:
    if equalsSMC(pseudonym, entry[0]):
        return entry

return None
```

4.2.2 Map tree-like

Tree-like data structures exist in many forms. One reason for preferring to use such data structures over lists is the logarithmic time complexity $\mathcal{O}(\log n)$ of tree operations.

Listing 4.2: SMC Lookup (binary search)

```
# Input 1: pseudonym of type PseudonymShare
# Input 2: sortedListMap of type SortedList[Tuple[PseudonymShare, MapValue]]
# Output: Reference to the matching sorted list entry
#         or None if the pseudonym is not in the sorted list

leftIndex = 0
rightIndex = len(sortedListMap) - 1

while leftIndex <= rightIndex:
    midIndex = (rightIndex + leftIndex) // 2

    compareResult = compareSMC(pseudonym, sortedListMap[midIndex][0])

    if compareResult == 0:
        return sortedListMap[midIndex]
    elif compareResult < 0:
        rightIndex = midIndex - 1
    else:  # compareResult > 0
        leftIndex = midIndex + 1

return None
```

When traversing a tree-like map, comparisons are performed at each step down the tree to decide whether to continue in one subtree or the other. Since these comparisons are done on the pseudonym keys, this operation has to be an SMC comparison. In Listing 4.2, binary search on a sorted list is used to demonstrate with pseudocode how such an SMC comparison can be applied in the context of a tree-like map.

4.2.3 Map with hashing

When performance is the important factor, maps are typically realized with hashing. This would require the availability of a fitting hash function, in this case an SMC hash function, which might be more complex to realize than an equality or comparison function. The best-case time complexity of map access with hashing is $\mathcal{O}(1)$, while the average time complexity is $\mathcal{O}(1+c)$ with c being the overhead caused by extra operations within a hash bucket in case there is more than one element in the bucket. These extra operations could be equality checks, if hashing with chaining is used. In such a hash map setup, lookups would first hash the requested key, which is a pseudonym, and then perform equality checks until the corresponding table entry has been found. Here, two SMC operations are required: One for hashing and one for equality checks. Listing 4.3 shows the pseudocode of an SMC lookup where hashing with chaining is the chosen hash map type. The linear search part there within a chain is equivalent to the map as list approach above.

Listing 4.3: SMC Lookup (hashing)

```
# Input 1: pseudonym of type PseudonymShare
# Input 2: hashMap of type List[List[Tuple[PseudonymShare, MapValue]]]
# Output: Reference to the matching hash map entry
#         or None if the pseudonym is not in the hash map

pseudonymHash = hashSMC(pseudonym)
hashIndex = pseudonymHash % len(hashMap)
chain = hashMap[hashIndex]

for entry in chain:
    entryPseudonym = entry[0]
    if equalsSMC(pseudonym, entryPseudonym):
        return entry

return None
```

4.2.4 Table entry format

The lookup table maps pseudonyms to table values. Such a table value consists of an endpoint, a lifetime and extra data. All of these values are specified by the user. Pseudonyms and endpoints are represented as integers for flexibility. The lifetime determines how long the pseudonym and the table entry as a whole is valid. If this expiration time has been reached, the entry can be removed by the providers and read requests to the pseudonym would result in an error. The extra data field acts as a placeholder and allows future extensibility of table entries. Extra data can be treated as a user chosen string of limited size. Users can choose to make extra data available to single providers in plain or to multiple providers in a secret shared form. Depending on the context, it could be reasonable to partially return extra data in pseudonym resolution requests made by other users. Less limitations on the size of the extra data string lead to higher memory and bandwidth requirements, especially on provider-side.

4.3 Provider log

To cover the use case of retroactive LI requests, each provider maintains a log which contains information about pseudonym related user requests. It is sorted by request time and the entries consist of the pseudonym request type, the identifier of the user who created the request as request origin, the pseudonym share and the endpoint share. Depending on the request type, the shares are either specified by the user or retrieved by the provider in the process of a lookup. Table 4.1 illustrates this log structure with example entries. New log entries are created by providers for each successful pseudonym related user request. The log can be queried by LEAs by sending corresponding LI

Timestamp	Request Type	Origin	Pseudonym Share	Endpoint Share
1562765976	Ψ Registration	User45	$[\Psi_{\text{User45},1}]_1$	$[E_{\text{User45},1}]_1$
1562766001	Ψ Registration	User62	$[\Psi_{\text{User62},2}]_1$	$[E_{\text{User62},2}]_1$
1562766090	Ψ Resolution	User45	$[\Psi_{\text{User62},2}]_1$	$[E_{\text{User62},2}]_1$
1562766259	Ψ Update	User78	$[\Psi_{\text{User78},1}]_1$	$[E_{\text{User78},1}]_1$
1562774593	Ψ Deletion	User23	$[\Psi_{\text{User23},3}]_1$	$[E_{\text{User23},3}]_1$

Table 4.1: Example log of provider 1

requests to providers. The sorting by time makes the log by default easily searchable by time. In case the log is needed to be searchable efficiently by other columns as well, it is possible to create and maintain additional search indexes, e.g. for the origin, pseudonym share or endpoint share column. Such share indexes could require the use of SMC, similar to the provider main pseudonym SMC lookup table. Search queries to the log for specific pseudonyms or endpoints would then have to be resolved with SMC operations. The main SMC lookup table data structure could be reused here to realize search indexes for pseudonyms and endpoints. Instead of mapping to endpoints, the log SMC table index could map to a list of associated log entries.

4.4 Provider behavior

Providers interact with other providers, users and LEAs. For each of these actors providers offer one interface. These three interfaces are explained in the next three subsections.

4.4.1 SMC interaction

In order to operate on the lookup table, providers have to cooperate and jointly perform SMC. While doing so, they have to make sure that their lookup tables stay in a matching consistent state. Otherwise, the type or number of SMC operations might vary among the providers when accessing the table, causing SMC errors. Therefore, the providers should coordinate lookup table operations and associated SMC executions.

Synchronization is required in two ways: Multiple parallel threads and SMC sessions on each provider for scalability need to be synchronized as well as the order of operations across all three providers for table consistency. Before any SMC method is called, a

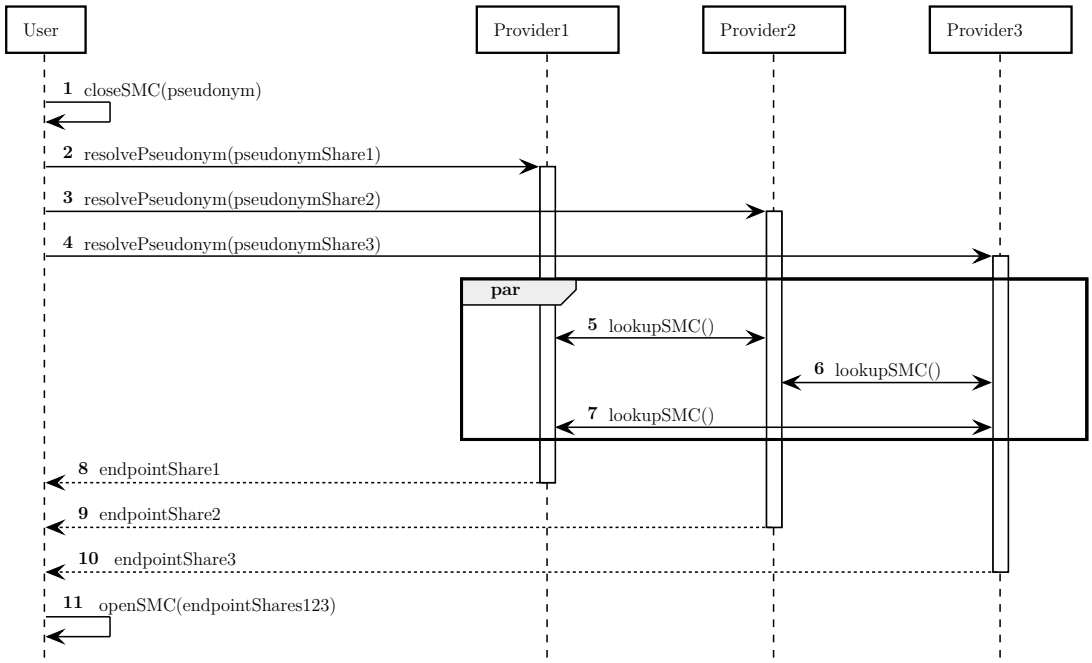

Figure 4.3: Pseudonym resolution

communication round takes place between the providers that prepares a coordinated execution. The order of operations can for example be determined by comparing the average request timestamps of the three providers.

4.4.2 Pseudonym requests

Providers offer an API which users can utilize for pseudonym related requests. These can be pseudonym registrations, resolutions, updates or deletions. If a user wants to create a request, the first step is to close any secret pseudonym and endpoint values of this request locally into secret shares. The user can then send one copy of the request to each provider with the exception of the shares which are unique for each request. The providers now have to perform an SMC lookup in their table to be able to read from or write to the requested table entry. After that, each provider returns a response to the user indicating success or failure and including requested data or other information. In case of a pseudonym resolution, the endpoint shares of the looked up table entry are returned by the providers. The user can then open these shares to discover the actual endpoint value. Figure 4.3 shows the pseudonym resolution process in a UML sequence diagram. Registrations, updates and deletions are done in a similar way. Depending on the request type, the user would include the desired new table value in the request.

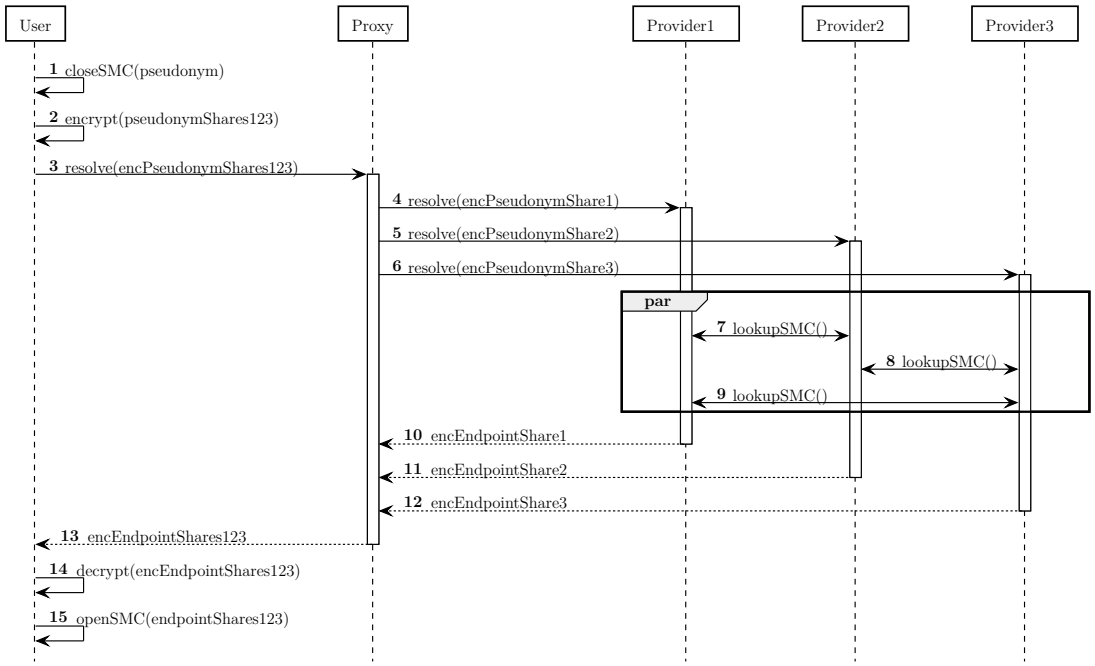

Figure 4.4: Pseudonym resolution with proxy

As an extension or alternative, a proxy or gateway between the user and the providers could be added to act as a mediator to simplify communication effort on user side. The proxy could collect messages from users and distribute them correctly among the providers. Responses from the providers would then be collected and sent back to the user. This would decrease the amount of messages users send or receive and save bandwidth. Additionally, the user would only have to communicate with the proxy instead of all three providers. The user could therefore outsource knowledge about where providers are located on the Internet to the proxy. Better administration in complex user networks with firewalls and other middleboxes could be achieved as well. Encryption can be used to protect critical message contents such as secret shares against the proxy. In theory, a proxy is not essential for a working system and could also be realized on user side. It complicates the system by being an additional party and could become a performance bottleneck or single point of failure. Nevertheless, a sequence diagram of the proxy concept is displayed in Figure 4.4.

Pseudonym updates and deletions require the requesting user to be authorized. Only the owner of a pseudonym can update or delete the corresponding table entry. In this context, authentication is also necessary to confirm the user identity. The design however does not elaborate on the details of how a secure authentication can be realized, but assumes it as given.

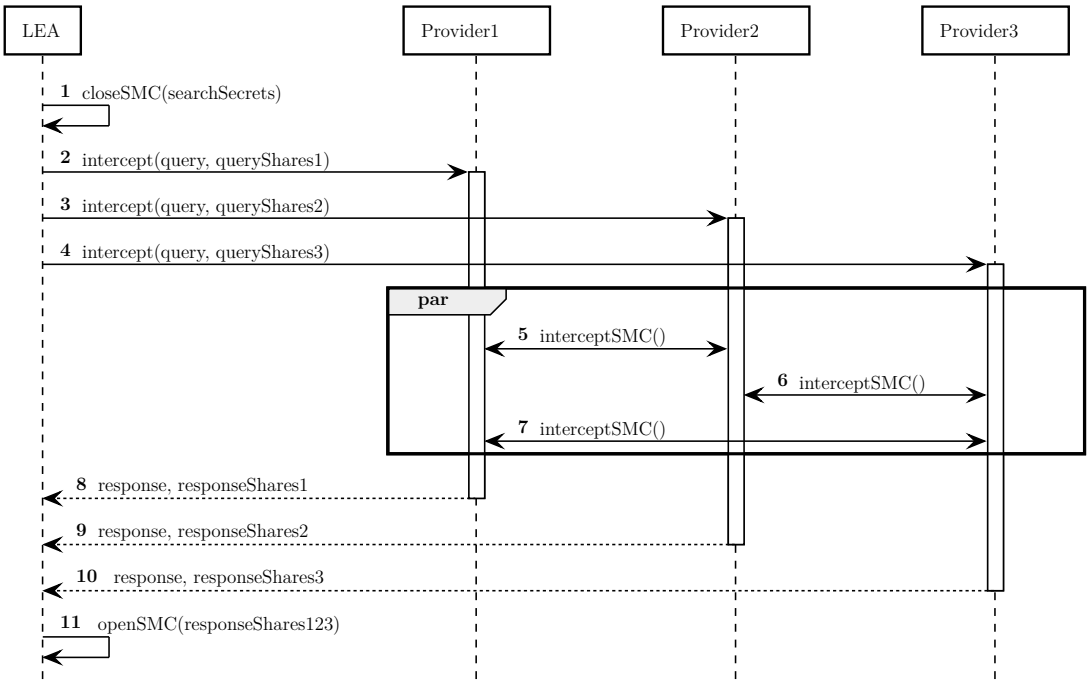

Figure 4.5: Lawful Interception

4.4.3 Lawful Interception requests

The third interface providers offer is the LI interface. This can be used by LEAs to request interceptions. Such requests contain search terms as query parameters which are used to specify the interception target. Search terms can be time ranges, user identifiers, pseudonyms and endpoints.

There are two types of LI requests: Proactive and retroactive. In the proactive interception case, providers are instructed to monitor or intercept active user requests and future provider interactions by specific users. Providers can realize this by checking the list of interception targets each time a new user request comes in to decide whether this special attention is required. In the retroactive interception case, LEAs request certain lookup table or log entries. Providers then have to find the entries corresponding to the given search terms.

If LEAs specify pseudonyms or endpoints as search terms, they do so in a secret shared way using SMC. In such a case, the providers would then also have to employ SMC either on active incoming user requests, the lookup table or the log in order to determine if a request, table entry or log entry matches the interception search terms. The request flow of this scenario is shown as a UML sequence diagram in Figure 4.5.

Chapter 5

Implementation

A prototype provider application, matching the designed components, is implemented using Java as programming language. Each provider can then run this application on own hardware. The SMC lookup table is realized using the SMC framework Fresco. A REST API allows users to send pseudonym related requests to providers. As a distributed system, synchronization and parallel execution play an important role. Some of the implemented components use multi-threading, for example the REST API, others are simplified to single-threading such as the lookup table, but kept extensible. Next to synchronization of methods within one provider, some operations also require coordination by all three providers. This kind of synchronization is also realized. The whole provider implementation consists of approximately 3300 lines of code.

5.1 SMC with FRESCO

The SMC functionality is implemented using the Fresco [2] framework in its current version 1.2.0. An overview of this framework is given in the background chapter in section 2.3. The chosen protocol suite is SPDZ, which is secure against malicious adversaries in a dishonest majority setting. SPDZ features an offline preprocessing phase and an online evaluation phase. Fresco provides by default a dummy preprocessing strategy with less security than theoretically achievable, which is seen as a tolerable limitation for a prototype.

The implemented class `SMCSession` encapsulates the initialization of Fresco and acts as SMC facade front, serving as entry point for calls to SMC operations from the normal provider code. It decouples SMC and Fresco from the other components, enabling

Operation	**Input**	**Output**
Close	Int	SMCInt
Open	SMCInt	Int
Equals	2 SMCInt	Boolean
Compare	2 SMCInt	Int -1/0/1
Hash	SMCInt	Int

Table 5.1: Implemented Fresco SMC applications

future modifications of the SMC back end without having to change the other provider components. In the setup phase, all providers connect to each other via dedicated and known ports for a new SMC session. Once the Fresco engine has started and connectivity is established, providers can execute SMC operations. Within one session, which only ends if the providers shutdown, multiple such executions can take place, but in a sequential way. Parallel SMC executions are possible in theory, if providers just start extra SMC sessions with different ports, but the implementation omitted this for simplification.

Secret shares are represented by `SMCInt` objects. Each of these SMC integers encapsulates a Fresco `SInt` object and therefore acts as a wrapper. The `SMCInt` class overrides the special Java methods `equals()`, `compareTo()` and `hashCode()` with corresponding calls to SMC operations. Together, this abstracts the secret shares and the SMC calls away from users of the class. This is especially useful in the lookup table data structure implementation, where `SMCInt` can directly be used as normal Java map key object.

Five SMC operations are implemented as Fresco SMC applications: Close, Open, Equals, Compare and Hash. SMC applications are composed of basic native Fresco SMC protocols. An overview of the function inputs and outputs is given in Table 5.1. The *Close* and *Open* operation are used to convert normal integers to secret ones and vice versa. *Equals* checks two SMC integers for equality and returns true or false accordingly. *Compare* acts similar to classic integer comparison functions, returning -1 if the first `SMCInt` is lesser than the second one, 0 if both values are equal and 1 otherwise. The *Hash* operation calculates a 30-bit hash of a secret value.

While the first four operation types utilize basic Fresco SMC primitives, the hash function is a bit more complex. Given a pseudonym as key, hashing is used, together with a subsequent modulo table size operation, to calculate the corresponding bucket respectively table index in the hash map. The hash function has to be consistent and

always return the same value for each key. From a security and privacy perspective, the hash value should not allow any significant conclusions about the secret pseudonym integer input. If this is not considered, the hash function could for example be chosen to just return the lowest n bit of the secret value. This would leak these lowest bits of the pseudonym, which could be tolerable if the total bit length is much higher. On the other end of the security spectrum, a cryptographic SMC hash function could be implemented. This would even provide collision resistance, but would also make the calculation much more expensive and slower. Since collision resistance is not an essential requirement in this SMC hashing context, the implemented hash function is chosen to be a one-way hash function, but not a cryptographic hash function. Literature describes for example the Rabin function as a one-way function [11]:

$$Rabin(x) = x^2 \bmod N$$

Here, N is a product of two prime numbers. Similar to RSA, security of the Rabin function relates to the difficulty of prime factorization: "[E]xtracting square roots modulo N is computationally equivalent to factoring N". Because this function offers the one-way property and is still rather simple, it is chosen as candidate for the SMC hash function. The prime numbers and the divisor N are fixed to constant values in the implementation for simplification, but could also be variables and dependent on the secret integer value or derived otherwise. This implemented *Hash* operation takes an `SMCInt` as input and returns a corresponding hash as normal integer.

The execution of each of the five implemented SMC operations described above can be started by calling the corresponding methods of the `SMCSession`. To make this process simple, the session is made available everywhere in the provider code by introducing the `SMCService`. Its `static` methods can be reached globally within the provider application and just forward the SMC calls to the `SMCSession`. This service is mainly used by `SMCInt` objects.

Important to remember is that SMC operations are always executed by all three providers simultaneously. Secret integers are represented by three `SMCInt` objects, distributed among the providers. Synchronized SMC executions have to be coordinated by the providers. How this coordination is implemented is explained later.

Associated Java files: `SMCService.java`, `SMCSession.java`, `SMCInt.java`, `EqualsApplication.java`, `CompareApplication.java`, `HashApplication.java`, `CloseApplication.java`, `OpenApplication.java`

Method	Arguments	Returned value
Put	SMCInt, TableValue	Boolean
Get	SMCInt	TableValue
Update	SMCInt, TableValue	TableValue
Delete	SMCInt	TableValue

Table 5.2: Implemented lookup table methods

5.2 Lookup table

The `LookupTable` is a data structure that maps pseudonym keys of type `SMCInt` to values of type `TableValue`. These value objects contain the associated endpoint as `SMCInt`, the expiration time, the extra data string and the user identifier of the pseudonym owner, which is used by the providers to authorize write requests on existing entries.

Four methods are defined in the `LookupTable`: Put, Get, Update and Delete. Table 5.2 gives a summary of the arguments and returned values of the four implemented lookup table methods. They follow classic calling convention rules. *Put* creates a new table entry for the given combination pseudonym and table value, if the pseudonym does not exist yet. *Get* looks up the value of a given pseudonym. *Update* overwrites an existing table entry with a new value and returns the old value. *Delete* removes a pseudonym from the table and returns the deleted table value.

The SMC lookup table is implemented in three variants. However, none of the table implementations actually need to or do understand the concept of SMC. They also do not directly start SMC operations or are aware of their executions. Instead, this functionality is hidden in the `SMCInt` objects, more precisely in their `equals()`, `compareTo()` and `hashCode()` methods. The `LookupTable` can therefore treat the keys of type `SMCInt` in the same way as non-SMC objects that implement these three methods. This indirection and separation between the lookup table, the SMC code and Fresco is shown in Figure 5.1.

The first variant is the `ListLookupTable`. It is based on the Java `ArrayList`. The list entries are tuples of the `SMCInt` pseudonym and the corresponding table value. When accessing this list map with either of the four table methods, the array list is searched linearly. For each entry, the `equals()` method of the `SMCInt` search key is called to

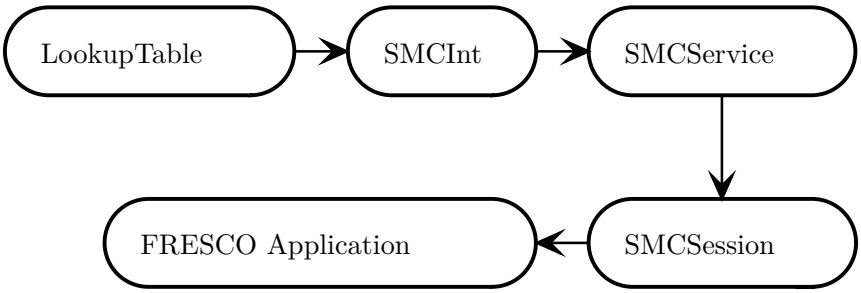

Figure 5.1: Call flow from the lookup table to FRESCO

check whether a matching entry has been found, which in turn triggers for each entry the execution of the corresponding SMC *Equals* operation.

The other two variants are the `TreeLookupTable` and `HashLookupTable`. Both directly utilize the existing Java map implementations `TreeMap` and `HashMap`. `SMCInt` can immediately be used here as key type without requiring any modification of the Java map classes. Internally, the methods `equals()`, `compareTo()` and `hashCode()` are called again by the map and the `SMCInt` then launches the correct SMC operation.

Computation-complexity-wise, the number of `equals()`, `compareTo()` and `hashCode()` calls depend on the used table variant and the current number of table elements n. Map access therefore has time complexity $\mathcal{O}(n)$ in the list map case, $\mathcal{O}(\log n)$ in the tree map case and $\mathcal{O}(1 + c)$ in the hash map case. Another performance influencing factor is the overhead introduced by the three different SMC operations.

Similar to the `SMCSession`, of which the providers currently only have one for simplicity, concurrency is also omitted in the three lookup table implementations. A multi-threaded table version is possible in theory, but would require a fitting synchronization process to make sure the provider tables do not deviate from each other internally. Randomness should also be avoided or used with care within a table implementation since it could also cause divergence. In general, the providers have to coordinate executions of the four lookup table operations to keep their tables in a matching consistent state.

Providers call lookup table methods if they receive pseudonym requests by clients and want to read from or write to the lookup table.

Associated Java files: `LookupTableService.java`, `LookupTable.java`, `ListLookupTable.java`, `TreeLookupTable.java`, `HashLookupTable.java`, `TableValue.java`

5.3 REST API

Clients can make lookup table related requests to providers through their API. The request types include pseudonym resolution, registration, update and deletion. The API that providers offer for this purpose is a REST API. The corresponding request methods are as follows:

- Pseudonym resolution: GET */api/pseudonyms/{pseudonymShare}*
- Pseudonym registration: POST */api/pseudonyms/{pseudonymShare}*
- Pseudonym update: PUT */api/pseudonyms/{pseudonymShare}*
- Pseudonym deletion: DELETE */api/pseudonyms/{pseudonymShare}*

Lookup table accesses are done by the providers using SMC. For this, matching SMC operations have to be executed in parallel by all providers. Before they can start resolving client requests in this way, the clients first have to send matching requests to all providers. The implementation therefore requires that each provider has to be queried by the client at roughly the same time for a valid pseudonym related request. The time window for request processing is chosen to be 30 seconds long. The timer on each provider starts after receiving the user request. If the request could not be processed within the time span, for example because one provider has not received a matching request yet, a timeout is triggered and the request is rejected. If however all three providers have received a matching request, they can together start to resolve it. Responses are then also created by each provider and sent back to the client.

All client requests also contain the user identifier as query parameter. In this context and as a prototype simplification, user ids emulate client authentication. The user id serves multiple purposes on provider-side. It is for example used by the providers to check whether a client has the necessary authorization for a request. This makes sure clients can only update or delete their own lookup table entries. Additionally, a rate limiting system could be based on the user identifiers, which has however not been implemented here.

While all four requests carry pseudonym shares and user identifiers in the URL, not every request and response contains an additional payload. Of the client requests, only pseudonym registrations and updates include the new or updated table value as JSON string. The endpoint share, expiration time and extra data are all part of the table value. Of the provider responses, only the pseudonym resolution returns the corresponding endpoint as a payload. An example pseudonym registration request is

shown in Listing 5.1. The share values are abbreviated there as they usually consist of about 450 characters each. Pseudonym resolutions do not carry a table value in the request, as displayed in Listing 5.2.

Listing 5.1: Example pseudonym registration request

```
POST http://localhost:8081/api/pseudonyms/114903048...006083527?user-id=1
{
    "endpointShare": "126226288...06083527",
    "expirationTime": "2020-09-06T09:12:18.707Z",
    "extraData": "test"
}
```

Listing 5.2: Example pseudonym resolution request

```
GET http://localhost:8081/api/pseudonyms/36767358481...9006083527?user-id=1
```

Associated Java files: `RestServer.java`, `RestAPI.java`, `PseudonymRequestHandler.java`

5.4 Coordinated provider actions

One task of the providers is to keep the internal structure of their lookup table in a matching and corresponding state compared to the table structure of the other providers. That means, every logical table entry has the same table index on each provider. Starting from such a consistent state, providers have to coordinate table operations to preserve the consistency, while considering that the pseudonym requests triggering these table operations are received by the providers in an asynchronous and unordered way. Each table access should behave similar on all three providers and call the same internal map methods. This includes the SMC operations which are carried out when traversing the lookup table to search for correct indexes. These SMC executions have to be coordinated by the providers. If the lookup tables are in a consistent state, each provider wants to execute the same number and type of SMC operations in a matching order. If this is not the case and the tables have diverged, one provider for example might start one extra SMC function while the others already found the correct index, resulting in an error situation.

The providers therefore have to coordinate the order in which they handle the user pseudonym requests, since these cause lookup table accesses. Before even being able to process such a user request, each provider needs to have received a matching request part. That means, providers have to communicate to each other which requests they

have received in order to find out the set of matching request triples that they then can choose an execution order for and after that start with the actual table accesses.

One simple coordination approach is to let each provider forward a user request to the other providers as soon as it is received. This would allow them to notice for each of their own user requests whether the other providers also received a matching request. What is not achieved by this approach is a correct table access order, since coordination messages have no clear order themselves and messages between two providers might get lost or delayed without the third provider noticing in time, causing providers to start table operations corresponding to different user requests.

For this reason, the coordination solution is implemented differently. Instead of immediately forwarding received user requests to the other providers, they are collected on each provider and then exchanged as batches after a certain interval. This is chosen to be 50 milliseconds long. Smaller intervals would increase network usage because of a higher rate of coordination message exchanges. The interval also influences the response time of pseudonym related user requests, since incoming user messages first have to wait for coordination before they are processed. Each of these batch exchange iterations increments a coordination round counter, which defines a clear batch order. After each exchange, the three batches are searched for matching user requests by comparing the user identifiers and request types contained in the request batches. These can then be sorted on all three providers by an average provider timestamp, resulting in a total order guaranteed to be the same on each provider. The ordered requests are finally scheduled for sequential detailed processing regarding lookup table access in the just calculated order.

All of this coordination functionality is encapsulated in the `SyncService`. It is called by the provider code that handles REST API user requests whenever the lookup table is about to be accessed. The processing time window of 30 seconds mentioned in the REST API section includes the coordination time. Providers cancel user requests after this time span, resulting in the user receiving an error message. This can for example happen if two providers received a matching request, but the third one did not.

Associated Java files: `SyncService.java`, `PseudonymRequestHandler.java`

5.5 Logging and LI

Each provider keeps a log of successful pseudonym user requests. LI rules might also require the logging of unsuccessful requests, but the focus is on successful requests here.

Log entries contain a timestamp, the pseudonym request type, the user identifier as request origin, the pseudonym share and the endpoint share. The log is implemented as a simple list. As mentioned in the design chapter already, this could be optimized though by using a more efficient data structure. Additionally, search indexes could be created for each entry column, which could even be realized by reusing the SMC lookup table data structure.

Retroactive LI is supported by the provider log. It can be queried for specific entries by a time range, request origin, pseudonym or endpoint. The list of log entries is then searched sequentially for matches. In case the LI request contains a pseudonym or endpoint as search term, the providers perform SMC equality checks using the secret shares provided by the LEA and the ones in the log in order to find the target entries.

Proactive LI could be implemented in a similar way. Instead of consulting the log, the providers would check incoming client requests for LI significance and act accordingly before processing the request normally. An according implementation has been omitted however for simplification.

Associated Java files: `LogService.java`, `Log.java`, `LogEntry.java`

5.6 Client secret sharing helper

Clients send and receive pseudonyms and endpoints to and from providers in a secret shared form. A client application would therefore require methods to convert normal values to secret shares and vice versa. These shares have to be formatted in such a way that they are understood by the providers, suggesting the use of the Fresco secret share format. To make a matching value-share-conversion available on client side, the *close* and *open* functionality of Fresco could be ported to the client application or the client could run a Fresco instance themself just for this purpose. To simplify the development of a test client application, the share conversion functionality is outsourced to the providers instead. The REST API is extended by a *close* and an *open* method:

- Close integer: GET */api/helper/close/{integer}*
- Open integer: GET */api/helper/open/{integerShare}*

Similar to pseudonym requests, all three providers would be queried at the same time to prompt them to execute the underlying SMC *close* or *open* application. However, clients should be aware of the privacy implications when using these two methods.

Providers have to be trusted to generate correct shares and to keep the privacy of the values intact. For testing purposes, this approach is acceptable though.

Associated Java files: `ClientHelper.java`, `RestAPI.java`

Chapter 6

Evaluation

The provider prototype implementation is tested for performance in a testbed environment using a small web client implemented for testing purposes. The goal of the measurements is to find out if and which scenarios deliver realistic execution times.

6.1 Setup

Performance tests are done with four parties in a testbed environment: Three hosts are used to act as providers and a fourth host is used as client. The client will later send test requests to the providers, that they then have to answer. The hardware properties of the host computers are as follows:

CPU: Intel(R) Xeon(R) CPU D-1518 @ 2.20GHz with 4 cores and 8 threads

RAM: 32 GiB

Network: 1 Gbit/s

All four hosts have the same operating system installed. Additionally, the providers have Java installed to be able to run the provider application software and the client requires Node.js so it can run the JavaScript client application. The detailed software properties are as follows:

OS: Linux 4.9.0-9-amd64 #1 SMP Debian (Stretch) 4.9.168-1+deb9u5 x86_64

Java: OpenJDK 1.8.0_222

Node.js: v10.16.3

6.2 Methodology

The goal of the performance tests is to find out how the table size, table type and network latency impact the execution time on provider-side of the SMC operations Equals, Compare and Hash, and the execution time on client side of pseudonym requests of different type.

Tests are executed with four parameters, which are constant throughout a test run, but vary over multiple runs:

Number of requests per pseudonym request type: Between 10 and 100
(equals maximum table size)

Repetitions: Between 1 and 3
(specifies number of test logic repetitions within a test run)

Table type: List, Tree or Hash

Network latency [ms]: 0.1, 1, 5, 10, 15, 20, 25

The request and repetition count was decreased as the latency increased for test execution time reasons which will become more clear when looking at the measurement results. The concrete chosen test parameters are listed in Table 6.1.

Latency [ms]	0.1	1	5	10	15	20	25
Requests per type	100	100	25	10	10	10	10
Repetitions	3	3	3	3	1	1	1

Table 6.1: Number of requests per pseudonym request type and number of repetitions by latency

The test logic incorporates a warm-up phase and a test phase. In both phases, pseudonym requests of all four types are generated by the client and sent to the providers. The requests are sent out sequentially, i.e. the client waits for a response before sending the next request. The warm-up phase lets the Java just-in-time compiler optimize the important code paths, assuring the first few measurements are comparable to the others. In the test phase, the client measures the response time for each request and the providers measure the duration of each SMC operation they execute. Within one test repetition, n requests of each pseudonym request type are made. The requests of different type are connected to each other by the user id which is contained in the request. n pseudonym registrations with an increasing user id are followed by n resolutions and n updates with the same increasing user ids. This means for example that the first pseu-

donym that is resolved is also the first pseudonym that got registered. The deletions however are done in reverse, i.e. the last registered pseudonym is deleted first and so on. This makes sure that delete measurements involving a list allow comparable scaling observations, which would not be the case when always removing the first element. The test logic steps are therefore:

1. Start providers
2. Warm-up phase (5 requests per request type, total 20)
3. Make n registrations (increasing user id)
4. Make n resolutions (increasing user id)
5. Make n updates (increasing user id)
6. Make n deletions (decreasing user id)
7. Repeat step 3 to 6 for each additional repetition

6.3 Test web client

A simple client application was implemented whose main purpose is to create test requests against the REST APIs of the providers. It consists of approximately 400 lines of code. As a web client, the chosen programming language is JavaScript, which allows it to be run either in a web browser like Firefox or with Node.js.

The client implementation supports the creation of pseudonym requests of type registration, resolution, update and deletion. New pseudonyms and endpoints can also be generated from client chosen values by querying the client secret sharing helper. In the same way, provider responses can be interpreted if they contain secret shared values. Additionally, the client offers an automatic generation of many requests, which is relevant for the performance measurements.

Associated JavaScript file: `main.js`

6.4 Measurement results

The first important result to note is the influence of latency on the execution time of SMC operations. Figure 6.1 pictures this relation for each of the three measured SMC operations *Equals*, *Compare* and *Hash*. The width of every box plot correlates

to the amount of sample data for the given latency. It is apparent that the execution time increases linearly as the latency increases. A doubled latency approximately also doubles the execution time.

The seemingly large variation of the *Compare* method can be explained by looking at the number of executed native SMC protocols. They are displayed in Figure 6.2. The native protocols are the basic Fresco SPDZ operations that are executed. These are for example additions and multiplications. More complex operations are combinations of multiple native protocols. The statistics for the *Compare* method show that its execution can end after two distinct numbers of native protocols. The lower point is reached if the result of the comparison returns 0, which means both values are equal. In this case, the *Compare* method has the same number of native protocol executions as the *Equals* method. The upper point on the other hand is reached if both values are not the same. Calculating which value is greater or smaller than the other is apparently much more computationally expensive.

The second main result are the absolute execution times of single SMC methods. Figure 6.1 already portrayed this. *Equals* operations take some tenths of a second to complete, in low-latency situations even only two-digit millisecond values. *Compare* executions take significantly longer in the non-equal case with a few seconds, in low-latency cases under one second. *Hash* computation times are similar to the ones of the *Compare* method.

The third important result is the answer to the question whether the SMC lookup table with its three types does actually scale as the table type suggests. This is indeed the case. The scaling of the lookup table when using the list map approach is linear and logarithmic for the tree map. The hash map has a constant scaling with additional chaining overhead sometimes in case of collisions. All three scalings are, among other things, visualized in Figure 6.3, 6.4 and 6.5 respectively. The three referenced figures show the pseudonym request timings for each table type and for each request type of the measurements taken with latency 1 ms. Every plot contains 300 data points, since 100 requests per type were repeated 3 times with latency 1 ms.

The fourth main result are the absolute execution times of pseudonym related lookup table requests. With a maximum table size of 100 and a latency of 1 ms, durations of single requests stay within a few seconds in the list map case of Figure 6.3, but grow linearly. Response times with a tree map vary depending on the request type. This can be seen in Figure 6.4, where read-only operations take only about half as long on average as write operations. Similar to the list map, the absolute execution times are in the area of a couple of seconds. However, the growth is logarithmic this time. By far

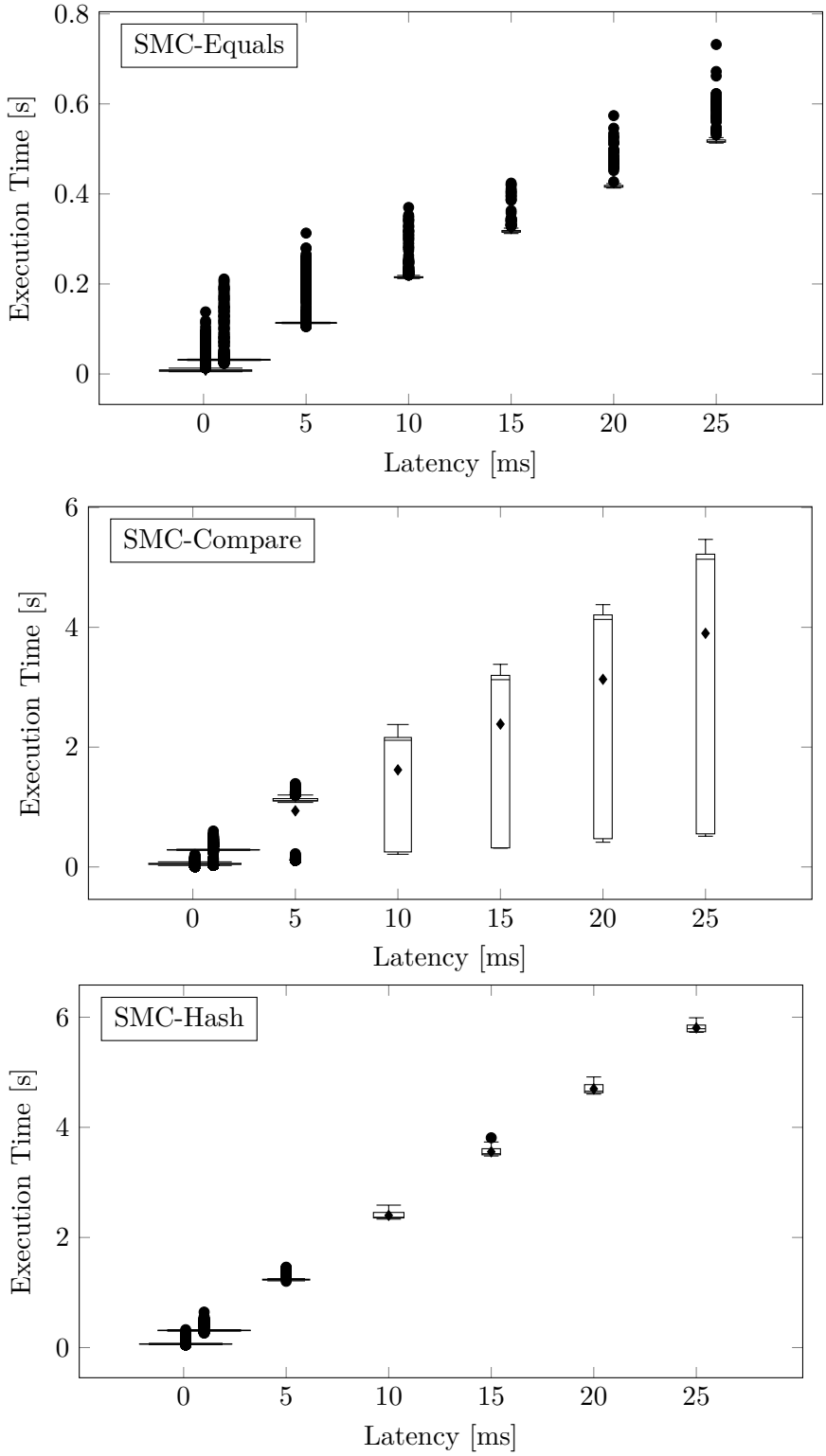

Figure 6.1: SMC method timings by latency

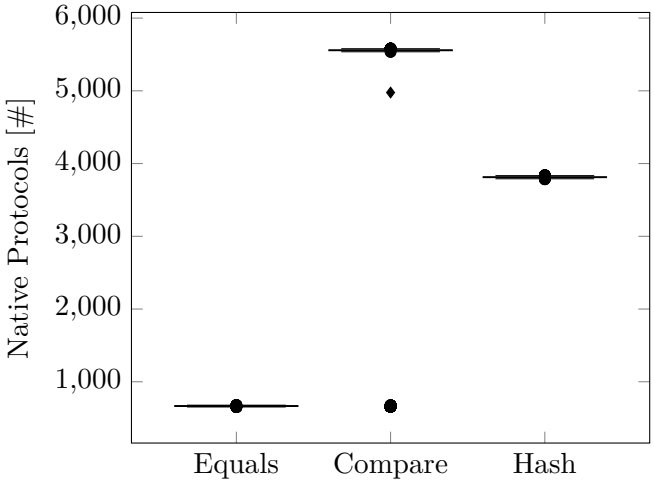

Figure 6.2: Native protocols by SMC method

the best results delivers the hash map approach, displayed in Figure 6.5. Here, requests take consistently under one second to complete, on average about 0.4 seconds, given a latency of 1 ms. Hash bucket collisions can also be seen, which result in additional *Equals* operations, causing outliers with increased execution time in Figure 6.5. The constant scaling and the low absolute execution time makes the hash map approach seem to be the best one for the realization of the SMC lookup table.

To summarize, low-latency scenarios combined with the usage of a hash lookup table deliver the best and even realistic response times regarding pseudonym related user requests.

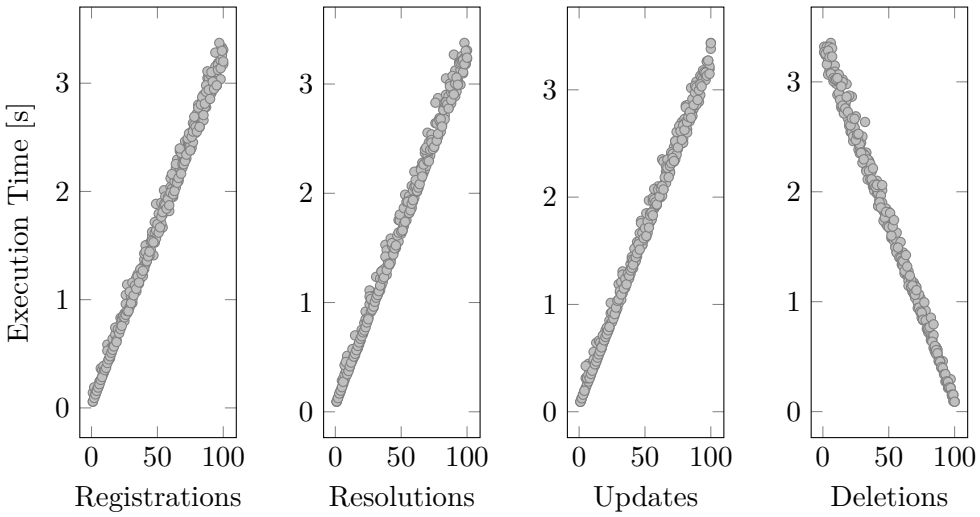

Figure 6.3: Pseudonym request timings (table type list, latency 1 ms)

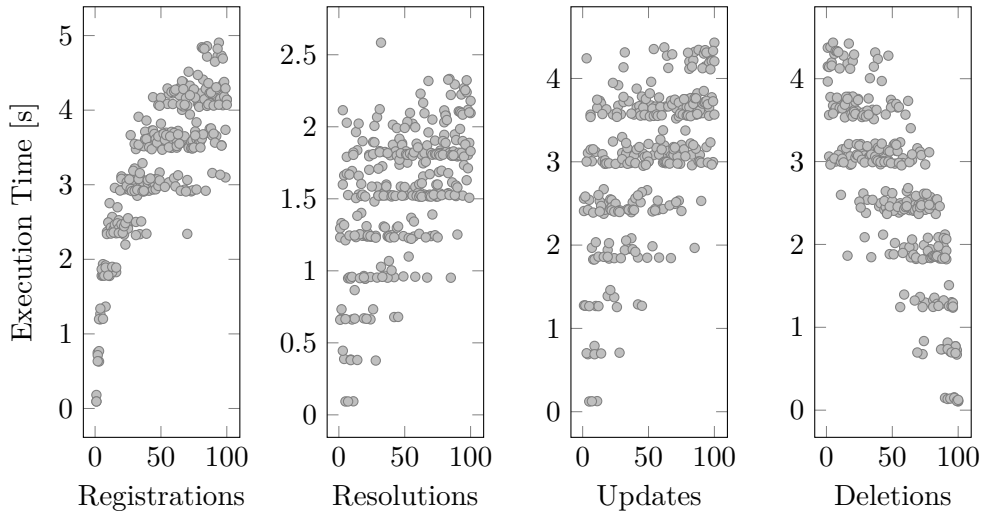

Figure 6.4: Pseudonym request timings (table type tree, latency 1 ms)

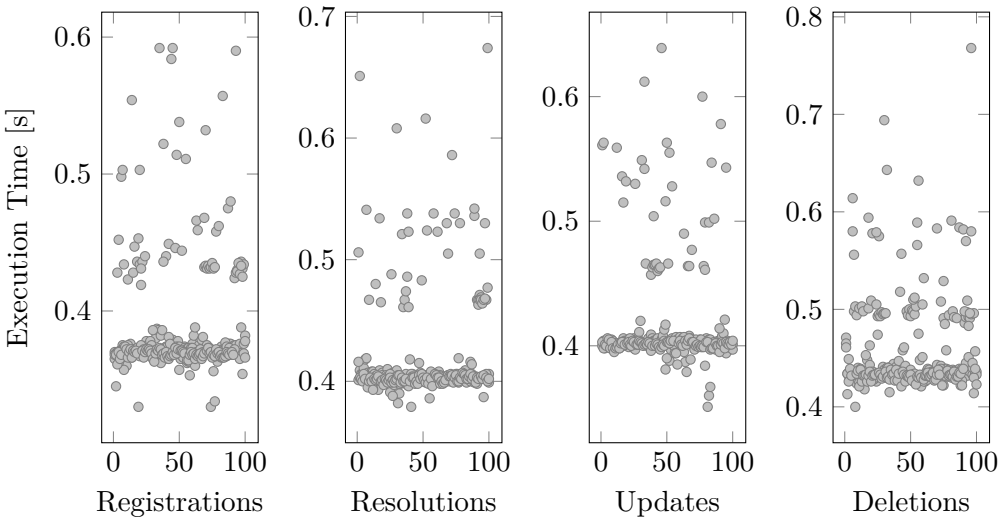

Figure 6.5: Pseudonym request timings (table type hash, latency 1 ms)

Chapter 7

Related work

This chapter presents existing work about related SMC applications and other solution approaches to some sub-problems encountered in this thesis.

An application of SMC in the area of VoIP has already been explored in 2015 by Archer and Rohloff [3]. Their goal was to provide audio data security in the sense of end-to-end encryption. For this, they created a prototype VoIP teleconferencing system using SMC based on linear secret sharing and homomorphic encryption. They did however not address the problem of metadata privacy in the signaling phase.

Next to client audio data and the metadata of phone numbers and IP addresses in the lookup table of the VoIP provider, metadata relating to the user identity can also be protected using SMC. In the Master's Thesis "Anonymous Authentication Using Secure Multi-Party Computations" written by Ahmad in 2011 [1], an SMC authentication process is developed which allows users to authenticate themselves at servers without actually revealing their identity simultaneously. A dedicated registrar service decouples the initial user identification from subsequent authentications at other services. That SMC authentication technique could be used to extend the designed and implemented VoIP provider system in this thesis with the goal of protecting the user identities themselves against passive observers on provider-side.

Another approach for protecting data in a lookup table is to use a map data structure with the *oblivious* property [15, 18]. Obliviousness would hide the access pattern of map operations and could be useful to limit the knowledge of providers about which table entries have been accessed by which users. The system design given in this thesis does not hide this information from providers. Instead, the contents of the lookup table

themselves are protected with SMC. It might be possible to create an oblivious SMC lookup table that protects both the table content and the access pattern.

In a similar direction go [12] and [14], who use table lookups with secret index values to realize SMC implementations for AES.

The problem of coordinating lookup table operations on provider-side also relates to the Byzantine fault tolerance problem [13]. The providers want to achieve a consensus about the order in which they execute the lookup table operations. This interactive consistency problem might also be addressed using a solution to the Byzantine fault problem, for example the "Practical Byzantine Fault Tolerance" algorithm [5], instead of the simpler approach of exchanging coordination messages time interval based, that the provider implementation uses.

Chapter 8

Future work

Extensions and variations of the designed and implemented system are imaginable in multiple ways.

On the design level, the number of in SMC participating providers could be varied for example. A set of backup providers could additionally be introduced to act as a fallback in case one provider is temporarily unavailable, which would increase the system robustness. Here, t-out-of-n secret sharing might be useful.

If the metadata on provider-side should not only be protected against passive attackers but also active ones, some sort of verifiable secret sharing could be applied in combination with public key infrastructure. This would allow callers to verify the integrity and authenticity of shares returned by providers in case of pseudonym resolutions.

Another idea is to let clients participate in SMC executions. The advantages and disadvantages of such an approach would have to be examined for this.

On the implementation level, performance of the implemented provider SMC application could be optimized. Full multi-threading would for example be a desirable feature for throughput scalability. Another aspect is the coordination process between providers. The used consensus algorithm could be replaced by a more efficient one.

The prototype provider implementation can also be extended in various ways to increase its realism regarding security. Transport encryption and authentication mechanisms have been assumed, but were not practically realized. The dummy preprocessing strategy of the used SMC protocol suite SPDZ can also be replaced with a real one.

Chapter 9

Conclusion

VoIP signaling operates on a significant amount of privacy-critical metadata. This thesis manages to improve the privacy of user metadata by applying SMC to VoIP signaling. The privacy goals of data minimization and unlinkability are achievable, while preserving the functionality. The presented SMC VoIP provider system protects privacy against a passive attacker monitoring up to $n-1$ out of n providers. LI has also been addressed in order to fulfill legal requirements and give LEAs the opportunity to request interceptions, despite the better protected user metadata.

On a more technical level, an SMC lookup table has been realized as central provider location service data structure. Secret sharing allowed the private table contents to be distributed among multiple providers. Functionality can be preserved if the providers cooperate and perform SMC on the secret shared user metadata. The developed design has been implemented as a prototype in Java using the Fresco SMC framework and tested in a testbed environment.

The results of the performance evaluation show that especially low-latency environments deliver realistic execution times, since SMC computations scaled linearly with latency. An SMC hash map approach delivered the best scaling and the best absolute execution times. In a setting with a latency of 1 ms for example durations of less than one second are achievable.

SMC is a promising privacy-preserving technique. It is a solution to the trade-off between data utilization and data privacy protection. Research is ongoing to improve SMC performance and thus increase its practical applicability. Many existing and future systems could benefit from SMC.

List of Acronyms

LEA	Law Enforcement Agency
LI	Lawful Interception
RTP	Real-time Transport Protocol
SIP	Session Initiation Protocol
SMC	Secure Multiparty Computation
VoIP	Voice over IP

List of Figures

List of Tables

Bibliography

[1] Maqsood Ahmad. "Anonymous Authentication Using Secure Multi-Party Computations". MA thesis. Norwegian University of Science and Technology and KTH Royal Institute of Technology, 2011.

[2] Alexandra Institute. *FRESCO - A FRamework for Efficient Secure COmputation*. `https://github.com/aicis/fresco`. [Accessed: 2019-06-10]. 2019.

[3] David W. Archer and Kurt Rohloff. "Computing with Data Privacy: Steps toward Realization". In: *IEEE Security Privacy* 13.1 (2015), pp. 22–29. ISSN: 1540-7993. DOI: 10.1109/MSP.2015.3.

[4] Bundesministerium der Justiz und für Verbraucherschutz und Bundesamt für Justiz. *Verordnung über die technische und organisatorische Umsetzung von Maßnahmen zur Überwachung der Telekommunikation (TKÜV)*. `https://www.gesetze-im-internet.de/tk_v_2005/index.html`. [Accessed: 2019-08-08]. 2017.

[5] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Transactions on Computer Systems* 20.4 (Nov. 2002), pp. 398–461. ISSN: 0734-2071. DOI: 10.1145/571637.571640.

[6] Council of the European Union. *Council Resolution of 17 January 1995 on the lawful interception of telecommunications (96/C 329/01)*. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:31996G1104`. [Accessed: 2019-08-07]. 1996.

[7] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. DOI: 10.1017/CBO9781107337756.

[8] Ivan Damgård et al. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits". In: *Computer Security - ESORICS 2013*. Ed. by Jason Crampton, Sushil Jajodia, and Keith Mayes. Vol. 8134. Lecture Notes in Computer Science. Springer, 2013, pp. 1–18.

[9] European Telecommunications Standards Institute. *TS 101 331 - V1.5.1*. https://www.etsi.org/deliver/etsi_ts/101300_101399/101331/01.05.01_60/ts_101331v010501p.pdf. [Accessed: 2019-08-07]. 2017.

[10] William A. Flanagan. *VoIP and Unified Communications: Internet Telephony and the Future Voice Network*. Wiley, 2011. ISBN: 9781118166048.

[11] Oded Goldreich. *Foundations of Cryptography*. Vol. 1. Cambridge University Press, 2001. DOI: 10.1017/CBO9780511546891.

[12] Marcel Keller et al. "Faster Secure Multi-party Computation of AES and DES Using Lookup Tables". In: *Applied Cryptography and Network Security*. Springer International Publishing, 2017, pp. 229–249. ISBN: 978-3-319-61204-1.

[13] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems* 4 (1982), pp. 382–401.

[14] John Launchbury et al. "Efficient Lookup-Table Protocol in Secure Multiparty Computation". In: *ACM SIGPLAN Notices* 47 (2012). DOI: 10.1145/2364527.2364556.

[15] Daniel S. Roche, Adam Aviv, and Seung Geol Choi. "A Practical Oblivious Map Data Structure with Secure Deletion and History Independence". In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 178–197. DOI: 10.1109/SP.2016.19.

[16] Jonathan Rosenberg et al. *SIP: Session Initiation Protocol*. RFC 3261. https://www.rfc-editor.org/rfc/rfc3261.txt. RFC Editor, 2002.

[17] Henning Schulzrinne et al. *RTP: A Transport Protocol for Real-Time Applications*. STD 64. https://www.rfc-editor.org/rfc/rfc3550.txt. RFC Editor, 2003.

[18] Xiao Shaun Wang et al. "Oblivious Data Structures". In: *Proceedings of the ACM Conference on Computer and Communications Security*. 2014, pp. 215–226. DOI: 10.1145/2660267.2660314.

[19] Nigel Smart. *Computing on Encrypted Data: How to do the impossible*. KU Leuven ESAT Inaugural Lectures. 2019.

[20] Peter Thermos. *Securing VoIP networks*. Ed. by Ari Takanen. Safari Books Online. Upper Saddle River, N.J.: Addison-Wesley, 2007. ISBN: 9780321437341 - 0321437349.

[21] Andrew C. Yao. "Protocols for secure computations". In: *23rd Annual Symposium on Foundations of Computer Science*. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.