Technische Universität München

Department of Informatics

Master's Thesis in Informatics

Flexible and Robust Orchestration of Secure Multi-Party Computation for Privacy-Preserving Services in Dynamic Environments

Stefan Smarzly

Technische Universität München

Department of Informatics

Master's Thesis in Informatics

Flexible and Robust Orchestration of Secure Multi-Party Computation for Privacy-Preserving Services in Dynamic Environments

Flexible und robuste Orchestrierung von Secure Multi-Party Computation für privatheitsschützende Dienste in dynamischen Umgebungen

Author	Stefan Smarzly
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Marcel von Maltitz, M. Sc., Dr. Holger Kinkelin
Submission Date	March 15, 2017

Informatik VIII
Chair of Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, March 15, 2017

Signature

Abstract

In recent years, sensor devices became ubiquitous in almost all areas of life. Whereas sensors are installed in public infrastructures and commercial buildings, an increasing number of applications turn mobile phones into sensors, taking advantage of their manifold sensing capabilities, to collect a huge amount of data about their environment. This makes people a living part of Smart Environments, that in turn enables services for the well-being of individuals, efficient resource management or safety, to name a few.

In practice, much more data tends to be collected on centralized systems than usually is needed to realize stated services. This results in an inherent risk for individuals' privacy if personally identifiable information is not protected thoroughly or used adequately.

In this thesis, we identify Secure Multi-Party Computation (SMC) to be a suitable technology for enforcing individuals' privacy and control over data. Even though real-world applications exist, practical applicability of current SMC frameworks is too complex due to high administrative efforts and their inability to cope with the flexible and dynamic nature of things. At last, this leads to bad acceptance. To solve these issues, we design and implement a flexible orchestration framework that strives for general applicability of SMC in practice while preserving privacy. Our performance evaluation shows that the impact of our orchestration framework is minimal and that overall results are promising from a dependent service's perspective.

Zusammenfassung

In den letzten Jahren haben Sensoren in fast allen Bereichen des Lebens Einzug gefunden. Während Sensoren einerseits in öffentlichen Einrichtungen und kommerziell genutzten Gebäuden präsent sind, benutzen andererseits immer mehr Anwendungen Smartphones als Sensoren und machen sich deren vielfältige Wahrnehmungsfähigkeiten zu Nutze, um große Mengen an Daten über ihre Umwelt zu erfassen. Auf diese Weise werden Menschen zu einem lebendigen Teil eines intelligenten Umfelds, was im Umkehrschluss Dienste ermöglicht, die sich zum Beispiel um das Wohlergehen der Individuen kümmern, Ressourcen effizient verwalten oder Sicherheit gewährleisten.

In der Praxis werden weit mehr Daten in zentralen Systemen gesammelt als eigentlich für die Erbringung genannter Dienste notwendig wäre. Dies hat ungeahnte Risiken für die Privatsphäre der Individuen zur Folge, falls Persönlichkeit identifizierenden Merkmale nicht umfassend geschützt oder angemessen verarbeitet werden.

In dieser Masterarbeit identifizieren wir Secure Multi-Party Computation (SMC) als eine adäquate Technologie, um für Individuen Privatsphäre und Kontrolle über ihre Daten zu garantieren. Obwohl Anwendungen dieser Technologie im Alltag bereits existieren, ist deren praktische Umsetzung mit verfügbaren SMC Frameworks zu komplex, da der administrative Aufwand hoch ist und die Frameworks unfähig sind, den flexiblen und dynamischen Anforderungen standzuhalten. Zuletzt ergibt sich damit die schlechte Akzeptanz der Technologie. Um den Problemen Herr zu werden, konzipieren und implementieren wir in diesem Rahmen ein flexibles Orchestrierungs-Framework, das nach genereller Anwendbarkeit von SMC unter Wahrung von Privatsphäre strebt. Die Performance Evaluierungen zeigen, dass der Einfluss unseres Orchestrierungs-Frameworks minimal ist und die allumfassenden Ergebnisse aus der Perspektive eines darauf aufbauenden Dienstes vielversprechend erscheinen.

Contents

List of Figures

List of Tables

Acronyms

BGW	Ben-Or, Goldwasser and Wigderson.
HbC	Honest-but-Curious.
HTTP/2	Hypertext Transfer Protocol Version 2.
HVAC	heating, ventilation and air conditioning.
ICMP	Internet Control Message Protocol.
IoT	Internet of Things.
LSS	Linear Secret Sharing.
PKI	public key infrastructure.
RPC	remote procedure call.
RTT	round-trip time.
SMC	Secure Multi-Party Computation.
SSS	Shamir's Secret Sharing.
TOFU	trust on first use.
TTP	trusted third party.

Chapter 1

Introduction

Within commercially used buildings, an increasing amount of sensors measure in-depth aspects of their physical environment. These are properties such as temperature, lighting conditions, air quality or presence of occupants. Based upon the collected sensor data, a Smart Building can provide a variety of services for the well-being of the occupants, resource management in an efficient way, and safety [1]. Not restricting our view to building equipped with sensors, most mobile devices are powerful sources of data as well, being carried around by many humans all day long while perceiving their environment with integrated sensors. A typical use-case could be that temperature readings, provided by each office or devices at a certain location, enable a central heating system to adapt its output to the needs. Energy savings can be even optimized by taking into account the number of occupants and the distribution of people in a building. In another use-case, aggregated presence data stating the frequency of spaces being occupied, could be visualized on public displays, e.g. to inform people when it is appropriate to visit a cafeteria.

Typically, all kinds of sensor data tend to be collected on centralized systems that are servers in the basement or online cloud solutions [2]. We believe that this centralism implies an inherent risk of sensitive data being disclosed or used inappropriately. Even more, by collecting all kind of data in raw form intentionally and unintentionally, processing and analysis is not bound to a certain purpose. Thereby, apart from the intended services, data gathering poses the risk of personal identifiable information being extracted without consent or control by the data origin. Individual insights even increase with knowledge about organization and structure within or across buildings and their environments. For example, monitoring the time of absence of employees or correlating their path when moving around, threatens their privacy. It might provide (false) indications to a person's health, productivity or habits. In fact, surveys among European citizen clearly show that the majority fears the collection of private information being used for other purposes as stated [3] [4]. Beside the personal threat, abusing data is often forbidden by law, e.g. the European data protection law [5, Sec. 2].

Many service do not act upon data of individuals, but on derived information. Hence, collecting individuals' data is often only a pure excuse for services to function, but related to the technical realization of information collection in reality. One promising field to target the fundamental issue of privacy-preservation for individuals is Secure Multi-Party Computation (SMC). SMC encompasses diverse approaches and protocols to jointly compute a function without revealing any party's data. As the approaches are quite diverse, they are suited differently for given premises and requirements. In our case, we focus on more generic SMC protocols and frameworks that support an arbitrary construction of arithmetic operations running on more than two parties. These properties create a good base to cover a wide range of services for Smart Environments, but are not restricted to them.

Fundamental work has been done in the past creating SMC frameworks and optimizing their protocols to render them efficiently and hence, making them considerable for real-world applications (see chapter 8). Consequently, SMC technologies are not a deal breaker in terms of protocol performance anymore. However, a huge downside to SMC-based applications is their lack of all-embracing structures that target ease of deployment and usability for long living services outside of well-controlled or static environments [6]. In this thesis, we aim for a widely generic approach to tackle these shortcomings on the one hand, and improve individuals' privacy on the other hand.

1.1 Objectives

The goal of this Master's Thesis is to elaborate shortcomings of SMC frameworks in their practical applicability, while always keeping an eye on privacy considerations and improvements. Based on these findings, we design and create an orthogonal framework to operate distributed SMC nodes in an autonomous way. Being orchestrated automatically on behalf of a simple query interface, the system aims for facilitating deployment and rendering it suitable for Smart Environments and beyond that through minimizing administrative efforts in the long term. Noticeably, our framework enhances the capabilities of existing SMC frameworks in a decoupled way, with the general aim to reuse existing technologies and to value its exchangeability as much as possible. A final evaluation analyzes the performance of the orchestration and SMC layer, using Fresco [7] as exemplary SMC provider, separately.

1.2 Research Questions

As follows, we define a set of research questions that will provide guidance by tackling relevant aspects to achieve the overall goal of this thesis.

1. How to protect raw data when collaboratively computing derived information?
2. Which issues and challenges arise from choosing Secure Multi-Party Computation as basis for privacy preserving services?
3. How to practically realize a flexible and robust aggregation infrastructure with Secure Multi-Party Computation at its core?
4. How is the performance impact of the provided orchestration framework on SMC-based computations? How is the overall performance of such a realized system from a service client's perspective?

1.3 Outline

The rest of this thesis is organized as follows:

Chapter 2 provides background information about privacy considerations and goals for preserving individuals' privacy, Secure Multi-Party Computation and the employed framework, being an exemplary representative for secure computations in this thesis.

Chapter 3 presents an in-depth analysis of typical use-cases, their privacy issues, and how to generalize them into a simple model in order to verify whether SMC can be applied practically. Then, we depict shortcomings in its practical deployment and with respect to privacy, finally being summarized in a list of requirements needed for a practical, privacy preserving architecture.

Chapter 4 describes the system design for operating and orchestrating distributed sensor nodes in a mostly automated way, in order to involve them easily for privacy preserving services.

Chapter 5 illustrates the implementation from an architectural perspective, albeit presenting all important components to realize the proposed design in practice.

Chapter 6 describes a performance evaluation conducted on behalf of real hardware, while highlighting the impact for different components of the architectural implementation and of SMC.

Chapter 7 conducts a brief requirements assessment, providing an overview about how we fulfill stated requirements of the analysis part.

Chapter 8 presents related work to this thesis, whose shown deficits in the practical deployment of SMC are a contributor to the reason for this work.

Chapter 9 summarizes our contributions and results. In addition to that, it depicts possible topics for future work.

Chapter 2

Background

Targeting privacy-preserving architectures, we first need to lay out some groundwork on this topic. First, we depict why privacy considerations should be valued more comprehensively by pinpointing recent failures in practice. This is followed by a systematization of privacy that enumerates fundamental protection goals. It serves as a guideline for our architecture further on. Thereafter, we introduce Secure Multi-Party Computation as it is a fundamental contributor to enforce privacy aspects from a technological perspective. Specifically, we highlight the protocols and attacker model that are relevant for the architecture and reflect the employed framework's properties. Rationales for choosing the specific framework are given last.

2.1 Privacy Preservation

Privacy is a core value of human being and democratic society as acknowledged by the European Convention on Human Rights [8]. It is a highly valuable asset that every individual should be entitled to have control over.

As the term *privacy* is overly broad and interpreted differently with regard to covered aspects and appropriate implementations, it is impossible for individuals to estimate how a service provider handles someone's sensitive data behind the scenes or which precautionary measures exist. For most users, it is even unclear what a provider should guarantee in order to preserve its client's privacy. It becomes even worse as implications of lax or neglected policies are realized by citizens, but also politicians, in the earliest when it is too late.

We want to briefly systematize privacy in the scope of Smart Environments with a more technical perspective, and enumerate which principles a framework needs to be based upon for a privacy-preserving foundation. Beforehand, we identify problems and lawful neglects in common practices that pose risks to individuals.

2.1.1 Failures and Risks in Common Practice

Since privacy aspects are commonly subject to some regulations in law, cloud providers and also manufacturers of interconnected devices with cloud integration have usually some kind of privacy section on their websites. Beside stating conformance to local law regulations, the focus is often on data security when talking about privacy. For example, some companies emphasize in their privacy section that data is safe by means of secure communication paths with state-of-the-art encryption or by protecting private data on their servers via encryption [9]. Others preclude that privacy is preserved by keeping all data on servers in the same country [10]. While data security is an important contributor to privacy, there are many other components as discussed later that are excluded in their privacy statements. Hence, this places an incomplete impression about privacy.

But also with regard to respective data protection laws, operators fulfill requirements only partially. For instance, there is the directive 95/46/EC on the protection of individuals in the European Union (EU) that has been active for a long time [11]. The directive aims to regulate how data containing personally identifiable information (*PII*) should be processed. Therefore, it states multiple principles to improve privacy for the citizens. Among them, there are principles such as purpose, consent and security. Recent disclosures demonstrate that some providers transmitted excessive PII which obviously are not necessary for providing their services, nor did it happen with consent of their users [12]. Otherwise, international companies with offices in the EU, have poor security practices for storing confidential data. Also personal data is kept longer than needed and stated to its users, as recent breaches show [13] [14]. Consequently, the current legal framework 95/46/EC is picked up as a set of recommendations, but fails in practice due to a broad range of recent technological challenges [15], and the divergence in how member states enforced the rules [4].

The core issue with many providers is their centralized approach in the market of Internet of Things (IoT) with Smart Buildings being a part of it: edge devices are responsible solely for data acquisition or actuating, and send their information (e.g. sensor measurements), to a cloud platform managed by the provider. This is also the place where all application logic resides. [2] While this strategy is comfortable and lucrative for a cloud provider, it poses manifold risks to an end-user. The ubiquitous collection of raw data along all edge devices enables the provider or (illegal) third-parties to analyze data beyond the purpose as intentionally stated. From a technical perspective, centrality greatly facilitates profiling and tracking of individuals. In addition to that, a breach disposes sensitive personal data related to many people compared to a few ones if a distributed approach was used. Furthermore, missing transparency and the lack of control over its own data places the user in a weak position. The detailed accumulation of data is therefore a valid concern by the majority of European citizens [4].

Finally, an end-user must trust its providers that state-keeping is done with privacy and security in mind, and only just as much data as justifiable by the services he consented to.

2.1.2 Privacy Protection Goals

To design a privacy-preserving architecture, it is helpful to devise a set of guidelines for orientation so that failures and risks mentioned in section 2.1.1 can be avoided at the earliest stage. The idea is to establish the paradigm *Privacy by Design.* Work by [16], [17] and technical aspects in [5] provide a solid foundation we consolidate for our purpose.

Rost *et al.* describe fundamental concepts and six goals with respect to data privacy and security in their work [16]. As it is the de-facto work in this area and some of its concepts are adopted by the new European General Data Protection Regulation (GDPR) [5], we use it as a basis for our conceptualization of privacy. Beside the classic concepts of information security, namely Confidentiality, Integrity and Accountability, it introduces the three privacy-relevant goals Unlinkability, Transparency and Intervenability. These are described as part of table 2.1.

In addition to that, Data Minimization defines a further goal that originates from the GDPR [17]. We consider minimization separately as it contributes a significant improvement to privacy by itself, even if other goals are not realized. It is fundamental for the guideline "Privacy by Default" both recommended by GDPR and Rost.

Table 2.1 states and summarizes all relevant privacy protection goals we identified based on works aforementioned.

2.2 Secure Multi-Party Computation

Secure Multi-Party Computation (SMC) embodies a group of technologies and protocols with the same goal: jointly compute functions on private input data, providing correct results for all parties *(output correctness)*, but without revealing any party's data *(input privacy).* For illustration, literature often refers to the *Millionaires' Problem* originally proposed by Yao [18]. Basically, two millionaires want to find out the richer one of both. This shall be done computationally without the need for a trusted third party. At the same time, it must avoid revealing any extra information about the other's wealth.

2.2.1 Definition and Categorization

To formalize this goal, let us introduce a definition for SMC to be used further on. Perry *et al.* [19] come up with an universal definition that covers our focus quite well:

Table 2.1: Seven abstract privacy protection goals for privacy preservation based on [16] and elaborated by [5].

Protection Goal	Description
Confidentiality Integrity Availability	are well-known key concepts of information security. They focus on data security.
Related	• Entity protection • Trust relationship • Communication Security
Data Minimization	requires to minimize collecting and processing privacy-relevant data in general. The aim is to maintain zero or as little information as possible to realize the stated functionality, but not more than that.
Unlinkability	ensures that it is infeasible to utilize privacy-relevant data for any other purpose than stated and acknowledged by the involved individual.
Related	• Data minimization • Purpose binding
Transparency	enables individuals to have insight how and where personal data is processed at any time. It is required to argue whether data collection, processing and use legitimates the purpose of the respective operation.
Related	• Prerequisite to intervenability • Openness
Intervenability	ensures that an individual has control over operations that process private data related to this person. So, any current or future data processing may allow a particular individual to opt-out whose privacy-relevant data is part of the operation.
Related	• Controllability

> A protocol for Secure Multiparty Computation of a class $\mathcal{C}$ of functionalities allows any number $n \geq 2$ of players, each holding a private input x_i, to compute an agreed-upon, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$, where f is any member of the class $\mathcal{C}$ of functionalities. [19]

Stepping into this definition, the defined class $\mathcal{C}$ of functionalities plays an important role for the differentiation of different secure computation approaches. Basically, there are two intersecting classes, in which common approaches can be categorized:

- *Specific functionality* — implements a specific computation that is restricted to a particular use-case. It is not Turing complete.
- *Generic functionality* — supports an universal model of computation. It is Turing complete.

For the former, a typical implementation is the computation of a secure sum. This means that multiple parties shall compute the sum of their inputs minimizing the risk that any input is disclosed. Exemplary protocols are *k-Secure Sum Protocol* [20] and its successor *Modified ck-Secure Sum Protocol* [21]. It basically breaks down to the partitioning of each party's data into k segments. For each round i, the parties accumulate their respective segment k_i in a ring-like manner. Therefore, a party adds its segment for the current round to the intermediate result received from its predecessor. This is passed to its successor. Each new round starts with the result determined before. In the end, the last party in round k, which is also the one that started the computation, disseminates the final result. To enhance privacy, the modified protocol enforces the permutation of the parties' position so that a single party never gets the same neighbors on the ring topology for all rounds. Still, privacy is a major issue with this approach, as all but one parties have to trust the outcome the last party announces (potentially on an individual basis).

In contrast to that, generic functionality is not restricted to additive calculations. It enables any type of computation due to its Turing-completeness. For instance, in addition to that we have seen before, this sort of SMC supports secure multiplications. This, in turn, renders possible secure comparisons. Thus, the combination of these primitives results in a huge amount of possibilities. Given that this thesis aims for great flexibility, we focus on this sort of SMC only.

2.2.2 Major Technologies

Research concentrates on some major technologies that support our requirements for generic functionality and $n \geq 2$ players. The following list gives a brief overview about relevant techniques and their representatives:

- **Linear Secret Sharing**
  Linear Secret Sharing (LSS) describes a partitioning scheme that divides a secret into multiple shares. These shares are usually disseminated to all parties. While the combination of a minimal number of shares allows the reconstruction of the secret, any incomplete set does not disclose any information. *Arithmetic operations* such as addition and multiplication, are the basis for secure computations. Due to its linearity, these operations inherit reactive computation properties. This means that computations may depend on previous undisclosed outputs so that only the final output is opened [22]. A practical example for LSS is Shamir's Secret Sharing [23]. Based on this building block, Ben-Or, Goldwasser and Wigderson (BGW) [24] propose a secure computation protocol with general functionality.

- **Garbled Circuits**
  Garbled Circuits (GC) was proposed by A. Yao in [18,25]. Despite it mainly targets secure two-party computation using *boolean operations*, GC is worth mentioning as it laid out groundwork for SMC in general. Put simply. the boolean circuit comprises the function to evaluate and is uniquely encrypted. By cooperation between the two parties, they can jointly evaluate the circuit without revealing secrets to any party. However, as it is restricted to boolean operations, we do not consider it any further in this work.

- ***Homomorphic Encryption***
  Homomorphic Encryption (HE) describes an encryption scheme that enables a party to apply *arithmetic operations* on ciphertext of encrypted values without prior decryption or knowledge about the key. On decryption by the sender party that is in obsession of the key, the applied operations are also visible in the cleartext value. Mostly, it is used in scenarios whereby a sender party wants to outsource computations to a powerful receiver party, without revealing any secrets. However, the commonly employed two-party setup renders it unattractive for distributed environments with multiple parties we target in this thesis.

In the context of this work, we employ a variant of the BGW protocol. It is easily comprehensible and there is good support in research and practical implementations. Thus, it is interesting for our employed SMC layer. Section 2.2.3 and 2.2.4 will handle its fundamentals from a technical perspective. For other major alternatives not regarded here in detail, Archer *et al.* give further insights and also consider performance in [22].

2.2.3 Shamir's Secret Sharing

The basic idea of Secret Sharing is to distribute a secret among several parties (*sharing*). Then, the *reconstruction* is only successful if a required number of parties collaborate to combine their shares revealing the original secret [23] [26]. A prominent example for Secret Sharing, and more specifically for LSS, is the polynomial-based sharing scheme introduced by Shamir [23]. Shamir's Secret Sharing (SSS) is the building block for multiple SMC protocols, such as for the Ben-Or, Goldwasser and Wigderson (BGW) protocol we refer to later on in section 2.2.4.

Shamir proposes a $(t+1, n)$-**threshold scheme**. Given the precondition that there are n parties, a secret s is divided into n shares and distributed among the parties [1] in a way so that the following properties hold:

- The collaboration and so the knowledge of $t+1$ shares makes it easy to efficiently compute the original secret s.
- t or fewer shares reveal nothing about the secret s.

This means that the threshold scheme tolerates t or less adversaries without compromising *privacy*.

To understand how this basically works, it is necessary to regard the technical construction of Shamir's approach. In a nutshell, it builds on polynomial interpolation in the 2-dimensional plane and utilizes the fact that $k = t+1$ points $(x_1, y_1), \ldots, (x_k, y_k)$ uniquely define a polynomial of degree t. In detail, let $\mathbb{F}_p$ be a finite field for some prime number p with the requirement that $p > n$. To **share** a secret $s \in \mathbb{F}_p$, let $q(x)$ be a randomly chosen polynomial of degree t as follows:

$$q(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_t x^t \tag{2.1}$$

$$a_0 = s \tag{2.2}$$

The coefficients $a_1, \ldots, a_t$ for eq. (2.1) are chosen uniformly at random from $\mathbb{F}_p$, while the secret s defines the first coefficient in eq. (2.2). Each party P_i receives an unique share $[s]_{P_i}$ that is deducted by evaluating $q(x)$ at distinct points as denoted below:

$$\forall i \in \{1, \ldots, n\} : [s]_{P_i} = q(i) \tag{2.3}$$

For **reconstruction**, we know that a set of $k = t+1$ points $(1, q(1)), \ldots, (k, q(k))$ exactly describes the polynomial $q(x)$ of degree t. Having such a set, $q(x)$ can be efficiently reconstructed via polynomial interpolation (e.g. Lagrange). The original secret is given then by $s = q(0)$. [27, sec. 3]

For illustration, we consider some decimal secret $s = 2.4$ to share via an arbitrarily chosen polynomial of degree $t = 3$ as depicted in fig. 2.1. To stay within the finite integer space, consider the values to be represented with fixed-point arithmetic internally, e.g. $2.4 * 100 = 240$. The set of points $\{(1, q(1)), (2, q(2)), (3, q(3)), (4, q(4))\}$ define exactly the blue polynomial $q(x)$. Now, let us say there is a passive adversary that manages to

[1] Of course, it is possible and of practical use to have less parties than shares. Given that, a party with j shares needs less collaborators $(t+1-j)$ to reconstruct the secret. Though there are useful applications to give more power to certain entities, for this work, each party needs to be involved to guarantee fairness in a distributed environment.

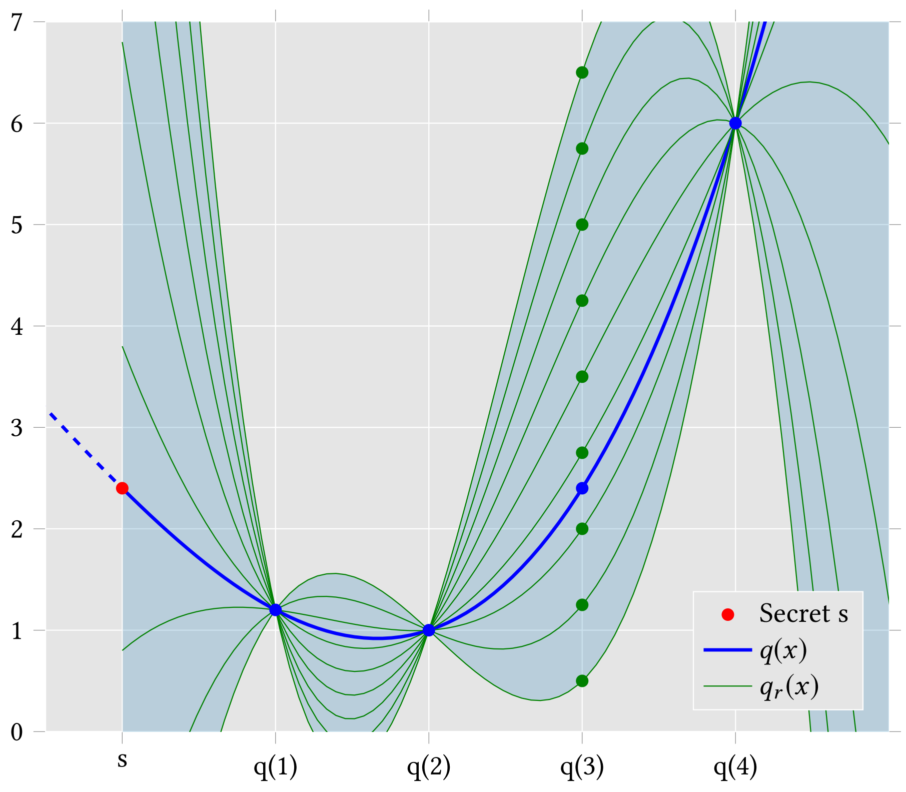


Figure 2.1: Shamir's secret sharing scheme: a secret s is shared by choosing a completely random polynomial $q(x)$ of degree 3 such that $s = q(0)$. The four blue markers are s-wise independent shares uniquely defining the blue polynomial $q(x)$ for secret s. Let $n = 4$ and given a $(n-1, n)$-threshold scheme, the single missing share $q(3)$ makes all green polynomials $q_r(x)$ equal likely, resulting in potential reconstructions for s of the same useless magnitude (blue area at $y = s$).

observe three of them. As for example, these are the points evaluated at $q(1), q(2), q(4)$. However, one share is missing, i.e. $q(3)$. If the adversary tries brute-forcing shares iteratively for $x = 3$, he ends up with $q_r(3)$ possibilities, where $r \in [0, p-1]$. On the solution side, this results in a range of potential secrets of the same order. Figure 2.1 demonstrates this fact: each sample not equal to the correct share $q(3)$, contributes to a bijective result in the solution space (blue area along $y = s$) that renders it useless to use techniques such as brute-force. Taken together, any subset of t or less shares does not reveal anything from an information-theoretic perspective as the likelihood of every possible polynomial $q_r(x)$ derived from t shares, is the same [27, sec. 3].

2.2.4 BGW Protocol

Ben-Or, Goldwasser and Wigderson (BGW) were one of the first to introduce a SMC protocol supporting generic functionality. Internally, BGW makes significant use of

Shamir's Secret Sharing (SSS). This is, in particular, for sharing a secret and reconstructing it at the end. Given the linearity of SSS, the BGW protocol categorizes into the LSS scheme. Thus, its operations rely on arithmetic calculations.

Ben-or *et al.* divide the protocol into three stages:

1. *Input stage* — each party closes its secret by splitting it into shares and sends one to each participant.
2. *Computation stage* — specifies how to commonly simulate a predefined arithmetic circuit, consisting of multiple additions and multiplications.
3. *Final (or output) stage* — one or each party gathers the final share from all other parties to resolve the result.

2.2.4.1 Input Stage

Based on the fact that all n parties $P_1, \dots, P_n$ evaluate a certain function or circuit[2], each P_i usually provides an input to be part of the computation. For secretly sharing an input, this stage uses the SSS scheme to generate n independent secret shares as previously described in section 2.2.3. These shares are distributed among all parties. As each party inputs a value, a single party receives up to $n - 1$ shares from the other parties for many circuits, e.g. to calculate the sum of all inputs.

2.2.4.2 Computation Stage

The computation stage evaluates the circuit that is known by all parties. Supported primitives are arithmetic additions and multiplications that can be placed in arbitrary order. To connect them, the output of two primitive functions (also referred to as *Gates* in the context of circuits) is wired to the input of the succeeding primitive, building an acyclic graph. Noticeably, the values keep secret between gates. No operation requires any secrets to be revealed in intermediate steps (*Reactive Computation*). Let secret shares $a \in [s]_{P_i}$ and $b \in [s]_{P_j}$ be an input for a regarded successor gate and let $q_i(x)$ and $q_j(x)$ be the respective underlying polynomials for party P_i and P_j, the following describes the construction of possible operations:

Addition
Since SSS is linear and the underlying polynomials are of the same degree, the addition is straight forward and does not require any communication between the parties. It is basically enough to add the shares together, i.e. $a + b$. This works as the polynomials $q_i(x)$ and $q_j(x)$ encode the secrets and this means that $q_s(x) = q_i(x) + q_j(x)$ encode $a + b$. [24]

[2]Conditional chaining of multiple arithmetic operations

Multiplication
While scalar multiplication works for the same reason and properties as addition, the multiplication of two secret shares $a * b$ requires some extra steps and synchronization.

Let $q_i(x)$ and $q_j(x)$ be the underlying polynomials of degree t for the shares a and b, respectively, multiplying them results in $q_s(x) = q_i(x) * q_j(x)$ where the degree of the resulting polynomial raises to $2t$. The problem is that $2t < n$ must always hold (remember n is number of parties participating in the computation). Otherwise, if there are more shares than participants, the reconstruction would fail as a party's n shares are not enough points to fix the polynomial $q_s(x)$ to open the result, i.e. calculating $s = q_s(0)$. It gets even more severe for multiple multiplications. To resolve this issue, the *degree reduction step* reduces the degree of the polynomial, i.e. $2t \longrightarrow t$. For the same reason as mention above, the reduction only works if $t < \frac{n}{2}$ holds.

Another issue: $q_s(x)$ is not random anymore after the multiplication. Based on this fact, there is the *randomization step* reestablishing privacy aspects. In a nutshell, it generates a new polynomial of degree $2t$ before reducing collaboratively the degree. This guarantees that the coefficients are independent. [27, sec. 4]

Altogether, the parties need to synchronize actively for each multiplication gate.

2.2.4.3 Final Stage

After the circuit evaluation, every party still holds secret values. This is P_i's retained share that is combined with other parties' shares during the computation stage. The final secret value needs to be revealed now. Thus, the final stage, or sometimes called output stage, requires each party P_i to send its remaining share to either all other parties $P \setminus P_i$ or to a certain group of parties that reconstruct the result (see section 2.2.3).

In case of present errors during reconstruction, the result can still be revealed correctly to some degree. This is possible due to properties of Shamir's Secret Sharing. The sharing scheme implicitly provide an error correction code. More precisely, the constructed shares $\langle q(1), \ldots, q(n) \rangle$ represent a Reed-Solomon code word. Used for decoding, the correction of t errors is possible if $t < \frac{n}{3}$ holds. [27, sec. 5]

2.2.5 Adversary Model

The primary goal of SMC is the preservation of participants' privacy and computations' correctness in presence of adversarial activity. As there is no solution to cover all possible types of attacks, the security of SMC protocols is classified according to different adversary settings. Table 2.2 describes the most relevant types for differentiation.

Referring to BGW (section 2.2.4), there are some important constraints to guarantee

Table 2.2: General attacker model for SMC (based on [19]).

Adversary Type	**Description**
Honest-but-Curious (HbC)	The HbC attacker (*passive* attacker) follows the SMC protocol, but tries to mine as many information as possible to reveal secrets from one or several participants. It clearly targets the parties' privacy.
Static	For the static case, the assumption is that the curious parties are fixed before the protocol starts. The roles will not change during execution.
Adaptive	Though the number of simultaneously curious parties stays the same for the adaptive case, the affected nodes can vary throughout the protocol execution.
Malicious	A malicious attacker (also called *byzantine* or *active* attacker) deviates from the protocol arbitrarily and manipulates messages with the goal to break the computation's correctness or to violate privacy.
Static	A static malicious adversary operates from a fixed set of nodes being the same throughout the protocol.
Adaptive	The adaptive variant is the strongest form of attack as the parties' behavior might vary during the protocol execution. This includes rational attacks that show a certain behavior only if probability is low being uncovered (*covert*).

both *privacy* and *correctness* side-by-side. This applies differently depending on the regarded adversary model [24]:

- Honest-but-Curious adversary:

$$t < \frac{n}{2} \tag{2.4}$$

- Malicious adversary:

$$t < \frac{n}{3} \tag{2.5}$$

Remember that t is the threshold of tolerated adversaries and equals the polynomial degree for Shamir's Secret Sharing. n is the number of participants which provide a share. For the HbC case from eq. (2.4), the attacker does not actively tamper with the transmitted data. Therefore, correctness is guaranteed implicitly. To preserve privacy, SSS is used in BGW. Its degree reduction step for the multiplication of shares only works if eq. (2.4) is satisfied. Static and active adversaries are equal in this scenario. In the presence of a static malicious adversary that incorporates wrong results during the

Table 2.3: Brief comparison of SMC frameworks that support arithmetic computations. The symbol ◗ refers to partial support.

Framework	FRESCO	SEPIA	VIFF	FairplayMP	SHAREMIND
Open-source	✓	✓	✓	◗[1]	✗
Provided tests run out of the box	✓	✓	✓	✗	✓
Support > 3 parties	✓	✓	✓	?[2]	◗[3]
Active development	✓	✗	✗	✗	✓

[1] No source code for compiler blob [2] Claim to work, but no testing possible due to fatal errors [3] Claim to have protocols therefore, but do not focus development in this direction

protocol execution, Cramer *et al.* show a full prove in [28, sec. 10.5] that eq. (2.5) needs to be fulfilled for resistance from an information-theoretic view. Basically, it is possible to recover from erroneous shares when applying techniques such as error correction codes that utilize implicit properties of SSS. This solves the problem for reconstructing a secret. However, an attacker could still share some degree-t' polynomials that are not compatible to t of the honest parties. Asharov *et al.* demonstrate with *Verifiable Secret Sharing* that correct behavior can be enforced as long as eq. (2.5) still holds. [27, sec. 5]

2.3 Choosing Fresco

Having discussed SMC from a protocol-oriented perspective, it is reasonable to decide upon an existing SMC framework implementation rather than creating a new niche. Remember that the overall goal of this thesis is to improve applicability of SMC. Hence, we aim for reusing existing frameworks and technologies as much as possible.

Concretely, we identified five major frameworks which should satisfy the requirement for arithmetic computations we focus on in the scope of this thesis. That is FRESCO [7], SEPIA [29], VIFF [30], FairplayMP [31] and SHAREMIND [32]. By a number of considerations presented in the comparative analysis of table 2.3, only the SMC framework Fresco remains that fulfills all stated requirements. Noticeably, chosen criteria assess the project's active and open-source development. We believe that these are important factors towards the acceptance of a regarded framework. At the same time, it means that a framework is tested more intensively with respect to applicability, functionality and performance. In fact, there are a recent real-world application [33], comparative benchmarks [34] and considerations [22], conducted on behalf of Fresco, that confirm the arguments given. The open development also contributes to security and privacy issues implementation-wise, as code is reviewed by more interested developers as it might be possible for a closed-source (commercial) project. Taking together, these factors set our decision for Fresco as a SMC provider for the implementation of this work.

In the following, we provide a short introduction to the chosen SMC framework. Fresco [7] is a comprehensive Java framework for Secure Multi-Party Computations that brings a rich set of protocols and all necessary groundwork to realize SMC-based applications without the need for in-depth knowledge about cryptography. The framework is mainly developed by the Alexandra Institute in Denmark, but attracts increasing attention from developers according to the amount of recently created issues and code contributions on the project's home.

Fresco's run-time features extensive modularization and abstraction that leads to high flexibility and re-usability on application developer's side. More specifically, a SMC application is constructed from a combination of high-level (and optionally low-level) secure functionalities, which are available as part of the run-time environment. The resulting application can dynamically adopt to current circumstances present right before protocol execution. For instance, this is the amount of participants. The well-thought abstraction also comes into play as written applications can be evaluated with different SMC protocols and network configurations dynamically rather than being tightly bound to a specific configuration supplied at the point when the run-time is started. This fact makes it comfortable to work with in the context of this thesis.

Please note that we implicitly mean Fresco version 0.1 as of October 2016 when referring to it throughout the thesis.

Chapter 3

Analysis

In the following sections, we analyze two exemplary use-cases and use them as basis for extracting common flaws of data-processing architectures nowadays in terms of mechanism to preserve individuals' privacy. We derive a generalization by pointing out technical commonalities with respect to what they need to realize the stated functionality. It helps to assess Secure Multi-Party Computation (SMC) as an alternative core technology to enforce privacy aspects. At the same time, we show present shortcomings and a proposal how to encounter its issues with applicability in practice, particularly applying to dynamic environments. Finally, a list of requirements summarizes aforementioned aspects to realize a privacy-preserving architecture for processing personally identifiable information on top of SMC.

3.1 Use-cases

First, we introduce two use-cases that illustrate typical technical realizations of architectures that process personally identifiable information by means of entities that can perceive their environment. Remarkably, the use-cases reflect common practice rather than showing a perfect solution. Specifically, they highlight privacy-related flaws that potentially leak sensitive information.

For keeping references short throughout the thesis, the first use-case is abbreviated as UC.1, and the second one as UC.2.

3.1.1 UC.1: Smart Environments and Presence-aware HVAC

Goal

Central goal of this use-case is to collect and process measurements from sensing devices in charge of a selected area in a building, and to make them available to central building systems being an operational requirement. In particular, an heating, ventilation and air conditioning (HVAC) system requests average temperature and sum of occupied rooms for a certain section in a building from an entity that provides data in aggregated form.

Context: Intelligent Buildings

The context is a Smart Environment, such as an intelligent building chosen for this use-case. We envision a modern commercial building with multiple floors split into segregated sections that in turn comprise a variety of offices, meeting rooms, kitchens and toilets. Most tenants are companies that rent either one or several floors, or certain sections.

For building automation, each room is equipped with stationary and mobile computing devices with either sensing capabilities, e.g. measuring temperature, humidity, etc., or actuation functionality, e.g. controlling the blinds of a room. An extensive network interconnects all automation-related devices by (wireless) LAN, but may restrict communication flow between individual floors.

Scenario, Actors and Information Processing

Sensing devices are small connected computers that individually generate data points from perceiving the environment for their responsible room(s) in a periodic or event-driven manner. For instance, measurements include temperature, humidity, lighting conditions and increasingly, presence of occupants. Interactions with occupants are mostly passive (by their influence on the environment), but rarely active (e.g. pressing buttons).

A variety of *central building systems* take care of the well-being of occupants, efficient resource management and safety within the building [1]. This is accomplished by means of actuators installed in various rooms. To make these systems work efficiently and not blindly, they depend on data that depicts perceptions in the building covering their specific needs. Also, instead of requiring a detailed representation (i.e. raw data), these systems request the data in a processed form, e.g. an average value.

A *management entity* bridges the gap between individual sensing devices and the need for aggregated measurements that comprise a certain category of data for selected sections in the building. It communicates with sensors from this section via the network, fetches periodically all kind of interesting measurements, aggregates them by category, and makes it available to depending services via an interface. The type of post-processing can be arbitrary due to the access to raw data.

Specific Example: Presence-aware HVAC

Energy savings shall be increased by means of a central HVAC system. Normally, HVAC adapts heating power based on temperature measurements from the offices and a time schedule to lower global power during nights. Given the building is divided in several heating sections (like a section comprising multiple rooms), a system could consider the respective number of present occupants for this area to optimize heat dissemination and output power. The information is fetched from a management and storage unit located in the basement that sums occupancy and temperature measurements on request for a desired section in the building. For the needed data, sensors' raw data are pulled regularly by the management and storage unit that collects them centrally.

Precondition: Device Anonymity

Sensing devices may not access raw measurements from neighboring or any other reachable device in the network.

Similar, the management entity may not disclose data other than in aggregated form. It may never grant access to raw data regardless of whether an entity requests it in terms of selecting an individual room instead of a whole section. Consequently, the exact origin of the measurements must remain anonymous.

Privacy Considerations

Maintaining device anonymity by means of aggregation is a suitable measure to improve privacy. So, aggregated results intended for a central building system, cannot be linked easily to their originating room. Like this, measures preserve an employee from being tracked and analyzed by a malicious service.

However, it does not cover the fact that technical aspects of a management entity or its operators can be a threat by themselves. By default, it receives and records a variety of personally identifiable information that are linked to a specific room's sensing device, and so referring to individuals. This is coupled with a global view since building automation might implement a single central entity for the generated data of the whole building. So, both intentional, being a lucrative goal for an attacker, and unintentional disposal of a stream of room-specific sensor information poses a high risk to all affected individuals and organizations.

3.1.2 UC.2: MeasrDroid

Goal

Central goal of this use-case is the computation of average velocity and distance from a set of mobile clients. The results enable participants to compare themselves instantly with the rest.

Context: MeasrDroid Project

MeasrDroid is a research project at the chair for Network Architectures and Services at Technische Universität München, developed by Dr. Johann Schlamp and several students [35]. Its aim is a large scale measurement framework that targets the ubiquity of mobile devices with the operating system Android and their extensive capabilities. Utilizing the power of crowd sourcing, a mobile application transforms practically every mobile device into a distributed measurement node that contributes extensive information about network properties and environmental perceptions for research.

More specifically, installed Android clients periodically conduct manifold measurements and upload the gathered data as encrypted blob to a web server.
On the one hand, collected data contains environmental sensor readings from the client's hardware, but also details about the employed hardware itself. In particular, these are hardware model and manufacturer, battery status, readings of acceleration, gravity and light sensors, and current device's location with details about the utilized provider (GPS or network) and accuracy, to mention a few examples.
On the other hand, collected data enumerates various aspects about cellular and wireless networks it uses for communication. This dates back to the original purpose of the research project to map the Internet topology from as many different entry points as possible from a geographic and network-wise perspective. A few representative data points in the data set are cellular network operator's name, type of phone and data network, signal strength and a list of neighboring networks in reach.

Uploaded data is decrypted and evaluated on an isolated server which is in possession of the respective private key. Commonly interesting statistical results are published to a website or provided via the mobile application itself. All conducted research always places a high emphasis on not publishing any sensitive information that could possibly identify an individual.

Scenario, Actors and Information Processing

Mobile clients use an integrated GPS module to keep track of the current device's location. Assuming that the owner carries his or her device most of the time close to the body, the location history should reflect precisely travels of the same. The device keeps track of several last locations in local memory.

In contrast to a default MeasrDroid setup, we modify the client application so that it processes subsequent location tuples i from recorded data to calculate the geographically traveled distance D_i and velocity V_i of the person locally on the device. It behaves as a new data provider and replaces raw GPS locations in the recorded time series. The latter are not necessary anymore for this use-case.

As aforementioned, the goal is to compare oneself to others with respect to average distance and velocity among several participants, all recorded data entries since the last synchronization are pushed to the project's *UploadDroid web servers* periodically. If network connectivity is unavailable, upload is deferred until it is established again.

The *data server* fetches all new records from UploadDroid and decrypts it. It calculates the average distance and velocity among all participants' data sets for the last x days. Technically, it is realized by a running average over D_i and V_i records choosing i to range over the given time span. Results are published regularly on UploadDroid so the mobile clients can fetch the average values to generate a comparative analysis for its owner.

Privacy Considerations

First privacy-preserving measures are aforementioned modifications that each device reports a time series of distance and velocity records instead of enumerating exact GPS locations. While this approach makes it harder for a malicious MeasrDroid back-end to track the exact location of a user, a quite precise estimation is still possible. Research in the field of mobile robotics and indoor localization demonstrates approaches to reconstruct movements based on a few different sensors, for instance the work by Li *et al.* [36]. Even the naive approach to combine the relative distances with direction information a digital compass (or a gyroscope) provides, enables an attacker to gain a rough trace. Combined with a few known fixed points (e.g. locations of wifi access points) quickly leads to a precise track.

Thus, if all recent and historic data accumulates in a centralized infrastructure, i.e. the back-end in this project, it is easy to gain deep insights in participating persons' life, correlate with each others or an external party, and predict future actions to eventually take control over someone's life.

3.2 Problem Statement

Reflecting the use-cases, both mention unsolved privacy considerations that can be traced back to the same core issue. Namely, there is always a centralized architecture that collects individual data in a central storage. Due to the strategy that all business logics reside in this central infrastructure, stored data comprises the following properties:

- Sensor data with high resolution
- Data from multiple origins largely linking to identifiable individuals
- Data origin part of recorded meta-data
- High amount of available data (high frequency of submissions)

Consequently, there is a huge amount of sensitive data that resides in a central place, out of direct control for their data origins. From a perspective of the affected individuals, this requires strong trust in infrastructural components, their operators, and ultimately the organizations behind it. To be even more explicit, this storage approach equals a trusted third party (TTP), effectively. Irrespective of using it indirectly or with consent, an individual implicitly articulates trust in uncountable technical and non-technical aspects that go far beyond an individual's ability to judge. Some of the essential ones are as follows:

- Trust in this component and its infrastructure with respect to
  - *Purpose binding* — components will not be programmed to misuse data or to replicate data to a different place.
  - *Confidentiality and integrity* — components itself and all communications are protected and secured by state-of-the-art technology. This is a matter of special importance if components or sensible communications are exposed to the Internet.
  - *Authenticity* — measures are taken to recognize and stop adversaries that try to replace or shadow central components pretending to be the original one.
- Trust in the honesty of
  - *Administrators* — do not abuse their role to access, redistribute or misuse the data.
  - *Operating organizations with access rights* — do not use the data for any other purpose than stated and required for operation.

In an ideal world, a centralized storage is often the most efficient solution with respect to costs, performance, scalability and maintenance. However, a TTP in the central position, liable for the most sensitive and fine-grained information, poses the risk that confidentiality of data is not guaranteed anymore if one of aforementioned trust relationships breaks.

Consequently, it becomes clear that a TTP approach lacks *confidentiality*. Judging the current solution according to the data protection goals we defined in section 2.1.2, and which are fundamental strategies for realizing a privacy-preserving architecture in this thesis, more flaws become visible.

Data minimization and unlinkability goals are not fulfilled as well. By collecting and storing all possible measurements from individuals and keeping note of the origin (meta-data), it is clearly visible that data could be used for further analysis beyond the stated purpose, e.g. tracking and tracing as mentioned in UC.2. Obviously, the vast amount of available sensor recordings is also not required to power a depending service with necessary statistical information, e.g. HVAC system requiring only averaged values and counts as for UC.1. Thus, data minimization does not apply here.

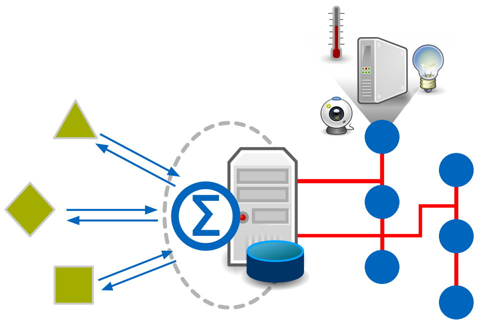
Same holds for *Intervenability*. Although being supported partly by the MeasrDroid project as the user can request to delete all stored data, continuous measurements within the scope of Smart Environments are difficult to control. In all cases, it seems infeasible to opt out for certain investigations by an individual, as he or she does not know what happens to a particular set or category of data. This directly links to *transparency* that is hardly provided in general with respect to the technical realization. So, a user again needs to trust that certain measurements stop, for instance. Therefore, it is impossible for an individual to take control even if the possibility was given.

In a nutshell, a centralized infrastructure with a single entity holding a vast amount of personally identifying data, is a bad idea with respect to individuals' privacy. In particular, the concept *Privacy by Design* as required by the upcoming EU Data Protection Regulation (see section 2.1.2), is not supported as the data protection goals describing fundamental strategies to achieve this, are not satisfied. Choosing a different technology is unavoidable to be compatible with future norms and regulations, but most importantly, to protect the well-being of humans with respect to privacy.

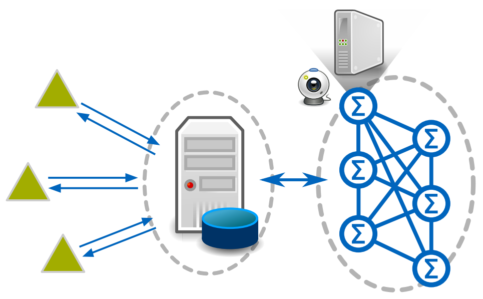
3.3 Abstraction

Although the introduced use-cases seem to differ widely at a first glance with respect to their scenario, they have similarities in their centralized architectures and relations between entities how data flows and where it is processed. This knowledge helps us further on to assess how a different technology is applicable to replace the centralized architecture which has obviously flaws to guarantee trust for the end-user.

First, we identify commonalities from an architectural perspective. Further on, they



(a) Central entity is a TTP. It receives sensitive data (red data paths) from data-generating producer devices and aggregates it for consumer devices depending on it.



(b) SMC based aggregation replaces need for TTP. Each data provider is owner of its sensitive data. It is never released in raw form.

Figure 3.1: Derived generalizations based on the use-cases. It also shows how to transform the realization of aggregation from a TTP (left side) to SMC (right side). Blue nodes are devices that generate data, for instance, by perception or knowledge. Σ symbol annotates entities responsible for aggregation. Green nodes visualize consumer devices or services which rely on different types of aggregated data as depicted by individual shapes.

are used to derive an abstract model that generalizes relationships and responsibilities without relying on details of the use-cases.

3.3.1 Architectural Commonalities

The use-cases discussed in section 3.1 can be summarized by means of architectural (functional and non-functional) aspects that they have in common. Figure 3.1a depicts an architectural overview that demonstrates all integral parts to support both of them. Based on that, we specify relevant commonalities in the following listing, while also pinpointing their respective implementations with regard to the use-cases.

When naming *central entity*, it means a piece of hardware or infrastructure that plays a critical role for realizing the stated application. It is the fundamental part of the business logic. Regarding the use-cases, it is equivalent to a management entity for a Smart Environment or the back-end infrastructure for the MeasrDroid project.

1. **Numeric values**
   All operations base on numeric values or tuples of the same. In case of non-numeric input, e.g. string of mobile operator, there is often a simple way to adequately map it to a number representation, e.g. enumerating, hashing.

   Use-case references: both use-cases generate data by perceiving their environment by means of a variety of sensors. Each sensor usually relies on numerics for efficient processing.

2. **Producer devices as raw data provider**
   A set of devices generates data by perceiving their environment either periodically or on demand. They follow an active or passive synchronization approach. Either, devices actively push recorded data to a server, or, they wait for an incoming request to return it. The selection which and how many records to provide, is static or dynamic.

   Use-case references: for both use-cases, sensor readings are turned in raw data. UC.2 actively connects to its back-end periodically to upload all records since the last synchronization. While this direction is necessary due to restrictions imposed by mobile Internet providers, a passive approach works as well in a local environment (e.g. wireless LAN). UC.1 supports both active and passive synchronization.

3. **Central entity receives raw data**
   A central entity receives or fetches newly generated raw data from producer devices. It keeps record of a given amount of historical data points from individual devices. Stored entries allow to differentiate different producers.

   Use-case references: for UC.1, the management entity fetches raw recordings from sensing devices on a room basis. Similar, mobile devices upload recent measurements to the back-end for UC.2.

4. **Central entity provides only aggregated data to consumer devices**
   The central entity hosts an API that consumer devices or services use to query for aggregated data of a certain category and time span, assuming prior authorization. Only aggregated data is returned to requesting consumer devices, though administrators or operators may access all producer devices' raw data directly from storage.

   Use-case references: management entity of the Smart Environment (UC.1) calculates the average of temperatures and the sum of occupants for all managed rooms. HVAC may only request this aggregated value. The MeasrDroid back-end (UC.2) calculates average distance and velocity. Consumer devices are the same as participants in this case. Still, devices may only request averaged values.

5. **Any edge device cannot access input from others**
   Both producer and consumer (≡ edge) devices do not have access to raw measurements of neighboring or any other edge devices that are reachable through a common network. Only an authorized central entity may gather this information. This is a measure to improve node anonymity.

   Use-case references: both use-cases realize this measure by means of encryption and authentication.

Noticeably, the points describe a centralized architecture that is quite common in

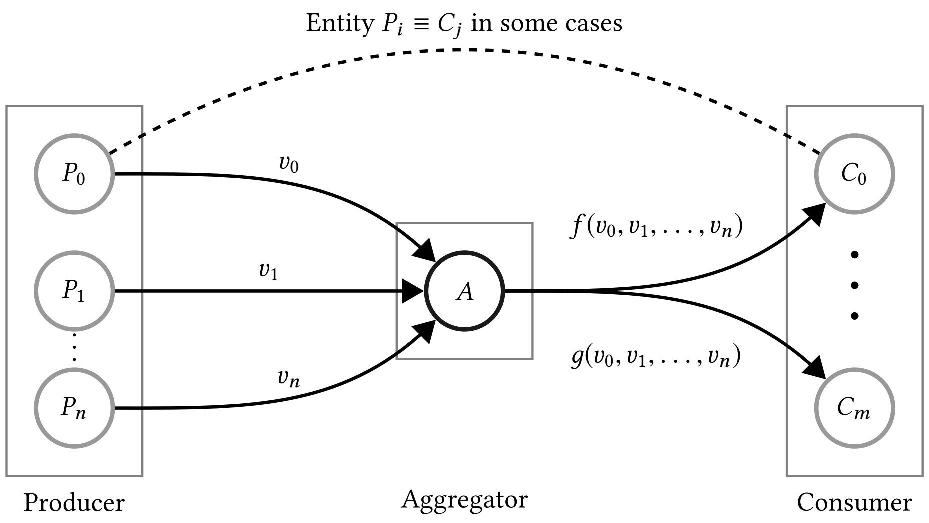

Entity $P_i \equiv C_j$ in some cases
P_0
P_1
P_n
v_0
v_1
v_n
A
$f(v_0, v_1, \ldots, v_n)$
$g(v_0, v_1, \ldots, v_n)$
C_0
C_m
Producer
Aggregator
Consumer

Figure 3.2: Architectural commonalities compressed to a simple generic model. $v_0, v_1, \ldots, v_n$ are raw input values provided by individual producer devices. f, g are (different) functions applying a chain of one or several arbitrary aggregation primitives on the input. Producer and consumer can be the same entity.

automation-related solutions. Therefore, the elaborated commonalities may apply to manifold use-cases which are not covered in this thesis, as well. This means that the intention is to mark integral aspects for orientation, but does not impose strict restrictions.

3.3.2 Generalization Model

Based on analyzing architectural commonalities, we identify three relevant groups of entities whose responsibilities can be further condensed to simple tasks. Considering the simplified model as given by fig. 3.2, the integral entities can be categorized as follows:

P_i **Producer entities**
Each producer runs on a dedicated device and generates raw data as input. Producer P_i provides v_i to the aggregation entity A for a given computation round. v_i is a vector with one or several numeric values. Length and type of supplied data must be consistent within a round.

A **Aggregation entity**
The input tuple $(v_0, v_1, \ldots, v_n)$ is processed by a commonly known aggregation function, i.e. $f(v_0, v_1, \ldots, v_n)$ or $g(v_0, v_1, \ldots, v_n)$ in the figure.

Such a function can be an arbitrary combination of aggregation primitives, such as

- Sum
- Multiplication
- Count
- Minimum / Maximum
- ...

In all cases, it must include at least one aggregation primitive that does not expose evidence about a single data origin. There is a preset of possible variants of aggregation functions, previously agreed upon by aggregation and consumer entities. Thus, functions are public. Based on what consumers demand, f and g might represent different functions of the preset.

C_j **Consumer entities**
A consumer chooses from a predefined set of aggregation functions that satisfies its requirement, i.e. selecting f or g, or any other function A supports.

Our model assumes that provided data from a set of producers is identical in type, e.g. all values represent temperature readings. This is not a restriction in general, but simplifies the scope of a single computation round. Further rounds might expand aggregations to other type of input. Still, the type must keep the same within a round so that aggregation makes sense. In addition, as denoted by fig. 3.2, a certain producer and consumer can be the same entity, so $\exists i, j \in \mathbb{N} : P_i \equiv C_j$ may hold in some cases. For example, this is the case when a data origin needs feedback about related entities for the purpose of comparison or optimization (see UC.2).

3.4 Secure Multi-Party Computation as Building Block

SMC is a promising technology that fits well to our overall goal for privacy-preserving services. In section 3.4.1 and 3.4.2, we analyze how SMC can be deployed while supporting the generalization model derived from the use-cases' architecture. It solves several privacy issues as identified in section 3.4.3. However, the use of SMC raises several new problems, and emerging challenges arising from the lack of a centralized architecture. These issues and a proposal to tackle them are discussed in section 3.4.4.

3.4.1 Model Compatibility and Deployment

With respect to the derived generalization model described in section 3.3.2, SMC can be a perfect candidate to replace the centralized architecture in a multitude of ways. Solutions based on Linear Secret Sharing have support for arithmetic operations. Combined with their reactive computation properties (see section 2.2.2), it allows to construct arbitrary circuits based on addition and multiplication gates without revealing secrets in intermediate steps. This allows to simulate any aggregation function composed of several operations. Hence, in the concrete case of using the BGW protocol, it fulfills technical requirements of the model and is a suitable alternative.

Still, there is an open question about how SMC is deployed in the concrete case, as it influences further analysis. In practice, we encounter two relevant constellations:

- **Simulated TTP** — the central aggregation entity A is simply replaced by a *dedicated set of privacy peers* that is usually smaller than the number of producer peers (mostly 3 nodes). The privacy peers perform aggregations with SMC and publish the result separately. Producer peers input their sensitive values by splitting them into shares that are distributed among the privacy peers. For example, deployments with 3 dedicated privacy peers are presented in [29] [37] [6].

- **Decentralized computation** — each producer device P_i provides an aggregation entity by itself as demonstrated in fig. 3.1b. So, each P_i is basically a dedicated privacy peer participating in SMC computations. There is no TTP with respect to aggregations. They even receive results as part of SMC. Practically, a similar deployment targeting more than 3 privacy peers is part of [38].

We choose to replace the TTP aggregation with a decentralized approach as it seems to maximize privacy with respect to the focused data protection goals. Its transformation is depicted in fig. 3.1. Suspected implications are part of the following analysis.

3.4.2 Prerequisites for Decentralized SMC Approach

As depicted in fig. 3.1b, the decentralized approach, we focus on, turns each producer, i.e. device with perceiving capabilities, into a privacy peer. This requires a few prerequisites for each single peer:

- Sensor node with computation resources (i.e. equipped similar to mobile devices nowadays)

- Local storage for persisting measurements

- Ability to communicate with all other local peers

A significant change arising from decentralization is the need for an individual local storage. As a sensor only publishes information in form of its contribution to the aggregation outcome, it needs to keep track of raw measurements by itself. In addition, the network available for communication must allow the direct exchange between peers to make SMC work efficiently.

3.4.3 Solved Privacy Issues

Required trust in infrastructure and its operating organizations was a huge concern for a centralized architecture as discussed in section 3.2. Given a decentralized data storage and aggregations being computed by SMC circuits, we analyze changed expectations for privacy in this section.

3.4.3.1 Confidentiality

Given the decentralized storage, the stored data belongs to the respective sensor. Measurements are only provided in form of aggregated values via SMC. The device and its concerned individuals do not have to trust any third party receiving their data in raw form. In particular, this originates mainly from two facts:

First, an administrator would need to withdraw data from several tens or hundreds of devices separately. If the effort was not high enough, further measures could report administrative access to the rooms' occupants to make them suspicious about the unusual actions. This would be not possible for a TTP as it is usually out of sight or control for anyone beside administration.

Secondly, an active participation in a SMC session has a fundamental advantage from a device's perspective compared to a simulated TTP as it is used quite commonly. In the latter case, the producer nodes provide a share for each privacy party. As the privacy peers are not under control of the data origins, they could collaborate to easily reconstruct the raw input from each producer without anyone to notice. In stark contrast to that, a participating peer divides a secret into multiple shares, but always keeps one back. The last share is combined with other shares as part of the aggregation. Hence, the raw input cannot be derived from an information-theoretic perspective for additions (see reconstruction of Shamir's Secret Sharing in section 2.2.3) or as long as the attacker model holds in the general case (see section 2.2.5).

In total, a decentralized SMC node strongly contributes to confidentiality of individual's data by means of its physical and logical separation, and most importantly, its active contribution in a SMC session.

3.4.3.2 Data Minimization

As it is in its own interest for privacy, a device reduces the collection of measurements to a required minimum. Other nodes' raw data is not present due to the use of SMC. In addition to that, local storage might restrict the amount of data a device collects, and hence the ability to create detailed profiles about a single individual. This makes it less attractive as an attacker goal than a centralized storage with perceptions originating from multiple sources over a long time period.

3.4.3.3 Unlinkability

Providing data only as part of enforced aggregation via SMC, the entropy of results is much lower than mining raw data. This renders it applicable for a specific purpose, but should reduce its usefulness for other purposes tremendously. In addition to that, aggregation functions and their technical realization details are previously known to all parties. On this basis, it is possible to argue whether an aggregation functionality is necessary to fulfill the stated task. Admittedly, the purpose of the operation might not be clear despite using SMC.

Similar, the composition of aggregation results cannot be backtracked. No individual party can be accused to be responsible for a certain outcome. Thus, relations between sensors' measurements are hidden, i.e. tracking an individual along several spatially distributed sensors is not gainful.

3.4.4 SMC as a Service: Problems and Challenges

Taking away responsibility from a central entity and moving significant portions to edge devices improves not only privacy aspects as aforementioned. The utilization of SMC massively complicates deployment and practical applicability that lowers its acceptance. General problems coming along with the application of SMC, are illustrated in section 3.4.4.1.

If we add the fact that deployments base upon the decentralized computation paradigm, things become even more complicated. To overcome these problem, we propose a list of properties an overall architecture must fulfill to render SMC a flexible foundation for applications compatible to the generalization model. As it targets dynamic environments, e.g. mobile clients that spontaneously collaborate in varying constellations, it should cover a wide range of applications which aim for combining applicability and privacy preservation in practice. This is presented in section 3.4.4.3.

3.4.4.1 Applicability Problems in Practice

With applying SMC in real systems, the benefit of improved privacy for individuals or organizations is huge. According to Bogdanov *et al.* who are responsible for renominated research around SMC, existing implementations are also not the bottleneck. What actually restricts applicability is its deployment and the need for a well-defined and stable environment. [6]

Expanding this perspective to the broad requirements of the generalization model, we concentrate present issues in three points:

- **Complicated deployment**
  Common implementations require administrators to pinpoint various settings beforehand to prepare a single computation. Among these, it requires to specify network locations and IDs for all privacy peers with respect to each other, define the concrete input set and configure the functions to apply to the data. Awaiting that all nodes are ready, the computations is triggered mostly manually as interfacing software is rare. Then, the output has to be collected from the privacy peers.
  For further computation rounds that require a slight change to any of the aforementioned parameters, the whole procedure starts over again. Due to the static wiring, it needs a lot of hand work and experienced operators.
- **Well-controlled environments**
  Both SMC implementations and applications that running on the privacy peers, expect a perfect environment with respect to availability and guaranteed participation of other peers. This means: if a single privacy peer drops out, everything fails and keeps broken until manual intervention by an operator. A change in the list of privacy peers even requires a complete manual reconfiguration of various components.
- **Inflexibility**
  Most deployment of SMC in real-life applications are tailored uniquely to its requirements and its environment. In particular, the task is always exactly the same and cannot be adapted without redeployment, e.g. to replace an averaging function with a standard deviation needs an developer to rewrite the SMC application. Even supporting infrastructure is often tightly coupled with the concrete use-case. Therefore, the resulting construction is very specific and most components are wired statically. This renders the architecture basically useless for any other use-case.

3.4.4.2 New Considerations for Dynamic Environments

In contrast to research papers which adopt SMC as core technology, we do not consider a well-controlled environment with static privacy peers. The decentralized computation approach foresees the nodes to be distributed and self-contained to maximize privacy. In addition to that, we consider a dynamic environment that partly comes along with the ubiquity of mobile or portable devices being employed as a replacement for former hard-wired entities.

Concomitant effects render it necessary to rethink architectural decisions:

- *Peers are moving.* A certain network and a trusted gateway becomes unavailable from time to time. It may become available again eventually. Additionally, a peer needs to adopt gateways of different networks as well, and switch between known one depending on their availability in different localities.

  For instance, a peer is a mobile device carried by the owner. Daily, the owner moves between its working place and home. Both places feature a wireless access point which acts as a SMC gateway, respectively. Due to different gateways and localities, the peer must adopt automatically to the new situation.

- *Topology or addresses changes in a network.* Endpoint addresses are never stable in a dynamic network environment. This also applies to the gateway as it is handled the same as any other device in a network.

  For instance, a peer or gateway uses stateless auto-configuration for its IPv6 link. Given the entity is attached to a common network via a wireless link, it might happen that it shortly looses connectivity. On reconnect, the entity might adopt a new network address as a IPv6 privacy extension[1] rotates the interface identifier part of the address.

- *Entities become unavailable sporadically.* A peer might be detached from the network by its owner as the host device might serve other purposes as well. Moreover, wireless links can cause short interruptions.

  For an example of the first case, assume that a portable computer is part of a SMC network. Most of the time, it is connected to a wired network at its owner's desk. Since it is used for presentations and team meetings, it is disconnected from the network from time to time. However, it is reattached later on.

Consequently, in a dynamic environment, entities cannot be assumed to be static or highly available anymore. Applied to a decentralized SMC computation, it is never guaranteed that a set of peers stays on-line for a long time period. Instead, participation can change quite abruptly and unexpectedly, hence, resulting in fluctuation of available peers. Current implementations expect that a previously defined set of privacy peers is always available. The high likelihood in this case that some peers become unresponsive, makes it fail in the long term. Similar, new suitable peers are not considered to be included due to manual configuration beforehand.

3.4.4.3 Proposal

A static approach suffers not only from issues mentioned before, but also does simply not work in a reliable and efficient manner for dynamic scenarios. We need a more flexible approach that makes SMC applicable easily in dynamic environments to raise

[1]RFC4941: Privacy Extensions for Stateless Address Autoconfiguration in IPv6

acceptance in general. The difficulty is to mimic a centralized infrastructure, e.g. cloud service, to the outside, but to control a spontaneous set of distributed peers in the inside, and eventually to benefit from this approach without exploding costs in operation. The following tries to pinpoint necessary properties for an distributed architecture so that SMC can be adopted as a service in practice:

- **Virtual central entity (gateway)**
  Clients relying on aggregation results provided by the sensor network, assume a well-known endpoint which provides a standard interface. It must hide the fact that a decentralized SMC computation does the job behind the scenes. The responsible entity must be highly available and easy to find. It basically acts as a gateway that mediates between client requests and specifics of a decentralized SMC network.

- **Simple deployment and operation**
  Operating a network of privacy peers usually requires expert knowledge to deploy and operate it as elaborated in section 3.4.4.1. This becomes even more complicated since all sensors are privacy peers in our case. Hence, most required steps should be done automatically, such as forming a SMC ring in a local network. Further, complexity of initial configuration should be comparable to a centralized service. The aspect of simplicity should also hold when replacing hardware.

- **Adaptive and robust**
  The dynamic nature in the stated use-cases makes it necessary that the SMC network adapts to changes in participation by individual sensors. Compared to a simulated TTP, this is important, because every sensor is a potential privacy peer. Thus, the sudden lack of a previously active sensor has to be tolerated. Similar, the network has to respect contributions of new sensors instantly. This attracts further importance if sensors' participation is timely and spatially limited, so resulting in sporadic contributions, e.g. a set of mobile devices acting as sensors.

- **Fast response times**
  A request from a client must be answered within an acceptable amount of time, i.e. a few seconds to be comparable with common cloud services. The perceived performance is crucial for the acceptance of SMC to be a replacement of existing centralized architectures.

- **Flexible tasks**
  SMC peers are not bound to a specific task or input. Based on their capabilities, they support a rich set of aggregations functions that can be applied to available sources, i.e. sensors in our case. Furthermore, devices' sensor capabilities can act as an implicit filter of which ones take part in a computation given individual devices might support or lack a desired type of sensor.

3.5 Requirements

Based on the analysis, we propose a list of requirements for a privacy-preserving architecture that is easily and flexibly applicable in practice while still basing upon decentralized SMC as core technology. It is categorized according to privacy (section 3.5.1), security (section 3.5.2) and deployment and operation (section 3.5.3).

When speaking of an "entity" in this context, it means either a device or the respective individuals whose data is present on this device, e.g. a sensor recorded a time-series of room occupancies which relate to people known to be regularly in this room.

3.5.1 Privacy

For fulfilling privacy aspects, we strictly align with data protection goals as introduced in section 2.1.2.

≪R.1≫ **Confidentiality**
An entity's sensitive data may not be disclosed to any other party than the data origin.

≪R.2≫ **Data Minimization**
Data collection has to be reduced as much as possible to realize the stated functionality, but not more than that.

≪R.3≫ **Unlinkability**
The data recipient shall not be able to trace back the input a single entity contributes to the overall result. The exposed data may not be usable for any other purpose than stated in a lucrative way.

≪R.4≫ **Transparency**
Each entity must able to gain technical insight about the operations on sensitive data and their exact purpose.

≪R.5≫ **Intervenability**
An entity must be free to opt out or to revoke access at any point of a current or future computation that processes its data .

≪R.1≫, ≪R.2≫ and ≪R.3≫ are already satisfied by the fact the architecture assumes distributed nodes with own storage whose results are aggregated by means of a decentralized SMC approach (see section 3.4). Transparency and Intervenability need consideration, though.

3.5.2 Security

For establishing trustful relationships between distributed nodes, data security is an important concern.

≪R.6≫ **Mutual Authentication and Encryption**
Any communication must be authenticated and encrypted by state-of-the-art technologies. This requirement inherits data security properties, such as confidentiality and integrity of data with respect to the transport layer.

≪R.7≫ **Verification of Identities**
All entities must ensure that they establish a connection only to peers whose identity is known and trusted.

3.5.3 Deployment and Applicability

For acceptance of SMC as a core technology, the deployment and applicability of the goal architecture should be as easy as using the service from a centralized architecture, e.g. a cloud provider. Due to decentralization and effects of dynamic environments, this results in the following requirements:

≪R.8≫ **Zero-Configuration**
When connecting to a network, sensors shall automatically locate and register at an entity (gateway) taking responsibility for locality and type of task.

≪R.9≫ **Virtual Centrality**
A standard service interface is provided by means of an virtually fixed endpoint. It mimics behavior of a centralized (cloud) architecture, but may be hosted on any entity. It should be easy to locate it.

≪R.10≫ **Fast Responses**
Request-response times should be within acceptable norms so it is useful for a depending service. Perceived performance must be adequate to typical web services.

≪R.11≫ **On-demand Sessions**
Computations are not run continuously, but might be triggered on demand by an authorized entity.

≪R.12≫ **Adaptiveness**
Nodes leaving or joining a network must be taken into account dynamically for future sessions. For long-running sessions, participants have to be informed to fix the session.

≪R.13≫ **Robustness**
If nodes become unresponsive or drop out during any operation, the network must recover and retry without user intervention. The system should take necessary measures to avoid manual administration or monitoring. Also a node itself should recover from errors, restart internal failed components and react appropriately if network parameters change that require action.

≪R.14≫ **Self-Sustainable**
Single nodes or any part of the composed system do not rely on services that require an outgoing connection to third party providers. This holds both for initial configurations and continuous operation.

Chapter 4

Design

In the previous chapter, we have identified SMC as a suitable technology to enforce fundamental aspects of privacy for individuals. To come up for deficits of current SMC frameworks in terms of applicability in practice and to even enlarge privacy aspects, section 3.5 lists several requirements an architecture must fulfill to resolve these issues.

In the following, we present a system design with the goal to make SMC easily usable and practically applicable, particularly with respect to dynamic environments and arising challenges. To achieve this, we strive to support all previously stated requirements.

We start of with a high-level view on the architecture and its fundamentals. Then, we move down to highlight specific components that contribute differently to satisfy the overall goal for robustness and flexibility.

4.1 Organization and Entities

This section presents design decisions and illustrates fundamental entities and their relationships from a high-level view which are essential to understand the overall system design. Moreover, it clarifies the scope of the design.

4.1.1 Hybrid Approach: Decentralization and Virtual Centrality

The analysis shows that a centralized infrastructure greatly simplifies design decisions and consequently reduces costs, but is an underestimated source of possible privacy violations. Its central storage makes it a lucrative target for both internal and external attackers, e.g. a curious boss spying on its employees, and thieves intending to steal expensive equipment during off-times of a company, respectively. Likewise, external attackers can steal the data itself in order to sell it.

On the other side, decentralization is a fundamental contributor to enforce privacy because of two major aspects: first, individuals' data tend to land on sensor devices associated with a certain locality that is visited by the same people. Resulting physical data distribution makes it difficult to collect data from all places, especially since physical access might be a prerequisite if devices offer no networked-faced possibility for data access. Secondly, by providing only insensitive data by means of a decentralized SMC network processing personally identifiable information, it contributes to confidentiality ≪R.1≫ and unlinkability ≪R.3≫ for individuals' data, and provides means to minimize local recordings effectively ≪R.2≫.

We propose a two-tier approach, bringing together the benefits of a decentralized architecture with small connected computing devices for each locality, and concepts of centralization for efficient management and simplified integration with existing services. Arising maintenance efforts from the decentralized nature, are tackled as well.

In particular, our approach adopts following functionality from both strategies:

- *Decentralization* — decentralized, device-owned storage and secure multi-party computations for maximized privacy guarantees, and self-management (configuration, adaption and monitoring) for applicability.
- *Virtual Centrality* — single generic endpoint for service requests (comparable to a cloud provider's endpoint), hiding complexity and fragility of the SMC network behind, and taking on the role as a leader for adaptive and robust orchestration.

4.1.2 System: Gateway, Peers and Service Clients

For the design, we divide our regarded system into multiple entities representing divergent responsibilities and tasks.

Figure 4.1 depicts a top-level view on the structure in an exemplary smart office use-case. It shows individual computing devices per each room that form an interconnected system for providing their measurements (perceiving their local environment) on behalf of SMC. This type of node is called *peer*. As there are multiple of them usually associated with a gateway, we refer to them as a collective named *peers*. The system is managed by a selected node which acts as a *gateway* to bridge outside and inside world. Outside world means external parties that need processed data from a locality to operate their services. They are called *service clients* in this context.

Although the illustration gives the impression of a dedicated gateway entity, there is actually no pre-determined node installed solely for this task. Rather, any node can potentially take this role, even in parallel to its role as a regular peer.

To provide deeper insights about specific tasks and relationships of mentioned roles, the entities *gateway*, *peer* and *service client* are described in more detail as follows. Please

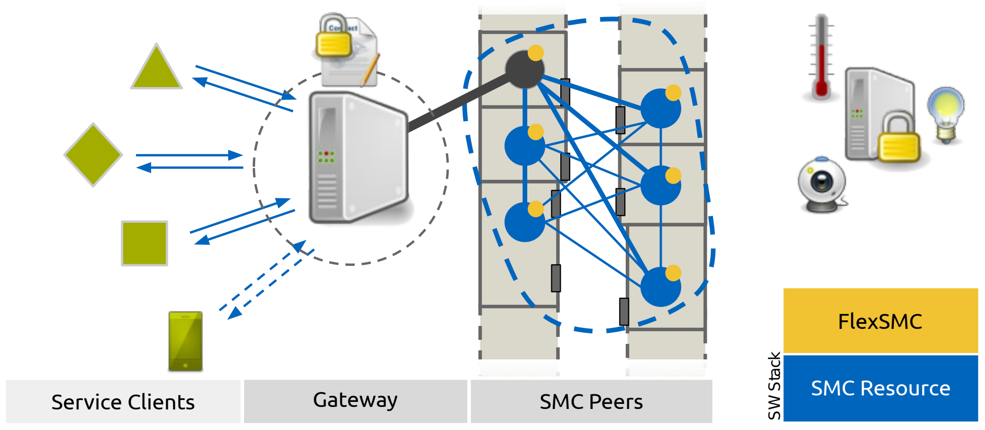


Figure 4.1: A complete architectural overview showing all relevant entities in an office showcase. The gateway (gray node among the SMC peers) unifies both aspects of a regular peer and leadership in this figure. Regular peers (blue nodes) build a computation network associated with this gateway. The green shapes on the left visualize different kind of clients with interests in different aggregation results.

note that we limit our design to a single locality for the sake of simplicity. It is called *domain* in this context and ensures that only a limited amount of nearby peers are attached simultaneously to the same gateway. More details on the motivation for that are given in section 4.1.3.

4.1.2.1 Gateway *GW*

Each domain has usually a single gateway *GW* that takes leadership and represents a central meeting point. On the one side, it manages and orchestrates a dynamic set of peers which employ SMC to process their local sensor input according to the request of a service client. In particular, it takes care of failing peers, monitors and handles problems during computations, and integrates recently joined peers. On the other side, it provides a public facing interface with whom client services get in touch to submit a task in order to receive processed sensor data for selected subsections the gateway is responsible for.

A gateway must not be a dedicated device that is solely placed and installed for this purpose by an administrator. Effectively, it can be any available computing device, optimally operative most of the time, that runs an additional piece of software. It is locatable in the network in order to reveal its public interface to depending service clients and to offer its available leadership to close peers for auto-configuration.

Use-case analogy: in UC.1, one of the sensing devices takes over the role as a gateway responsible for its installed locality. At the same time, it acts as a regular sensor to not

loose perceiving capabilities, i.e. a peer joining its own network.
Regarding UC.2, the gateway role could be taken by a router to whom multiple mobile clients are connected regularly.

4.1.2.2 Peer *P*

A domain consists of several powerful sensors that are called *peers* in this work. In contrast to "dump" sensors of centralized solutions, peers are the smart work force in this system. They feature sufficient computational power and can communicate with each other beside their sensing and potentially acting capabilities.

A peer automatically joins a nearby gateway, that matches with its capabilities, in the network it is currently attached to. Like this, it gets part of a spontaneous peer network to contribute its recorded sensor measurements on behalf of secure computations. Though the gateway coordinates which tasks are executed at what time, a peer is free to refuse any request, eventually after some pre-computations with its suggested collaborators.

When loosing the connection to the previous gateway, it tries reattaching or looks for another appropriate and trusted gateway in oder to maintain an robust integration with an available domain.

Specifically, a peer accomplishes the following tasks:

- Automatically registers with a nearby gateway and remembers it
- Perceives its environment and keeps some historical records
- Shares processed measurements via secure computations on demand
- Controls and monitors requested computations
- Intervenes action and notifies owner on abnormalities

Use-case analogy: in UC.1, a sensing devices is basically a peer contributing its perceptions, albeit having enough computational power to participate in SMC sessions.
Regarding UC.2, any mobile client resembles a peer. Given a potent hardware platform, it is a good foundation for secure computations required as part of a peer.

4.1.2.3 Service Client

A service client is a stationary or mobile consumer entity which is interested on processed data sourcing from the peers of a specific domain. To gather respective data, it locates the responsible gateway in the network and issues an ad hoc query on its public interface.

Service requests could also be issued by peers themselves as they might conduct a comparison with their neighboring devices for the purpose of optimization. In this case, service client and peer are the same device.

Use-case analogy: for UC.1, a HVAC system acts as a client to request current room occupancy and average temperature for a certain section in the building to control heat dissemination.
Regarding UC.2, mobile clients retrieve averaged velocities and distances themselves to compare to the rest.

4.1.3 Scope Limitation by Locality and Composite Pattern

We need to make some assumptions to not overly complicate system design with regard to properties incurred by basing upon SMC.

A large system is represented by multiple locally separated gateways that cover adjacent sections of moderate size. Locality here is interpreted in terms of small network latency and geographical proximity. Such a section is also called **domain** in this context. Within a section, all devices can freely communicate with each other. Peers attach to a gateway of a nearby section based on their current position that is derived from their associated network resource or by means of other location providers. Moreover, a peer actively serves a single gateway only at a time.
For illustration, imagine a corporate building as described in UC.1. Each floor of the building is represented by a different gateway in form of a wireless router. All smartphones located on a certain floor join the network of the respective gateway.

The reason for *splitting responsibilities by locality* leads back to scalability properties of SMC: networks are usually organized the way that nearby devices connected to the same network, have short paths to each other. This results in low network latency and high bandwidth availability for a certain location. Secure computations require all peers of a domain to communicate with each other, meaning quadratic communication complexity. They scale well as long as latencies are low and network bandwidth is not exhausted [29]. We aim to guarantee this requirement by restricting the scope to secure computations in a particular local network.

A problem that arises from this approach, is the efficient fusion of multiple domains. For instance, a service client, such as a central heating system, may want to retrieve temperature readings from a larger area, constituted by several independent domains. An efficient solution to this problem is proposed in [52] and seized practically in [38]. Basically, it suggests a hierarchical organization of gateways and peers in a tree-like structure, also referred to as *composite pattern* in this context. More specifically, gateways simulate a peer role to a higher-level gateway that in turn aggregates their results by means of SMC, recursively. Then, the root gateway presents the result to the requesting service client. Remarkably, this approach enables parallelization for independent branches on each level of the tree.

However, being beyond the scope of this thesis, the approach is neither part of the design nor its implementation. In the rest of this thesis, we limit our scope to a single domain. Though, it is a suitable topic for future work and demonstrates that it is possible to scale our system efficiently for large scenarios.

4.2 Core Components of Self-Managed Networks

Our design does not presume a statically configured and centralized architecture around well-known endpoints. The lack of infrastructure for centralism and the fact that entities can be mobile with regard to dynamic environments, make it necessary for employed entities to organize themselves.

In this section, we discuss core components which found the base layer for self-organization and are the prerequisite for forming reliable and secure integrations between peers and gateways.

4.2.1 Discovery

Given a peer connects to a new network, it needs to locate a gateway that is compatible to its requirements in terms of locality and support for its sensors. Due to the dynamic nature of the participants and their environment, it is also not guaranteed that a remembered network address stays valid all the time. It relates to the fact that there are no fixed endpoints as it is for a centralized architecture. Therefore, a peer needs to find a suitable gateway in a dynamic fashion.

For this, discovery is a fundamental mechanism to enable zero-configuration ≪R.8≫ and like this, facilitate the deployment and operation of a SMC network as final goal. Due to the requirement for a self-sustainable architecture ≪R.14≫, a decentralized approach provides the foundation for the discovery mechanism, e.g. using DNS-based Service Discovery [39].

4.2.1.1 Gateway Casting its Properties

Each gateway announces regularly its existence in the local network so peers take notice of them. The other way round, a peer can issue a query to retrieve a list of all available gateways if it recently joined a network and so, missed some announcements.

Table 4.1 enumerates the descriptive properties which accompany each announcement message.

Table 4.1: Gateway and its properties retrievable via local discovery.

Property	**Example value**	**Description**
ID	`gw-58c1ef..abaf1c.flexsmc.local`	An unique identifier for a whole building or larger scope within a certain area.
API Number	`0001`	Minimum API level support.
Role	`primary_gateway`	Tells which gateway to prefer if there are multiple ones for close localities.
Location	`R.08.03.123`	Known position of the gateway. Within a building, a room number is provided. Otherwise, a GPS position could be used.
Supported Sensor Types	`temperature, humidity, occupancy`	The set of sensor types a gateway takes responsibility for and is competent.
IP Address	`2001:db8::2:42`	The address a peer can reach a gateway's interface in the local network.
Port	`50051`	The respective port number.
Signed Hash	`b86d04..d64bda`	Secure hash over previous properties signed by gateway's public key.

The properties *ID* and *Signed Hash* have special relevance in terms of authenticity (contributors to ≪R.6≫ and ≪R.7≫). Hence, we briefly elaborate on them. The *ID* uniquely identifies a gateway globally in a network. As an example, the ID must be unique for the whole network backbone of an intelligent building. So, a service client such as a HVAC system may only reach one instance for a given ID. For being able to verify the authenticity of the retrieved result, the ID contains the fingerprint of the gateway's certificate. This in turn is used to prove whether fingerprint matches the presented certificate during pairing (see section 4.2.2). If true, it further allows to utilize the *Signed Hash* property to verify the integrity of all properties before finishing the pairing.

4.2.1.2 Matching Capabilities on Peer Side

In a network, there might be multiple gateways that respond to a peer's query. Moreover, not every gateway is suitable for a peer. It is the peer's duty to select a suitable gateway according to the following strategies with respect to the received properties:

- *Location* should be close to the peer's location itself. Closeness is calculated by means of distance metrics or heuristics, e.g. a room number indicating the same section of a building is preferable over two distinct sections.

- *Supported Sensor Types* should form an non-empty intersection with the device's own capabilities of perception.
- If a gateway is already known to a peer, i.e. a gateway ID is known and trusted since a successful pairing, it is a preferable candidate to join its network if the other two properties are still fulfilled.

4.2.2 Pairing

While the discovery process identifies a suitable gateway, a peer usually does not know this gateway given it is new to a network. Hence, the peer needs to establish a trust relationship to this gateway.

As we do not presume a hierarchical public key infrastructure (PKI) which would require administrative competence we aim to exclude ≪R.14≫, all entities including any gateway generate self-signed certificates if no other means are available. The task of pairing is the commissioning of those certificates on both parties that are used for mutual verification of identities ≪R.7≫ and authenticated and encrypted communication ≪R.6≫ later on. The outcome, i.e. mutual authentication material, is stored locally on both peer and gateway. Specifically for the gateway, this information is also linked in the gateway's directory component (as described in section 4.4.1) as it is responsible to manage the communication channels to the peer.

The process of pairing involves a series of interactions between a gateway and a peer in order to mutually exchange and verify their certificates by an out of bound method. Figure 4.2 illustrates the flow of necessary actions that are elaborated below:

1. The regarded peer locates a gateway that matches its capabilities. As there are no known and trusted gateways in the current network, it initiates a pairing.
2. The peer starts a TLS session with the gateway. The peer's certificate P_{cert} is exchanged as part of a TLS client authentication. The gateway's certificate GW_{cert} is always sent to the peer as part of a server's role in the protocol. Both assume that there is no MitM attack and authenticity is provided. Hence, all communication from now is secured and authenticated (based on the previous assumption). Any of the following message exchanges is part of the same session.
3. The peer sends a certificate commissioning request to the selected gateway. It contains the peer identity. Further checks verify that the announced identity is the same as encoded in P_{cert}. The gateway refuses the request if the identity is already reserved. Otherwise, the entity temporarily accepts it and its associated P_{cert}.
4. The next step requires the only manual user intervention to verify that exchanged certificates are actually the same on both entities. For instance, an administrator

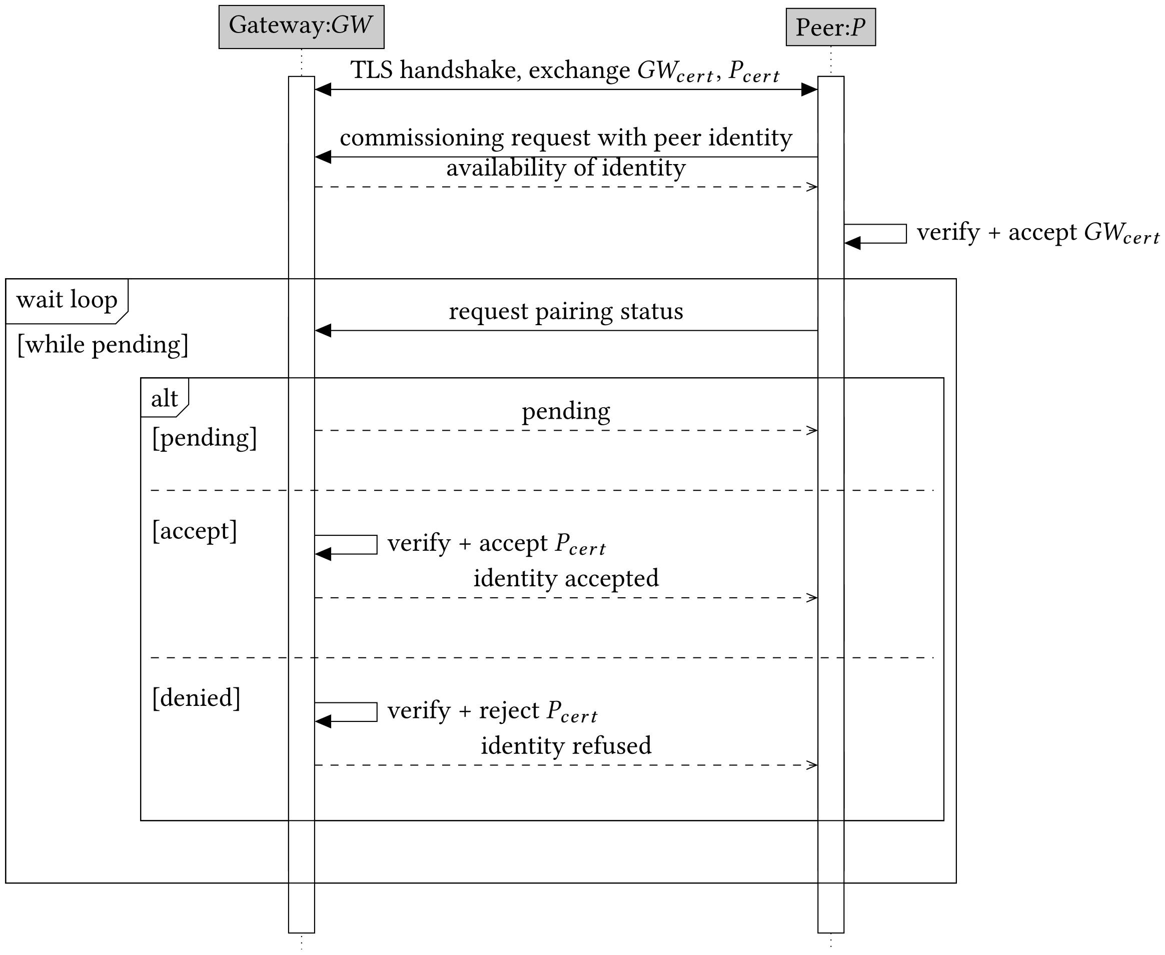

Gateway:GW
Peer:P
TLS handshake, exchange GW_{cert}, P_{cert}
commissioning request with peer identity
availability of identity
verify + accept GW_{cert}
wait loop
[while pending]
request pairing status
alt
[pending]
pending
[accept]
verify + accept P_{cert}
identity accepted
[denied]
verify + reject P_{cert}
identity refused

Figure 4.2: Pairing: sequence flow for a process of certificate commissioning on gateway and peer. All communication is part of an unauthenticated TLS session which is proved afterwards to be free of any MitM attack.

could note the certificates' fingerprint displayed on each device, and compare them with each other, i.e. providing an out-of-band verification. Only if this check is successful, he first accepts GW_{cert} on the peer device, then P_{cert} on the gateway. At the same time, matching fingerprints mean that all previous and future communication should be safe within the same TLS session. So, the original assumption now holds.

5. When GW_{cert} is accepted on the peer, it regularly queries the gateway about the status of pairing. It may take some time until the request is reviewed by the person in charge of the gateway. On success, the peer and gateway persist GW_{cert} and P_{cert}, respectively, for future communication. On failure, an exception handling routine should be invoked, which e.g., could connect to another gateway if discovery revealed suitable alternatives nearby.

4.3 Robust Communication Approach

A robust communication network is an important prerequisite for reliable orchestration of a number of peers, given that devices are part of dynamic environments. Due to the lack of a centralized infrastructure, the aforementioned components were introduced as building blocks for realizing self-managed networks, but need to be put now into a broader context to accomplish a reliable communication network.

This section proposes an approach how to continuously keep peers and suitable gateways connected albeit dynamic changes. In the following, we differentiate between responsibilities of peers in section 4.3.1 and a gateway in section 4.3.2, respectively, to accomplish this goal.

4.3.1 Continuous Peer-side Connectivity and Recovery

It is a peer's task to keep a stable and continuous connection to a compatible gateway in order to take part in computations and eventually also benefit from that. Accompanied with a dynamic environment, there are multiple eventualities and uncertainties as pinpointed in section 3.4.4.2. A peer needs to overcome them.

The finite state machine in fig. 4.3 depicts a robust approach that applies core aspects of self-managed networks and combines them to achieve robustness and quick recovery for changing circumstances (needed for ≪R.12≫, ≪R.13≫). In particular, it handles

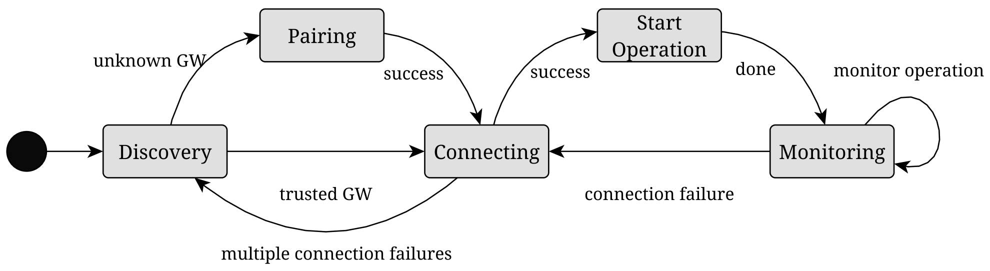


Figure 4.3: Continuous peer-side integration into a network.

the following cases:

4.3.1.1 Attaching to available and compatible Gateway

If a peer enters a network, it starts *Discovery* (section 4.2.1) to find all available gateways. This renders a list of gateways and their capabilities. If it was already associated with a gateway in the past (so it trusts the device), it prefers connecting to this device in the case that capabilities, i.e. location, supported sensor types, etc., still match its requirements. Otherwise, it selects the best-fitting gateway with respect to its advertised capabilities and starts *Pairing* process (section 4.2.2). Once successfully paired, it enters the *Connecting* state.

From there, the peer establishes a communication channel to the gateway and enters *Operation* mode. This means it is ready now to receive commands and process SMC matters. Implicitly, a monitoring component is started to continuously observe the healthiness of the communication channel.

4.3.1.2 Handling Failures

If the monitoring component detects a problem with communication, the peer first tries to reestablish the existing communication channel a few times. Therefore, it assumes that gateway's network address is still the same. This strategy covers minor unreliabilities in the network, such as failures on the transport layer.

In case of an unrecoverable failure, i.e. the gateway is not reachable anymore, it falls back to *Discovery* for the following reasons:

- Locate the gateway previously connected to, if peer is still within the same network, but network address changed.
- Look for another suitable gateway if the first point does not work out.
- Determine the gateway responsible for a different network the peer recently connected to, for instance, if it moved.

If discovery reveals the previous or another suitable gateway, the peer invokes aforementioned steps to attach to it as described in section 4.3.1.1.

4.3.2 Gateway-side Monitoring and Handling

While a peer does the heavy-lifting regarding a robust communication to a gateway, the latter contributes its share for stable operation. After all, the gateway needs to know exactly which peers are currently available and whose communication channels are established successfully in order to involve them in SMC operations. Collecting and maintaining this information is part of the gateway's directory component, as described in section 4.4.1.

The employment of two mechanism on gateway side allows evidence if peers become unreliable:

- *Periodic heartbeats* — a peer is advised to regularly send a simple ping message to the gateway that it is still alive. If there is no message for too long, it is safe to mark the peer for revocation later on.

As a side note, the ping message may carry an additional payload, containing recent peer-related information that needs regular updates, e.g. the current port number at which other peers may contact this peer.

- *Monitoring of communication channels* — analyzing response patterns of a certain peer provide insight if something is wrong, potentially related to communication. If all communication channels stall to this peer, i.e. no replies are received after several requests, or are broken, the gateway must cleanly tear down all channels.

If one of the mechanism gives clear evidence about issues with current means of communication, the gateway resets and tears down all remaining channels and frees resources. On the one hand, this approach signals the respective peer that it needs to reconnect, given any channel is still functional. On the other hand, any potential conflicts for future attempts to attach to the gateway are fixed as resources and queued commands are flushed.

4.4 Task Orchestration

Having laid the groundwork for forming spontaneous peer networks attached to a suitable gateway in the previous sections, these constructs shall be the basis to realize privacy-preserving services on behalf of SMC. In this context, the gateway is of fundamental importance to get together the out-side world, i.e. service clients interested in gathering processed data, and the internal sensor network represented by a fluctuating number of peers.

This section focuses on responsibilities and contributions of a gateway to achieve an robust orchestration of tasks from an end-to-end perspective, while illustrating at the same time how peers integrate with the gateway to safely provide their SMC resources.

4.4.1 Conceptual Overview and Data Flow

Before highlighting specific aspects of task orchestration, it is beneficial to get a top-down view on fundamental components and how they are related to each other first, which make up for large parts of a gateway's functionality.

Figure 4.4 illustrates all relevant components and is the basis for all explanations. The data flow starts with an incoming request (top left corner of the figure) stating a task for the acquisition of processed data, as usually invoked a service client. This kicks off a series of actions while data blazes a trail through related components. The flow and details about these components are presented in chronological order in the following breakdown.

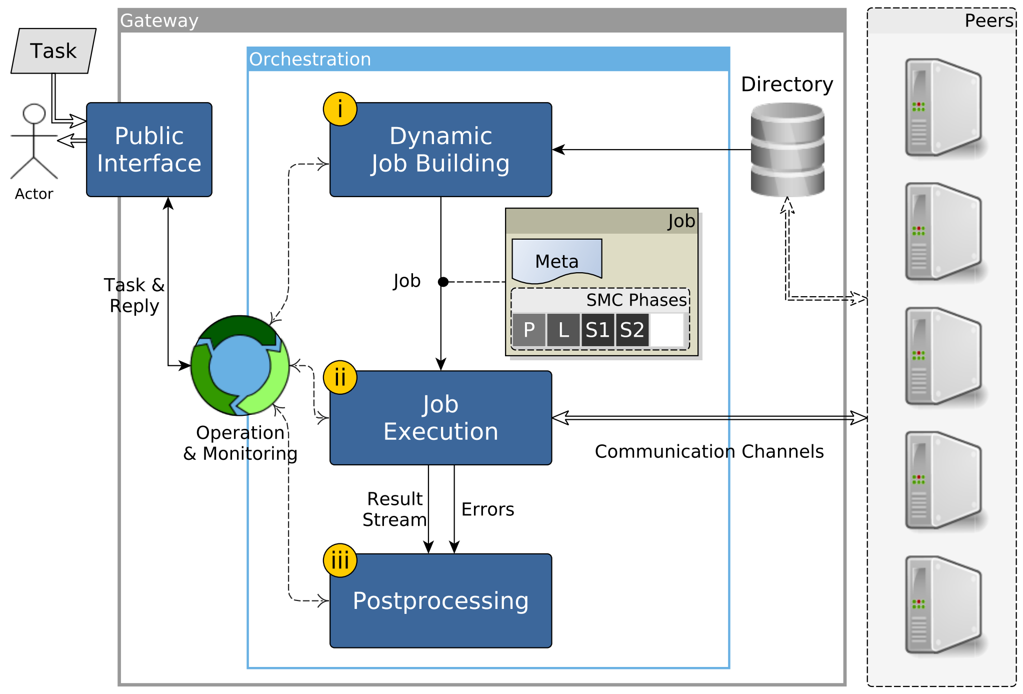


Figure 4.4: Components for robust task orchestration.

4.4.1.1 Task Description

A task is a generic description for service clients' demands for retrieving processed data from a specific location represented by a computation network. It acts as an entry point for task orchestration.

Details are elaborated in section 4.4.2.

4.4.1.2 Public Interface

The public interface provides a standard public-facing way for service clients to retrieve information without knowledge about internals (≪R.9≫). Due to the fact that any node can potentially take the role of a gateway, the interface of an interesting gateway is locatable by means of discovery mechanism as discussed in section 4.2.1.

More specifically, the interface component handles incoming requests containing task descriptions for execution. To do so, it hands the task over to the orchestration component and waits for a result. A client decides to retrieve the result either in a synchronous fashion, i.e. blocking for it, or to be notified asynchronously by the interfacing component.

Moreover, it verifies the validity and integrity of a task and whether the originating service client is authorized to issue this request by analyzing the respective fields of the task description (see section 4.4.2).

4.4.1.3 Directory

The directory component maintains a detailed view about a gateway's associated peers. It is the central data collection point for encompassing the dynamic aspects within a gateway's controlled part of a network. To the most part, it keeps track of the following information for each peer:

- Availability
- Associated sets (e.g. a peer belongs to an specific office)
- Supported sensor types
- Endpoint description (i.e. IP addresses, port)
- Endpoint authentication (i.e. an authentication ID and at least one valid certificate being provided by each peer during pairing)

This data stems from a variety of sources in the architecture. First, the certificate commissioning during pairing (section 4.2.2) informs the directory about peer-specific authentication material to provide a basis for secure and authenticated communication. Furthermore, the mechanism of regular heartbeats (section 4.3.2) conducted by a peer, contributes two pieces of information to the directory: it learns the current endpoint address of a peer by deriving it from the incoming ping message. In addition to that, the heartbeat message may contain an additional data field, specifying the port number at which the respective peer expects other peers to direct their requests for communication when forming an SMC network. During a computation, learned endpoint addresses and ports are presented to the participants so they know how to reach out to each others. Secondly, the heartbeat is a strong indicator about the availability of a peer.
Last but not least, the directory needs to be aware about the skills and properties of a peer. In particular, these are associated sets and its supported sensors. While the former one is configured manually after the pairing by an administrator or alternatively links to a default set if unspecified, supported sensors are reported by the peer on interaction with the gateway. Therefore, the directory exposes a separate registration interface. When attaching to a gateway, peers get in touch with the interface to inform about available sensors they want to contribute to the network.

Beside the above state-keeping, another sub-task of the directory is the management of communication channels to all registered and available peers. While establishment of these channels is driven by the peers themselves (as described in section 4.3.1), they need to be hold and monitored. Like this, the orchestration component can simply request needed communication channels for an upcoming computation session from the directory. This takes management work from other components that interface with peers.

4.4.1.4 Orchestration

When receiving a verified task from the public interface component, it needs to be executed on behalf of a dynamic set of peers collaborating via SMC. In order to largely meet applicability goals as stated in section 3.5.3, with an emphasis on robustness and flexibility, the orchestration component divides the process into three interlocked subprocesses. As illustrated in fig. 4.4, the basic flow is characterized by three consecutive steps:

(i) *Job Building*
A job is a SMC-specific sequence of instructions, called phases. It is formed by combining current states about the peers from the directory and demands of the provided task description.
Section 4.4.3 elaborates on this step in detail.

(ii) *Job execution*
The augmented job is executed in a synchronized manner. The instructions for each phase are transmitted to the peers and all results are collected before proceeding with the next phase.
Section 4.4.4 elaborates on this step in detail.

(iii) *Postprocessing*
While executing the job, incoming results of any phase are streamed directly to the postprocessing component. Its task is simple: it verifies consent among all peers' responses. Inconsistent peers are logged and reported to the directory component. Behavioral analysis opens up possibilities to identify and ban suspicious or misbehaving peers. However, this is beyond the thesis's scope.

An all-embracing unit, named *Operation & Monitoring* unit in fig. 4.4, waits for the final results and returns it to the interface component, given everything goes smooth. In case of any problems with peers' collaboration or their communication link, this unit controls the resumption of a halted job while dropping erroneous peers as default strategy. If a job's progress makes any repair impractical, e.g. if a SMC session was already running for some time, the job is started from scratch. Section 4.4.4.3 describes error handling in more detail.

4.4.2 Generic Task Description

A generic task description is an abstract instruction format that establishes interoperability between service clients and peers. It provides service clients with a simple, but

Table 4.2: Shows all fields of a task description. It enables a service client to interface with a peer network in an abstracted way.

Field	**Example values**	**Description**
Set	`"offices", "toilets", "meeting rooms"`	Specifies a group of peers whose participation is desired. If not set, it assumes all peers of a network.
Purpose	`"HVAC", "personal comparison"`	Service client must state the reason for data acquisition.
Data Source	`temperature, occupancy_count`	Selects a specific type of data if multiple sources, e.g. sensor types, are supported.
Preselector	`now, last_xMins, last_xHours`	Controls the amount of historical raw data to process on each peer.
Aggregator	`sum, average, std_deviation`	Specifies the aggregation method for the secure computation. It is an identifier representing an specific *aggregation entity* (see section 3.3.2).
Ticket Signature	`084c79..839cc0`	Signature by gateway that signs all previous fields from here.
Timestamp	`1487868900`	States the time point when the request is issued.
Query Signature	`859f0b..c047cd`	Signature by service client that signs all previous fields from here.

powerful tool to describe their need for certain information that can be extracted from a set of data providers, i.e. peers in our case.

Table 4.2 depicts the structure of a task description. A task's generality covers a broad range of use-cases outreaching the state ones in the scope of this thesis by implementing central aspects of the generalization model as defined in section 3.3.2. In particular, the fields `Set, Data Source, Preselector` provide a generic way to specify the input to the computation, hence, defining a set of *Producers* and which data they contribute to the computations. The field `Aggregator` relates to the entity of the same name in the model, so specifying the method applied during secure computations. The resulting generality is an important contributor to the goal that external service clients do not require any specific insights or knowledge about SMC or the peers.

The purpose of the remaining fields, i.e. `Purpose, Ticket Signature, Timestamp, Query Signature`, is manifold, but concentrates in two coherent objectives in the scope of this thesis: as a task is accessible for all peers during a computation, it provides a foundation for transparency (≪R.4≫). This is the prerequisite for intervenability (≪R.5≫), i.e. a peer knows the higher-level goal and origin of the request to decide whether the computation is appropriate to be executed. Further, it is a proof for authorization and integrity of a task issued by a service client.

4.4.3 Turning Tasks into Multi-Phase Jobs

A generic task establishes a common layer for interoperability between service client's demands and peers realizing it, but does not solve properties of SMC being inflexible and prone to fail in dynamic environments.

Therefore, this section presents the concept and rational of the multi-phase job which is built upon both the information of a task description and current knowledge about the dynamic environment.

4.4.3.1 Components

A job comprises two major components:

- *Meta-data* — list of applicable peers with their configuration. This is depicted in section 4.4.3.2.
- *SMC instructions* — sequence of specialized instructions, called phases, to control the SMC operation in a flexible and robust manner. This is elaborated starting from section 4.4.3.3.

4.4.3.2 Adaption to Peer Variations in Dynamic Environments

To accommodate to changing circumstances in dynamic environments, a job needs to integrate instantaneous states about all peers associated with a gateway. At any time, it is possible that previously active peers become unresponsive and therefore, the same set of peers would not work anymore for further computation sessions.

The required information is acquired from the directory component. As elaborated in section 4.4.1.3, it provides, for instance, current information about a peer's availability, its associated sets and supported sensor types.

Using this information, a gateway decides on which peers to include for a task's intended computation. It proceeds as follows in ascending order:

1. **Translate set**
   Map `Set` identifier of a task to a list of potential peers. As set mappings are configured manually during pairing, it is comparable to a static dictionary containing several peers that have associated with the specific set in the past.

2. **Match capabilities**
   Ignore peers that lack any of the capabilities stated in the task, e.g. supported sensor types of a peer does not support a task's required `Data Source`.
3. **Select available peers**
   Select only the peers which are currently available and recently acknowledged their willingness to the directory. This is the most important information to adapt a task to the current situation. Concretely, it concerns the requirement for adaptiveness ≪R.12≫.

The result is a list of participating peers that should be ready to process the task. It is provided as meta-data in the job.

4.4.3.3 Task Segmentation into SMC Phases

With the help of aforementioned meta-data, the orchestration component knows by now which peers are planned for the next secure computation session. For efficient execution on a SMC network, it is beneficial to segment the task into multiple sub-tasks.

Therefore, a computation session is formed by a combination of differently-purposed phases (that are chained as described later on in section 4.4.3.4):

P **Prepare phase**
This phase provides the peers with two major pieces of information: first, it disseminates the original task description to all peers. Secondly, it enumerates all peers that are intended to collaborate in the secure computation. As this information is forwarded to all related peers, the gateway provides a transparent view on the intended operation (≪R.4≫). In particular, the phase depicts the following information:

- Original task description (as elaborated in section 4.4.2)
- List of participants with
  - *Temporary peer ID* — defines a peer's individual rank within a set of peers. It provides the peers with a common context to uniquely address each other without the need to defer it from IP address and port combinations.
  - *Endpoint description* — specifies the IP address and port combination. An interface will be available at this endpoint during a SMC session in order to establish an interconnected network.
  - *Authentication ID* (optional) — relates to a peer's fixed identity that is used during the process of authentication with the gateway. As it also identifies a peer's certificate, a peer may use it to randomly verify

the authenticity of its collaborative parties, given it has access to the respective certificates by any trusted means, e.g. a pre-configured set of certificates of trusted peers.

Under normal circumstances, the authentication ID will not change and uniquely identifies a peer, in contrast to other non-permanent attributes, e.g. an IP address.

On peer side, this information lays out a peer's foundation for decision making. Either, a peer starts preparatory work based on the received information. This includes allocating a stream of input data that takes values from the selected `Data Source` and filters it according to the `Preselector` setting. Furthermore, it initializes SMC components and pre-processes data for the secure computation if required.

Otherwise, the command's transparency gives a peer the chance to analyze the same information to decide in an early stage whether to refuse collaboration temporarily or even in the long term (≪R.5≫). So, a peer may deny a request due to a suspiciousness that it might reveal private data, e.g. if participation is very low or most of the participants are sitting on the same host device. This can become a threat with respect to the supported attacker model as discussed for the employed BGW protocol in section 2.2.5. Or, it may limit requests for a certain combination of data input and employed SMC aggregation to an upper bound per minute to save resources.

Although peers dropping out for a session occasionally, the early stage of handling allows the gateway to repeat an adapted phase for reconfiguration, without having to start from scratch. As a result, established communication channels and allocated resources on remaining peers' side can be reused. It increases performance and robustness (≪R.10≫, ≪R.13≫).

L **Link phase**
This phase instructs all peers previously enumerated in the Prepare phase, to interconnect with each other, being a prerequisite for the upcoming secure computations as part of SMC. Like this, sessions can be started on-demand involving a dynamic set of peers (≪R.11≫). Hence, it does not require particular peers to maintain a permanent network, causing a constant high usage of peers' resources and additional network load. The peers report the results of individual connections to the gateway, including any peer-specific problems occurred during connection establishment.

In particular, each peer tries connecting to the peers' endpoints known since the Prepare phase. If interconnecting works flawlessly, all related peers immediately signal a positive outcome to the gateway, which, in turn, continues with the next phase. If peers fail to establish a connection to one or more other parties until a given timeout, they inform their gateway about the problem. Knowing the specific origin of fault, the respective peer IDs are appended to the error report. On broad consensus, the gateway may decide to exclude the faulty peers and supply the remaining peers with a new list of parties as part of an adapted Prepare phase. As a result, the gateway quickly recovers the computation network to continue operation (≪R.13≫).

S **Session phase**
This phase triggers a secure computation on all linked peers based on the common task. Specifically, it signals the peers to simultaneously start running the functions that are associated with the `Aggregator` field of the task defined during the Prepare phase. Having finished computations, each peer is in possession of identical results if it actively collaborated. Results are reported individually by each peer to the gateway. Given received values are consistent (verified as part of the postprocessing in fig. 4.4), orchestration component provides them to the task's originator.

For flexibility and efficiency reasons, multiple computations can be run in a row by repeatedly rolling out the Session phase. This is interesting for long running services which continuously needs fresh input from peers of a certain location in short intervals, i.e. as many requests so that it is more efficient to keep up the current SMC network.

Beside the kick-off announcement, the Session phase introduces additional parameters to optimize the operation of secure computations based on expectations about the amount of data to process, knowledge about the peers from the directory and historical experience gathered during orchestration. For instance, there is the parameter `parallelBatchOps` that sets the degree of parallelism to process data inputs in batches. An orchestration component may increase parallelized operations if past requests took quite long although peers are running on potent hosts. If resulting execution times decrease, it may remember this adaption in future. As parameters are quite specific for the employed SMC solution, these fields are optional and not deepened in this context.

To summarize this part, three phases, namely Prepare, Link and Session phase, are introduced to segment a generic task (from a service request) into SMC-specific instructions. This scheme and arising possibilities contribute strongly to achieve robustness (≪R.13≫) though supporting intervenability (≪R.5≫), performance (≪R.10≫) and on-demand sessions (≪R.11≫) in the overall orchestration.

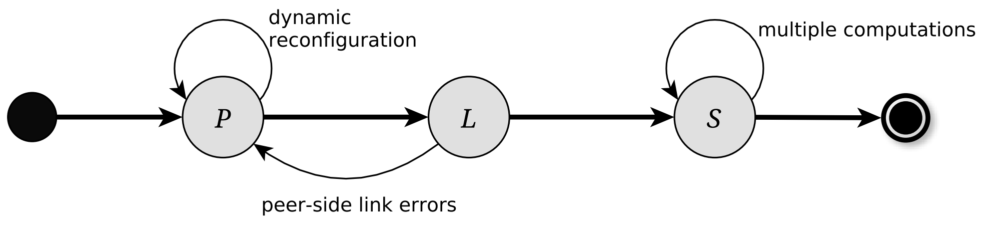


Figure 4.5: Default sequence of phases and possible deviations. *P*, *L* and *S* denote the Prepare, Link and Session phase, respectively. Thick arrows illustrate the default sequence of phases if the setup works as expected and only demands for a single computation per session. Thin arrows depict adaption to usual flow in order to either recover from unexpected changes in the set of properly operating peers, or to extend number of computations per session.

4.4.3.4 Sequence of Phases

As introduced in the beginning, a job comprises a certain sequence of aforementioned phases in order to realize a task on behalf of SMC. It can be imagined as a buffer holding multiple phases for instructing the controlled peers, which are invoked one-by-one by decision of the gateway. The finite state machine in fig. 4.5 illustrates all possible sequences of phases a gateway might roll out and which peers would accept, based on the task's requirements and the current dynamic situation. It simulates allowed phase transitions for a single computation session. Once reaching the end state, the current computation session finishes and denies any further usage as resources are freed.

As depicted in fig. 4.5, the roll-out of phases is directed under normal circumstances as follows: first, a Prepare phase applies the configuration to all related peers. Then, the Link phase instructs them to interconnect to each others, being a precondition for SMC operation. When done, one or several Session phases trigger specified secure computations, each. This means that there might be multiple consecutive Session phases, triggering the same secure computations on top of the current peers' input. Results are transmitted individually by each peer to the gateway before the current phase ends.

Possible deviations from normal operational flow are needed to handle unexpected circumstances arising from the nature of dynamic environments:

- *Repetition of Prepare phase* — intends to reconfigure peers dynamically. It is necessary if peers drop out or refuse the current task.
- *Transition from $L \rightarrow P$* — has the same goal to invoke a reconfiguration of all collaborative peers, since some hosts are not reachable from each other.

Origins of both special cases are explained in more detail in the respective phase descriptions of section 4.4.3.3.

4.4.4 Executing Jobs

A job encompasses all essential information and instructions for orchestration the task it is based upon. For its realization, the job execution component follows a step-wise approach with the ability to handle errors, as described as follows.

4.4.4.1 Preparation

Before a new job can be executed on behalf of the peers being elected for secure computations (as enumerated in the job's meta-data), some preliminary work has to be done.

First, the orchestration component retrieves a bi-directional communication channel for each selected peer from the directory component. As peers are filtered in terms of availability, i.e. being reachable and having resources for a new SMC session, no extensive checks are needed at this step. These channels are used as command, control and reply channels for the instructions further on. Moreover, they are buffered, i.e. not blocking, to behave like a mailbox system. This allows entities to proceed with computational-intensive operations first and check mailboxes if input is needed.

In addition to that, an unique session ID is generated for a new job. By attaching this session identifier to each message directed to the peer network, peers are able to support multiple tasks at the same time.

In case of a resumed job, i.e. a job that is stopped due to an error, then reconfigured and queued again for execution, aforementioned preparation work is skipped as communication channels and session ID are already allocated. However, if the adapted job excludes peers which were previously considered, respective channels are teared down in this step.

4.4.4.2 Phase-based Synchronous Execution

Since preparation is finished, job execution starts. A job's phases are processed according to following consecutive steps:

1. The next phase from a job's buffered series of phases is fetched. A phase contains all the instructions intended to be sent to all related peers.
2. The current phase's instructions are wrapped into an envelope, personalizing the message for a specific peer. In particular, these fields are:
    - *Session ID* — relates to current computations the instructions are intended for. It is needed to support multiple SMC sessions in parallel on peer side.

   - *Peer ID* — defines a peer's rank within the set of participants where all of them are identified by temporary peer IDs for the current session.

3. The individualized messages are transmitted to the respective peers asynchronously via the previously established communication channels.

4. On peer side, incoming instructions open a new session or link to an existing one based on the supplied session ID. The instructions are executed on behalf of the supported SMC resource if they do not violate allowed phase transitions as illustrated in section 4.4.3.4. Replies are sent back using the same communication channels as instructions arrived from.

5. The orchestration component waits for all peers to process sent instructions. Any incoming replies are streamed directly to the postprocessing component for consensus checks and aggregation (see item (iii) in section 4.4.1.4).

   Noticeably, the job execution component waits until all peers of a phase finish and reply before proceeding. As a result, it acts similar to a synchronization barrier in a phase-wise manner. The rationale behind this behavior relates to the expectations of SMC. It requires all peers to be in the same state in terms of preparations and done secure computations in order to proceed with further operations. Otherwise, the peer network gets out of sync and dependencies on expected inputs are not fulfilled, hence causing a deadlock.

6. Given that all peers' replies show a successful execution, this process starts from step 1 again until all phases are executed on behalf of the peers.

Unexpected situations that interfere with any of these steps, are tackled in the error handling, as described in section 4.4.4.3.

4.4.4.3 Error Handling

There are cases in which job execution must deviate from aforementioned steps. In particular, it is triggered by one of the following scenarios:

- Communication channel fails for specific peers
- Peers do not reply within a given time frame
- Peers refuse participation for a given session. Either, this is due to the unavailability of resources, or the peer's reasoning decides to opt out, e.g. to minder the risk for disclosing sensitive data

In all these cases, a job pauses execution at the currently handled phase. Control is handed over to the *Operation & Monitoring* unit who is responsible for the overall procedure of task orchestration (cf. fig. 4.4). It tries fixing the issue differently according to the progress which phases have been accomplished so far. Following the possible phase state and transitions from section 4.4.3.4, an error might occur either during a Prepare, Link or Session phase.

Given the case that peers become unavailable or deny collaboration during the first two phases, it is quite easy to roll them out. The *Operation & Monitoring* unit creates a new job based on current states of the directory, compares new and paused job in terms of participation, and drops those peers from the paused job missing in the new one. In both cases, it needs to resume from the Prepare phase to transfer the corrected list of participants to all peers. Already allocated resources on peer side can be preserved in large parts. The modified, but paused job is resumed by queuing it in the job execution component again.

If a problem occurs during a Session phase, it is more serious. In a first attempt, the *Operation & Monitoring* tries to repeat this phase. If the outcome is still erroneous, a new independent session is spawned which adopts the current task description. Then, it is put to execution. The present job is teared down instead to free acquired resources.

In summary, the step of error handling is a fundamental part to put together the benefits of task segmentation into phases to achieve robustness (≪R.13≫) in the overall system.

Chapter 5

Implementation

Having laid out the design for our orchestration framework in the previous chapter, the following sections present the concrete architecture of the implementation and highlight specific details that are essential to achieve the overall goal for robustness and flexibility.

For realization, we attach great importance to a strict separation of responsibilities. The result is a high degree of modularization that enables to easily exchange or modify certain components without affecting the system as a whole. This makes the prototype interesting for future work. Moreover, we adopt technologies and frameworks that are known to work well in dynamic environments.

The heart of the implementation is the orchestration component *FlexSMC*. By moving the SMC logic into an independent service, it is highly reusable for any SMC framework other than Fresco as well.

5.1 System Architecture

Due to the distributed nature of our design, a system consists of several instances of an application and means of communication between them. Concretely, fig. 5.1 illustrates all constituent parts, with gray boxes visualizing self-contained applications which mainly interact via messages being passed to each other. Noticeably, beside the obvious communication links in between gateway and peer constellations, inner-host responsibilities have been split as well.

In total, we differentiate three major system components. This is *FlexSMC*, *Flex Connector* and interactions powered by *Protocol Buffer messages*.

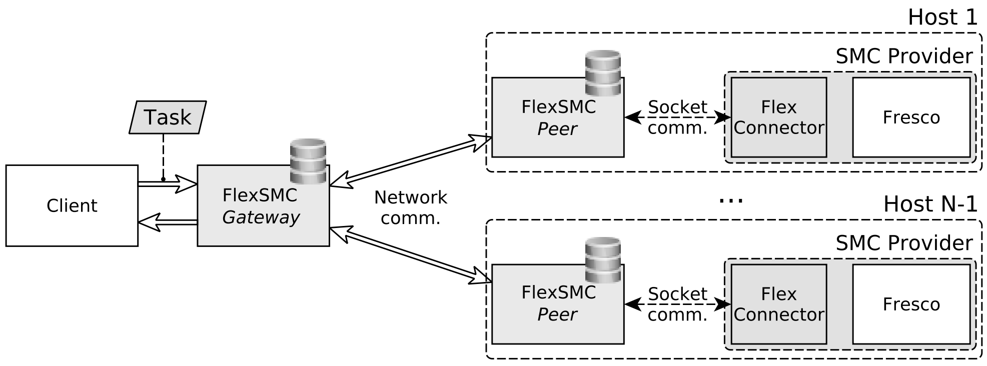


Figure 5.1: System overview involving all relevant parties. Gray boxes visualize parts that we have contributed on behalf of this thesis. Fresco is the SMC framework which is integrated as a reference to provide secure computations.

FlexSMC

Language:	Go
Project source:	`https://github.com/grandcat/flexsmc`

FlexSMC is the core contribution of this thesis implementation-wise. It is the key to bring together self-configuration, forming secure and authenticated networks, and robust task orchestration on behalf of locally attached SMC providers. Based on the runtime configuration, a FlexSMC instance inherits the role of either a gateway or a peer. Nevertheless, divergent roles can be operated on the same host without interference.

Section 5.2 elaborates on its implementation in more detail.

For realization, it is based upon two other important projects which were developed in the scope of this work as well:

- **Zeroconf**

Language:	Go
Project source:	`https://github.com/grandcat/zeroconf`

  Zeroconf is a minimalistic, but robust implementation of zero-configuration networking aspects. In particular, it focuses on service discovery in local networks by implementing essential aspects of both RFC 6762 (Multicast DNS) [40] and RFC 6763 (DNS-Based Service Discovery) [39].

  Although there are great implementations for Linux, e.g. Avahi [41], the integration with the Go language is widely unsupported and seems to over-complicate development with inaugurated dependencies.

  A self-contained, but incomplete implementation for zero-configuration exists in the Go community. Zeroconf adapts several parts from it, but rewrites large fragments to fix its flaws in robustness, reliability and maintenance. Thereafter, it is comparably to a new implementation. Noticeably, we reference the original project in source code's license file.

- **srpc**

Language:	Go
Project source:	`https://github.com/grandcat/srpc`

  srpc is a tiny layer around Google's remote procedure call framework *gRPC* [42], which is an easy-to-use and robust communication layer on top of language-agnostic protocol buffer messages [43]. The srpc project exposes the same functionality as gRPC, though transparently integrating a modularized architecture for more flexible authentication methods, including a mechanism to commission TLS certificates on the fly. Moreover, it integrates the project Zeroconf for resolving services.

  More details about the composition of srpc's implementation are given in section 5.2.2.1.

Flex Connector (SMC provider)

Language:	Java
Project source:	`https://github.com/grandcat/flexsmc-fresco`

Flex Connector is tightly coupled with the FlexSMC project. Its task is to provide Secure Multi-Party Computation as a controllable resource, without having to deal explicitly with its specifics. It is also called SMC provider in this context.

More specifically, it is a stand-alone application compiled to a single Java Archive (`.jar`) that employs and controls the SMC framework Fresco [7] under the hood to provide secure computations. It manages opened sessions, and does some basic validation and reasoning about a requested task which, in turn, might be refused conditionally in order to preserve privacy. For input to the computations, Flex Connector is intended to be integrated with its host's perceiving capabilities as data source for requested secure computations, e.g. takes sensor readings such as CPU's temperature and uses it for collaboratively calculating the average. Currently, inputs mainly consist of fixed values for reproducibility, though respective interfaces are available for extensions. Similar, validation and reasoning about a task is quite basic as it follows a strict rule set instead of dynamically learning from bad requests.

Communication between FlexSMC and this connector is provided by means of protocol buffer messages exchanged via a local gRPC interface. Details on that are illustrated in section 5.3.3.

Communication via Protocol Buffer Messages

Language:	language-agnostic
Project source:	`https://github.com/grandcat/flexsmc/tree/master/proto`

For stable means of communication between either two FlexSMC instances, or a FlexSMC and a Flex Connector instance, we specify a global set of platform-independent message definitions. Therefore, we utilize the protocol buffer scheme [43]. It comes with a compiler to generate language-specific message (de-)serialization code, i.e. turning the abstract specification into Java and Go code in our case. It resembles an important component in this design to guarantee smooth interoperability on the one side, and to preserve compatibility to older instances on the other side, since definitions might evolve with time.

Section 5.3 elaborates more precisely on employed message definitions and remote procedure call (RPC)-based communication.

5.2 FlexSMC Implementation

FlexSMC is the heart of this work's implementation. Written in the Go programming language, its goal is to make SMC practically usable in distributed and dynamic environments. This section provides deep insights about the architectural design and its components first. Thereafter, important aspects for the realization of task orchestration are illustrated with an end-to-end perspective, involving both tasks of gateway and peer, as it is a building block for enabling privacy-preserving services on top of SMC.

5.2.1 Design Facts

To make FlexSMC a sustainable solution in demanding environments and reusable for ongoing work, special emphasis during development is paid to the following aspects:

- **Modularity and Flexibility**
  FlexSMC consists of a variety of separated modules which interact with each other using clearly defined interfaces. The result is a strong separation of components as elaborated in section 5.2.2. Instead of being tied to a particular use-case, all fundamental structures are further designed to be dynamic, easily extensible and exchangeable to make the implementation highly flexible.
- **Stand-alone**
  The application and almost all of its dependencies, including discovery and networking, are compiled into a statically linked binary. As the Go compiler allows to easily cross-compile for other targets such as ARM (e.g. for Raspberry Pi), the deployment of the software is very easy. Moreover, the overhead of such an

application is significantly lower. The only loose dependency is the SMC provider, attached by message passing.

- **Concurrency**
  Inner architecture is built with concurrency in mind. Decoupled parts run asynchronously on behalf of lightweight Go routines that communicate by passing messages via Go channels. Moreover, task orchestration is designed to enable parallelism using a worker pool.

- **Loose coupling**
  A characteristic feature of this architecture is a loose coupling to its sole dependency, namely the SMC provider. Instead, its integration is realized by message passing and a well-defined interface. This renders it possible to restart the SMC framework on failures, use different programming languages, and exchange the SMC provider on the fly, without having to restart or redeploy FlexSMC.

5.2.2 Architecture and Components

FlexSMC's flexibility and manifoldness tribute to a relatively large code base, considering that this is a prototype implementation. To keep complexity low during development, we attached importance to modularization and abstraction. The result is a manageable amount of modules, from which a large fraction is easily exchangeable.

Figure 5.2 depicts an overview about the building blocks of FlexSMC. The implementation can be divided into three major functional blocks. As elaborated with more details as follows, this is the common base layer called *srpc*, and functional responsibilities for both gateway and peer, respectively.

5.2.2.1 srpc: Common Base Layer

For shared functionality in terms of secure communication, authentication and discovery integration, FlexSMC settles on top of the *srpc* framework that is also contributed as part of this thesis. It features four fundamental components as follows.

Discovery employs multicast messages in the local network to provide a decentralized service directory. It is provided by *Zeroconf* that is based on multicast DNS (mDNS) and DNS Service Discovery (DNS-SD). Endpoints of dialed instances, identified by their service name, are resolved automatically on behalf of these techniques. It further watches for updates about an endpoint's address to which currently a connection is established. On change of the destination's IP, new connections deploy the updated IP.

Implementation: `zeroconf`, `srpc/registry/*`

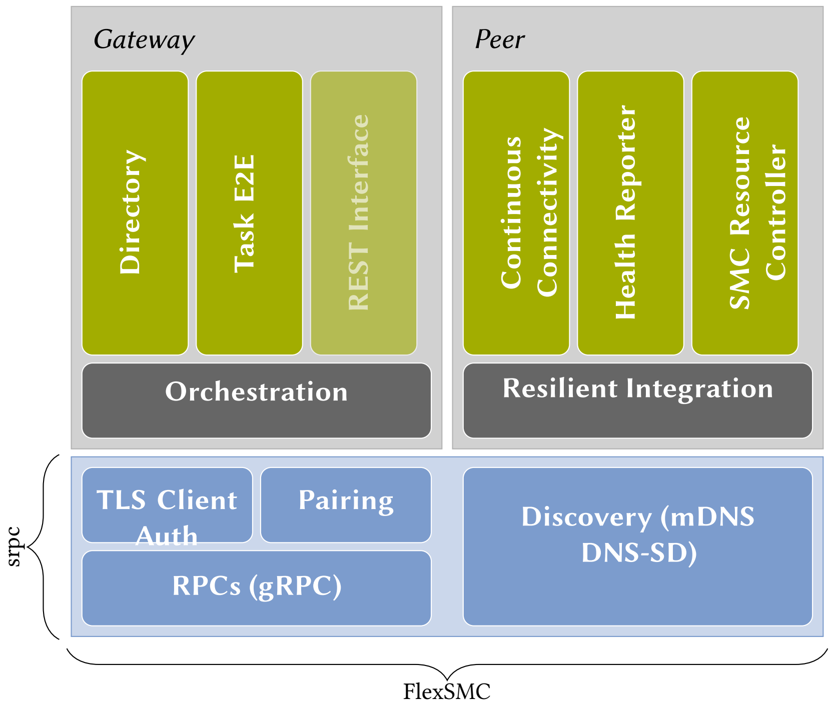


Figure 5.2: Layered node architecture of FlexSMC. The blue bottom layer depicts the shared core layer named *srpc*. The top layer parts show responsible components for gateway and peer, respectively. The indicated REST interface is not part of this thesis and open for future work.

Remote Procedure Calls (RPCs) are the basis for all communications via network and local sockets. RPC calls connect FlexSMC instances via networked channels on remote side, and establishes a communication link between FlexSMC and a SMC provider via local sockets on host site.

Implementation-wise, gRPC [42] provides the base layer for remote procedure calls, but is enveloped by a thin layer to modularize integration of authentication and pairing abilities. Messages are specified, serialized and de-serialized using protocol buffers.

Implementation: `srpc`

Authentication (TLS Client Auth) employs client-side authentication to enable mutual identification and verification of learned identities. Therefore, it manages a system-independent directory with X.509 certificates of trusted peers. Each certificate is associated with a role to allow several certificates per peer in parallel, e.g. primary, backup and revoked certificates. Authentication module persists all certificates in local storage, and are restored on startup.

Implementation: `srpc/authentication/*`

Pairing realizes the dynamic commissioning of self-signed X.509 certificates between a gateway and a peer following the principle of trust on first use (TOFU), but expects an out-of-band verification by a person to ensure that packets have not been altered by a third party. Pairing utilizes the fact that the TLS protocol implicitly exchanges certificates in both directions if client authentication is employed. When accepted on both sides, certificates are stored in respective local stores on behalf of the authentication component. This is elaborated more precisely in section 4.2.2.

Implementation: `srpc/pairing/*`

5.2.2.2 Gateway: Orchestration

The following components realize a gateway's functionalities with high emphasis on the realization of its orchestration capabilities.

Directory maintains a map of registered peers, their properties (associated sets, supported sensor types, etc.), their states about availability, and manages bidirectional communication channels to these peers.

Implementation: `flexsmc/directory/*`

Task End-to-End Orchestration is manifold: the orchestration transforms a generic task to a SMC-specific job within a modular *preprocessing pipeline*, incorporating the current situation from the directory component. Valid jobs are enqueued for execution. A free *worker* routine self-assigns a queued job, opens communication channels to the respective peers, and consecutively sends job's instructions (sequence of phases as described in section 4.4.3.4) to them. Replies are handled by an *aggregator*, responsible for postprocessing. Any unexpected errors hold the current job. A simple, autonomous error handling is done by the parent orchestration component.

Implementation: `flexsmc/orchestration/*`

REST Interface opens an RESTful API to third parties in the network in order to submit requests for data acquisition. Although having implementing a native request interface in Go, it is not connected to a server instance for offering it as a REST endpoint. Hence, an RESTful API is open for future work.

5.2.2.3 Peer: Resilient Integration

The following components realize a peer's functionalities with high emphasis on the realization of its robust network and SMC integration.

Continuous Connectivity maintains robust communication channels to a gateway and restarts depending components, i.e. the SMC Resource Controller and Health Reporter, if they fail due to an connectivity issue. Moreover, it actively locates a gateway via discovery mechanism if previously resolved IP endpoint fails to respond.

Implementation-wise, the service name of the gateway must be supplied currently as configuration parameter to tell auto-configuration which gateway to choose.

Implementation: `flexsmc/node/peer.go`, `flexsmc/node/modules/modules.go`

Health Reporter implements a connection-alive check whether the associated gateway is still available on the one hand. In addition, it regularly tells the gateway being connected to, that this peer is alive and connection is healthy. If something breaks, the parent component, i.e. Continuous Connectivity, is notified.

Implementation: `flexsmc/node/modules/health.go`

SMC Resource Controller is two-fold: first, it manages and monitors connections and running SMC sessions to a local SMC provider. Secondly, it spawns command and control (C&C) channels to the gateway for separate SMC sessions and bridges them to available SMC resources. When a SMC provider signals the tear-down of a SMC resource, e.g. that is a finished or an aborted session, it notifies the gateway, cleans occupied resources and spawns a new channel for a future SMC session if resources are available.

Implementation: `flexsmc/node/modules/smcadvisor.go`, `flexsmc/smc/*`

5.2.3 Task Orchestration

In order to make task orchestration work, a multitude of components on both gateway and peer side are involved. In this section, we guide through all relevant parts of the whole process by regarding a task's execution from the start all the way down to the SMC provider and back. In particular, the focus will be on technical aspects of job execution that is introduced as an abstract design in section 4.4.4. In contrast to the design elaboration, this section also highlights respective functions on the peer side.

Figure 5.3 illustrates an overview of the components we will concentrate on. As it is quite complex as a single piece, it is broken down into multiple parts as follows.

5.2.3.1 Directory with Peer Information

Task orchestration requires some essential control structures which are provided by the directory component. Therefore, we start with this part first.

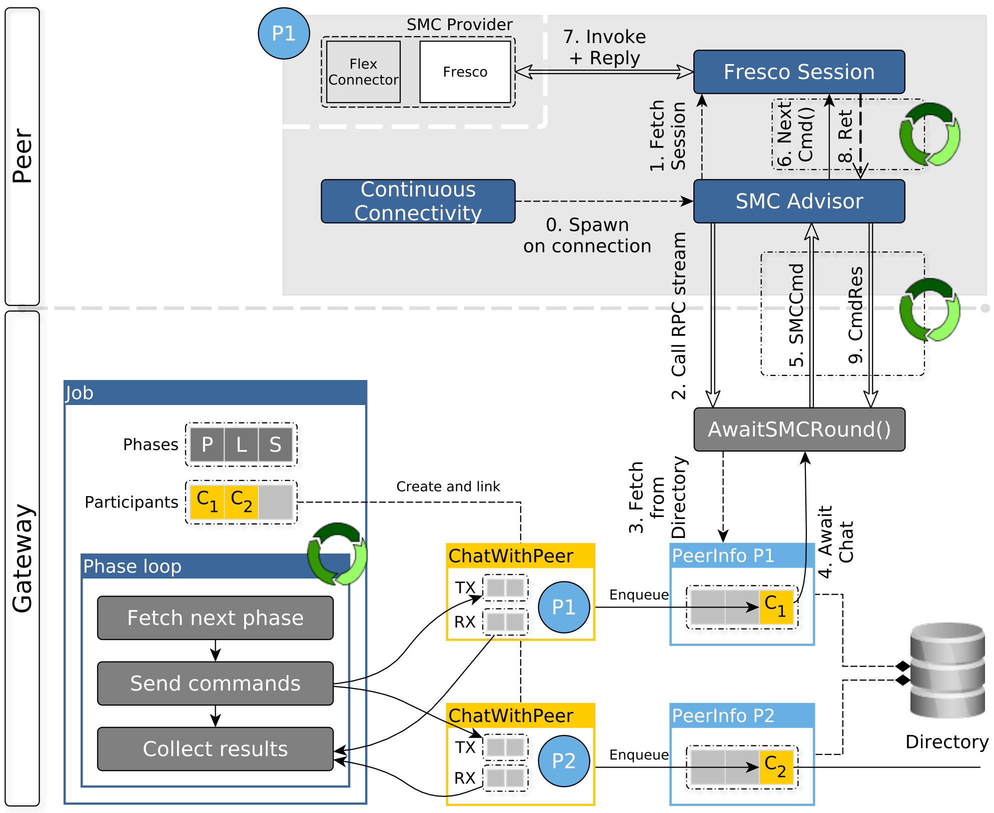


Figure 5.3: Simplified process of job execution, showing both gateway's and peer's main functions. Green circles denote a repeating process. Thin solid edges $\longleftrightarrow$ illustrate data flows between components within a FlexSMC instance, while thick ones $\Longleftrightarrow$ show network or socket connections. Dashed edges ←--→ depict either function calls or highlight relationships, depending on the context.

When a peer registers with a gateway the first time, the `Registry`, being the responsible element of the directory, creates a new entry and holds it in a map. That is the structure `PeerInfo` which consolidates both mostly static and dynamic properties of this particular peer. For instance, static fields are properties such as capabilities, authentication identifier and version of a peer.

In this context, the dynamic fields of `PeerInfo` are of interest. Most importantly, there are:

- `Addr` — maintains the current information about a peer's endpoint. Beside its reachable IP address, this includes a port number available for future SMC operations.

- `lastPing` and `stateNotifier` — provides means to monitor a peer's recent activity. While the first records the elapsed time since the last heartbeat message, the

Listing 5.1: Chat structure for bidirectional and asynchronous communication between two Go routines.

```go
type smcChat struct {
    // Reference to the concerned peer.
    peer *PeerInfo
    // TX channel to send instructions to a particular peer.
    to chan *pbJob.SMCCmd
    // RX channel to listen for feedback from the same peer.
    from chan *pbJob.CmdResult
    // [..]
}
```

latter keeps track of available and active command and control channels to this specific peer.

- `requestedSessions` — is a buffer holding outstanding communication requests in form of chat objects (as visualized by C_1 and C_2 in fig. 5.3). The job execution component in a gateway creates these objects when intending to communicate with this particular peer.

Noticeably, the buffer `requestedSessions` is the central tool to steer communication to the respective peer. As elaborated later on, this element is queried by a peer's RPC call for waiting sessions. Given job execution involves the peer related to this particular `PeerInfo` structure, it enqueues a shared chat object used for interactions, as elaborated next.

5.2.3.2 Chats for Asynchronous Communication

In this implementation, the key to communication with multiple peers in an asynchronous manner from a gateway perspective are chat objects, provided by the structure `smcChat` in code. In fig. 5.3, chats are depicted in yellow color on gateway side (bottom half).

Listing 5.1 shows the structure of `smcChat`. Most importantly, it contains two buffered channels that are essential to asynchronism. *TX* is directed towards a peer and supports forwarding enqueued commands to the referenced peer. Let us assume that multiple chats are opened. Then, job execution component can load next instructions to all involved peers without having to wait for an explicit reply at this stage. For each peer, a dedicated Go routine cares about the direct communication with the peer on behalf of the running command and control channel, initiated by the peer with an RPC call. Similar for the receive channel *RX*, each peer's responsible Go routine enqueues an incoming result independently (with respect to any other linked peers). Then, job execution can collect all results from those peers who answered after a while. This makes it more robust to failing connections, job execution would wait for and potentially loose messages of the remaining peers otherwise.

Not loosing sync among multiple peer's progress, the buffer's capacity is limited to one element in the prototype implementation. This comes from the fact that each rolled-out phase requires all peers to be in sync before proceeding.

5.2.3.3 Integration and Processing from Peer Side

In order to orchestrate any tasks, peers must be on-line and be attached to the gateway via command and control channels. Given this precondition is fulfilled, instructions are passed towards a SMC provider available on a peer. That is illustrated as part of this subsection.

The process starts from the *Continuous Connectivity* component on a peer, as earlier introduced in section 5.2.2.3. It finds suitable gateways, establishes a reliable gRPC connection and probes them with heartbeat messages. On success, the following steps are invoked consecutively as enumerated below. Note that numbering relates to edges' label in the right side of fig. 5.3 starting from the peer.

0. First, *Continuous Connectivity* component spawns an instance of `SMCAdvisor`. The latter is responsible for spinning up command and control channels to the gateway and bridging communication to the SMC provider.
1. Before being part of a computation, the `SMCAdvisor` acquires a local SMC session from the SMC manager (`FrescoConnect struct`). The latter regulates the amount of parallel sessions and establishes an RPC connection to *FlexConnector* via a socket. Only when a session is available, it proceeds.
2. Next, `SMCAdvisor` calls the RPC function `AwaitSMCRound()` on remote side to establish a bidirectional command and control channel (gRPC stream).
3. On gateway side, gRPC spawns a dedicated Go routine to handle the running stream. By querying the directory, it fetches the respective peer's information (`PeerInfo` object) in order to wait for the job execution to issue a communication request in form of a chat object.

A peer's attachment to the gateway pauses here if the gateway has no intend to talk to the peer. Let us assume that job execution queued a chat object for the regarded peer (e.g. chat C_1 in fig. 5.3) and interacts with this peer using the chat's buffers, the following steps are repeated as long as the chat is not teared down, i.e. more phases are to come:

4. Once a chat is available for this particular peer, the respective Go routine continues execution. In a loop, it first reads enqueued commands for transmission and sends

them down the stream to the peer. Then, it waits for a reply with a result message from the peer and writes it to the receive buffer of the chat. If connection breaks, the same buffer is used to notify job execution with a special message.

5. Commands meant for the peer, are sent via the established command and control channel. The message encodes a `SMCCmd` structure. It is described in section 5.3.2.
6. On peer side, the received command is directly passed to the running SMC session by calling the function `NextCmd(...)`. As it is a blocking function call, the reply is handled in item 8.
7. The session (instance of `frescoSession`) handles communication with the SMC provider. Basically, it just passes the commands down to the *Flex Connector* via synchronized RPC calls and receives the reply.
8. The reply is handed back to `SMCAdvisor`.
9. Before passing the received reply from the local session to the gateway via the stream, `SMCAdvisor` checks the message itself. If it indicates that the SMC provider intends to stop the current session, the advisor will cut the stream after having sent the result message `CmdRes`. The message format is described in section 5.3.2.

As noted above, steps 4 to 9 are repeated until any party decides to tear down the current computation session, including this command and control channel. To allow future sessions, the `SMCAdvisor` takes care keeping always at least one open command and control channel to the gateway, so effectively starting over from item 1.

5.2.3.4 Job Execution on Gateway Side

Realizing design decisions from section 4.4.1.4, we will concentrate on job execution from an implementation-specific perspective in this subsection. It is the missing piece of task orchestration from a gateway's view. With respect to fig. 5.3, this subsection highlights the bottom left part.

Assuming a task description is submitted by a service client, and parent orchestration component has transformed the task into a more concrete SMC-specific instruction set as described in section 4.4.3, job execution is next to complete the overall process.

A key element to job execution is the `job` structure itself, holding all important instructions and the current state of execution. The following depicts important fields of a `job` structure (in `orchestration/worker/job.go`):

- `instructions` — holds the information generated by translating a generic task into concrete SMC-specific steps, while involving a specific set of peers. In particular, it contains:

    1. `Tasks` — is a list of SMC phases to execute in provided order on behalf of the participants listed next.
    2. `Participants` — list of participating peers' `PeerInfo`.

- `chats` — is a map of chat objects, providing means of communication with all participants listed in the instruction. Chats are allocated on demand in the early phase of job execution.
- `progress` — keeps track of currently rolled out SMC phase. That is necessary for resumption when a job is paused due to present errors or communication losses. Default is `0`, i.e. no progress made.

A `job` is inserted into a shared queue. According to the Multiple-Producer-Multiple-Consumer paradigm, a worker routine from a pool of workers fetches a job instance from this queue and is exclusively responsible for its execution.

Prior to start communicating with the peers related to this task, the worker must first **establish chats** to them: for all participants listed in the job's instruction, the worker requests a bidirectional chat. This causes the chat object to be enqueued in the respective peer's `requestedSessions` queue (as described before in section 5.2.3.1). At the same time, the worker keeps references to all chats in the job's `chats` field.

Given established chats, the main loop rolls out each phase as follows:

1. **Fetch next phase**
   Based on the current `progress` of this job, its subsequent phase is fetched from the instructions stored in the task. Basically, a phase equals a specific SMC command that needs to be rolled out on relevant peers.
2. **Send commands**
   Prior to sending the fetched phase to all participants the job got chat objects for, the phase is enveloped in a container that holds information specific for the destination's peer, such as its role among all other peers and the current session's ID. This is the message structure `SMCCmd` as described in more detail in section 5.3.2.

   Having built the outgoing messages, they are enqueued in the respective chats' transmission buffer. From here on, the Go routines, waiting since peers calling the RPC function `AwaitSMCRound()` for the command and control channel, take over to deliver the message asynchronously. Replies are stored in the chat's reception buffer.
3. **Collect results**
   The phases are meant to be conducted synchronously on a set of peers to satisfy requirements originated by the deployment of SMC. This means that it is not beneficial in the most cases if a peer was one or several phases ahead of the

rest. Consequently, all peers need to stay in sync in spite of asynchronously communicating with them. As replies from the peers are buffered in the chat objects, the job's worker routine collects incoming results and waits until all peers complete prior to continue as part of the function `queryTargetsSync(...)`. Given a phase is complete, the replies are forwarded to the postprocessing component (see section 4.4.1).

In case that some peers do not reply until a given timeout or refuse collaboration by replying with an error, the job is paused at this stage and reported to the all-embracing orchestration component to resolve the issue. If everything runs smooth, `progress` is incremented in order to fetch the next phase in the next cycle of this loop. Hence, it starts from item 1 again until all phases are completed, thus the computation session is finished.

As mentioned, an error pauses the job and control is handed back to the orchestration component in order to handle the error, as described in section 4.4.4.3. In this state, the chats are kept alive to accelerate the resumption of a repaired job. Analogous to a new job, a recycled job is added to the job queue and executed by a free worker routine. Due to its `progress` field, execution is resumed from the correct state. Noticeably, if recovery needs to start over from an earlier phase for proper resumption, the progress is altered before queuing the job again.

In all cases, the continued chats accelerate overall execution due to two facts: on the one hand, command and control channels to FlexSMC peers and ongoing channels reaching to the SMC providers are still sound. On the other hand, and more importantly, the SMC provider is still tied to the current session, meaning, that needed resources are still allocated within the used SMC framework.

If a computation session is finished, all established chats are teared down. This causes all depending resources to be freed. More specifically, the current command and control channel is also shut down, as a new one is spawned immediately by the peer for a future computation session.

5.3 Communication and Messages

In this section, we present the means of communication that enable reliable interactions between decentralized FlexSMC instances on the one side, and also describe the links between FlexSMC and Flex Connector on the other side.

5.3.1 Used Techniques and Rationale

For any communication, that outreaches an application's border, we adopt well-established techniques for defining protocols, message serialization and de-serialization, transmission and reception, and securing resulting message exchange.

Protocol Buffer and gRPC

Instead of using platform-dependent and application-specific serialization, we define the structure of our messages in an Interface Definition Language. In particular, we adopt the Protocol Buffer scheme [43]. It comes with a set of compilers and tools to generate all (de-)serialization code for a variety of languages, including our needed ones. That are Go and Java. Although the compiled message interchange is a binary format, it preserves interoperability between different versions. This is of practical importance as applications in a distributed environment may have been deployed over time.

Message serialization does not make up for communication alone. To provide interfaces on both FlexSMC and Flex Connector, gRPC [42] comes handy and continues the concept of generic definitions for RPC endpoints. It seamlessly integrates with the aforementioned protocol buffers and is a good match for our server-client interfacing, therefore. Not least, that it brings the necessary server and client code for both Go and Java. Taken together, it is a robust foundation for communication in a distributed environment.

Secure Channels with TLS and Mutual Authentication

With respect to cryptography, communication between FlexSMC instances is secured by means of the Transport Layer Security (TLS) protocol. Each peer maintains a host-independent set of certificates of those peers who gain trust through a pairing process (see section 4.2.2 and section 5.2.2.1). This allows two nodes, in our case a gateway and a peer, to mutually authenticate each other, leveraging the protocol's capability for client authentication. Instead of integrating cryptographic primitives in a custom protocol, TLS provides the advantage that there is a lot of research about its security guarantees, hence, recent versions are a reliable choice.

Implementation-wise, the communication between FlexSMC instances enforces at least TLS version 1.2. Like this, we mitigate several attack vectors that arise in previous versions as summarized in [44].

5.3.2 Command and Control: Messaging between Gateway and Peer

Command and Control Streams

FlexSMC peers are the system's providers for SMC support. In oder to make the resource available, a peer keeps at least one bidirectional gRPC stream to its gateway, that is used as a command and control channel for a computation session. Moreover, an open stream means that a SMC resource is waiting and ready.

Depending on the performance of a peer's host, it may offer multiple SMC sessions in parallel. On implementation side, it spawns the same number of streams to the gateway as parallelism allows to. While it sounds like a heavy approach in terms of having many open network connections, it actually leverages specific properties of gRPC: parallel streams to the same destination share the same connection for transport by multiplexing the data streams logically. Concretely, Hypertext Transfer Protocol Version 2 (HTTP/2) is the underlying transport protocol which realizes this feature by means of a framing layer. [45]

Message Definitions

For starting and controlling a SMC session, a gateway sends `SMCCmd` messages to a specific peer via the established stream. Each peer replies to an incoming command with one `CmdResult` message. The stream is kept alive until either gateway or peer closes it. A peer signalizes a clean shutdown by replying with a `SUCCESS_DONE` status.

Listing 5.2 depicts the **command structure** for instructing peers in Protocol Buffer notation. Basically, it is divided in two logical sections.

The fields `sessionID` and `smcPeerID` define the peer's context. When sending a command to the peer, these are the fields, a gateway alters individually with respect to its intended peer destination and session. While the session ID is the same for all targeted peers, the identifier is usually changed for independent sessions, while the task's instructions might still be the same, e.g. when a service client requests the same task consecutively within a short amount of time. More importantly, `smcPeerID` are individual IDs that are assigned uniquely to each peer. The assignments are randomized with respect to different computation sessions. Within a SMC session, these IDs are needed by the peers to know their rang among the others. Due to a bug in Fresco, the identifiers must be a consecutive series of numbers [46]. This is of a special concern due to the employed reconfiguration if peers fail. Consequently, a failing peer means a hole in the sequence. On reconfiguration, it is taken into account by assigning new IDs to all peers.

The other section of the command holds the actual SMC instructions. Therefore, each

Listing 5.2: SMC command.

```
message SMCCmd {
// Peer context
string sessionID         = 1;
int32 smcPeerID          = 2;
// Payload packet
oneof payload {
    PreparePhase prepare = 3;
    LinkingPhase link    = 5;
    SessionPhase session = 4;
    DebugPhase debug     = 9;
}
}
```

Listing 5.3: Simplified command result.

```
message CmdResult {
enum Status {
    // Class: success and info
    SUCCESS      = 0;
    SUCCESS_DONE = 1;
    // Class: recoverable errors (33 - 63)
    ERR_CLASS_NORM  = 32;
    UNKNOWN_CMD     = 33;
    DENIED          = 34;
    // Class: irreversible errors (65 - 127)
    ERR_CLASS_FAULT = 64;
    ABORTED         = 65;
    // Class: communication errors (129 - 255)
    ERR_CLASS_COMM  = 128;
    STREAM_ERR      = 129;
    // Combined error classes
    SEVERE_ERROR_CLASSES = 192; //      64 + 128
    ALL_ERROR_CLASSES    = 224; // 32 + 64 + 128
}
Status status    = 1;
string msg       = 2;
SMCResult result = 3;
}
```

command holds exactly one single phase that describes the SMC-specific task. This part is not altered by the gateway and is exactly the same for all involved peers. Hence, it simulates a broadcast. Most fields of the respective phases are already elaborated in section 4.4.3.3. In addition to that, we introduce a `DebugPhase`. Its purpose is mainly to gather statistics about the performance of the implementation. Basically, it carries a ping message and key-value pairs to evaluate different parts of a FlexSMC peer and the attached Flex Connector. By default, those debug phases are ignored completely, unless explicitly enabled by a run-time argument.

Listing 5.3 depicts the **reply structure** that constitutes a peer's reply to a foregoing command. Most prominently, it contains an extensive `Status` field, reporting about the success or failure of a command. Its most significant bits categorize the type of error into three classes, namely communication, hard (faults) and soft errors. The remaining bits define error details. Figure 5.4 gives concrete examples for illustration.

The reason for the categorization into error classes comes into play for the job execution on gateway side. Given, any peer raises an error, the task orchestration component may specify which one to handle. If the occurred error matches the handled error classes, job execution pauses the current progress and hands control back to the orchestration logic. There, it fixes the problem, i.e. by dropping erroneous peers in the prototype, and signalizes job execution to resume the current task. If an error class is not stated to be handled, the job execution module will tear-down the faulty session and , hence, the

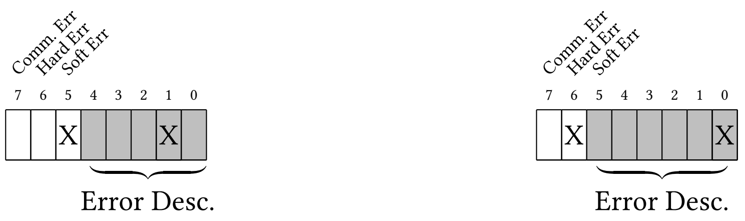


Figure 5.4: Two exemplary error status. Concretely, left byte field shows a recoverable error, indicating its refusal to a task. Right byte field represents a (non-recoverable) fault, indicating an aborted task on peer side.

Listing 5.4: Flex Connector's RPC interface for FlexSMC.

```
service SMC {
    rpc ResetAll (FilterArgs) returns (job.CmdResult) {}
    rpc Init (SessionCtx) returns (job.CmdResult) {}
    rpc NextCmd (job.SMCCmd) returns (job.CmdResult) {}
    rpc TearDown (SessionCtx) returns (job.CmdResult) {}
}
```

Listing 5.5: Session context.

```
message SessionCtx {
    string sessionID = 1;
}
```

requested task fails.

The `result` field of `CmdResult` carries the final result that is calculated on behalf of SMC. It is a simple value storage for floating-point numbers as we currently only support numeric results.

5.3.3 Inner-Host Communications

Due to the split of responsibilities within a host, in particular employing FlexSMC for task orchestration and communication, and Flex Connector for integrating SMC functionality, there are two independent applications that need a common mean of communication. By reusing largely the same message definitions as introduced for interactions between remote FlexSMC instances (see section 5.3.2), it aims for reducing system's complexity and maintenance effort to keep communication in sync.

Flex Connector runs a gRPC server to which a FlexSMC peer can attach as client via a local Unix socket. It exposes the RPC interface as depicted in listing 5.4.

Interaction takes place mainly on behalf of three RPC methods, namely `Init`, `NextCmd` and `TearDown`. For any new computation session, a FlexSMC peer first calls the `Init` method in order to allocate the session on Flex Connector side. It is a required step as the latter component must keep state about all active sessions.

On success, a peer simply bridges each phase that comes from the gateway, to the Flex Connector using the `NextCmd` method. As the `SMCCmd` message structure is reused, no conversion is necessary and further simplifies FlexSMC's responsibility. Same applies to the reply sent by the connector. It is important to stress that the session ID used for initiation, is remembered on FlexSMC side, as it is always passed along as meta-data with each call to `NextCmd` method. Implementation-wise, the content of the meta-data is shown in listing 5.5. On the other side, the Flex Connector only trusts the passed meta-data for session association. Like this, a gateway cannot hijack other sessions by changing `sessionID` field in the command structure.

When a computation session is done, the `CmdResult` message from the SMC provider sets its `status` field to `SUCCESS_DONE`. Then, a peer calls the `TearDown` method to conjointly signalize resource freeing and finishing any unfinished tasks on both sides, before finally forwarding the reply to the gateway. Noticeably, a peer also invokes a teardown if the connection to the gateway permanently breaks for the current session.

Chapter 6

Performance Evaluation

Based on the implementation of our orchestration framework, this chapter evaluates performance aspects on a setup built up of real hardware and a typical network infrastructure found in commercial buildings.

Due to the complexity of the overall architecture, we mostly perform *black-box* measurements beside some special investigations we introduce for deeper insights. In order to not mix results up originating from different components, we first evaluate the performance impact of the orchestration layer *FlexSMC*. Thereafter, we present measurements from a service client's perspective that also involve simple secure computations, as it would be in a real-world application. The used SMC provider is Fresco in all measurements.

6.1 Experimental Setup

This section describes the setup we used for all tests. The idea for the test setup is to have a similar network topology as it can be found in small offices or within departments of larger organizations, so particularly applying to Smart Environments as discussed in UC.1. For the majority, this is a simple switched environment with departments living within divergent subnets and virtual LAN environments. Though, a router may forward layer 3 packets if the target is another department. [47]

The following subsections elaborate our setup in detail.

6.1.1 Network Organization and Roles

There are two computing isles R and S equipped with six identical computers (called *nodes* or *peers*) each. For evaluation, each node is identified by a sequential integer. Nodes from isle R receive an ID from $[1, 6]$ and nodes from isle S an ID from $[7, 12]$.

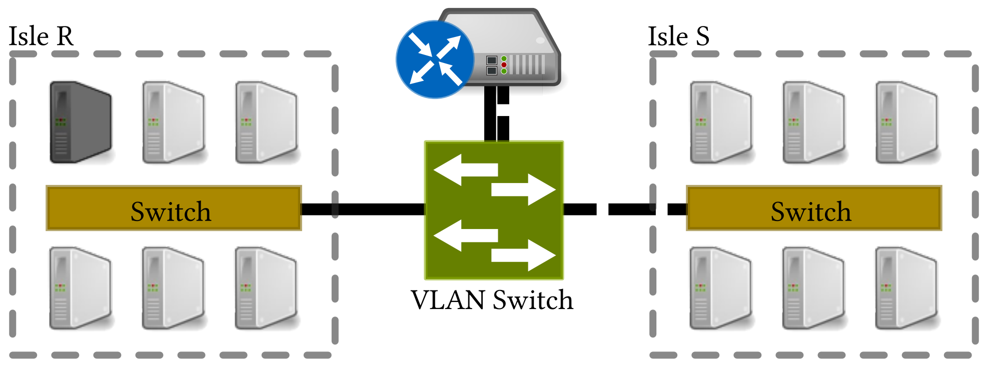


Figure 6.1: Network organization. Six computers are connected to the central switch (colored green) via an individual switch per isle (colored ocher). All traffic needs to pass the router at the top when communicating with the neighboring isle. The dark colored node acts as a dedicated FlexSMC gateway and orchestrates the other 11 nodes.

Figure 6.1 depicts an architectural overview for orientation. Regarding one isle, all assigned nodes are connected to an unmanaged switch via `1 Gbit` links. This switch is wired to a managed switch. In the figure, this is the central green part. The central switching unit splits packets to and from both isles into disjunct virtual LANs. This means that the isles are isolated logically and cannot reach each other by default. It is necessary that a computer with routing capabilities bridges this gap and forwards IP traffic between those virtual LANs. Same as for the other links, a single `1 Gbit` link transports the tagged traffic to the router in both directions.

To give a rough impression with respect to packet transmission time, consider an IP packet that targets a node of the neighboring isle. In total, it needs to pass one router and meets a switch four times for a single direction. So, we expect latency and jitter to be way higher than within a single switched isle without outbound communication.

The dark-colored node in fig. 6.1 acts as a FlexSMC gateway. It is chosen arbitrarily as the same circumstances hold for every of the 12 usable nodes. Noticeably, its only task is the management and orchestration of the other nodes for the tests. It does not run any SMC instance by its own to improve measurement accuracy. Though, it is a typical use-case we support in FlexSMC.

6.1.2 Hardware and OS Details

For the purpose of evaluation, we employ a homogeneous test setup. On the one hand, it simulates the fact that hardware is often chosen to be equipped similar in large offices to facilitate administration. On the other hand and more importantly, artifacts in measurements that arise from a small set of computers, are unlikely to relate to the hardware itself. Like this, we can omit this consideration.

Thus, all computers that are part of the SMC network, are identical in construction during the tests. They feature the following specification:

- Intel(R) Xeon(R) CPU E3-1265L V2 @ 2.50GHz
- 8 Cores, with 8192 KB cache size
- 16GB main memory
- Debian GNU/Linux 8.5 Jessie (64 bit), non graphical version
- Linux kernel 3.16.0-4

These nodes are mostly idle beside the computations running in the scope of our tests. Likewise, their respective Ethernet links are not limited and can be utilized up to their maximum supported bandwidth of 1 Gbit/s. The main uncertainty is the router node that forwards IP traffic between both isles. It is not under our control and the load is unknown. Normally, it should be quite low during our test period. Still, it might skew our results slightly in an unpredictable manner. Same applies to some background processes (e.g. NTP, DHCP client) that run by default on the machines. Though, under normal circumstances, they are negligible within the scope of these tests.

6.2 Methodology

6.2.1 Primary Parameter

One major goal of FlexSMC is the flexibility to orchestrate the same task to a varying number of nodes automatically. Therefore, the primary parameter in this work's tests is the amount of participants for a certain task or computation that is changed dynamically without any reset or manual reconfiguration. Linked to our discussed use-cases, it is interesting to explore how the system behaves and whether overall responsiveness still satisfies the needs of real environments. In particular also due to the fact that scalability is a concern for SMC in general.

The parameter is always increased by two, so adding two new peers with each change of configuration. Thus, the resulting configurations for this parameter are $\{3, 5, 7, 9, 11\}\#peers$.

6.2.2 Black-box Measurements

If not stated otherwise, we measure the results at publicly accessible endpoints (GW interface) to achieve the same results as a regular client would obtain when issuing tasks to the system. We will call it *request-response* or *end-to-end* latency. This means that FlexSMC or certain parts of it are treated as black-box with respect to response time testing. Consequently, we do not distinguish explicitly between different peers

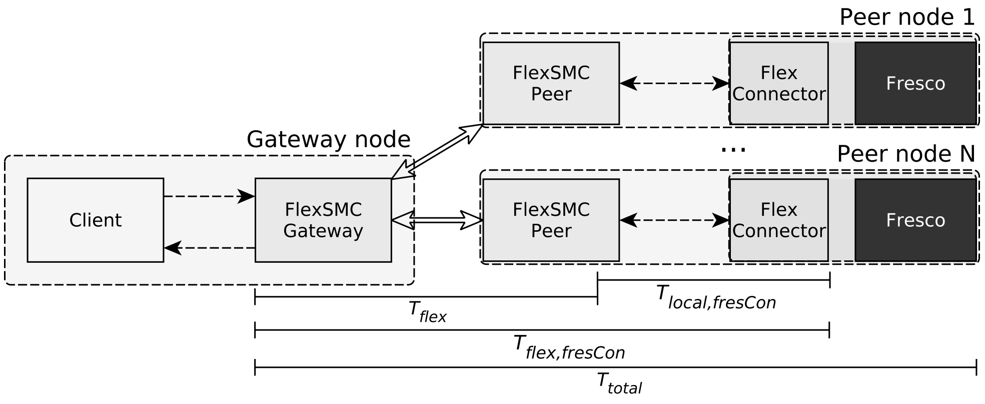


Figure 6.2: Test metric for the complete communication path. T always denotes the time for a bi-directional communication flow. Dotted edges $\dashleftarrow\!\dashrightarrow$ visualize local communication on a host. Solid edges between FlexSMC instances $\Longleftrightarrow$ depict any routed and switched network.

involved in a computation. Rather, the behavior as an anonymous group of peers is regarded. A fundamental characteristic of black-box testing is its behavior comparable to a synchronization barrier. This means that the slowest peer per phase defines the global performance. The reason is simple: SMC requires every peer to contribute in all aspects of the computation so that every participant has the same impact. If one is slower than the rest, others cannot proceed. In all cases, the last round requires synchronization so all peers are able to deduct the final result of the operation.

Depending on the outcome of specific black-box tests throughout this chapter, it might reveal interesting facts that make us delve deeper into the architecture in order to locate the origin. In that case, we give explicit notice.

6.2.3 Metrics

In order to discuss the results of the evaluation, we introduce several time-based metrics. Figure 6.2 depicts all components and communication paths of a FlexSMC network and defines the interesting temporal segments. One can see that we do not examine each core segment individually which would be possible alongside each part of communication and processing unit. Rather, we focus on composites of them. As discussed before, a black-box perspective provides good indications about the performance of the implementation. By regarding different composites of core segments, we get suitable measures for both FlexSMC's basic functionality and behavior as a production system, also considering properties of the employed network (section 6.1.1). On the other hand, employing measurements for all possible core segments one could possibly partition the whole path, is hard to accomplish and does not guarantee more meaningful insights. It would be even misleading as it would hide specific facets only visible during the interplay of multiple areas.

All in all, we concentrate on the following set of metrics in this evaluation:

- $\boldsymbol{T_{local,fresCon}}$ — is the request-response time for local communication between a FlexSMC instance and a connector bundling a Fresco instance on the same host.
- $\boldsymbol{T_{flex}}$ — measures the time from issuing a task at the FlexSMC gateway until receiving and processing the response of all peers. In case the gateway sends special ping messages, peers answer the request directly without passing it any further.
- $\boldsymbol{T_{flex,fresCon}}$ — same as T_{flex}, with the difference that all packets are handled by passing them to the Flex connector via RPCs over local socket communication.
- $\boldsymbol{T_{total}}$ — is the total request-response time for a use-case based task. So, $T_{total} = T_{flex,fresCon} + T_{fresco}$, where T_{fresco} is the time Fresco needs for completely processing a SMC task. This includes all phases, such as preparation, linking nodes, and performing the actual calculations.

6.3 Platform Performance

Before proceeding with system tests covering the overall performance (section 6.4), it is essential to get an impression about how the platform architecture performs itself. Most importantly, the expectation is that the orchestration framework makes up for a minor part only compared to the SMC counterpart. Thus, it shall support SMC in terms of flexibility, robustness and transparency, but shall not decrease its operational performance. This is subject to evaluation in the next sections.

6.3.1 Test Parametrization

For testing, subsets of FlexSMC's platform can be activated differently by adding special attributes to the task submitted to gateway's orchestration component. This allows to differentiate effects of internal and external influences. Note that we still employ black-box testing, but the "box" can be influenced.

In particular, the following two parameters are adjustable:

EP.S **Stage of message handling**
Incoming commands from the gateway can be handled in different stages on peer side.

EP.S$_1$ A peer handles the message directly before invoking a SMC backend. Thus, it never reaches the backend. This is similar to a simultaneous ping to all participating peers and described by T_{flex}.

EP.S$_2$ FlexSMC peers operate as usual. They initiate a session to their Flex Connector counterpart on demand via gRPC socket connections. Then, the connector replies to the message without actively invoking Fresco. In particular, $T_{flex,fresCon}$ is the corresponding metric.

Effect: design decisions of the architecture and its induced overhead can be assessed with more detail. Generally speaking, it allows to differentiate *architectural overhead* in the measurements.

EP.M **Number of messages in a row**
While a session is established among all peers, the gateway can exchange multiple messages with its peers. This is covered by either T_{flex} or $T_{flex,fresCon}$ depending on EP.S.

EP.M$_1$ Gateway sends a single request over the established channel.

EP.M$_{10}$ Gateway sends 10 consecutive requests over the established channel.

Effect: *network latency* becomes the dominating factor given multiple messages are exchanged[1]. Thus, costs for establishing and tearing-down the communication channel loose significance.

For the practical evaluation, we automatically triggered more than 1000 requests for every combination of participating number of peers, EP.S and EP.M. All requests were issued sequentially.

6.3.2 FlexSMC Layer

In the first test, we examine the responsiveness up to the FlexSMC layer. This is T_{flex}. More precisely, beside varying the number of nodes within $\{3, 5, 7, 9, 11\}$ peers, messages are handled already in the first stage (EP.S$_1$). Further, we split the test by means of requesting a single message versus ten messages in a row. This allows to deduct the system overhead in the first place and separate it from network aspects in the second place.

6.3.2.1 Single Message Exchange

First, a task is constructed that instructs the gateway to send a single request to all participating peers (EP.M$_1$). It then waits until receiving a reply from all. This maximum

[1]We assume that CPUs are not exhausted. Noticeably, regular checks confirmed it.

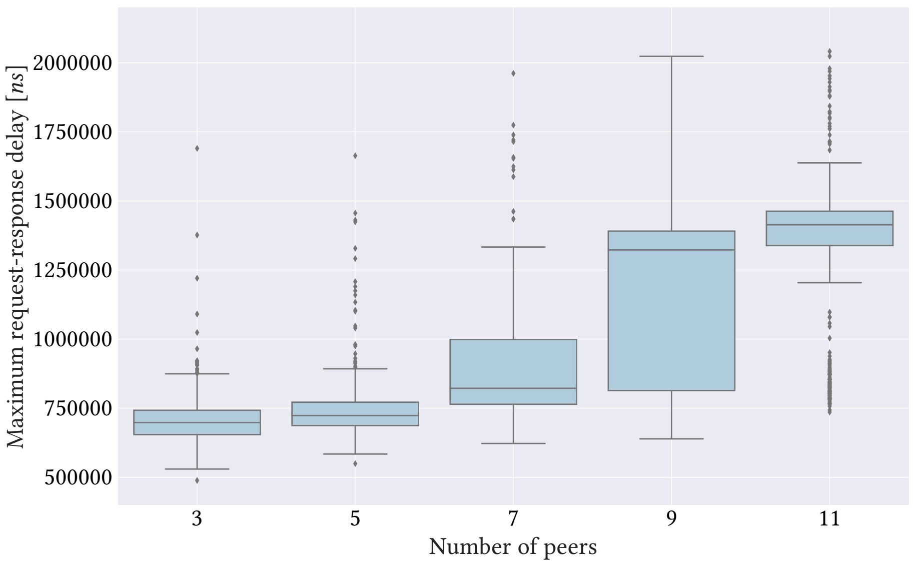


Figure 6.3: FlexSMC request-response time for a single ping message. The used parameters are (EP.S_1) and (EP.M_1). Appendix A.2.1 enumerates the underlying statistics.

delay contributes a data point to the raw measurement dataset. Figure 6.3 depicts the maximum request-response delay for this scenario. Precise statistics are listed in appendix A.2.1.

One can see that it takes on average between $0.71ms$ and $0.74ms$ until the response arrives for 3 and 5 peers, respectively. In comparison, the RTT for an Internet Control Message Protocol (ICMP) packet is around $0.17ms$ on average for the same path, but with respect to a single peer. The experienced jitter is almost the same when consulting the standard deviation as a measure for the RTT of our request-response communication effort and the ICMP packet exchange. Obviously, the delay is almost three times higher for our implementation. In the first place, it seems that efforts to parallelize communication do not fully work out yet with respect to these numbers. This will clarify later on.

At this point, recall that network topology and node placement is organized in a way, that one gateway and up to 5 participants (IDs are from $[1, 6]$) operate on the same isle as shown in section 6.1.1. Thus, packets pass a single switch only on the way to any target with respect to a single direction. Now, breaking through this barrier, we expect a considerable increase in RTTs. What we get to see is that the median delay $median(T_{flex}^{M_1}(n))$ reaches $1.414ms$ given $n = 11$, so all nodes are participating. The ratio $\frac{median(T_{flex}^{M_1}(11))}{median(T_{flex}^{M_1}(5))} \approx 1.96$ shows that the median delay almost doubles when migrating from one fully employed isle with 5 peers to all 11 peers on both isles taken together. This result falls behind our expectations. As we see later on in EP.M_{10}, the origin of this observation is narrowed down to an inefficiency in initial arrangement of the connections. Nevertheless, the high jitter for the test configuration with 9 peers is exceptional and unexpected. As jitter regresses and sharpens with even more peers, it is to be assumed that the router's network load peaked during this part of the test. Thus, it should be abstracted away for further analysis.

All in all, the prototype's results are already considered to be very suitable, considering the use-cases and that SMC will make up the most dominant part regarding the relative time shares (see section 6.4).

6.3.2.2 Multi Message Exchange

To obtain a deeper understanding on which parts are likely to cause the effects observed for the single message case, the idea is to increase the usage of an established communication channel (EP.M_{10}). The assumption is that costs are high in FlexSMC to initially establish an end-to-end communication channel to each peer.

First, we discuss $T_{flex}^{M_{10}}$ by the results up to the FlexSMC stage (EP.S_1) depicted in fig. 6.4. With 3 and 5 peers, it takes around $4.3ms$ on median until a request with 10 messages is done. This means the slowest peer responded by then. Moving on to all peers, median request-response time increases by $1,062ms$. In order to compare it with the single-message case, the multi-message ratio $\frac{median(T_{flex}^{M_{10}}(11))}{median(T_{flex}^{M_{10}}(5))} \approx 1.25$ reveals useful information: it is now magnitudes lower than before. The direct conclusion is that using an established communication channel more intensively exposes less costs in term of RTTs for individual messages. Explained the other way round, connection management accounts for a denotative share in T_{flex}.

While the total delay for multiple messages recorded as a whole, gives rough indications, a view on RTTs of individual message exchanges (per established communication channel) reveals more detailed insights which help to understand the origin of described differences. Note that we look now into the black-box. Analyzing fig. 6.5a for EP.S_1 (FlexSMC stage only) shows prominent peaks centered around $380\mu s$ and $450\mu s$ for 5 and 11 peers, respectively. If additionally taking statistics from appendix A.2.2.2 into account, the trend looks quite linear with a slight increase of RTTs in the order of a few $\frac{1}{100}ms$ given an increasing number of peers. But, more importantly, there is a second local maximum centered around $0.58ms$ and $0.67ms$ representing approximately at least one tenth of all RTTs. Correlation with raw recordings clearly show that this accumulation originates from the first one in the sequence of request-response messages. Interestingly, the third local accumulation at $1ms$ apparently relates to the last message in the same sequence. In contrast to the first, it appears in a non-deterministic manner, but is more likely for high number of peers. While the latter is presumingly a synthesis of multiple effects (e.g. connection tear-down, gRPC runtime, garbage collec-

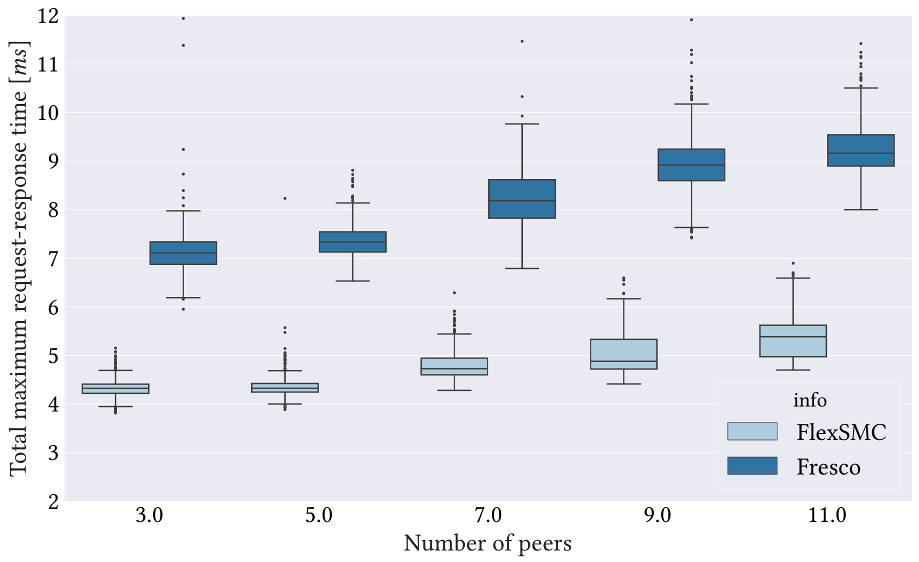


Figure 6.4: Total maximum request-response time for 10 consecutive ping messages over a channel including initial preparation and follow-up delays. The labels FlexSMC and Fresco correspond to stages of message handling EP.S$_1$ ($T^{M_{10}}_{flex}$) and EP.S$_2$ ($T^{M_{10}}_{flex,fresCon}$), respectively. Appendix A.2.2.1 enumerates the underlying statistics.

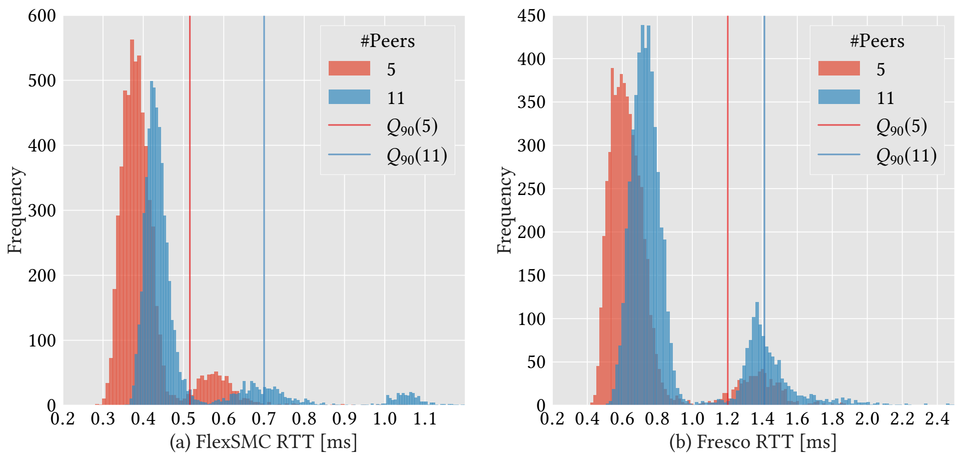


Figure 6.5: Distribution of RTTs individually per message for 10 consecutive ping messages over the established communication channel. The sub-figures FlexSMC and Fresco correspond to stages of message handling EP.S$_1$ ($T^{M_{10}}_{flex}$) and EP.S$_2$ ($T^{M_{10}}_{flex,fresCon}$), respectively. Appendix A.2.2.2 enumerates complementary statistics.

tion, etc.), the first clearly indicates some inefficiencies in the initial establishment of a communication flow.

One informal remarks: the horizontal shift in time between both *#peers* configurations arises in large parts from the fact that the maximum latency is higher when gateway's communication has to cross isles' border.

6.3.2.3 Retrospective: Implementation

Looking into the implementation, FlexSMC uses a new gRPC stream for each session. Therefore, each peer maintains always at least one standby stream to the gateway that is spawned asynchronously and independently from a new task. Due to gRPC multiplexing all streams over HTTP/2 via a single connection [45], there should be no evident overhead connection-wise. However, without looking into specifics of gRPC's implementation, the supporting structures and agreements of a stream may be established first as soon as the first packet is queued for transmission. This would explain why a session with a single request and response has the observed extra delay compared to the case when multiple messages are sent. Consequently, these effects presumably exist due to how FlexSMC uses gRPC streams for this special test scenario.

6.3.2.4 Conclusion

When initiating a FlexSMC session for sending a single request to 11 peers, median delay is $median(T^{M_1}_{flex}(n)) < 1.42ms$ until all responses arrived. As the result did not meet expectations and the origin of the cause was unclear, another test with 10 consecutive messages over the same channel was conducted. It revealed some inefficiencies that arose from connection establishment and also occasionally its tear-down. Though, for more extensive use of an open channel, the median delay for individual messages dropped to $< 0.44ms$ while all peers were active. Like this, performance is clearly better than previously anticipated in direct comparison to the single message case. Last but not least due to the fact that applications like SMC need multiple synchronized rounds of communication. Therefore, it represents most of the multi message case.

6.3.3 Fresco Layer

Due to the architecture, the SMC component is decoupled from the FlexSMC part. It does its job within a separate process. There, the Java process runs an additional gRPC server which handles requests coming from a local socket connection. For this test case, we investigate responsiveness same as the previous test case in section 6.3.2.2, but this time involving the Fresco layer, too. Therefore, the primary test metric is the delay $T_{fresco,fresCon}$. This is discussed in section 6.3.3.1. For deeper investigation, we briefly analyze local communication timings $T_{local,fresCon}$ for all nodes as of section 6.3.3.2 and followings.

6.3.3.1 Request-Response Delay $T_{fresco,fresCon}$ for Multi Message Exchange

The goal of this specific test scenario is to discuss the overhead induced by separation of orchestration and SMC components into separate processes. This is measured in terms of overall request-response delay spanning the gateway's instruction until all peers' final replies arrived. For each round, 10 consecutive messages are sent via the same communication channel (EP.M_{10}) that ends at the peers' SMC connector. Figure 6.4 depicts the result on the right for message being handled by the Flex Connector (EP.S_2). Noticeably, we omit the single message scenario now as it is not relevant for our SMC use-case.

It is easy to see that for increasing participation, the distribution of end-to-end delays follows the same pattern as for tests conducting the first stage EP.S_1. First, the jitter increases significantly when communication crosses the isles' border from $5 \longrightarrow 7$ $\#peers$, as the quartiles dilate. The observation is the same as for EP.S_1, though the jitter is constantly higher now and more steady in range of $[0.7, 0.9]ms$. It is due to the fact that measurements consider two communication paths instead of one with both being subject to jitter independently. Additionally, both are regarded over a complete round that accumulates the individual timings. Second, there is a constant offset of about $[3, 4]ms$ in RTT depending on the number of participants. For discussion, we also consider RTTs for individual messages as depicted in fig. 6.5. The most significant local maximum clearly shows that the peak of messages is handled within a delay of approx. $0.8ms$ and $1ms$ in total for a configuration with 5 and 11 peers, respectively. So, in theory, local communication accounts for an additional median RTT of maximal $median(T^{M_{10}}_{flex,fresCon}(11) - T^{M_{10}}_{flex}(11)) < 0.4ms$. However, the second local maximum indicates that opening the initial communication channel suffers from an even reinforced delay compared to the former test case's delay. Although this is of no noticeable consequence for rounds with multiple exchanged messages as for SMC, it is worth investigating the origin precisely in the following section.

6.3.3.2 Local Communication $T_{local,fresCon}$ in Detail

In theory, $T_{local,fresCon} = T_{flex,fresCon} - T_{flex}$ gives good inference of RTTs for local communication if the system behaves well. As previously seen, initial establishment of communication channels in an end-to-end perspective for EP.S_2 seems to induce almost two times the delay as for EP.S_1. This is unexpected as the assumption is that local communication should take a minimal share in overall request-response times. In order to gain evidence about the origin, fig. 6.6 depicts the distribution of $T_{local,fresCon}$

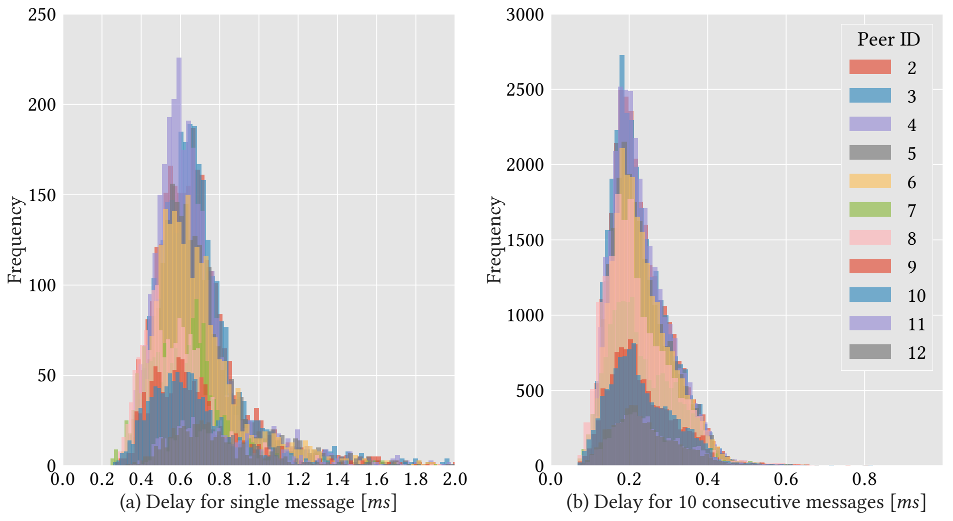


Figure 6.6: Distribution of RTTs for local communication $T_{local,fresCon}$ in comparison. Both figures show the delay per individual message. Peers with smaller IDs are part of more computations and appear more often in the measurements accordingly. The left and right sub-figure show RTTs for a single message EP.M$_1$ ($T^{M_1}_{local,fresCon}$) and multiple consecutive messages EP.M$_{10}$ ($T^{M_{10}}_{local,fresCon}$), respectively. The first message per round always includes delays caused by establishing the socket connection.

measured on all 11 peers locally and individually. It is important to note that only relative frequency rather than absolute values are relevant: the test was operated in multiple steps with two more peers added each time. Thus, peers with smaller IDs (i.e. 2, 3) did participate more often than ones within the higher ID range (i.e. 11, 12). In addition, these delays appear independently from the one in T_{flex} seen before as interactions happen only within the same host in this case.

To begin with, the horizontal shift between single (a) and multi message case (b) indicates that costs for initiating connections shows up again for communication channels within the borders of a local host. This is due to the fact that the delay for a single message exchange $T^{M_1}_{local,fresCon}$ through a new channel is magnitudes of time longer on average than exchanging 10 messages through the same channel $T^{M_{10}}_{local,fresCon}$. In general, the additional delay holds for every first message exchanged through a new channel as correlation with raw data shows. Combining multiple facts, $T^{M_1}_{local,fresCon}$ that is fully affected by the additional costs for a channel's establishment, is the missing part to bridge the gap between the likewise affected parts in EP.S$_1$ ($T^{M_{10}}_{flex}$) and EP.S$_2$ ($T^{M_{10}}_{flex,fresCon}$) depicted in fig. 6.5. More explicitly, this means their local maxima on a histogram plot. Given $fmax(x)$ denotes the global maximum (dominant peak) in the frequency distribution of x by its values binned into multiple bands. And, given $fmax_2(x)$ is the closest local maximum beside the global maximum $fmax(x)$. Let $\#peers = 11$ be the peer configuration. Then, the above statement is practically confirmed by:

$$fmax_2(T^{M_{10}}_{flex}) + fmax(T^{M_1}_{local,fresCon}) \cong fmax_2(T^{M_{10}}_{flex,fresCon}) \\ \cong 0.7ms + 0.7ms = 1.4ms \tag{6.1}$$

Noticeably, the delay $1.4ms$ of eq. (6.1) is the typical request-response latency for our specific test case EP.S_2 with 11 participants.

As RTT for communication through open channels $T^{M_{10}}_{local,fresCon}$ is magnitudes lower (peak around $0.2ms$) than for the single message case, it needs some clearing up from a technical point of view in section 6.3.3.3.

6.3.3.3 Retrospective: Robustness vs. Performance for Local Communication

From implementation side, there are multiple facts which contribute to the overall delay. On the one hand, the gRPC implementation itself plays a minor role. At the time of writing, benchmarks show that the reference implementation for Java responds slower to unary requests than the same does written in Go language. Though, the difference is only in the range of normally some $10\mu s$ up to $200\mu s$ in rare occasions. [48]

On the other hand, an implementation detail in the SMC controller within FlexSMC is most likely to be responsible for the major share of the initial delay. To open a communication channel to Flex Connector in the Java process that also encapsulates SMC logic, a new local socket connection is initiated. As each new session opens a dedicated channel, supporting structures have to be created in gRPC during every first call. Thus. sessions consisting only of a single message exchange, suffer the most as overhead cannot amortize among several messages utilizing an open channel. This means that performance degrades if the majority of sessions only employs a single message exchange, e.g. similar to our simple ping test. Since a session usually consists of at least three messages, it should be an acceptable trade-off.

More importantly, the current way improves robustness and enables simplified deployment: the Java process can be replaced at any time without the need to reload FlexSMC. Instead, each session tries connecting to the SMC backend independently. This provides a quick recover. Like this, small changes can be deployed quickly without interrupting the system as a whole. The same applies if the Java process fails for some reason and is restarted automatically. Admittedly, multiplexing over a shared pool of local connections would be an alternative. Nevertheless, its introduced complexity for bookkeeping and alive-checks could easily exceed its performance benefit.

6.3.3.4 Conclusion

This section discussed the performance for all orchestrating components with respect to end-to-end delay. Beside FlexSMC, this also covers connector logic on Fresco side and all intermediate communication paths. For requests with 10 consecutive message exchanges to 5 and 11 peers, the majority of last replies arrives below $8ms$ and $10ms$, respectively. Individual measurements on a per-message basis approve that even 90% of the requests are answered within $1.4ms$ with all peers participating. Though results are good, local communication was responsible for more delay than anticipated. In-depth analysis localizes this unexpected delay to arise from initially establishing a local communication channel for each session. This could be improved at the expenses of higher complexity and eventually decreased robustness. Altogether, architectural delays are still a fraction compared to our SMC use-case. Therefore, it is a good fit.

6.4 SMC Performance

After having examined the platform performance covering all orchestration components, we enlarge benchmarks to conduct secure multi-party computations which are representative for our use-cases. The goal is to analyze how the system performs for a typical task under real conditions.

6.4.1 Test Details

The scope for these tests embraces a real-world perspective measuring the total request-response latency T_{total} starting from the point in time when a task is submitted to the orchestration layer. By embedding additional measurement points per phase, we keep track of how a phase contributes to T_{total}. As we see later on, it is useful to interpret the black-box result correctly with respect to unexpected behavior by the underlying SMC framework.

Since it is easy to generalize a lot of use-cases in such a way that interactive computations are reduced to a simple sum of multiple inputs, the task is chosen to be the same for all tests. More specifically, the following are the parts a SMC framework needs to take care of:

1. Each peer retrieves data from a local source. It is aggregated to a single decimal.
2. The peers collaborate to calculate the secure sum of each peer's input.
3. Each peer receives the result as part of the secure computation.

Fresco is the supporting SMC framework we used for all tests. For every combination of participating peers, we automatically triggered more than 1000 requests. All of them were issued sequentially.

6.4.2 Results

As part of the design, a generic task is translated to a job that is associated with the current session. A job comprises multiple phases with peer-specific instructions being exchanged in a synchronized fashion. So, a phase only completes if all peers respond to gateway's request correspondingly. Although FlexSMC implements aforementioned structure to fulfill requirements for a high degree of flexibility and robustness, the mechanism to synchronize operations between phases provides a foundation to remotely evaluate performance of SMC aspects.

For each job, the following phases are carried into execution in ascending order:

1. **Prepare Phase** — tells a peer about task details. It also informs each peer about all other peers participating in the same task.
2. **Link Phase** — instructs a peer to connect its Fresco instance with the other peers' one. Like this, the initial delay for interconnecting peers is separated from the actual computations.
3. **Session Phase** — starts the secure computation over the established channels. When it finishes, each peer reports its result individually to the gateway.

Figure 6.7 shows the measured latency for each phase. We discuss the results individually. Finally, fig. 6.8 depicts a real-world or client perspective encompassing all aspects of the system.

6.4.2.1 Prepare Phase

The first phase has informal character for the peers and their SMC units. While it is necessary for planning, it does not rely on any part of SMC. Though, the other way round holds. Beside adding some constant delay for some Java instructions, its main latency sources from $T_{flex,fresCon}$. This means it should be similar to the test case EP.S$_2$ discussed in section 6.3.3. This is an assumption we want to prove briefly.

Results depict that the median delay is in the range of $[1.4, 1.6]ms$ for up to 11 peers. With increasing participants, the result slowly rises. Remember that this is the first message exchange on the complete communication path. This means it suffers from initial delays to establish the structures. Considering eq. (6.1) that represents the RTT for initial message exchange with the Flex connector on SMC side, it states a value of $fmax_2(T^{M_{10}}_{flex,fresCon}) \approx 1.4ms$. So, the result for the prepare phase is quite close to the reference value. This indicates that it behaves similar beside a low offset delay for operations in the JVM.

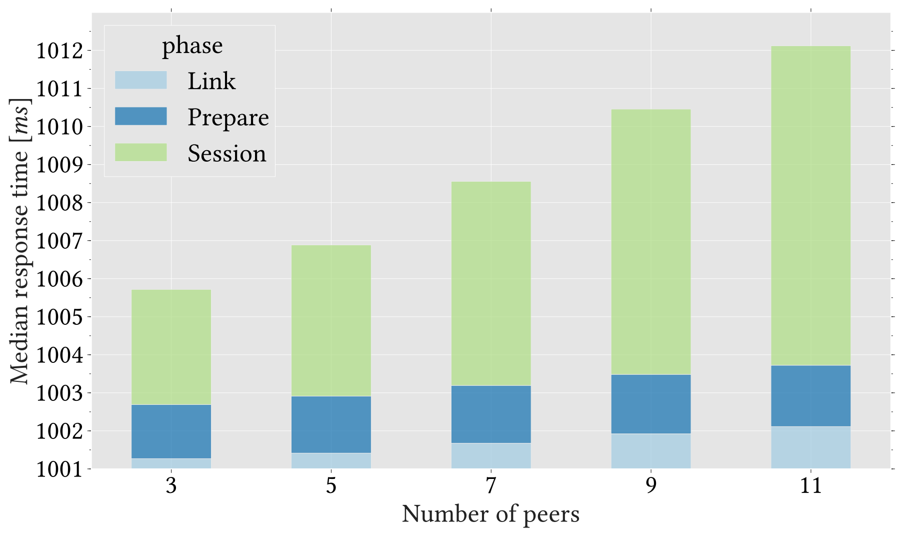


Figure 6.7: Median request-response delays for computing the secure sum. Differently colored bars identify the respective phases' duration. Note that plotting starts from $\approx 1s$ as the most expensive part is the link phase with its inter-connection of SMC peers operated by Fresco.

Table 6.1: Complementary statistics for conducting secure sum tasks, grouped by phase and number of peers. The conducted sequence of phases differs from the shown order. The median of above graphic is represented by the label 50%. Further note that it states the relative time per phase rather the absolute one. Unit: milliseconds [ms]

		min	max	mean	std	25%	50%	75%	90%
phase	peers								
Link	3	1001.0	1503.0	1120.2	213.0	1001.2	1001.3	1001.6	1501.2
	5	1001.1	1504.0	1022.8	101.1	1001.3	1001.4	1001.5	1001.6
	7	1001.3	1501.8	1006.7	49.744	1001.6	1001.7	1001.8	1002.0
	9	1001.4	1503.3	1007.0	49.782	1001.8	1001.9	1002.1	1002.2
	11	1001.6	1503.4	1011.3	66.869	1002.0	1002.1	1002.3	1003.0
Prepare	3	1.114	3.207	1.433	0.133	1.362	1.422	1.491	1.556
	5	1.250	3.320	1.528	0.188	1.441	1.497	1.563	1.641
	7	1.308	3.213	1.551	0.176	1.469	1.520	1.587	1.667
	9	1.315	3.267	1.608	0.236	1.505	1.562	1.629	1.723
	11	1.382	3.364	1.663	0.225	1.552	1.611	1.693	1.840
Session	3	2.676	4.875	3.056	0.244	2.944	3.027	3.113	3.183
	5	3.587	7.615	4.066	0.435	3.876	3.973	4.079	4.217
	7	4.612	11.147	5.493	0.607	5.192	5.364	5.568	5.998
	9	6.058	12.431	7.104	0.617	6.731	6.974	7.264	7.893
	11	6.889	17.375	8.565	0.807	8.147	8.396	8.780	9.319

6.4.2.2 Link Phase

To render Fresco ready for work, it needs to connect all participating peers with each other via bi-directional communication channels. Therefore, it bases upon the library SCAPI [49] that manages the tunnels. As an established communication layer is fundamental prior to doing any secure computations and its necessity for recreation when topology changes, it is interesting to measure timings separately from the actual computations.

Evaluation shows that the link phase is the most dominant one. It takes approx. $[1001, 1002]ms$ at a median delay for our peer configurations. Noticeably, it scales considerably better than these huge values might anticipate. The minimal delay of $1s$ indicates that a timer in SCAPI waits at least for this duration before checking whether connections are ready. The standard deviation shows another interesting property of the network layer: being quite large for a single isle configuration (3 and 5 participants), it decreases drastically with even more peers. Correlation with raw data shows that link times are discrete values ($1s$ and $1.5s$) with jitter in the range of a few milliseconds. Generally speaking, fewer peers are more likely to take $1.5s$ instead of $1s$ until interconnection is successful. This contradictory behavior would need some additional investigation if the system was considered for on-demand tasks with real-time requirements.

6.4.2.3 Session Phase

The session phase is finally the place where secure computations take place. It measures the time to calculate the secure sum of distinct inputs per peer based on the BGW protocol described in section 2.2.4, plus the time for re-sharing the secrets so that each peer can deviate the final result individually prior reporting it to the gateway.

Due to the nature of SMC, each peer interacts with all other peers. This implies an $O(n^2)$ communication complexity. For a secure sum, there are two such rounds of synchronization: first, a peer deducts n shares from its local input and distributes them among all peers, while expecting the same from the others. Second, local processing on each peer generates a new share that is announced to all.

Given this knowledge, it is surprising that results imply a linear scale with increasing number of participants. Several facts explain this behavior. First of all, there are only two communication rounds. As the computation makes use of sums only, a lot of work is done locally. Due to the lack of a multiplication, there are no dedicated rounds for re-sharing their secrets derived from a new polynomial. Thus, the volume of exchanged messages is quite low. In addition to that, message input and output happens in parallel. So, incoming messages are queued until being processed asynchronously. Outgoing messages are buffered until transmission kicks in (non-blocking). Another advantage is the locality of the nodes. In particular, RTTs between any two peers are below one or two milliseconds. This is of special interest as large RTTs have an even bigger impact on the overall performance since $O(n^2)$ communication complexity stills holds from an information-theoretic perspective. To summarize these facts, SMC performs well for applications with fast and minimized inter-communication. Then, it is possible to reduce quadratic costs.

Our test setup even shows an average request-response time below $9ms$ for the maximum peer configuration. This satisfies our requirements for instant feedback and renders it suitable for most general use-cases. For soft real-time applications, the low standard deviation becomes interesting. It clearly shows that jitter is no issue throughout the computations.

6.4.2.4 Total Request-Response Latency

The previous phase-wise evaluations give detailed insights what parts of SMC take which amount of time. The missing piece is an all-embracing perspective onto the whole application combining all parts of this thesis. Thus, the aim of this section is a brief investigation of the total request-response time for the known secure sum task. Differently to taking phase-wise results together, these measurements additionally consider local processing times for the orchestration components. Or more generally, it provides a real-world view from a client using the system as a service.

Figure 6.8 depicts the results for T_{total} of requests issued at the gateway. As expected, the main contributions from link and session phase dominate the offset and relative alignment of the plot's boxes, respectively. Similar, the standard deviation is strongly ruled by the link phase's one. This is not surprising due to the link phase's leader position in terms of absolute numbers. Same as aforesaid, the tests conducted with 3 and 5 peers, are affected the most from SCAPI's non-determinism in terms of latency bouncing between discrete steps of $\approx 500ms$. Other $\#peers$ configurations cause the jitter to behave normally. We identified two direct implications. First, there is a high chance that a request takes $T_{total}(\#peers < 6) \approx 1.5s$ to complete for a scenario with low participation. Given this specific behavior shows up alike in other environments as well, there might be privacy implication as a client gets to know if few peers are participating for a specific task. Second, high jitter makes it difficult to cope with requirements of real-time systems as planning becomes unstable.

Beside that, the overall response latency increases linearly with more peers. As described for the session phase (see section 6.4.2.3), $O(n^2)$ communication complexity will gain significance with more complex tasks. This is a present limitation of many SMC frameworks and more specifically related to the employed BGW protocol. It is up to research to make it more efficient for complex scenarios. Nevertheless, the numbers are promising and should be adequate for most services as most computations should be

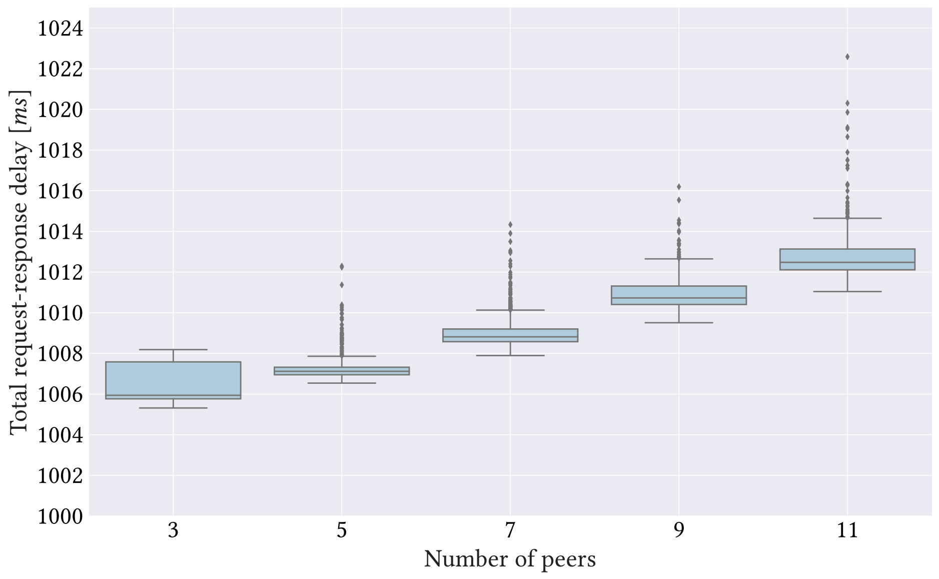


Figure 6.8: Computation of a single secure sum per round with different number of participants. The result represents an end-to-end perspective on the system measuring the elapsed time for starting a request, traversing all phases, its network round-trips, the preparation and execution on Fresco side, and returning the result back to the client.

Table 6.2: Complementary statistics for total request-response time of secure sum task. Unit: milliseconds [ms]

	min	max	mean	std	25%	50%	75%	90%
peers								
3	1005.3	1508.2	1124.9	213.0	1005.8	1005.9	1007.6	1505.9
5	1006.5	1510.0	1028.6	101.1	1006.9	1007.1	1007.3	1008.5
7	1007.9	1509.6	1014.0	49.749	1008.6	1008.8	1009.2	1010.1
9	1009.5	1513.2	1015.9	49.792	1010.4	1010.7	1011.3	1012.0
11	1011.0	1514.4	1021.8	66.877	1012.1	1012.5	1013.1	1013.9

quite simple, i.e. peers usually need only a small amount of rounds for synchronizing with each other.

In direct comparison with individual phases taking together per peer configuration, T_{total} stays largely the same. As expected, the orchestration layers by FlexSMC account for negligible delays which are of no consequence. Thus, the architectural design fits well to the requirements of SMC applications without compromising operational performance.

6.4.2.5 Conclusion

For a real-world perspective, FlexSMC was set to full operation integrating Fresco as a SMC provider. On the one hand, end-to-end measurements conducted from a client showed that median request latency ranged from $1005.9ms$ to $1012.5ms$ for an increasing number of peers. In spite of quadratic communication complexity from an information-theoretic perspective, it increased linearly due to minimized and parallelized communication effort of our simple task. A phase-wise investigation showed that the majority of time is spent in the link phase responsible to interconnect Fresco peers. Within Fresco, it is provided by a library called SCAPI. It was responsible that connect times bounced between $1s$ and $1.5s$ as discrete steps. Unexpectedly, fewer $\#peers$ suffered more often from higher delays, i.e. $1.5s$, while more participants mostly had a shorter connection time, i.e. $1s$. Based on the environment, this might potentially disclose additional information to an attacker. Consequently, it might be useful to replace Fresco's SCAPI network backend with a hardened implementation in future work.

Chapter 7

Requirements Assessment

In the analysis in section 3.5, we derive a set of requirements for a privacy-preserving architecture on the basis of SMC, in order to render it applicable for dynamic environments. Or roughly speaking, making it an attractive choice in practice without the need for expert knowledge in the domain of SMC.

This chapter presents a brief overview about how and where we tackle the stated requirements in the design, implementation and practical evaluation of this thesis.

7.1 Privacy

This section states the answer how this thesis fulfills privacy requirements as possessed in section 3.5.1.

≪R.1≫ - ≪R.3≫ **Confidentiality**, **Data Minimization**, **Unlinkability**
These privacy requirements are established by using an appropriate SMC protocol, executed on behalf of distributed computation nodes who generate the data and are their solely owners (see section 4.1.1). More specifically, using SMC enforces data to be shared in a predefined and purpose-bound way. This means that raw data, owned by the distributed nodes, is not revealed in clear-text (Confidentiality). We only slice partial information from the raw data which is currently needed to fulfill a defined task, and this slice should be insufficient to reuse it for other purposes (Data Minimization, Unlinkability). For instance, calculating a secure sum prevents the receiver of the result to derive a standard deviation from that.

Remarkably, these requirements hold as long as the attacker model is not violated (see section 2.2.5 for BGW protocol's guarantees as used in our realization).

≪R.4≫ **Transparency**
The post-processing of raw data is known to the data owners (peers) by letting

themselves take part in the computation (see section 4.1.2.2) on the one hand. On the other hand, we use a generic task definition which states the exact purpose and kind of computations in a signed document (see section 4.4.2). Moreover, a peer receives an even more specific sequence of descriptive phases which contains a list of participants and their unique IDs (see section 4.4.3.3). This provides the peers with additional information for autonomous decision making.

≪R.5≫ **Intervenability**
Given a rich foundation for reasoning about the task and circumstances from the above Transparency requirement, an intelligent peer takes full responsibility about its data and can autonomously decide whether to take part in any computations. The prerequisite therefore is given by the separation of a task into multiple SMC-specific instructions (see section 4.4.3) in order to preserve robustness. The separation prevents a computation session to turn completely useless, given a peer should opt out in an early stage (cf. section 4.4.3.4 and 4.4.4.3).

In our concrete prototype implementation of the SMC provider, a peer exercises his right for Intervenability if overall amount of collaborators is below 3, for instance. Hence, a disposal of raw input is likely with respect to the violated attacker model.

Some general remarks:
In our design, we introduce a gateway role that inherits a (virtually) central position and maintains a directory of available peers (cf. section 4.1.2.1 and 4.4). One might argue that a gateway's centrality imposes a major concern with respect to peer's privacy. In fact, it is not a privacy problem: in the former architecture as described by the use-cases in section 3.1, sensible data is collected centrally. This is not the case anymore. Instead, only technical data for robust and secure orchestration of the computations is stored centrally (cf. section 4.4.1.3). Furthermore, most of the information comes solely from the peers. Hence, they have full control, which information to disclose to the gateway. The remaining meta data, e.g. the IP address of a peer and whether it is on-line, is also known to every other peer during a secure computation.

7.2 Security

This section states the answer how this thesis tackles security requirements as possessed in section 3.5.2.

≪R.6≫ **Mutual Authentication and Encryption**
Gateway and peer exchange self-signed X.509 certificates during initial pairing (see section 4.2.2). Trust is established by means of the trust-on-first-use paradigm and a manual out-of-band verification. Based on this, communication is encrypted

and mutually authenticated using TLS with client authentication (cf. section 5.3.1).

A current drawback regarding the SMC provider's implementation is the missing encryption and authentication between peers during a secure computation session. So, malicious (active) adversaries could be outsiders, hence, not taking part in the computations, but still be able to tamper with the protocol. Noticeably, a gateway could detect irregularities as an attacker might not be able to influence all decentralized peers at the same time.

Anyways, securing intercommunication is important to be tackled in future work.

≪R.7≫ **Verification of Identities**
As aforementioned, client authentication in TLS allows to mutually verify identities when a peer connects to a gateway. Still, the missing verification between intercommunicating peers is an open issue in the current SMC provider's implementation.

7.3 Deployment and Applicability

This section states the answer how this thesis realizes deployment and applicability requirements as possessed in section 3.5.3.

≪R.8≫ **Zero-Configuration**
This requirement is realized by an automated local discovery mechanism as described in section 4.2.1. Implementation-wise, a fundamental contributor is the *zeroconf* component, that was developed separately as part of this thesis (see section 5.1)

≪R.9≫ **Virtual Centrality**
The gateway is the responsible point of virtual centrality that hides complexity originating from applying SMC and decentralized peers in a dynamic environment, behind a simple interface (see section 4.4.1.2). Prerequisite for the interface's usability without specialized knowledge about SMC is the generic task description (see section 4.4.2).

≪R.10≫ **Fast Responses**
As demonstrated in the practical performance evaluation in chapter 6, the response times of the orchestration layer is significantly below 1 or 2 *ms* most of the time (cf. section 6.3). Combined with the consumed time our utilized SMC provider takes for linking all peers and doing secure computations each time on demand, a single, isolated request-response adds up to a maximum of 1014*ms* on the majority (90%) for a secure addition involving 11 peers (cf. section 6.4).

≪R.11≫ **On-demand Sessions**
An incoming request from a service client invokes a gateway's orchestration component. This triggers a series of actions to identify a set of peers and to interconnect them on demand, by instructing the peers with a special Link phase, as described as part of section 4.4.3.3.

≪R.12≫ **Adaptiveness**
A gateway's directory component continuously keeps track of all its attached peers (see section 4.4.1.3 and 5.2.3.1). Before a task is executed on behalf of a set of peers during task orchestration, available and suitable peers are identified as illustrated in section 4.4.3.2. In a nutshell, the process of preparing a task considers the current circumstances in the environment by requesting the directory first.

≪R.13≫ **Robustness**
Being a fundamental requirement of this thesis, robustness is targeted by various aspects of the architectural design, that act together to achieve the goal. From a peer's perspective, an essential contributor towards robustness is its resilient integration with an available gateway (see section 4.3.1). On gateway side, a cooperation of task orchestration (see section 4.4) and the directory's monitoring component (see section 4.3.2) contribute significantly to the overall robustness. More specifically, the separation of a task into multiple small phases that constitute a specialized SMC job (cf. section 4.4.3), the fault-tolerant job execution (cf. section 4.4.4) and especially its practical realization in terms of reliability and decoupling (see section 5.2.3) are the key to success.

≪R.14≫ **Self-Sustainable**
No part of either design nor implementation relies on any external service. Self-configuration makes solely use of special capabilities in local area networks, e.g. multicast messages. Beside that, the main implementation *FlexSMC* compiles to a self-contained binary, runnable on most platforms without any further dependencies thanks to the Go compiler. The concrete SMC provider we contribute on basis of the SMC framework Fresco [7], is packed to a single Java archive (see section 5.1). Hence, this part requires a current Java runtime installed on the host. This is the case anyway most of the time.

Chapter 8

Related Work

In recent research, Secure Multi-Party Computation has been chosen increasingly as building block in real-world applications when coping with computations on sensible data while preserving privacy of the data origin. In fact, high computational and communication overhead were ancient issues that extinguished interest in SMC outside of research. By now, fundamental work has been done to optimize computation primitives, protocols and leverage parallelization in order to make SMC a (mostly) considerable choice in practice. In the following, we first name some representative work in this domain before pinpointing common deficits that might render it still impractical to be run continuously, without extensive maintenance or under the influence of dynamic environments, as discussed as part of this thesis.

The first well-known deployment of SMC in a large setup was a Danish sugar beet double auction in 2008, which provided a secure auction platform for farmers to trade their production outcome, and for buyers to give bids, resulting in an optimal assignment without exposing unmatched participant's inputs. In their related work [50], Bogetoft *et al.* describe their architecture to consist of three fixed servers, dedicated to execute the auction-specific secure computations on demand. In a previous step, participating farmers use an applet to specify their offer, that in turn gets derived shares to be encrypted with the public keys of the computation servers before uploading the data to a centralized collector. In the computation step, three actors decrypt and load the inputs meant for their respective computation server they are in charge, from the central database, and finally kick off the secure computations. Clearly visible, the cons of this approach are that it involves many manual steps and administration requires specific knowledge about the system.

In the work [29], Burkhart *et al.* present the novel library *SEPIA*, that arose from the intend to evaluate practicability of SMC for multi-domain network security analysis. Beside a full set of primitive functions to lay out a base layer for arbitrary secure computations, the authors provide efficient SMC protocols for comparison and vector addition operations, but also more specialized protocols for event correlation and computation of network statistics. In an evaluation in a local and a wide area network with up to 9 distributed computation nodes, their implementation demonstrates promising performance results. Though, SEPIA and their deployment lacks automatism: for evaluation, actors generated input files from traffic captures and provided them manually to the respective nodes. Moreover, in terms of communication security, they assume that certificates have been deployed securely beforehand. Similar, privacy peers need to be supplied manually with their mutual network locations. For more nodes to be added on demand, the needed administrative effort is not practical. With regard to failing computation nodes, they mention an interesting side note though: by leveraging redundancy properties of Shamir's Secret Sharing during reconstruction, some failed peers could be mitigated without effecting the overall result. Being a very interesting approach to tackle this problem at a lower level than we do, it is not implemented, unfortunately.

While centering around a different use-case, the work in [37] practically uses the SEPIA framework to realize a secure trading system of CO_2 allowances in the aerospace sector. Their way of deploying SMC is not unique to this solution, but similarly applied from time to time in other solutions as well. So, while not going into detail in general, this particular deployment is briefly regarded. There are several SMC input clients which apply secret sharing to split the participant's row data into multiple shares. These shares are then distributed among the computation nodes. In this case, these are three hosts installed in a separate network, presumingly administered by a single person named *Cloud System Admin* in the paper. This arises two concerns: first, an evil administrator could combine the submitted shares to reconstruct the inputs. Secondly, the SMC clients loose ownership over their data. As they do not participate actively in the computations, the collective of SMC computation nodes acts as some kind of virtual or simulated TTP. Further, the SMC clients, being interested in the overall outcome, must trust the computation network to reply the result truthfully.

An extensive real-world deployment of SMC is demonstrated in the work [6] by Bogdanov *et al.* The overall aim is the practical usage of SMC for the members of an Estonian consortium in the information and telecommunication sector to jointly and securely analyze their financial data regularly. Faster results are useful for the members for better decision making regarding their businesses. Specifically, the proposed deployment is interesting in this context: it is based on *SHAREMIND* [32] and employs three distributed hosts for secure computations called data miners, hosted by independent companies. To collect the necessary data, each host runs an additional web server. Using the hosted web page, the consortium members manually provide their raw data in a web form. A locally executed JavaScript library then handles secret sharing the input and submits shares securely to all three data miners individually. When the consortium agrees, the servers employ secure computations on behalf of the submitted data and generate a report. Although it seems to be a working real-world example and maintenance effort is reduced to hosting a web page for the end user, the authors admit that the current effort for maintaining the servers by individual parties was undesirable in the long run. While the companies are still offering their hardware, a single administrator controls all three miners and kicks off computations manually. Similar to the aforementioned work, this approach arises some concerns: the administrator could manipulate the miners to collude in order to reconstruct the inputs, and, eventually, to disclose it. Further, the need for this separate administrator raises questions how to simplify deployment and to automate the miners' control and execution. Last but not least, to keep in mind that similar to aforementioned work, ownership is lost as soon as shares are transferred to the miners.

Taking together aforementioned works, all approaches realize complete systems with the aim to be practically applicable for real-world applications. While they contribute great optimizations such as improving SMC protocols, and leveraging concurrency in modern computers and networks, they face common deficits in terms of practical deployment (being potentially set up by unexperienced people), robustness for dynamic environments (e.g. failing or unreachable nodes), flexibility and ease of maintenance for a long living service. In addition, due to input entities omitting active collaboration, the possibilities of SMC computations are often reduced to a virtual or simulated TTP. Hence, input parties basically loose ownership and transparency over their data in favor of having a simpler initial deployment.

As noted, the previous works often assume also that input is provided manually in separate steps, whereas our work centers around automatism. We make computationally equipped and mutually networked peers to participate in secure computations, while automatically providing input on their own by means of environmental perception. Rather than having deployments with simulated TTP, our work insists on decentralization in terms of data storage and computations.

While there is some research such as the work [51] by M. Ambrosin *et al.*, that considers the same IoT scenario with smart nodes, relying on cooperation (via consensus-based information fusion), that even works in dynamic environments, we have not found a comprehensive solution that focuses on bringing everything together for improving general applicability while having privacy and easy deployment in mind as well.

Remember that our framework does not provide any SMC functionality for most of its privacy guarantees by its own, but relies on a SMC provider, such as one of the frameworks above, to augment its capabilities. Generally speaking, our contribution is to resolve the mentioned shortcomings and hence, improve general applicability of privacy preserving services on behalf of SMC, without creating a new niche framework.

Chapter 9

Conclusion and Future Work

In this Master's Thesis, we contribute *FlexSMC*, a flexible and robust solution for Secure Multi-Party Computation (SMC) frameworks to raise their friendliness towards applicability in practice, or more loosely speaking, to escape the need for laboratory conditions. As performance of SMC is not a prominent bottleneck [6] in recent frameworks and utilized protocols anymore, applicability in this context concentrates on providing ease of deployment, adaptiveness for needs of dynamic environments, robustness towards offering long living services, and ease of use without expert knowledge. Beside privacy contributions from applying secure computations in a decentralized setting, we further lay out the groundwork for enabling transparency and intervenability. When used, it gives intelligent sensors more insight about a task and circumstances to autonomously take decisions about its behavior (e.g. shall a node collaborate in a certain computation knowing most of the participants live on the same host, indicating a possible Sybil attack?) with the final goal to preserve data owner's privacy.

To achieve this, we first analyze two use-cases' realizations and their improper regarding of privacy aspects. That is the foundation to investigate how SMC can be applied and what problems arise in practice. Finally, this leads to several requirements an architecture shall fulfill to support privacy, security and applicability. From there, we design and implement a modularized and decoupled solution that has no dependency besides an existing SMC framework, and is easy to adopt from a service client's perspective. Performance evaluations show that the overhead of the introduced orchestration layer is minimal compared to the runtime of the SMC framework. While the overall performance results, i.e. inclusive SMC, are promising in our simple test cases, more extensive testing is desirable to elaborate potential inefficiencies and integration issues of FlexSMC with its specific SMC provider.

Despite our far-reaching accomplishments and promising results in the scope of this thesis, there are still plenty of possibilities for improvement and enhancements to expand applicability of FlexSMC-based solutions in real-world applications. We attach an incomplete list of issues and possible enhancements to cope with assumed limitations and known gaps, worth investigated in future works proceeding upon this thesis. Remember that following proposals are ideas that would need further comprehensive analysis in terms of privacy implications before considering their deployment:

Replication of Certificates Currently, the gateway node is in obsession of all peers' certificates for mutual authentication and encryption which it collected during pairings. Though, the peers usually have only the certificates of their known gateways. By replicating all certificates on all nodes, every peer could potentially take over the role of a gateway. Running a consensus protocol beforehand should allow to find a trusted gateway alternative. This removes the present need that a peer explicitly must trust a new gateway first. Moreover, this could tackle the present weakness of a single gateway gaining too much responsibility.

Scalability The current design assumes a limitation in scope by locality, so effectively limiting the number of possible active peers. Bigger areas are equipped with multiple separated gateways that are responsible for nearby devices, but are currently regarded as isolated units implementation-wise. As suggested in the design, a hierarchical structure that applies the composite pattern [52] [38], looks like an adequate solution to unite gateways if a request was targeting all peers of a building, for instance. The composite pattern means in this context that most gateways act like peers towards a parent gateway in order to parallelize computations from the leaf peers over to intermediate gateways/peers up to a root gateway.

REST Integration Unlike requested in the analysis and design, the currently implemented orchestration is exposed by an internal Go interface only, but does not provide any outside facing API yet. A well-designed REST API is a missing building block urged to be added.

Fall-back for mDNS For zero-configuration in terms of network discovery, FlexSMC includes the *Zeroconf* package that employs multicast traffic to announce services. Despite being a great choice regarding self-sustainability and maintenance effort, it might not work in some scenarios with routed traffic or extensive firewall setups. FlexSMC might include an alternative for service discovery. For instance, this could be some bootstrapping mechanism based on a known device and its address.

Appendix A

Evaluation Details

A.1 Phase-wise Round-trip Times for multiple Messages

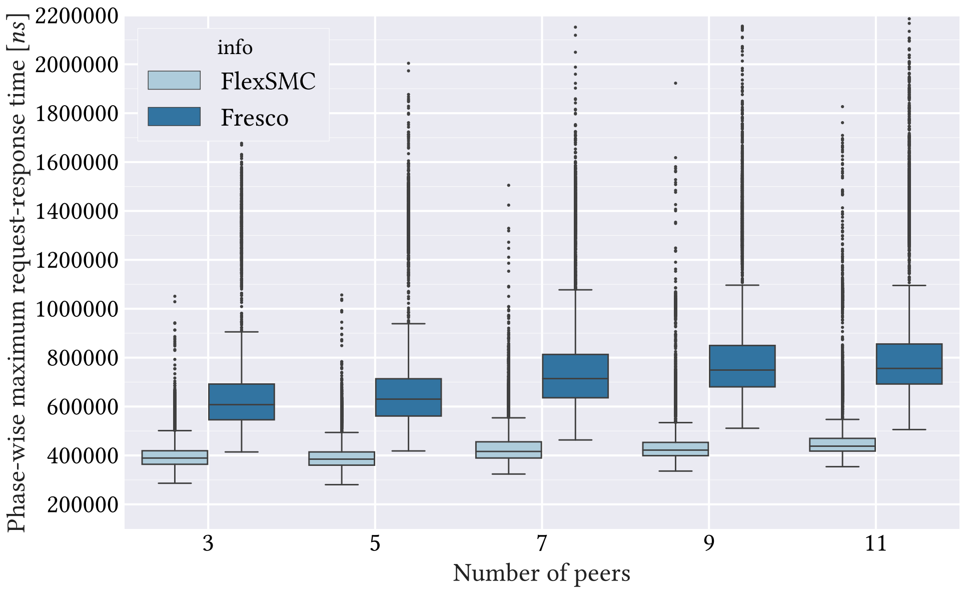


Figure A.1: Individual RTT per message for 10 consecutive ping messages over the established communication channel. The labels FlexSMC and Fresco relate to different stages of message handling (EP.S). Appendix A.2.2.2 enumerates the underlying statistics.

A.2 Table Statistics

A.2.1 Single Message to FlexSMC Layer

	min	max	mean	std	25%	50%	75%
peers							
3	0.488	1.690	0.707	0.094	0.654	0.698	0.742
5	0.549	1.664	0.743	0.107	0.687	0.723	0.772
7	0.622	1.962	0.894	0.197	0.764	0.822	0.998
9	0.639	2.023	1.175	0.312	0.814	1.323	1.391
11	0.736	2.041	1.324	0.273	1.338	1.414	1.463

Table A.1: Statistics for a single ping message to FlexSMC layer. Unit: milliseconds [*ms*]

A.2.2 Multiple Messages to FlexSMC and Fresco Layer

A.2.2.1 Summed Round-trip Times including Preparation and Follow-up

		min	max	mean	std	25%	50%	75%	90%
info	peers								
FlexSMC	3	3.817	5.153	4.328	0.174	4.222	4.324	4.411	4.525
	5	3.887	8.233	4.355	0.248	4.248	4.326	4.425	4.541
	7	4.283	6.290	4.796	0.276	4.602	4.728	4.946	5.142
	9	4.415	6.597	5.026	0.393	4.723	4.882	5.332	5.504
	11	4.701	6.900	5.369	0.444	4.976	5.388	5.624	5.994
Fresco	3	5.953	14.268	7.182	0.734	6.877	7.112	7.340	7.595
	5	6.533	14.206	7.428	0.746	7.129	7.336	7.545	7.854
	7	6.791	15.188	8.301	0.867	7.826	8.185	8.618	9.011
	9	7.419	15.507	9.012	0.890	8.599	8.921	9.247	9.632
	11	8.002	15.536	9.294	0.771	8.898	9.162	9.543	10.004

Table A.2: Statistics for summed analysis of 10 consecutive messages to FlexSMC and Fresco layer. The latter also includes the former in terms of delay. Unit: milliseconds [*ms*]

A.2.2.2 Phase-wise Round-trip Times

info	peers	min	max	mean	std	25%	50%	75%	90%
FlexSMC	3	0.286	1.051	0.404	0.070	0.364	0.389	0.419	0.479
	5	0.280	1.056	0.403	0.075	0.360	0.385	0.414	0.515
	7	0.324	1.505	0.447	0.102	0.389	0.416	0.456	0.595
	9	0.336	1.923	0.466	0.140	0.399	0.422	0.453	0.637
	11	0.354	1.827	0.499	0.175	0.418	0.438	0.470	0.700
Fresco	3	0.414	2.138	0.682	0.243	0.546	0.608	0.692	1.122
	5	0.418	2.004	0.704	0.253	0.561	0.630	0.714	1.190
	7	0.463	2.152	0.790	0.255	0.636	0.714	0.813	1.268
	9	0.511	2.156	0.856	0.287	0.681	0.749	0.850	1.367
	11	0.506	2.186	0.879	0.307	0.692	0.756	0.856	1.407

Table A.3: Statistics for phase-wise analysis of 10 consecutive messages to FlexSMC and Fresco layer. The latter also includes the former in terms of delay. Unit: milliseconds $[ms]$

Bibliography

[1] J. K. W. Wong, H. Li, and S. W. Wang, “Intelligent building research: A review,” *Automation in Construction*, vol. 14, no. 1, pp. 143–159, 2005.

[2] R. Roman, J. Zhou, and J. Lopez, “On the features and challenges of security and privacy in distributed internet of things,” *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.

[3] TNS Opinion & Social, “Special Eurobarometer 359 - Attitudes on Data Protection and Electronic Identity in the European Union,” Tech. Rep., 2011. [Online]. Available: http://ec.europa.eu/public_opinion/archives/ebs/ebs_359_en.pdf

[4] TNS Opinion & Social, “Special Eurobarometer 431 - Data Protection,” Tech. Rep., 2015. [Online]. Available: http://ec.europa.eu/public_opinion/archives/ebs/ebs_431_en.pdf

[5] G. Danezis, J. Domingo-Ferrer, M. Hansen, J.-H. Hoepman, D. L. Metayer, R. Tirtea, and S. Schiffner, “Privacy and Data Protection by Design - from policy to engineering,” European Union Agency for Network and Information Security (ENISA), Tech. Rep., 2015. [Online]. Available: http://arxiv.org/abs/1501.03726

[6] D. Bogdanov, R. Talviste, and J. Willemson, “Deploying Secure Multi-Party Computation for Financial Data Analysis,” in *Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, A. D. Keromytis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 57–64.

[7] Alexandra Institute (Security Lab), “FRESCO: a FRamework for Efficient Secure COmputation,” 2016, Accessed on 2016/11/21. [Online]. Available: https://github.com/aicis/fresco

[8] European Court of Human Rights, “Treaty No.005 - Convention for the Protection of Human Rights and Fundamental Freedoms,” Rome, 1950, Accessed on 2017/02/03. [Online]. Available: http://www.coe.int/en/web/conventions/full-list/-/conventions/rms/0900001680063765

[9] Nest Labs, "Frequently asked questions about privacy." 2017, Accessed on 2017/01/31. [Online]. Available: https://nest.com/privacy-faq/

[10] Telekom Deutschland GmbH, "10 Gründe, warum sich Magenta SmartHome lohnt," 2017, Accessed on 2017/01/31. [Online]. Available: https://www.smarthome.de/stories/10-gruende-warum-sich-magenta-smarthome-lohnt

[11] European Parliament and European Commission, "Directive 1995/46/EC on protection of individuals with regard to the processing of personal data on the free movement of such data," pp. 31–39, 1995.

[12] J. B. Ullrich, "CVE-2015-1600 - Netatmo Weather Station Cleartext Password Leak," 2015, Accessed on 2017/02/01. [Online]. Available: http://seclists.org/bugtraq/2015/Feb/108

[13] B. Krebs, "Yahoo: One Billion More Accounts Hacked," 2016, Accessed on 2017/02/01. [Online]. Available: https://krebsonsecurity.com/2016/12/yahoo-one-billion-more-accounts-hacked/

[14] L. Tung, "Dropbox bug kept users' deleted files on its servers for six years," 2017, Accessed on 2017/02/01. [Online]. Available: http://www.zdnet.com/article/dropbox-bug-kept-users-deleted-files-on-its-servers-for-six-years/

[15] D. Korff, "Comparative Study on Different Approaches to new Privacy Challenges, in particular in the light of Technological Developments, Working Paper N° 2, Data protection laws in the EU: The difficulties in meeting the challenges posed by global social and technical developments," 2010. [Online]. Available: http://ec.europa.eu/justice/data-protection/document/studies/files/new_privacy_challenges/final_report_working_paper_2_en.pdf

[16] M. Rost and K. Bock, "Privacy by Design und die Neuen Schutzziele: Grundsätze, Ziele und Anforderungen," *Datenschutz und Datensicherheit (DuD)*, vol. 35, no. 1, pp. 30–35, 2011.

[17] European Commission, "Proposal for a Regulation of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation)," vol. 0011, pp. 1–119, 2012. [Online]. Available: http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf

[18] A. C. Yao, "Protocols for secure computations," *23rd Annual Symposium on Foundations of Computer Science (SFCS)*, pp. 1–5, 1982.

[19] J. Perry, D. Gupta, J. Feigenbaum, and R. N. Wright, "Systematizing Secure Computation for Research and Decision Support," *Security and Cryptography for Networks - 9th International Conference (SCN)*, pp. 380–397, 2014.

[20] R. Sheikh, B. Kumar, and D. K. Mishra, "Privacy-Preserving k-Secure Sum Protocol," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 6, no. 2, p. 5, 2009.

[21] R. Sheikh, B. Kumar, and D. K. Mishra, "A Modified ck-Secure Sum Protocol for Multi-Party Computation," *Journal of Computing*, vol. 2, no. 2, pp. 62–66, 2010.

[22] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and Performance of Programmable Secure Computation," *IEEE Security Privacy*, vol. 14, no. 5, pp. 48–56, 2016.

[23] A. Shamir, "How To Share a Secret," *Communications of the ACM (CACM)*, vol. 22, no. 1, pp. 612–613, 1979.

[24] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault Tolerant Distributed Computation," *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 1–10, 1988.

[25] A. C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (SFCS)*, 1986, pp. 162–167.

[26] G. R. Blakley, "Safeguarding cryptographic keys," *International Workshop on Managing Requirements Knowledge*, p. 313, 1979.

[27] G. Asharov and Y. Lindell, "A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation," *Cryptology and Information Security Series*, vol. 10, no. 189, pp. 120–167, 2013.

[28] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. New York, USA: Cambridge University Press, 2015.

[29] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics," *Proceedings of the 19th USENIX Conference on Security*, p. 15, 2010.

[30] VIFF Development Team, "VIFF, the Virtual Ideal Functionality Framework," Accessed on 2017/03/13. [Online]. Available: http://viff.dk

[31] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: A System for Secure Multi-Party Computation," *Proceedings of the 15th ACM conference on Computer and communications security*, pp. 257–266, 2008.

[32] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ser. ESORICS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 192–206.

[33] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft, “Confidential Benchmarking based on Multiparty Computation,” *Cryptology ePrint Archive*, 2015.

[34] B. Pinkas, F. Kerschbaum, F. Hahn, T. Schneider, M. Zohner, and R. Rebane, “PRACTICE: D11.2 An Evaluation of Current Protocols based on Identified Model,” BIU, Tech. Rep., 2015. [Online]. Available: https://practice-project.eu/downloads/publications/Deliverables-Y2/D11.2-Evaluation-Protocols-Identified-Model-PU-M24.pdf

[35] J. Schlamp and G. Carle, “measrdroid,” 2014, Accessed on 2017/02/05. [Online]. Available: http://www.droid.net.in.tum.de/

[36] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao, “A reliable and accurate indoor localization method using phone inertial sensors,” *ACM Conference on Ubiquitous Computing (UbiComp)*, pp. 421–430, 2012.

[37] M. Zanin, T. T. Delibasi, J. C. Triana, V. Mirchandani, E. Álvarez Pereira, A. Enrich, D. Perez, C. Paşaoğlu, M. Fidanoglu, E. Koyuncu, G. Guner, I. Ozkol, and G. Inalhan, “Towards a secure trading of aviation CO2 allowance,” *Journal of Air Transport Management*, no. February, pp. 1–9, 2016.

[38] G. Zyskind, O. Nathan, and A. Pentland, “Enigma: Decentralized Computation Platform with Guaranteed Privacy,” *CoRR*, vol. abs/1506.03471, 2015. [Online]. Available: https://arxiv.org/abs/1506.03471

[39] S. Cheshire and M. Krochmal, “DNS-Based Service Discovery,” Internet Engineering Task Force (IETF), RFC 6763, February 2013. [Online]. Available: https://tools.ietf.org/html/rfc6763

[40] S. Cheshire and M. Krochmal, “Multicast DNS,” Internet Engineering Task Force (IETF), RFC 6762, February 2013. [Online]. Available: https://tools.ietf.org/html/rfc6762

[41] L. Poettering and T. Lloyd, “Avahi Service Discovery Suite,” Accessed on 2017/03/04. [Online]. Available: https://github.com/lathiat/avahi

[42] Google, “gRPC - a high performance, open-source universal RPC framework,” Accessed on 2017/02/22. [Online]. Available: http://www.grpc.io

[43] Google, “Protocol Buffers,” Accessed on 2017/03/04. [Online]. Available: https://developers.google.com/protocol-buffers/

[44] Y. Sheffer, R. Holz, and P. Saint-Andre, “Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS),” Internet Engineering Task Force (IETF), Technische Universität München, RFC 7457, February 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7457

[45] V. Pai, "gRPC Design and Implementation," 2016, Accessed on 2017/01/20. [Online]. Available: http://platformlab.stanford.edu/SeminarTalks/gRPC.pdf

[46] K. Damgård, "Fresco: BGW suite - consecutive IDs," 2015, Accessed on 2017/03/05. [Online]. Available: https://github.com/aicis/fresco/issues/2

[47] D. Oxenhandler, "Designing a Secure Local Area Network," 2003, Accessed on 2017/01/14. [Online]. Available: https://www.sans.org/reading-room/whitepapers/bestprac/designing-secure-local-area-network-853

[48] Google, "Perfkit - gRPC Performance Multi-language," 2017, Accessed on 2017/01/25. [Online]. Available: https://performance-dot-grpc-testing.appspot.com/explore?dashboard=5652536396611584

[49] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell, "SCAPI : The Secure Computation Application Programming Interface," *IACR Cryptology ePrint Archive*, no. 629, 2013. [Online]. Available: http://eprint.iacr.org/2012/629.pdf

[50] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, "Secure multiparty computation goes live," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5628 LNCS, pp. 325–343, 2009.

[51] M. Ambrosin, P. Braca, M. Conti, and R. Lazzaretti, "ODIN: Obfuscation-based privacy preserving consensus algorithm for Decentralized Information fusion in smart device Networks," *CoRR*, vol. abs/1610.0, October 2016. [Online]. Available: https://arxiv.org/abs/1610.06694

[52] G. Cohen, I. B. Damgård, Y. Ishai, J. Kölker, P. B. Miltersen, R. Raz, and R. D. Rothblum, "Efficient multiparty protocols via log-depth threshold formulae (Extended abstract)," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8043 LNCS, no. 2, pp. 185–202, 2013.