Interactive Private Multi-Party Calendar Scheduling

Jonas Nagy-Kuhlen

June 15, 2018
Abstract

Calendar scheduling software enjoys great popularity as a basic tool for collaboration, but generally requires users to disclose information on their personal timetables. This thesis presents a novel approach for decentralized scheduling based on secure multi-party computation (MPC), allowing multiple parties to agree on an appointment date in a privacy-preserving manner. Previous work on MPC tended to look at the underlying cryptographic concepts in isolation, only vaguely relating to real-world applications. In contrast, the primary contribution of this work is the implementation of a fully functional prototype that incorporates privacy enhancing techniques while also taking practical requirements such as usability and flexibility into account. Scheduling is performed by first carrying out a public session setup phase, followed by the actual secure computation which evaluates a date selection algorithm expressed in boolean logic. It shows that the developed prototype comes close to existing scheduling software in terms of usability and allows interactive running times in the order of few minutes even for large numbers of participants, thereby proving that MPC techniques are indeed applicable in practical use cases.

1This abstract was not included in the submitted version of this thesis.
# Contents

1 Introduction 1
   1.1 Motivation ........................................... 1  
   1.2 Thesis Goal ........................................ 2  
   1.3 Outline ........................................... 3  

2 Background and Related Work 5  
   2.1 Secure Multi-Party Computation .................. 5  
      2.1.1 Protocols ....................................... 6  
      2.1.2 Private Set Intersection ...................... 9  
      2.1.3 Frameworks .................................... 9  
   2.2 Calendar Standards ................................ 11  
   2.3 Summary ........................................... 12  

3 Use Case and Requirements 13  
   3.1 Threats ............................................. 15  
   3.2 Scheduling Schemes ............................... 16  
   3.3 Requirements ...................................... 17  
      3.3.1 User Interface ................................. 18  
   3.4 Framework Choice ................................ 19  
   3.5 Summary ........................................... 20  

4 Conceptual Approach 23  
   4.1 Circuit Design ...................................... 26  
      4.1.1 Basic Building Blocks ......................... 26  
      4.1.2 Scheduling Circuits ........................ 30  
   4.2 Summary ........................................... 32  

5 Implementation 35  
   5.1 Circuit Generation ................................. 37  
      5.1.1 Static Optimization ........................... 39  
      5.1.2 Building Block Representation .............. 40
## Contents

5.1.3 Secure Evaluation ........................................ 42  
5.2 Invitation Management and Networking .......................... 43  
  5.2.1 Invitation Broadcasting .................................. 43  
  5.2.2 Session Setup .......................................... 44  
5.3 Graphical User Interface ...................................... 46  
5.4 iCalendar Integration ........................................ 48  
  5.4.1 Input Extraction ........................................ 49  
5.5 Current Limitations .......................................... 51  
5.6 Summary .................................................... 52  

6 Evaluation .................................................... 55  
  6.1 Performance Measurements .................................. 55  
    6.1.1 Running Time ........................................ 56  
    6.1.2 Network Utilization ................................... 61  
  6.2 Threat Assessment .......................................... 64  
    6.2.1 Attacks on Correctness ................................ 65  
    6.2.2 Attacks on Privacy .................................... 66  
  6.3 Practical Use ............................................... 68  
  6.4 Summary .................................................... 69  

7 Discussion .................................................... 73  

8 Conclusion .................................................... 75  
  8.1 Future Work ................................................. 76
Chapter 1
Introduction

1.1 Motivation

Finding a free time slot for a meeting with a number of employees is every-day routine at most companies. It is in fact so common that there exist a lot of easy-to-use software tools specifically for this task, the most prominent arguably being Doodle\(^1\) and Calendly\(^2\). Although these services promise an increase in productivity by providing an uncomplicated workflow to schedule appointments, they are problematic from a privacy perspective. Users need to make their calendar public not only towards their colleagues, but also towards a third party with own economic interests. The shared information can be used to draw conclusions about the user’s daily structure and current workload. In the worst case, knowing the availability and non-availability of certain employees can help social engineers to mount an attack against the associated company. Therefore, from a user’s point of view, it is desirable to leak as little information as possible while accomplishing the task of scheduling an appointment.

At first glance, it appears to be impossible to unite the two contrary goals of keeping a user’s calendar private on the one hand, and still agreeing on an appointment date that is approved by every user on the other hand. It turns out, however, that this problem can be solved with methods of secure multi-party computation (MPC), a cryptographic field which became practical in the last couple of years. MPC algorithms allow a number of parties to compute the result of a specified function without learning any inputs of other parties or intermediate values. This mechanism can be used as a basis for implementing a secure appointment scheduling application that

\(^1\)https://doodle.com
\(^2\)https://calendly.com
treats the users’ calendar information as private input and nevertheless allows to collaboratively evaluate a scheduling function whose output, a free time slot for the appointment, is shared between all participants.

1.2 Thesis Goal

The goal of this work is to develop a secure calendar scheduling application that fulfills the following properties:

- It allows users to agree on a date for an appointment while keeping their timetables private from the others.

- Scalability must be achieved for a number of participants that is common in real-world scenarios. As a reference, a count up to 30 is considered reasonable.

- Agreeing on an appointment date remains interactive even with a high number of collaborators. The required time for finishing the protocol needs to be in the order of seconds or few minutes to guarantee a convenient workflow.

- The application exposes a clean and straightforward graphical user interface and integrates into established calendar applications such as Microsoft Outlook, Thunderbird Lightning and Calendar on Mac operating systems.

- The underlying protocol is designed for flexibility and extensibility and does not impose unnecessary restrictions regarding its use case. In particular, the scheduling scheme, i.e. the algorithm which decides on an appointment date dependent on the user’s private information, should be easily customizable.

Using an MPC protocol as a basic building block for such an application is the natural choice since it directly, by definition, provides a solution for the problem of ensuring privacy of the users’ inputs. Scalability, on the other hand, is an inherent problem of existing $n$-party MPC protocols. In the absence of a central trusted server, each party needs to communicate with every other party directly [Sma16] and thus the resulting theoretical quadratic runtime complexity needs to be considered as part of this work.

Although there exist multiple frameworks for practical MPC [BDNP08; Ejg+12; Gei07; Dam+13], this technology has not really been used in real-world applications with the exception of a sugar beet auction in Denmark.
1.3 Outline

Instead of being another cryptographic playground, this work aims to create a complete practicable system that could directly be integrated into a company’s working routine. Therefore, non-cryptographic requirements such as user experience, integration into existing workflows and adherence to common standards must be ascribed importance to. When considering security, the classical adversary models for MPC cannot be discussed in isolation anymore but need to be complemented by an analysis of external threats, such as man-in-the-middle attacks. Besides an MPC protocol as the central component, other cryptographic primitives, especially those which provide data confidentiality and authentication, are necessary to ensure secure communication and ultimately form a secure system.

The preceding paragraphs were intended to give an intuitive understanding why secure calendar scheduling is desirable and which factors need to be considered for a practical implementation. In the following chapter, the current state of research with respect to secure computation and calendar interoperability is examined. It is detailed how MPC protocols work, how their security can be characterized and which frameworks exist for this purpose. Furthermore, an overview of calendar standards that can serve as a common interface towards established calendar applications is given. Chapter 3 describes the assumptions that are made regarding the use case in which secure scheduling is applicable. The emerging requirements as well as a general approach to fulfill them is presented. A more technical specification of the necessary steps follows in chapter 4, including a comprehensive section on how different scheduling algorithms can be formalized in boolean logic as a prerequisite for secure evaluation. Whereas chapter 4 primarily presents a mathematical perspective, chapter 5 covers the implementation of the accompanying prototype application. An overview of the developed software system is given and different architectural components are identified, listed together with tools and libraries that were used as a starting point. It is detailed how each component is modeled in order to fulfill its purpose and how the interfaces are designed with extensibility in mind. In chapter 6, the implemented prototype is evaluated with respect to the requirements formulated in chapter 3. Measurements of different performance metrics are presented and analyzed with the main focus on scalability, followed by an attempt to characterize the impact of potential attacks on privacy. Chapter 7 briefly discusses to which extent the findings of this work can serve as a basis to promote privacy in other application areas, leaving the narrow scope
of calendar scheduling. Finally, in the last chapter, a conclusion is drawn on how privacy was achieved in the context of this thesis and opportunities for future improvements as well as ensuing research questions are pointed out.
Chapter 2

Background and Related Work

There exist two ingredients which are essential to a practical, secure scheduling application. First, MPC techniques provide means to protect the users’ inputs and therefore achieve privacy. Second, the adherence to common calendar standards ensures a seamless integration into the existing landscape of time management software. The development in these areas of research as well as recent directions are briefly presented in the following sections, focusing on insights regarding the practical use rather than giving an exhaustive description of the underlying theoretical principles.

2.1 Secure Multi-Party Computation

Protocols for secure function evaluation are subject of cryptographic researchers for many years, going back to Yao in 1982 and the work of Goldwasser, Micali and Wigderson five years later [Yao82; Yao86; GMW87]. At this time, it was a purely theoretic field that became practical only recently [Pin+09; Orl11]. The goal of MPC is to be able to evaluate any function collaboratively with a number of parties such that the following two requirements are fulfilled: First, every party must be in possession of the correctly computed function output at the end of the protocol execution. In addition, no party should be able to learn any input of another party with the exception of information that can be directly derived from the output. As an analogy, the MPC protocol can be seen as a replacement for a trusted server that receives the inputs from all parties, computes the function and transfers the result back to each party. MPC protocols can be classified by their security against adversaries, the natural representation of the evaluated function and the supported number of parties, in general referred to as secure two-party and n-party computation.
When reasoning about the security of MPC protocols, cryptographers usually focus on adversaries that are involved in the collaborative computation. External attackers can be excluded using a secure communication channel, as provided by means of classical cryptography. Participating adversaries are assumed to be either semi-honest or malicious [Gol+05; CDN12]. The former, also referred to as passive or honest-but-curious, correctly follows the protocol, but tries to use exchanged information to learn as much about the other parties’ inputs as possible. In case of multiple semi-honest adversaries, they might collude to extract even more information. Malicious (or active) adversaries, on the other hand, are those who can freely deviate from the protocol steps in order to acquire private information or even manipulate the outcome of the computation. Resistance against active attacks is generally much harder to achieve and comes with higher computational costs for each party [DO10].

2.1.1 Protocols

Two major MPC concepts, namely Yao’s garbled circuits and the GMW protocol, were published in the 1980s and provide the basis for practical MPC frameworks of today. In their fundamental form, both are secure in a semi-honest setting and rely on a representation of the evaluated function as a boolean circuit.

The two-party protocol presented by Yao assigns different roles to its participants, namely the creator and the evaluator [Yao82; Yao86]. The creator first garbles the circuit by mapping the two possible values of each wire in the circuit to randomly drawn ciphertexts. The evaluator can then consecutively evaluate each gate based on garbled truth tables provided by the creator, acquiring the ciphertext corresponding to a gate’s output based on the garbled input values. Finally, the creator publishes the mapping from ciphertext to plain bits for those wires which correspond to the function output, whereas the evaluator shares the respective garbled values, allowing both parties to obtain the final result. The creator does not reveal her own input since she only sends the matching ciphertexts. The evaluator, on the other hand, can select the ciphertexts for his input bits by the means of 1-out-of-2 oblivious transfer (OT). In general, 1-out-of-n OT is a cryptographic primitive which allows party A to query one single value out of n values provided by another party B without learning the remaining n – 1 values and without B knowing which value was chosen [Rab05; NP01].

There exist various improvements for Yao’s protocol [Sny14], worth mentioning especially those which make the computation secure and efficient even in presence of a malicious adversary [She+13; FJN14]. Further opti-
mizations drastically reduce communication overhead, most importantly a technique which allows to evaluate XOR gates basically at no costs [KS08]. Despite the conceptual design as a two-party protocol, specialized constructions for up to five parties have been proposed recently [MRZ15; Cha+17]. As of yet, the attempts by Chandran et al. to generalize the garbled circuit concept for \( n \) parties are highly theoretical though and therefore cannot be considered a practical multi-party solution.

As opposed to the protocol by Yao, the GMW protocol supports an arbitrary number of parties involved in the computation and does not assign distinct roles [GMW87]. The evaluated function is assumed to be solely constructed from gates of types XOR, AND and NOT. This set of logical connectives is functional complete and can thus be used to express any conceivable function. Note that binary XOR and AND operations correspond to addition and multiplication in \( \mathbb{F}_2 \), respectively, and are thus used interchangeably in the following. The idea of the GMW protocol is to split the actual value \( v \) of each wire into \( n \) linear shares \( v_1, \ldots, v_n \) for parties \( P_1, \ldots, P_n \) such that, altogether, \( v = v_1 \oplus \ldots \oplus v_n \) holds. Party \( P_i \) can easily create a valid sharing for each private input bit \( u \) by drawing random shares \( u_j \) for every other party \( P_j \neq P_i \) and deriving the own share \( u_i = u \oplus (\bigoplus_{j \neq i} u_j) \). It becomes clear that the individual shares \( u_j \) do not reveal any information on the input \( u \), even if \( n - 1 \) parties collude and add up their shares. Assuming that shares were successfully distributed for all inputs, circuit gates with input wires \( u, v \) and output wire \( w \) can then be evaluated according to the following rules:

- **XOR gate**: Each party \( i \) locally computes its new share \( w_i = u_i \oplus v_i \). Since a linear sharing scheme is used, the gate’s invariant \( w = u \oplus v \) holds.

- **NOT gate**: In order to obtain \( \neg v \) from \( v \), it is sufficient that the first party \( P_1 \) locally negates its share \( v_1 \). Thereby, the resulting sharing is consistent in that \( \neg v = \neg v_1 \oplus v_2 \oplus \ldots \oplus v_n = \neg (v_1 \oplus \ldots \oplus v_n) \).

- **AND gate**: \( w = u \cdot v = (\bigoplus_i u_i) \cdot (\bigoplus_i v_i) = (\bigoplus_i u_i v_i) \oplus (\bigoplus_{i<j} u_i v_j \oplus u_j v_i) \) and thus, the first term \( u_i v_i \) can be computed locally. For the second term, every pair of parties \( (P_i, P_j) \) needs to collaborate to obtain a random share of \( u_i v_j \oplus u_j v_i \) each. This two-party sharing is possible by means of a single 1-out-of-4 OT, as detailed in [Gol98].

As a last step, all parties need to publish their shares for those wires that represent the final function output. By adding up all \( n \) shares, a wire’s true value is finally revealed.
Whereas XOR and NOT operations can be performed basically for free, carrying out $O(n^2)$ OTs per AND gate clearly is the limiting factor for executions of the GMW protocol. However, there exist improvements to make evaluation of AND gates more efficient. First, OTs of AND gates independent of each other can be performed in parallel, encouraging circuit design with low multiplicative depth [Cho+12]. Furthermore, OT extension techniques have been presented which allow fast OTs being derived from a small number of base OTs to evaluate large circuits more efficiently [Ash+13; PSZ14]. Finally, researchers proposed enhancing mechanisms that make the GMW protocol resistant against active attacks at the cost of additional computational complexity [GIP15].

Various other $n$-party MPC protocols emerged from the initial approach by Goldwasser, Micali and Wigderson. The BGW protocol presented in 1988 is based on sharing polynomials instead of bits and allows the secure evaluation of a function represented as an arithmetic circuit, defining integer addition and multiplication with respect to a large field $\mathbb{F}_p$ [BOGW88; AL11]. Depending on the functionality that needs to be implemented, it is necessary to carefully assess whether boolean or arithmetic circuits are the natural choice. Although it is possible for either representation to emulate the other, performance greatly varies, as the following conversions indicate:

- $\mathbb{F}_p$ to $\mathbb{F}_2$: When restricting integers to the values 0 and 1, AND gates are equivalent to multiplication gates. Since $x \oplus y = x + y - 2xy$ holds, formerly free XOR gates can be emulated at the cost of one additional multiplication [Sma16]. Note that simply using the modulus $p = 2$ is not an option because protocols based on arithmetic circuits usually require a large $p$ for security.

- $\mathbb{F}_2$ to $\mathbb{F}_p$: Integers of fixed bit length can be added and multiplied by means of circuits similar to those used in arithmetic logic units of hardware processors. Costly AND gates are necessary for computing the carryover of bitwise additions.

Thus, it is important to know the protocols at one’s disposal and their performance implications when designing efficient applications based on MPC.

Another modification of the GMW protocol, the BMR protocol, allows the joint function to be evaluated in a constant number of communication rounds between the parties, independent of the size of the underlying boolean circuit [BMR90]. This improvement is particularly beneficial in networks with high latency, where every communication round results in a significant delay of the overall protocol execution.
2.1.2 Private Set Intersection

As detailed in the previous section, there exist different generic MPC protocols that allow any function to be evaluated by a number of parties, as long as this function can be expressed in terms of a boolean or arithmetic circuit. Private set intersection, on the contrary, is a field specialized on one particular function, namely computing the intersection of sets provided by each party [FNP04]. Set intersection is a common task that often requires to be performed securely, for example when companies or governmental institutions want to match customer data without granting each other full database access.

Practical cryptographic protocols for both two-party and \( n \)-party set intersection were published in the last years [DCT10; HV17]. However, as of yet, no ready-to-use implementations are available. Furthermore, it has been shown for the two-party case that such specialized protocols are not inherently faster than generic MPC solutions where set intersection is expressed as a boolean circuit, especially for sets over a small domain [HEK12]. Performing set intersection by means of generic MPC protocols greatly increases flexibility, due to the fact that additional functionality can be easily integrated into the evaluated circuit.

2.1.3 Frameworks

As soon as MPC became practical in recent years, various frameworks were developed that allow the integration of secure computations into new software products without the need to have a deep understanding of underlying cryptographic mechanisms. Conceptually, the programming interface an MPC library provides is very simple: Given a suitable description of a boolean or arithmetic circuit, a representation of the communication network (such as IP address and port for every party) and a list of private input values, the framework runs the computation as a black box and returns the circuit result when finished. Knowledge of the implemented cryptographic protocol is not necessary when using such APIs, as long as the required privacy guarantees are met. Although a lot of effort has been put specifically into efficient two-party implementations, there also exist several practicable open-source solutions for \( n \)-party computation. A concise overview is given in table 2.1.

*FairplayMP* emerged from the two-party implementation *Fairplay* published in 2004 [Mal+04; BDNP08]. The framework defines its own high-level language for expressing the shared function, which can then be compiled into a boolean circuit for secure evaluation via the BMR protocol. It is fully written in Java, making prototyping and deployment on different operating
systems very straightforward. However, the codebase is in a disorganized state with major technical shortcomings and an unclear license agreement. These problems are unlikely to be addressed because the project appears to be abandoned since 2015.

Another \(n\)-party implementation based on the GMW protocol is provided by \textit{SCAPI}, short for \textit{Secure Computation API}, a general purpose library that implements various cryptographic building blocks for MPC protocols \cite{Ejg+12}. The library was originally developed in Java and ported to C++ later on for performance reasons. Although the GMW implementation is purely written in C++, there exists a thin wrapper that allows using the same functionality directly in Java. Experimental changes to the protocol are complicated by this architecture due to the fact that both Java and C++ code most likely needs to be modified and recompiled. Since SCAPI is designed as a toolbox for own secure protocols rather than a ready-made MPC suite, it does not include a high-level circuit compiler. Instead, it is the user’s responsibility to construct a boolean circuit for evaluation. The SCAPI project is actively maintained, well documented and exposes a clean, structured and well-conceived codebase.

\textit{SPDZ} can be considered as the direct successor to the discontinued \textit{VIFF} framework \cite{Gei07; Dam+09; Dam+13}. Both projects are similar in the way that they implement custom MPC protocols based on sharing polynomials and thus support arithmetic circuit representations. Instead of manually defining circuits, the user can write Python code to express the evaluated function. Whereas the VIFF library comes with optional security against less than \(n/3\) malicious adversaries, the privacy guarantees of the SPDZ protocol are even stronger: Even if \(n - 1\) parties are actively attacking the computation, no private input can be recovered. Message authentication codes are used to ensure that no party deviates from the protocol. To increase

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|l|l|l|l|}
\hline
Name & Protocol & Circuit Representation & Threat Model & Language & Status & Reference \\
\hline
FairplayMP & BMR & Boolean & Passive & Java & Abandoned & \cite{BDNP08} \\
SCAPI & GMW & Boolean & Passive & Java/C++ & Supported & \cite{Ejg+12} \\
VIFF & Custom & Arithmetic & Passive/active & Python & Discontinued & [Gei07; Dam+09] \\
SPDZ & Custom & Arithmetic & Active & Python/C++ & Supported & \cite{Dam+13} \\
\hline
\end{tabular}
\caption{Overview of existing open source MPC implementations that support an arbitrary number of parties.}
\end{table}
efficiency, both VIFF and SPDZ distinguish between an offline and an online phase. The offline phase is exclusively composed of operations independent of the parties’ inputs and can thus be performed prior to the potentially time-critical computation, allowing much better load balancing.

2.2 Calendar Standards

Exchanging calendars and scheduling appointments quickly became an important application of the world wide web. Thus, iCalendar was proposed in 1998 as a text-based, open file format for calendar data [Des09]. At present, it is widely adopted and supported by major calendar software as a common basis for importing and exporting calendars as well as for sending proposed dates via email.

BEGIN:VCALENDAR
VERSION:2.0
BEGIN:VEVENT
CREATED:20170801T150123Z
DTSTAMP:20170801T150123Z
UID:378243e0-39f9-4f94-985c-86ee5ba1816a
SUMMARY:An Important Meeting
ORGANIZER;CN=Alice;EMAIL=alice@rwth-aachen.de
ATTENDEE;CN=Alice;PARTSTAT=ACCEPTED;
EMAIL=alice@rwth-aachen.de
ATTENDEE;RSVP=TRUE;CN=Bob;PARTSTAT=NEEDS-ACTION;
EMAIL=bob@rwth-aachen.de
DTSTART;TZID=Europe/Berlin:20170812T143000
DTEND;TZID=Europe/Berlin:20170812T153000
END:VEVENT
END:VCALENDAR

Listing 2.1: Contents of an example iCalendar file with a single event. The time zone definition has been omitted for conciseness.

A calendar as defined by the iCalendar standard is made of a variable number of components, such as events and free or busy time slots. These components usually contain the scheduled time frame, textual descriptions of the subject and contact information on the organizer and potential attendees. An example is shown in listing 2.1, including a meeting organized by Alice that has a pending invitation for Bob. Note that iCalendar files only describe a snapshot at a certain time. The standard is intended purely as an exchange format and therefore does not define mechanisms for scheduling
appointments, notifying participants of changes to events, or transferring files to the designated receiver. Agreeing on a common notation but decoupling it from transport and scheduling tasks allows flexible usage of such calendar files.

For collaborative calendar access, the standard CalDAV was developed on top of WebDAV and HTTP [DW05; DDD07]. In contrast to the iCalendar standard, CalDAV specifies how clients can access calendars provided by servers over the network. Users may synchronize with shared calendars in a way that changes are propagated to other collaborators, relying on provided mechanisms to resolve potential conflicts. Integrated access control prevents unauthorized clients from inspecting or modifying private calendars. The iCalendar format is still used in CalDAV to represent calendar resources that can be accessed via common HTTP requests.

2.3 Summary

This chapter gave an overview of secure multi-party computation and common calendar standards as the two major technologies relevant for developing a secure scheduling application.

The purpose of MPC is to allow multiple parties to jointly compute a given function in a way that no party can deduce any information on the inputs provided by other parties. The most common protocols are Yao’s garbled circuits for two parties and the GMW protocol for \( n \) parties based on linear secret sharing. Both concepts rely on functions modeled as boolean circuits, however there exist extensions for arithmetic circuits. The former representation comprises logical gates such as AND, XOR and NOT, whereas the latter allows addition and multiplication of integers. Various practical MPC frameworks have been developed, the major ones that allow generic \( n \)-party computation being SCAPI and SPDZ. Whereas SCAPI provides a fast GMW implementation that directly operates on boolean circuits, SPDZ offers the highest privacy guarantees and comes with a high-level language compiling into arithmetic circuits. There also exist secure protocols specifically tailored to frequently needed functionality such as set intersection, which circumvent intermediate circuit representations altogether.

As far as calendar interoperability is concerned, the iCalendar standard is widely used as a common file format. It defines a straightforward notation for a list of events, deliberately avoiding any assumptions on underlying transport mechanisms in order not to limit the area of application.
Chapter 3

Use Case and Requirements

The secure calendar scheduling software developed in the context of this work is primarily intended to be used by employees within a company or a working group therein. Whenever an appointment is due, the scheduling application provides the following workflow to agree on a date:

1. An initiator opens up the user interface and specifies the time frame in which the meeting should take place, together with a short description on the subject. This information, from now on called the invitation, is then sent to other potential participants, either by a broadcast in the corporate network or by external communication channels such as email.

2. Recipients of the invitation who are willing to attend the appointment establish a connection with the initiator, thereby signaling their cooperation. As soon as the initiator observed all required confirmations, the scheduling protocol is ready to start.

3. All participants now need to determine for each time slot in question whether it is free or occupied, either by entering this information manually or by importing the events from their favorite calendar application. Based on this input data, the secure scheduling computation begins to run in the background.

4. When the protocol completes, every party is notified of the selected date, which can be carried over to the user’s standard calendar management system again.

This workflow heavily relies on connectivity over the corporate network. Since advanced topics on network communication are beyond the scope of this work, the following simplifying assumptions are made:
The machines of all users participating in the protocol can directly communicate with each other via IP addresses and port numbers. In practice, this means that no middleboxes such as firewalls or devices for network address translation restrict connectivity between users.

For broadcasting invitations, it is required that all machines potentially taking part in the computation are within the same broadcast domain. This broadcast domain should not exceed the boundaries of the company’s network.

The corporate network is protected against access from outside the company. However, it is assumed that messages sent within the network can be read by any connected host. This includes employees from other working groups not participating in the appointment scheduling, as well as potential visitors with temporary access to the internal network.

Figure 3.1 illustrates the different roles with regard to the company’s network. Although the network is assumed to be protected against external access, it is possible for internal hosts to eavesdrop on messages of the scheduling protocol even if they do not actively participate.

![Diagram](image-url)

**Figure 3.1:** Roles of hosts in the considered use case. Hosts taking part in the scheduling protocol need to be inside the corporate network since a direct communication channel is required between each pair of participants. Hosts without access to the network are referred to as external.
3.1 Threats

The ultimate goal of this work is to design and implement a scheduling system that preserves privacy of its users’ calendars. The definition of privacy is straightforward in this context: When Alice takes part in scheduling an appointment, she is guaranteed that no person other than herself is able to learn any information on her calendar. In other words, it must be ensured that no data ever leaves her local machine that would leak such information. Analogously, only the users who participated in the protocol should see the result of the computation, namely the date of the appointment that fits best.

In contrast, scheduling invitations are not considered sensitive because they do not include any private information and need to be broadcasted within the network to reach potential attendees. The broadcast domain is assumed to end at the corporate network’s boundaries and thus, invitations cannot be overheard by external hosts.

Adversaries may misuse their role within the network to mount different attacks intended to compromise privacy. The following adversary types can be distinguished:

- **Semi-honest adversary:** Users taking part in scheduling an appointment can try to reconstruct private information of other participants based on exchanged messages and local protocol state. Multiple adversaries might collude in an attack. Since semi-honest employees correctly follow the protocol, they are not at risk of being detected.

- **Malicious adversary:** Instead of just passively trying to infer additional information, malicious adversaries involved in the scheduling process might also modify or drop protocol messages whenever it benefits their goal of finding out private inputs of other parties. These attacks usually change the outcome of the computation and can thus potentially be detected.

- **Non-participating adversary:** Other employees or guests in the corporate network are capable of eavesdropping on an ongoing scheduling session even if they are not participating. Although these adversaries do not hold any protocol state, they might read and tamper with exchanged messages in order to gain insights or actually impersonate legitimate users.

Throughout this work, a semi-honest setting is considered sufficient because of the following reasons:
• It is reasonable to assume that companies have software restriction policies in place. Technical measures to enforce such policies prevent malicious adversaries from executing code that deviates from the correct protocol implementation.

• Employees at the same company enjoy each other’s trust to a certain extent. Since there is a mutual interest in executing the protocol correctly and ultimately getting a job done, potentially destructive attacks are implausible. In contrast, trying to obtain information on other employees from existing protocol data out of curiosity seems to be much more likely since the corporate workflow remains unaffected.

• There is a chance that active attempts to deviate from the protocol are detected, impacting the working atmosphere and resulting in disciplining measures in case that the adversary can be identified. Partial knowledge of another employee’s calendar is probably not considered valuable enough to justify this risk.

Non-participating adversaries within the network, on the other hand, can be easily excluded from all protocol-related communication by means of classical cryptography. Mechanisms to create secure channels for data confidentiality, integrity and authentication are well understood and standardized, most prominently as Transport Layer Security protocol (TLS) [Res01; Die08]. The task of providing a secure communication layer is orthogonal to the goal of designing a privacy-preserving calendar scheduling system and will therefore not be focused on in this work.

3.2 Scheduling Schemes

In this application, the purpose of the secure computation is to choose an appropriate time slot for an appointment based on free and busy time of every participant. Mathematically speaking, every party \( i \in \{1, \ldots, n\} \) provides input bits \( b_1^i \ldots b_m^i \), where \( m \) is the number of time slot candidates. A time slot can be any time interval, reaching from half-hours to full days, dependent on what was agreed on during the invitation phase. For a certain time slot \( j \), input bit \( b_j^i = 0 \) indicates that party \( i \) is busy during this time slot, whereas \( b_j^i = 1 \) means that the time interval is available for the appointment in question. Based on this notation, the following scheduling schemes can be defined that need to be implemented as a secure computation:

• **First match:** The scheduling returns the first time slot \( j \) where every party is available, i.e. the smallest \( j \) for which \( b_j^i = 1 \) holds for all \( i \), if
• **Best match:** Instead of requiring that all participants attend the meeting, it might be desirable to find a date where as many as possible out of the $n$ parties are available. That is, compute a time slot $j$ such that $(\sum_{i=1}^{n} b^j_i) \geq (\sum_{i=1}^{n} b^k_i)$ for every $k \neq j$.

• **Weighted best match:** In reality, it is often the case that some personnel is more indispensable than others for certain meetings. For example, the project manager’s presence is usually vital for appointments to succeed at all. Thus, there is need for a weighted scheduling scheme which takes a participant’s identity into account for prioritization. Formally, weighted best match finds a time slot $j$ that satisfies $(\sum_{i=1}^{n} p_i b^j_i) \geq (\sum_{i=1}^{n} p_i b^k_i)$ for every $k \neq j$, with $p_1, \ldots, p_n \in \mathbb{N}$ being fixed priorities associated with each party. The higher a party’s coefficient $p_i$, the more important this participant is rated in the computation.

It is likely that the need arises for additional scheduling schemes that proof beneficial in special corporate environments not considered yet.

## 3.3 Requirements

The requirements for a secure scheduling application can be refined with respect to the goals presented in section 1.2 and the use case illustrated in this chapter:

• **Privacy:** The application should rely on existing protocols for secure $n$-party computation. With a suitable encoding of a user’s calendar as inputs, privacy is preserved both against semi-honest adversaries and non-participating attackers within the network.

• **Scalability:** It is necessary that the scheduling process, independent of the selected scheduling scheme, completes in reasonable time even for a large number of participants. Organizing meetings for up to 30 attendees should be supported. Since available MPC protocols do not scale favorably for high numbers of parties, optimizations must be considered in terms of protocol improvements as well as for reducing the complexity of the underlying circuits.

• **Workflow integration:** At virtually every company, common calendar applications such as Outlook are used to keep track of personal
timetables. Adherence to standard calendar protocols is necessary to allow easy transfer of information on free and busy time slots from existing applications to the scheduling protocol, as well as for saving the final appointment date after a successful computation. For the user's convenience, the workflow should require as few manual actions as possible.

- **Flexibility:** Different scheduling schemes need to be implemented so that the scheduling functionality is applicable in diverse settings at various companies. While the schemes listed in section 3.2 serve as presets, improving the application's usefulness with additional scheduling schemes should be a straightforward task using a well-defined interface.

- **Extensibility:** Although the reference use case for this work restricts communication to a local network, appropriate abstractions are required to integrate new communication technologies in the future, particularly in regard to usage over the Internet. Therefore, it is desirable to abstract the scheduling protocol from the communication layer that provides basic services for exchanging messages between the parties. Similar considerations need to be made when designing the other components of the software, so that the implemented mechanisms can be transferred to new application areas without problems.

### 3.3.1 User Interface

In order to provide a convenient workflow, a graphical user interface needs to be designed which makes the scheduling process interactive and intuitive. The following tasks must be easily achievable by the user:

- **Initiate scheduling:** An input mask should enable the user to enter a textual description for the new appointment and select time slots candidates as well as the desired scheduling scheme. When broadcasted, other users within the network should be notified of the pending invitation and get the opportunity to accept or reject it. The initiator, on the other hand, must be provided with tools to exclude unapproved contenders during the invitation phase.

- **Select free and busy time slots:** When taking part in the scheduling process, users need to review and modify their personal timetable by means of a clear graphical overview. They must be offered the option of importing an iCalendar file to automatically mark time slots as busy when they overlap with existing events.
3.4 Framework Choice

Finally, the choice of a suitable MPC implementation is of central importance for the application and determines whether the requirements can be fulfilled. SCAPI appears to be the most practical choice for a number of reasons:

- The library implements the GMW protocol and thus provides security in the semi-honest setting. Mechanisms against malicious adversaries included in other protocols require additional computational resources, contradicting the scalability requirement.

- Boolean circuits promise to be a more flexible representation than arithmetic circuits for modeling different scheduling schemes. Whereas the \textit{first match} scheme can be defined in a natural way using only bitwise operations, the other schemes require a certain amount of arithmetic operations. However, the necessary counting is limited to small integers of fixed size and can thus be easily emulated based on binary gates. The rationale of preferring binary over arithmetic circuits is discussed in [Cho+12], which also lists binary circuit layouts for common building blocks such as arithmetic operations.

- SCAPI is well-documented, well-structured and can be easily integrated in Java code. These properties facilitate fast prototyping and ultimately the development of an accessible, platform-independent MPC application.

A minor drawback of SCAPI is the lack of a high-level language that compiles into the underlying circuit, as provided by other MPC frameworks. Crafting the required boolean circuits by hand allows for manual optimizations with respect to the overall number of gates, though, potentially increasing the computation’s performance.

Another conceivable option would be to use specialized protocols for private set intersection to avoid circuits as an intermediate representation entirely. The \textit{first match} scheme can then be implemented in a straightforward manner: Intersecting the free time slots of all participants directly yields the desired result. However, other scheduling schemes cannot be modeled only
based on set intersection. Hence, this approach proves to be too inflexible with respect to the requirements.

Although this work is planned to rely on SCAPI, the possibility of exchanging the MPC framework in the future must be kept in mind during the application’s design process. A clean interface encapsulating the secure computation is necessary to adapt to changing requirements later on, for example to allow usage in settings with malicious adversaries.

3.5 Summary

In this chapter, the use case was fixed to a corporate environment and the emerging threats to privacy were identified. On this basis, the requirements for a secure scheduling application were formulated and a framework choice was made.

Simplifying, yet realistic assumptions were made in terms of a corporate setting in order to retain the complexity of this work. Most importantly, the existence of a local network is presumed in which hosts can directly communicate with each other. In addition, this network defines a unified broadcasting domain. Different types of adversaries who intend to uncover private information can be distinguished in this context: Whereas semi-honest attackers correctly follow the scheduling protocol and only analyze rightfully received messages, malicious attackers can interfere with the protocol in any conceivable way to achieve their goal. The third type of threats, non-participating eavesdroppers, can be easily coped with using classical encryption mechanisms. Malicious attacks are expected to be prevented by external measures such as software restriction policies, leaving semi-honest adversaries as the main threat.

Three different algorithms were formalized for finding an appropriate time slot based on the users’ free and busy times. The first match scheduling scheme returns the first time slot that every party marked as available, whereas the best match scheme determines the time slot with the highest number of votes. The weighted best match scheme functions similar to the best match scheme, but the votes of each party are prioritized depending on their role.

Incorporating the preceding considerations, it was possible to specify several requirements. Besides the essential prerequisite that privacy of the users’ personal timetables is maintained, the scheduling application is required to scale favorably even with a large number of participants. Furthermore, usability is important in that user interaction is as intuitive as the workflow provided by existing appointment scheduling software. Finally, the imple-
mented system needs to be flexible enough to support the addition of new scheduling schemes as well as to provide clean interfaces for exchanging the underlying MPC framework or network technology if necessary.

On the basis of these requirements, SCAPI was chosen as the central framework for secure computations since its GMW implementation guarantees privacy in a semi-honest setting. The representation of functions as boolean circuits is adequate for modeling different scheduling schemes and promises efficient evaluation. In addition, SCAPI is well-documented and offers an accessible Java API.
Chapter 4

Conceptual Approach

In the previous chapter, an overview of the required actions from a user’s perspective was given. This chapter, on the other hand, describes a more technical view. Besides explaining which general steps are necessary to initiate and successfully carry out secure scheduling, the following sections focus on formalizing how to construct scheduling circuits for secure evaluation.

For the benefit of a clearer understanding, the overall scheduling process can be subdivided into two distinct phases, the invitation phase and the computation phase, which fulfill different objectives. The invitation phase can be seen as prior setup and its sole purpose is to provide a meaningful context for the subsequent computation. First of all, the initiator broadcasts an invitation to the pending appointment within the network, as illustrated in figure 4.1a. The following information is included:

- **Textual description**: A human-readable description of the appointment in question informs the receivers about the general subject. On this basis, interested parties can decide whether or not to accept the invitation.

- **Session identifier**: Since multiple scheduling attempts might be ongoing at the same time, a unique session identifier needs to be assigned in order to correctly match exchanged messages with the corresponding appointment.

- **Initiator address**: The invitation contains information on how to reach the initiator over the network. Thus, recipients are able to establish a connection in case they wish to accept the request.

- **Time slot candidates**: Invitations include a list of time slot candidates as selected in advance by the initiator. More precisely, a mapping
is provided that associates each time slot index 1, \ldots, m with a time interval bounded by start and end date. Since the actual scheduling computation exclusively works with discrete time slot indices, this mapping is required to give inputs and outputs a meaningful interpretation in the real-world time domain.

- **Scheduling scheme identifier:** Lastly, a unique identifier of the used scheduling scheme is distributed to potential participants as part of the invitation. This also comprises fixed parameters not dependent on the parties’ inputs, such as the weight coefficients used in the *weighted best match* scheme for prioritization.

Recipients who are willing to participate in the scheduled appointment establish a connection with the initiator, depicted in figure 4.1b. The initiator serves as central hub and notifies the other parties of any topology changes. As a result, each party eventually learns how to reach every other party over the network. Based on this information, a pairwise connected peer-to-peer layer is established in preparation for the ensuing computation phase as soon as all participants signal their readiness. The resulting topology is shown in figure 4.1c. Note that, once this final configuration is set up, the initiator no longer plays a special role anymore and is instead treated exactly like any other peer during the remaining protocol steps. Furthermore, the set of participating parties is considered immutable from this point on, preempting later connection attempts.

![Diagram](attachment:image.png)

**Figure 4.1:** Connection patterns over the course of a scheduling session. An invitation is sent to all hosts within the network (a), gradually evolving into a star topology as a result of interested participants connecting to the initiator (b). Performing the final scheduling computation requires to establish direct links between each pair of hosts (c).

During the computation phase, the actual scheduling is performed in a secure manner. From a high-level view, this phase interacts with the user
by means of iCalendar files as a common description of events, schematically depicted in figure 4.2. Each participant locally selects a calendar file that contains free and busy times, usually exported from a common calendar application. By matching the listed events with the time slot candidates shared during the invitation phase, a binary value can be derived for each of the \( m \) slots indicating whether or not it is occupied. These \( m \)-bit words are used as inputs for the following MPC execution. Based on the number of participants \( n \), the number of time slots \( m \) and information on which scheduling scheme should be used, a boolean circuit is constructed that represents the scheduling function. Evaluating this circuit using the MPC framework of choice, every party eventually holds a binary description of the scheduled time slot. Again relying on the public mapping from time slot indices to time intervals, a new iCalendar file is created which contains a meaningful description of the final appointment and can be imported back into external calendar applications. Note that the underlying MPC protocol is invoked as a black box and no assumptions are made other than that it must operate on binary instead of arithmetic circuits. Thus, it is straightforward to exchange the semi-honest GMW implementation of SCAPI for a protocol with higher security guarantees later on.

Figure 4.2: Data flow during the computation phase from a high-level perspective. iCalendar files serve as a direct interface towards the user, whereas the underlying secure computation operates on arrays of bits representing discrete time slot information.

A point that is worth emphasizing is that, while iCalendar files are integrated into the described scheduling approach as an exchange format, the CalDAV standard does not find its use. This is a deliberate choice since
CalDAV directly competes with the planned secure scheduling application in terms of its purpose. The concept of CalDAV is to share calendars between several users as a measure to facilitate scheduling, whereas the fundamental principle of this work is to keep the users’ calendars private. Although the joint goal is to provide a working solution for scheduling an appointment, the approaches to accomplish this goal are contradictory with respect to the exposure of personal information.

4.1 Circuit Design

Scheduling is required to be performed in a secure manner, which means that the scheduling algorithm is executed by the underlying MPC framework. It is therefore necessary to construct an equivalent boolean circuit for every scheduling scheme, taking the number of parties $n$ and the number of time slots $m$ into account. In the following, construction rules are presented for circuits computing the scheduling schemes defined in section 3.2.

In the context of secure multi-party computation, it is reasonable to define boolean circuits to be composed exclusively of the binary operators AND and XOR as well as the unary NOT gate, denoted by $\cdot$, $\oplus$ and $\neg$, respectively. In addition, the symbolic constants 0 and 1 may be used instead of variable wire values. Note that including these constants does not increase expressiveness but only allows for a more concise notation. In accordance with this definition, higher-level building blocks can be constructed to facilitate the translation of scheduling schemes into boolean circuits.

4.1.1 Basic Building Blocks

Integer arithmetic plays an important role in counter-based scheduling schemes. Let $X = x_1 \ldots x_\ell$ and $Y = y_1 \ldots y_\ell$ describe $\ell$-bit words, each interpreted as an unsigned integer in the range between 0 and $2^\ell - 1$, inclusively. Note that the bits are listed in order of increasing significance, in contrast to the prevalent conception, for the benefit of a more comprehensible circuit notation. Based on the work by Kolesnikov et al. [KSS09], summarized in [Cho+12], different integer operations can be modeled by boolean circuits in the following way:

- **Addition:** This function adds the numeric values of two $\ell$-bit integers $X$ and $Y$ and returns the result $Add(X,Y) = z_1 \ldots z_\ell$ as another $\ell$-bit integer, ignoring the potential carryover of the most significant bit. Computing the carryover bits $c_1 \ldots c_\ell$ as an intermediate step, the
corresponding circuit can be constructed as follows:

\[ c_i = \begin{cases} 
0 & \text{if } i = 1, \\
{c_{i-1}} \oplus ((x_{i-1} \oplus c_{i-1}) \cdot (y_{i-1} \oplus c_{i-1})) & \text{otherwise}
\end{cases} \]

\[ z_i = x_i \oplus y_i \oplus c_i \]

- **Comparison:** The comparator \( \text{GreaterThan}(X, Y) \) returns 1 if and only if the numeric value of \( X \) is strictly greater than the value of \( Y \). Again, a number of intermediate bits \( c_1 \ldots c_{\ell+1} \) are assigned, where \( c_i = 1 \) indicates that the \( \text{GreaterThan} \) relation holds true when only the \( i - 1 \) least significant bits of both operands are considered. For convenience, additional comparison functions can be derived from the definition of \( \text{GreaterThan} \).

\[ c_i = \begin{cases} 
0 & \text{if } i = 1, \\
x_{i-1} \oplus ((x_{i-1} \oplus c_{i-1}) \cdot (y_{i-1} \oplus c_{i-1})) & \text{otherwise}
\end{cases} \]

\[ \text{GreaterThan}(X, Y) = c_{\ell+1} \]
\[ \text{GreaterOrEqual}(X, Y) = \neg \text{GreaterThan}(Y, X) \]
\[ \text{LessThan}(X, Y) = \text{GreaterThan}(Y, X) \]
\[ \text{LessOrEqual}(X, Y) = \neg \text{GreaterThan}(X, Y) \]

- **Shifting:** The operation \( \text{Shift}_k(x_1 \ldots x_{\ell}) = 0 \ldots 0 x_1 \ldots x_{\ell-k} \) shifts all bits by \( k \) positions towards higher significance. As long as no set bits are truncated, this operation is equivalent to multiplying the numerical value of \( X \) by the factor \( 2^k \). Note that, in contrast to other listed primitives, shifting does not add any gates to the circuit and can instead be considered a *rewiring* of existing gates.

- **Constant multiplication:** Based on the addition and shifting rules described previously, it is possible to define the function \( \text{Multiply}_k(X) \), which computes the product of \( X \) and the natural number \( k \) as another \( \ell \)-bit integer. It is assumed that \( \ell \) is chosen sufficiently large to guarantee that no overflow occurs. Furthermore, it is worth to emphasize that \( k \) is a constant which is specified when constructing the circuit, and thus can not depend on variable values during circuit evaluation. Let \( k_0, k_1, \ldots \) denote the bits of the binary representation of \( k \) such that \( k = \sum_{i \in \mathbb{N}_0} k_i 2^i \) holds. The product can then be defined as follows, where the sum refers to the *Add* operation:

\[ \text{Multiply}_k(X) = \sum_{i \in \mathbb{N}_0, k_i = 1} \text{Shift}_i(X) \]
Since shifting by \( i \) corresponds to a multiplication by \( 2^i \) and the sum only considers the finite set of indices \( i \) for which \( k_i = 1 \) hold, this equation effectively computes \( \sum_{i \in \mathbb{N}_0} k_i 2^i X = kX \).

Besides integer arithmetic, there exist several utility functions which prove beneficial when designing circuits for scheduling schemes. The following operations can be applied to arbitrary \( \ell \)-bit words, independent of their interpretation:

- **Multiplexing**: Since the concept of dynamic branching does not exist in boolean circuits, it is often necessary to select one out of two inputs \( X \) and \( Y \) depending on a selection bit \( s \). More precisely, the function \( \text{Multiplex}(X, Y, s) \) returns \( X \) in case that \( s = 0 \) holds, and \( Y \) otherwise. The individual bits of the result \( z_1 \ldots z_\ell \) can be computed as follows:

\[
z_i = x_i \oplus (s \cdot (x_i \oplus y_i))
\]

- **Filtering**: With regard to the disclosure of information, it might sometimes be desirable to mask certain bits of a result. For this purpose, the function \( \text{FilterFirst} \) clears all bits of the input, except the first set bit, if it exists. Formally, the result \( \text{FilterFirst}(x_1 \ldots x_\ell) = z_1 \ldots z_\ell \) satisfies the following identity:

\[
z_i = \begin{cases} 1 & \text{if } x_i = 1 \text{ and } x_j = 0 \text{ for all } j < i, \\ 0 & \text{otherwise} \end{cases}
\]

This function can be transformed into a boolean circuit as follows, computing intermediate bits \( c_1 \ldots c_\ell \) in a way that \( c_i = 1 \) if and only if \( x_j = 0 \) for all \( j < i \):

\[
c_i = \begin{cases} 1 & \text{if } i = 1, \\ \neg x_{i-1} \cdot c_{i-1} & \text{otherwise} \end{cases}
\]

\[
z_i = x_i \cdot c_i
\]

- **Bitwise operations**: In practice, it is often required to apply operations on a number of bits in parallel. Thus, it is helpful to generalize the basic gates AND, XOR and NOT for bit-words of arbitrary length:

\[
\text{And}(x_1 \ldots x_\ell, y_1 \ldots y_\ell) = x_1 \cdot y_1 \ldots x_\ell \cdot y_\ell \\
\text{Xor}(x_1 \ldots x_\ell, y_1 \ldots y_\ell) = x_1 \oplus y_1 \ldots x_\ell \oplus y_\ell \\
\text{Complement}(x_1 \ldots x_\ell) = \neg x_1 \ldots \neg x_\ell
\]
The aforementioned building blocks serve as a basis for constructing complex circuits. Finally, it needs to be discussed how operations can be combined efficiently. A particularly common task is the aggregation of multiple values. For example, one might be interested in the sum of given integers $X_1, \ldots, X_k$, which can be achieved by computing the $Add$ function $k - 1$ times. Since integer addition is associative and commutative, the result does not depend on the order in which the $Add$ operations are evaluated. However, the depth of the constructed circuit greatly varies, as shown in figure 4.3. By arranging the operations according to a tree layout, the depth complexity can be reduced to $O(\log k)$, compared to $O(k)$ in the sequential case. Circuit depth directly affects the round-complexity of most MPC protocols and therefore the overall running time, which makes its minimization a highly desirable goal [Bue+16].

![Figure 4.3: Summation of integers with a sequential (left) and tree-like (right) arrangement of Add operations. The latter results in a much lower circuit depth.](image)

The preceding considerations can be formalized to allow a variable number of arguments for any associative, binary function $f$. The resulting function $F(X_1, \ldots, X_k)$ can be derived from $f(X, Y)$ according to the following, recursive definition:

$$F(X) = X$$

$$F(X_1, \ldots, X_k) = \begin{cases} 
F(f(X_1, X_2), \ldots, f(X_{k-1}, X_k)) & \text{if } k \text{ is even}, \\
F(f(X_1, X_2), \ldots, f(X_{k-2}, X_{k-1}), X_k) & \text{otherwise}
\end{cases}$$

Every recursion step translates into one layer of the corresponding operation tree, where each node represents a single evaluation of $f$. Pairs of consecutive arguments are aggregated into single values, which in turn form the inputs to the next layer. Eventually, the overall result can be obtained as the output of the final layer.

When constructing higher-level circuits during the course of this chapter, the function $Add$ may be used with more than two arguments to compute the sum of multiple integers, following the above aggregation rules. Analogously, a generalized version of the bitwise $And$ operation can be constructed.
4.1.2 Scheduling Circuits

The previously defined building blocks can be combined to form boolean circuits for each scheduling function. As formalized in section 3.2, each of the \( n \) parties provides input bits \( b_i^1 \ldots b_i^m \), where a bit \( b_i^j \) of party \( i \) is set if and only if the time slot with index \( j \) is considered free. A scheduling circuit \( S \) processes these bits and returns the time slot \( j \) which fits best according to the implemented scheduling scheme, or a symbolic value indicating that no matching time slot was found. Since encoding the index \( j \) as a binary integer would incur additional circuit complexity, it makes sense to restrict the scheduling computation to adhere to the following mapping:

\[
(b_1^1 \ldots b_1^m, \ldots, b_n^1 \ldots b_n^m) \mapsto z_1 \ldots z_m
\]

The output must be formed in such a way that at most one bit is set. \( z_j = 1 \) indicates that the time slot \( j \) has been successfully scheduled, whereas an all-zero result stands for an unsuccessful scheduling attempt. Scheduling circuits should be constructed in a way that other outcomes are logically impossible. However, in practice it is still advised to verify the well-formedness of the result to detect incorrect circuit evaluation, which might point towards a flawed MPC implementation or active attacks.

In the following, it is shown how boolean circuits can be constructed for different scheduling schemes:

- **First match**: This scheduling scheme requires that every party is available for a time slot to be selected, which means that \( b_i^j = 1 \) for every party \( i \). Finding all matching time slot candidates can be directly achieved via a bitwise \( \text{And} \) operation. Since only the first matching time slot should be returned, a supplementary filtering step is necessary to give the desired result:

\[
\text{FirstMatch}(b_1^1 \ldots b_1^m, \ldots, b_n^1 \ldots b_n^m) = \text{FilterFirst}(\text{And}(b_1^1 \ldots b_1^m, \ldots, b_n^1 \ldots b_n^m))
\]

To obtain a depth-efficient circuit, the bitwise \( \text{And} \) operation uses the tree-like aggregate construction as presented in the previous section.

- **Weighted best match**: Given fixed priorities \( p_i \) for each party \( i \), scores can be assigned to every time slot. Party \( i \) contributes to the score of slot \( j \) with its assigned priority if and only if \( b_i^j = 1 \). The goal then is to find the time slot with the highest score. When translating this procedure into a boolean circuit, the first step is to accumulate
the scores $S_1, \ldots, S_m$ for each time slot, represented by $\ell$-bit integers. $\ell$ is chosen in a way that the maximum possible score $\sum_{i=1}^{n} p_i$ does not exceed the upper representable limit $2^\ell - 1$.

$$S_j = \sum_{i=1}^{n} \text{Multiply}_p(B_i^j)$$

The sum again denotes the aggregated $Add$ operation and $B_i^j$ is the $\ell$-bit integer representation of the numeric value of $b_i^j$, obtained by copying $b_i^j$ to the least significant bit and zero-filling all remaining bits. This construction is equivalent to computing $B_i^j = \text{Multiplex}(00\ldots0, 10\ldots0, b_i^j)$.

It is now necessary to find the time slot $j$ with highest score $S_j$. To this effect, it is necessary to incrementally assign the auxiliary bits $c_1, \ldots, c_m$ and counters $S_{j,\max}^1, \ldots, S_{j,\max}^m$. A set bit $c_j$ indicates that the score of time slot $j$ is equal to or higher than the scores of all subsequent slots $k > j$, whereas $S_{j,\max}^j$ keeps track of the highest score out of all those time slots succeeding $j$. The following circuit construction models this behavior:

$$S_{j,\max}^j = \begin{cases} 0\ldots0 & \text{if } j = m, \\ \text{Multiplex}(S_{j+1,\max}^j, S_{j+1}^j, c_{j+1}) & \text{otherwise} \end{cases}$$

$$c_j = \text{GreaterOrEqual}(S_j, S_{j,\max}^j)$$

The purpose of the $\text{Multiplex}$ operation is to increase the counter $S_{j,\max}^j$ only in case that a higher score was found, ensuring a monotonically nondecreasing behavior for declining index $j$. The reason for starting recursion at the highest index $j = m$ instead of $j = 1$ is that a simple filtering operation now suffices to obtain the time slot with highest score and thus the final result:

$$\text{BestMatch}_{p_1,\ldots,p_n}(b_1^1, \ldots, b_m^1, \ldots, b_1^n, \ldots, b_m^n) = \text{FilterFirst}(c_1, \ldots, c_m)$$

Selecting the first set bit $c_j$ ensures that no time slot with index $k < j$ has a higher score than $j$, otherwise the corresponding bit $c_k$ would be set as well. Furthermore, directly following from the definition of $c_j$, there cannot be an index $k > j$ with higher score, leaving only $j$ as the correct result.

- **Best match**: The best match function can be trivially constructed from the weighted best match scheme, namely by assigning the priority $p_i = 1$ to all parties. The score of each slot $j$ then exactly corresponds to the number of participants available at this time.
Figure 4.4 shows small sample circuits that were constructed according to the above rules. For simplicity, it is assumed in the weighted best match case that 3-bit integers suffice to represent the time slot scores $S_j$, which means that the sum of all priorities $p_1, \ldots, p_4$ does not exceed $2^3 - 1$.

An alternative, conceivable approach would be to generate scheduling circuits without the final filtering step and allow more than one set bit in the result. Selecting the first set bit would instead be performed locally after successfully evaluating the circuit, reducing the complexity of the secure computation. However, this procedure contrasts strongly with the goal of privacy, since the output potentially conveys additional information. By allowing only a single bit to be set, participants cannot draw any conclusions about free and busy time slots of other parties except for the one particular slot selected by the scheduling algorithm, which necessarily needs to be revealed.

4.2 Summary

In the preceding sections, a general protocol for initiating and carrying out a secure scheduling computation was presented. It is split into two distinct phases, starting with the invitation phase that allows potential participants to gather in a session dedicated to the appointment in question. During session setup, the parties agree on the scheduling scheme as well as the mapping from abstract time slot indices to real-world time intervals. Thereby, the secure computation is embedded in a meaningful context. As soon as the set of participating parties is fixed, the following computation phase makes use of an MPC framework to securely evaluate the scheduling function. Conceptually, users interact with the system via iCalendar files as a common calendar representation. The private input bits indicating whether a certain time slot is free or occupied can be extracted from imported iCalendar files. Analogously, the computation result is transformed back into the iCalendar format so that it can be further processed. Whereas communication during the invitation phase is centralized in the sense that the initiator serves as a central hub, the computation phase is built on top of a pure peer-to-peer topology with a separate communication channel between each pair of hosts.

For the purpose of securely computing the scheduling function, an equivalent scheduling circuit is constructed ad hoc, dependent on the number of parties $n$ and time slots $m$. Section 4.1 covered how the different scheduling circuits can be formally defined in terms of boolean logic. In order to facilitate the design, construction rules for various auxiliary building blocks were listed, including basic integer and bitwise operations as well as specialized
functionality such as filtering. Finally, to complete the toolbox, aggregation rules were derived that allow to combine operations in a depth-efficient manner.
Figure 4.4: Sample circuits modeling different scheduling schemes for $n = 4$ parties and $m = 3$ time slots. Each box stands for a basic building block and each wire for an individual bit.
Chapter 5

Implementation

This work is accompanied by the implementation of a fully functional prototype of a private scheduling application, developed in Java. This language choice allows simple deployment to different operating systems and fast prototyping since the language encourages clear and robust code. In addition to an extensive standard library that already comes with network communication and file access, there also exist many third-party solutions written in Java. In particular, designing an interactive GUI is a well-covered task. Since SCAPI includes a Java wrapper for the GMW protocol, using this language during the course of this work is the natural choice.

![Figure 5.1](image-url)
From a high-level perspective, different components can be identified as parts of the application’s overall architecture, as shown in figure 5.1. The following list gives a brief overview of their individual responsibilities and relations.

- **Graphical user interface**: The GUI is the part of the application that the user directly interacts with. Its purpose is to accept input from the user and present computation results on the screen. In particular, it allows to initiate and join scheduling sessions as well as keeping track of active sessions in a straightforward manner.

- **Network layer**: This layer encapsulates the communication between the local machine and a number of remote hosts. It supports sending and receiving messages between the designated parties as well as broadcasting information in a specified broadcast domain.

- **Invitation management**: Whenever the user intends to initiate the scheduling process, information on the appointment is passed from the GUI to this component. The invitation management then relies on the network layer to broadcast the event and keeps track of connected participants. As soon as every designated receiver accepts the notification presented by the GUI, the negotiated invitation data is forwarded to the computation phase, which relies on the circuit generator and the MPC black box components.

- **iCalendar converter**: After a user selected an iCalendar file in the GUI, the task of this component is to extract a binary array representing free and busy time slots from the included list of events. The necessary description of time slot intervals is obtained as part of the invitation data. In addition, the converter’s responsibility is to transform the binary result of the finished computation back into a meaningful iCalendar file that other calendar software can process.

- **Circuit generator**: The circuit generator takes the number of parties \(n\), the number of time slots \(m\) and an identifier of the desired scheduling scheme as arguments and outputs a boolean circuit that models the correct scheduling behavior. This components comprises a system for defining higher level building blocks from basic logic gates as discussed in the previous chapter, simplifying the generation of complex scheduling circuits.

- **MPC black box**: Based on the boolean circuit generated in advance and the binary time slot description derived by the iCalendar con-
In order to allow exchanging the MPC framework later on, it is necessary to decouple the circuit generation process from the way in which circuits are stored. For example, SCAPI requires boolean circuits to be present in a text file, adhering to a prescribed file format. However, writing these specific circuit files directly for each scheduling scheme is not optimal since it is likely that other MPC frameworks define their own deviating file format, or even rely on in-memory representations that bypass persistent storage altogether.

To address this problem, a distinction is made between the two important concepts *circuit builder* and *circuit generator*. Their corresponding programming interfaces are shown in listing 5.1. Concrete circuit builder implementations are tightly coupled to individual MPC frameworks and specify how the basic gates AND, XOR and NOT are stored in the desired circuit format. Furthermore, wires can be marked as inputs or outputs of certain parties. The sole purpose of circuit generators, on the other hand, is to model algorithms as boolean circuits that are constructed using the given circuit builder instance. Although the words *circuit builder* and *circuit generator* might have a similar meaning in natural language, the *builder* terminology strictly refers to the software design pattern as formalized by the Gang of Four [Gam+95]. As the authors state, this pattern is applicable when “the algorithm for creating a complex object should be independent of the parts that make up the object and how they’re assembled” and “the construction process must allow different representations for the object that’s constructed”, which exactly matches the described use case.
Chapter 5. Implementation

abstract class CircuitBuilder {
    Wire and(Wire leftWire, Wire rightWire) { ... }
    Wire xor(Wire leftWire, Wire rightWire) { ... }
    Wire not(Wire wire) { ... }
    Wire input(int partyId) { ... }
    void output(Wire wire, int partyId) { ... }
}

interface CircuitGenerator {
    void generate(CircuitBuilder builder);
}

Listing 5.1: Circuit builder and circuit generator API as defined in the prototype’s Java code.

On this basis, it is already possible to define simple circuits, as the example in listing 5.2 illustrates. Given two collaborating parties providing private input bits $a$ and $b$, the circuit computes the value $a \cdot b$ and outputs it to both participants.

class SimpleGenerator implements CircuitGenerator {
    void generate(CircuitBuilder builder) {
        Wire inputA = builder.input(0);
        Wire inputB = builder.input(1);
        Wire result = builder.and(inputA, inputB);
        builder.output(result, 0);
        builder.output(result, 1);
    }
}

Listing 5.2: Example of a simple circuit generator that performs a single AND operation on two input bits.

It is important to note that this method does not actually perform any computations, although its structure resembles a sequence of executable commands. Calling generate can instead be considered a recording step. Instead of evaluating the listed gates directly, the given circuit builder adds them to its internal circuit representation. As soon as the method is fully traversed, the recording process finishes and builder holds a complete description of the generated circuit. This behavior is also reflected by the signatures of the circuit builder operations: input returns a circuit wire that is later on used to hold a single input bit of the specified party, instead of directly requesting private input from the user. Analogously, and and xor
5.1. Circuit Generation

determine the wire that eventually represents the operation’s result, given the wires which would hold the corresponding inputs.

The indirection has some implications when writing generator code. Dynamic branching, for example, is not carried over to the generated circuit. Instead, the condition is evaluated once when performing the actual recording, hard-wiring the respective branch into the circuit. Similarly, counting loops are unrolled during this step, generating a fixed number of gates. On the other hand, when keeping these potential pitfalls in mind, this API offers a natural and expressive way to design circuits directly in Java code, independent of the internal circuit representation used by the MPC framework.

5.1.1 Static Optimization

The operations AND, XOR and NOT together are functional complete, which means that any conceivable function can be modeled as a boolean circuit composed of no more than these gate types. Nevertheless, it is often convenient to supplementally use the constants 0 and 1 in logical expressions, as already became apparent when designing scheduling circuits in section 4.1. For instance, it is natural to model the increment $X + 1$ of an integer $X$ by the term $Add(X, 10 \ldots 0)$. A building block for general integer addition already exists, so it would be highly redundant to assemble a separate circuit solely for this special case. MPC frameworks do not necessarily provide a built-in mechanism for constant wires, which is also not available in SCAPI. Although there are workarounds addressing this limitation, such as requiring a party to pass in additional input bits with guaranteed constant value, the approach taken in this work is to apply a static preprocessing step which eliminates potential constants. As a side-effect, the total number of generated gates is significantly reduced for certain expressions.

For this purpose, symbolic wires for both constants are introduced, represented programmatically by the immutable static fields Wire.ZERO and Wire.ONE. These symbols only exist during construction time, meaning that the constructed circuit does not comprise the notion of constant wires anymore. As a consequence, constant wires cannot be marked as circuit outputs, which would be of little use anyway.

Let 0 and 1 denote the respective symbolic wires and $w$ an arbitrary wire, either symbolic or conventional. When generating a circuit, the underlying circuit builder statically replaces expressions containing symbolic wires according to the following replacement rules (symmetric cases omitted for
conciseness):

\[
\begin{align*}
    w \text{ AND } 0 & \rightarrow 0 \\
    w \text{ AND } 1 & \rightarrow w \\
    w \text{ XOR } 0 & \rightarrow w \\
    w \text{ XOR } 1 & \rightarrow \text{NOT } w \\
    \text{NOT } 0 & \rightarrow 1 \\
    \text{NOT } 1 & \rightarrow 0
\end{align*}
\]

In all but one of these cases, the circuit builder immediately returns a reference to the resulting wire without emitting an actual gate. Also note that it is likely for these rules to produce another constant output which causes further shortcuts being taken, potentially triggering a cascading effect. Eventually, as all symbolic wires are eliminated, the circuit builder holds a description of the optimized but functionally equivalent boolean circuit.

The benefit of this static optimization phase is that generalized building blocks can be combined with constants to obtain specialized operations, without paying for unused gates in terms of circuit complexity. Thus, circuit generators can be designed with readability and logical consistency in mind, while still constructing optimal circuits for secure evaluation. For example, it does not matter if the bit length of integers is chosen too conservative during construction, since unused bits filled with zeros are simply optimized away. The practical relevance can also be quantified: A statically optimized circuit for the best match scheduling scheme with parameters \(n = 3\) and \(m = 5\) comprises 127 gates in total (42 AND gates), whereas a non-optimized circuit with explicit shared wires for constant values is made of 144 gates overall (44 AND gates). Circuit construction in this example followed the formalization from section 4.1.2, which regularly makes use of constants in favor of a clearer specification.

### 5.1.2 Building Block Representation

Circuit builders provide the most basic operations for constructing boolean circuits. However, it was shown in section 4.1 that circuit design can be greatly facilitated by identifying common building blocks, which in turn serve as a toolbox for higher-level constructs. A straightforward approach to represent these building blocks in an object-oriented programming language would be to create a single `Toolbox` class which exposes all necessary operations, such as integer addition and comparison, in the form of static methods. Each method would expect a circuit builder and a list of input wires as arguments and return a number of output wires, generating the interconnecting gates.
with help of the provided builder instance. This strategy has two major drawbacks, though:

- **Extensibility:** It is likely that the need for additional building blocks arises later on. Augmenting the Toolbox class with the respective static methods would require access to the source code and thus is not an option for third-party libraries using this API. Instead creating a new static class ExtendedToolbox would bypass this limitation, but ultimately leads to a fragmented codebase that is difficult to maintain.

- **Aggregation:** Section 4.1.1 covered how arbitrary associative binary functions can be transformed into functions accepting a variable number of arguments, presenting rules for efficient circuit construction. For every binary operation, this transformation would need to be expanded explicitly into a second static method with a variable argument list.

The general approach taken during prototype development to address these shortcomings is to represent building blocks as actual objects, deriving from either UnaryBitwiseOperator or BinaryBitwiseOperator as shown in listing 5.3. Each operator instance holds an implicit reference to the circuit builder used for generating gates. This way, custom building blocks can easily be added independent of existing ones by creating a new operator class that overrides the respective apply method. Also note that classes deriving from BinaryBitwiseOperator implicitly inherit the method applyAggregated, which encapsulates the previously described tree-like expansion. As a result, no manual work is necessary anymore to enable aggregation for custom operations.

```java
abstract class UnaryBitwiseOperator {
    abstract Wire[] apply(Wire[] operand);
}

abstract class BinaryBitwiseOperator {
    abstract Wire[] apply(Wire[] leftOperand,
                          Wire[] rightOperand);
    Wire[] applyAggregated(Wire[]... operands) { ... }
}
```

Listing 5.3: Abstract base classes for bitwise operations. The apply method needs to be overridden to implement custom building blocks.

On this basis, the operations formalized in section 4.1.1 can be easily translated into Java code. Relying on these building blocks, it is then
straightforward to implement a circuit generator for each scheduling scheme that takes the parameters $n$ and $m$ into account.

## 5.1.3 Secure Evaluation

Circuit builders were introduced to decouple the generation process of circuits from their internal representation, which is determined by the incorporated MPC framework. This segregation also needs to be sustained when encapsulating the protocol for secure circuit evaluation in order to allow easy interchangeability. For this purpose, the MPC protocol is accessed via an abstract interface, as shown in listing 5.4. As soon as every participating party calls `evaluate`, the actual MPC instance is initiated. It is the network layer's responsibility to synchronize the connected hosts. Note that the computation expects a circuit generator as an argument, describing the algorithm that is about to be evaluated in an abstract way. Passing a circuit builder in this situation would not be an option since the calling code is not aware of the concrete MPC implementation and thus the expected circuit format. Besides a description of the evaluated function, the interface requires an identifier of the impersonated party as well as the private input encoded in a byte array. Eventually, the computed output bits are returned. The input bits are correlated with the input wires specified during circuit generation in a sequential fashion, i.e., the $i$-th input bit is assigned to the wire generated by the $i$-th invocation of `input` for the respective party.

```java
interface SecureComputation {
    byte[] evaluate(CircuitGenerator generator,
                    int partyId,
                    byte[] inputBits);
}
```

**Listing 5.4:** Interface that encapsulates the protocol for secure circuit evaluation. The computed function is represented by the specified circuit generator.

It is of high importance that all participants refer to the same circuit generator for joint evaluation. Otherwise, the MPC protocol is expected to desynchronize and abort as soon as mismatching gates are encountered. In particular, this means that deliberate local circuit modifications disqualify as a measure to alter the computation result without the other parties' agreement.

This abstraction allows to integrate the GMW protocol provided by SCAPI in a way that it is easily exchangeable in the future. As far as the application's prototype is concerned, the class `ScapiSecureComputation`
5.2 Invitation Management and Networking

encapsulates all SCAPI-specific code. From a high-level perspective, its evaluate method is implemented such that it performs the following steps:

1. A custom circuit builder is created which stores circuits according to SCAPI's proprietary file format.

2. The generate method of the given circuit generator is called, passing the previously created builder instance as an argument. Thereby, the constructed circuit gates are transformed into the framework's internal file-based representation.

3. An instance of the GMW protocol is set up via the SCAPI programming interface, pointing to the file handle of the fully generated circuit. Given IP addresses and port numbers of the participating hosts, SCAPI establishes its own peer-to-peer network layer that serves as a basis for executing the actual MPC protocol. After successful evaluation, the output bits are transferred back to the caller.

Analogous to this approach, wrappers for other MPC frameworks that rely on boolean circuit representations can be implemented later on.

5.2 Invitation Management and Networking

The network layer provides mechanisms for broadcasting invitations, joining scheduling sessions and coordinating secure computations. Since the use case is restricted to direct connections within a local network, the socket implementations included in the Java standard library are sufficient for realizing a working prototype. Plain TCP connections serve as a basis for direct peer-to-peer communication, whereas broadcasting functionality is covered by the UDP protocol. Reading and writing objects to network packets or streams is generally accomplished with the help of the Java serialization API [Sch07]. The following sections describe how the different stages of network connections are implemented as part of the prototype application.

5.2.1 Invitation Broadcasting

When the application is started, a UDP socket is set up and bound to the default network interface. As soon as a user initiates a scheduling session, the corresponding invitation data is broadcasted in the local network via this socket. Since the socket is also configured to listen for incoming packets,
invitations of other peers can be recognized and forwarded to the user interface. A 128-bit universally unique identifier [LMS05] is assigned to every newly created scheduling session in order to determine whether an invitation has already been handled by the user and to correlate subsequent connection attempts. For the purpose of establishing a permanent TCP connection, broadcast information also includes the local machine’s IP address and a randomly drawn port number at which the initiator is reachable later on for this specific scheduling session.

For the realization of a working prototype, UDP broadcasts were chosen as the primary method for distributing invitations because they allow for a straightforward implementation. One might picture many more ways to handle this task. For example, the initiator might want to export the invitation as a file and send it via an external channel, such as email. Another option would be to rely on higher level network services to deliver invitations over the Internet. These potential changes were kept in mind when designing the software: Invitation management only relies on an abstract broadcast channel and not the concrete UDP socket implementation. A broadcast channel needs to support the following two operations:

- **Broadcasting invitations**: This function allows to transfer invitation data to all relevant network nodes, depending on the concrete distribution mechanism. As a consequence, the broadcast channel defines its own broadcast domain.

- **Registering callbacks**: Other components might want to get notified as soon as an invitation is received over the network. For instance, the GUI is expected to show a dialog to the user whenever a new appointment is due. Therefore, callbacks can be registered such that incoming invitations are detected and forwarded by the broadcast channel.

Complying with this generalized interface, the software can be augmented with additional ways to broadcast scheduling session invitations over the network.

### 5.2.2 Session Setup

The purpose of the setup phase is to introduce those parties to each other that wish to collaboratively schedule the given appointment. It thereby functions as a preparation for the subsequent secure computation. When a user receives a scheduling invitation, she can either ignore it or indicate her interest by establishing a TCP connection targeted at the initiator’s IP address and port, which is part of the broadcasted information. As a result, the network
topology converges towards the star layout described in chapter 4, with the initiator serving as central hub.

Each participant provides the following party state, which needs to be synchronized between all connected hosts:

- **User name**: A textual identifier that allows to distinguish between hosts when displayed in the user interface. Note that the user name can be freely chosen within the application and therefore does not provide any authentication.

- **Network address**: IP address and ports that are used by the MPC framework to establish a peer-to-peer network layer in order to perform the secure computation.

- **Readiness status**: A boolean flag which indicates whether or not the party is ready to engage in the final scheduling phase.

The party state is made available to every participant in this session by means of a simple network protocol. Let $P_1, \ldots, P_n$ denote the participating peers and $s_1, \ldots, s_n$ the corresponding local party states. $I = P_1$ is the initiator serving as central hub with bidirectional connections to the clients $P_2, \ldots, P_n$. Whenever a client’s state $s_i$ changes locally, it is retransmitted from $P_i$ to $I$. The initiator, in turn, forwards received client state updates as well as changes to the own state $s_1$ to all $n - 1$ connected peers. Thereby, every participant eventually learns the other parties’ states, as exemplarily illustrated by the sequence diagram in figure 5.2.

![Figure 5.2: Exchanged messages for state synchronization with three participants. Each connecting client $P_i$ eventually sends the local state $s_i$ to the initiator, who then distributes a copy of all known party states to every available peer.](image-url)
In addition, the current implementation uses a soft-state approach for the benefit of straightforward error handling. Even if there are no changes to the local state, each client $P_i$ transmits the latest $s_i$ to $I$ in a regular time interval, fixed to 2 seconds in the prototype. This periodic message serves as a heartbeat to indicate that the client is still interested in taking part in the scheduling computation. An erroneous connection is thus easily detected by the initiator, who can exclude the associated peer from the ongoing session as a consequence.

Session setup is considered complete once every party confirmed its readiness, as indicated by the respective flag included in the party state. In this case, the star topology is shut down and the program proceeds to the computation phase. It is now the responsibility of the MPC framework to establish a meshed peer-to-peer network based on the participants’ network addresses gathered during the setup phase. As already mentioned in section 5.1.3, SCAPI interconnects each pair of hosts by an individual TCP-based channel. The implications of separating the two network layers are pointed out in section 5.5, discussing the prototype’s limitations.

### 5.3 Graphical User Interface

Section 3.3.1 listed the requirements concerning the application’s graphical user interface in order to allow a convenient workflow and to hide the technical details of the internal secure computation from the user. The Java installation already comes with the powerful GUI framework JavaFX\(^1\), which is suitable for this purpose and was therefore used during the course of the prototype development.

Most importantly, the application enables its users to easily create their own events and invite others to collaboratively find an appropriate date. As displayed in figure 5.3, the initiation process consists of two steps: First, a general free text description of the appointment needs to be specified, as well as the scheduling scheme that is used in the computation phase. Afterwards, the user can select an arbitrary number of time slot candidates. Since this list is published as part of the invitation, it is advised that the time intervals are chosen independent of the initiator’s personal timetable to avoid revealing private information. Just like any other participant, the initiator gets the option to mark certain slots as unavailable later on.

As soon as the user confirms the entered appointment information, the corresponding invitation is distributed over the network. On the receiving

\(^1\)https://www.java.com/javafx
5.3. Graphical User Interface

(a) General appointment settings  (b) Time slot selection

Figure 5.3: Dialogs which allow to initiate a scheduling session. A textual description of the appointment can be specified (a) as well as a freely chosen number of time slot candidates (b).

side, a notification is displayed that contains the appointment’s description.

Active sessions, either self-initiated or joined by means of a received invitation, can be managed in the application’s main view. An illustrating screenshot is shown in figure 5.4a. The left portion of the user interface lists the user names of the currently connected peers as well as their readiness status. On the right, users can enter their private input by marking each of the given time slot candidates as either free or busy, highlighted in different colors to provide a clear visualization. Furthermore, there exists an option to automatically extract free and busy times from an imported calendar file, as detailed in section 5.4. Each open scheduling session is represented by a separate tab, meaning that multiple appointments can be scheduled at the same time. As mentioned in the preceding network section, the actual scheduling computation begins as soon as all parties explicitly state their readiness by checking the corresponding option in the GUI.

During execution of the MPC protocol, a simple loading screen is displayed to indicate that no user actions can be carried out at that time with respect to the active session. Finally, the computed time slot is presented on screen, as exemplified in figure 5.4b. In addition, an option for exporting the scheduled appointment as an iCalendar file is given, allowing further external processing.
5.4 iCalendar Integration

It was an important goal during prototype development to provide mechanisms for interoperability with existing third-party calendar software. By relying on the iCalendar standard as a common exchange format, the scheduling application can be easily integrated into the users’ workflow.

Two distinct tasks need to be accomplished by the iCalendar component. First, the events from user-supplied calendars need to be extracted and matched with the time slot candidates specified in a scheduling session. Whenever an overlap is detected, the corresponding time slot can be marked as unavailable in the user’s private input. The second task is to convert a successfully scheduled appointment back into an iCalendar file in order to be processable by external calendar programs. Generally, both problems require parsing functionality in terms of the iCalendar file format. For this purpose, there exist ready-to-use solutions such as iCal4j\(^2\) or the more lightweight biweekly\(^3\) library. The decision was made for biweekly since the API is highly intuitive and exactly covers the given use case, not comprising unnecessary features and thus keeping the number of induced dependencies low.

\(^2\)http://ical4j.sourceforge.net/index.html
\(^3\)https://github.com/mangstadt/biweekly
The latter task of transforming an abstract appointment description into a standardized calendar file is straightforward: With the help of biweekly, an iCalendar file with a single event can be easily created. The scheduled time slot comprises a start and end time, which is copied to the event’s corresponding DTSTART and DTEND fields. The SUMMARY and DESCRIPTION properties are populated with the appointment’s textual description, whereas the unique identifier matches the GUID already used during session setup. Listing 5.5 shows an iCalendar definition as exported by the application.

```
BEGIN:VCALENDAR
VERSION:2.0
BEGIN:VEVENT
DTSTAMP:20180228T130043Z
CREATED:20180228T130043Z
LAST-MODIFIED:20180228T130043Z
SUMMARY:PETs4DS Working Group Meeting
DESCRIPTION:A public discussion of new cryptographic techniques that allow to maintain privacy in data science.
DTSTART:20180302T130000Z
DTEND:20180302T140000Z
UID:aa20bc5c-5460-4742-afe6-3d19870559c1
END:VEVENT
END:VCALENDAR
```

**Listing 5.5:** Calendar file generated from a successfully scheduled appointment. The appointment’s start and end time as well as the human-readable description are part of the result.

The following section discusses how to address the remaining task, namely marking time slot candidates as free or busy dependent on an imported iCalendar file.

### 5.4.1 Input Extraction

As described in section 4.1.2, party $i$ needs to supply the scheduling circuit with private input bits $b_1^i \ldots b_n^i$, indicating whether or not the respective time slot is considered free. Although these values can be directly set via the GUI, it is usually more convenient to automatically derive them from entries in the personal calendar. For this purpose, the prototype comprises an option to import a local iCalendar file representing the user’s timetable, which can be easily generated by common third-party calendar software. The goal is to
find exactly those time slot candidates from the ongoing scheduling session that do not conflict with any appointment already planned.

Let $T$ be the set of occupied time intervals, where every interval $[t_1, t_2) \in T$ corresponds to exactly one event included in the imported calendar file. $t_1$ and $t_2$ each denote an instant in time, defined by the event’s start and end time, respectively. Analogously, the session-dependent time slot candidates can be formalized as the time intervals $S_1, \ldots, S_m$. Then, the party’s private input bits $b_1^i \ldots b_m^i$ can be derived as follows:

$$ b_j^i = \begin{cases} 0 & \text{if there exists } T \in T \text{ such that } S_j \cap T \neq \emptyset \\ 1 & \text{otherwise} \end{cases} $$

A value of 1 is assigned to $b_j^i$ if and only if there is no event which overlaps with the corresponding time slot $j$.

Programmatically, checking for a non-empty intersection of two time intervals $S = [s_1, s_2)$ and $T = [t_1, t_2)$ can be done efficiently: $S \cap T \neq \emptyset$ holds true precisely if $s_1 < t_2$ and $s_2 > t_1$. However, intersecting every time slot candidate $S$ with the time interval $T$ of every event in the user’s calendar becomes prohibitively expensive for timetables of practically relevant sizes.

In order to reduce the computational complexity, the prototype’s implementation makes use of a hash table as an intermediate data structure. The hash table is constructed from the time slot candidates $S_1, \ldots, S_m$, hashed by the day the time interval covers. Since it is likely that there are several time slots proposed for a single day, each bucket potentially contains multiple entries. In the unlikely case that a single time slot spans more than one day, this slot is added to all affected buckets. The hash table’s structure after inserting the time slot candidates is illustrated in figure 5.5. As a second step, the algorithm iterates over every time interval $T \in T$ and checks

![Figure 5.5](image)

**Figure 5.5:** Exemplary structure of the constructed hash table, which allows for efficient lookup of time intervals by day.
for intersections only with those time slot candidates that fall into the same
day as $T$, which can be efficiently obtained from the hash table. As soon
as an overlap is found with a time slot $S_j$, a value of 0 is assigned to the
corresponding input bit $b_j^i$.

The described optimization utilizes the fact that a hash table’s lookup and
insertion operations can be performed in $O(1)$ on average [Cor+09, ch. 11,
pp. 253–285]. Thus, the hash table can be constructed in $O(m)$. The subse-
quent intersection test has a runtime complexity of $O(k \cdot |T|)$, where $k$ de-
notes the average number of time slot candidates per day. In practice, only
a handful of appointments can be attended on a single day and therefore, it
is reasonable to consider $k$ bounded by a small constant. As a result, the
implemented iCalendar component computes the user’s private input from a
personal calendar in $O(m + |T|)$, whereas the naive approach would require
$O(m \cdot |T|)$ of time.

Another option for efficiently intersecting the time slot candidates with
the events from the user’s calendar is to use an interval tree instead of a
hash table [Cor+09, ch. 14.3, pp. 348–355]. This specialized data structure
allows to find overlaps between a given interval $S_j$ and a set of intervals $T$
in $O(\log |T|)$. However, it is computationally expensive to initially construct
the search tree from the provided timetable and thus, it must be carefully
assessed whether a more sophisticated algorithm pays off in terms of increased
performance in a practical scenario.

### 5.5 Current Limitations

Although constituting a fully functional scheduling application, the proto-
type developed during the course of this work still has some limitations.
Most importantly, both the setup and the computation phase make use of
plain TCP connections that provide neither confidentiality nor authentica-
tion. This choice kept the prototype simple and allowed to integrate the
SCAPI framework, which inherently relies on TCP sockets, without modifi-
cations. However, the use of unencrypted channels means that noninvolved
nodes in the local network can eavesdrop on the session setup and uncover
the scheduled date. It is important to highlight that the participants’ private
inputs cannot be compromised regardless, since they are protected by means
of the GMW secret sharing scheme. Another problem is that separate TCP
connections are established for the setup phase and the secure computation.
By forging the IP address provided by a legitimate participant during session
setup, an attacker could impersonate this party in the subsequent MPC pro-
tocol execution. In a corporate setting, any dishonest employee can take part
in the scheduling process in another colleague’s place by choosing a misleading user name. In its current state, the prototype provides no mechanism to verify the identity of participating users.

The solution to all these potential attacks is to introduce secure channels based on the TLS standard. Symmetric encryption ensures that eavesdropping is not possible anymore, whereas digital certificates allow to uniquely identify all parties that are connected to a scheduling session. By displaying the subject name included in the certificate in the GUI instead of a freely chosen user name, impersonation attempts are effectively averted. Changes to the SCAPI codebase would be required such that a single TLS-based network graph can be used as a basis for both the session setup and the secure circuit evaluation. As a result, it is guaranteed that network endpoints do not change between the different protocol stages, nullifying redirection attacks that rely on injecting spoofed IP addresses. The increased security comes at the cost of additional configuration effort: Before initial operation in a corporate environment, certificates must be issued to every potential user to maintain a valid chain of trust. Since detailed insights into secure networking exceed the scope of this work and the developed prototype is primarily intended as a simple-to-use proof of concept, the described TLS-based mechanisms are not included in the current implementation and instead left as future work.

An indirect implication of the lack of authentication is that asymmetric scheduling schemes, i.e. those treating users differently based on certain characteristics, have to rely on honesty. This is for example the case in the weighted best match scheme, which allows to assign different priorities to each party. Without reliable methods to proof a user’s identity, it is easy for dishonest participants to masquerade as higher-rated individuals. A TLS-based approach, on the other hand, allows to directly encode priorities into the users’ digital certificates. This way, the values are verifiably linked to the identity of the respective party and cannot be changed retroactively. Due to these restrictions, prioritization is not part of the prototype’s user interface, even though there exist functioning circuit generators for the weighted best match scheme that can be evaluated in a testing environment.

5.6 Summary

This chapter described the steps that were necessary to develop a functioning prototype of a secure scheduling application using the Java programming language. Different software components were identified and their inner workings as well as implementation strategies were examined.
First, it was necessary to design an intuitive API for circuit generation, reflecting the mathematical model formalized in the previous chapter. For this purpose, a distinction was made between circuit builders that encapsulate the concrete circuit representation and circuit generators as a description of the abstract algorithm. As a result, scheduling circuits can be defined independent of SCAP, allowing to freely exchange the MPC framework later on. Furthermore, a static optimization phase was introduced which eliminates constants at construction time, encouraging a clean circuit notation and potentially reducing the circuit size.

Another important component is the network layer, which enables users to interact by sending invitations and connecting to a common session. Plain sockets were used as a basis for implementation. Whereas UDP broadcasts allow to distribute invitations in the local network, TCP connections serve as direct channels between hosts. A simple network protocol is used to synchronize the local party states within a scheduling session, assuming a star-like topology with the initiator as central mediator. In regular intervals, each connected party sends its state to the initiator, who in turn forwards the information to all other peers. Whereas relying on plain sockets allowed a straightforward implementation, it is also the prototype’s primary limitation, as discussed in section 5.5. At its current state, there exist no mechanisms for data confidentiality and authentication during session setup. This problem can be solved by establishing encrypted channels by means of TLS, which needs to be addressed in future work.

The prototype comprises a graphical user interface realized with the help of JavaFX, allowing an intuitive workflow. There exists a dialog for initiating scheduling sessions and notifications are displayed reminding the user of received invitations. When participating in a session, time slots can be manually marked as free or busy by means of a central management panel.

Finally, it was covered how the iCalendar standard can be integrated into the application. For the purpose of parsing iCalendar files, the prototype incorporates available open-source libraries. Building on this, a system was implemented that automatically marks time slots as occupied as soon as there is an event in the user’s personal calendar that overlaps with the respective time interval. It turned out that utilizing a custom data structure based on a hash table resulted in reduced runtime complexity of the accompanying matching step.
Chapter 6

Evaluation

After the successful implementation of the working prototype application, it is necessary to validate that the requirements were met and that it performs as expected. This chapter discusses the measurements of different performance metrics, guided by the evaluation that accompanies the practical work of Choi et al. [Cho+12]. It is particularly interesting to direct one’s attention to how the protocol scales with 20 parties and above. Currently, there seem to exist no real-world measurements of the GMW protocol with values of $n$ in this order, the most promising reference being the aforementioned evaluation with up to 13 participants.

Another aspect of this chapter covers to which extent unquantifiable goals such as privacy and workflow integration were accomplished. An overview of theoretical attacks on the protocol is given as well as an assessment of the relevance of these threats in regard to the users’ privacy. Finally, the prototype is compared to existing non-secure scheduling software in terms of usability and its practical use in a corporate setting is examined.

6.1 Performance Measurements

This work focuses on overall running time and network utilization as the two primary performance metrics, both as a function of the number of parties $n$, the number of time slots $m$ and the selected scheduling scheme. All presented measurements strictly refer to the computation phase of the protocol, i.e., the successful evaluation of the scheduling circuit given the inputs of every party. Session setup is excluded because it requires user interaction and its computational overhead is considered negligible compared to the subsequent MPC protocol execution.

As a testing environment, measurements rely on common consumer hard-
ware since the given use case is best reflected by this choice. All parties are simulated on a single Windows 10 office notebook computer with a dual-core Intel i5-6200U processor running at 2.30GHz, as well as 8GB RAM. The individual processes communicate with each other over the localhost, eliminating unpredictable network factors such as medium access and existing traffic. In order to emulate network latency in a controllable manner, the lightweight network tool Clumsy\textsuperscript{1} finds its use.

Measurements were performed for both the first match and the best match scheduling scheme. Note that the weighted best match scheme is not evaluated separately due to the fact that its circuit construction is nearly identical to the non-weighted case. Although the choice of the users’ priorities has an impact on the required bit length of the internal counter registers, the overall asymptotic complexity remains unaffected.

It is also worth emphasizing that the benchmark results do not in any way depend on the concrete values of the input bits for each party. This is because boolean circuits do not comprise a concept of dynamic branching. In fact, the security of the GMW protocol hinges on the assertion that the actual values of circuit wires are not known to any participant until the computation is completed. Any variations of externally measurable physical quantities would enable attackers to mount side-channel attacks on an ongoing multi-party computation.

### 6.1.1 Running Time

The overall running time of the scheduling protocol is critical for the perceived interactivity of the application. An important requirement when developing the prototype was that the computation finishes reasonably fast, in a way that the waiting time until an appointment is successfully scheduled does not interfere with the user’s general workflow. In order to obtain meaningful data, the values presented in the following were each obtained by averaging over three independent measurement runs. Each run measures the total time it takes to evaluate the generated boolean circuit via the GMW protocol, where the timer stops as soon as every party is in possession of the scheduled appointment date. Circuit generation itself is not included in the measurements since it can be performed offline and constructed circuit files can be cached to speed up subsequent runs. Furthermore, the construction process completes in an order of milliseconds and is thus insignificant for the overall delay perceived by the user.

Figure 6.1 shows the running time of the computation for different

\textsuperscript{1}http://jagt.github.io/clumsy
6.1. Performance Measurements

Figure 6.1: Overall protocol running time in a LAN setting with negligible latency.

Scheduling schemes and an increasing number of time slots \( m \), plotted against the number of parties \( n \). No artificial network latency was introduced, simulating a LAN environment. It becomes evident that first match scheduling does not take much longer than one minute, even for party numbers up to 30. Primarily, \( n \) is the determining factor on the overall running time, whereas the number of time slots \( m \) only has a minor impact. This is not the case anymore as far as the best match scheme is concerned: A high number of time slots significantly increases the measured running time. However, scheduling still finishes within the order of minutes.

In order to analyze how the running time scales with \( n \) and \( m \), it is necessary to understand the inner workings of the GMW protocol. Whereas XOR gates only require local computations and can thus be considered free, it is necessary to perform a fixed number of OTs between each pair of parties whenever an AND gate is evaluated. An individual party is expected to communicate with \( n - 1 \) hosts, meaning that in theory, the overall effort per AND gate scales linearly with the number of parties. However, there is room for massive parallelization: Two gates are called dependent on each other only if the output of one gate \( A \) directly or indirectly forms the input of the other gate \( B \). In this case, it is mandatory to fully compute the sharing of the output of \( A \) to allow processing of \( B \). Otherwise, the gates can be evaluated simultaneously, resulting in significantly increased performance. The degree of possible parallelization is best reflected by a circuit’s multiplicative depth, which is the maximum number of consecutive AND gates found in the circuit. In other words, the multiplicative depth is the number of depth layers that
are mutually dependent, whereas the gates that are part of the same layer can be processed in parallel. Assuming perfect parallelization of independent AND gates, the multiplicative depth becomes the determining factor for the time it takes to securely evaluate a given circuit.

<table>
<thead>
<tr>
<th></th>
<th>( n )</th>
<th>( m = 10 )</th>
<th>( m = 100 )</th>
<th>( m = 1000 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>101</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>102</td>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>102</td>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>103</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>103</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>103</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>103</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>14</td>
<td>104</td>
<td>1004</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>104</td>
<td>1004</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.1:** Multiplicative depth of the evaluated circuit for different values of \( n \) and \( m \).

Table 6.1 lists the multiplicative depth of the generated scheduling circuits for increasing numbers of parties and time slots. It becomes apparent that the multiplicative depth with respect to the *first match* scheduling scheme grows in \( \mathcal{O}(\log n + m) \), which can be justified with the construction rules specified in section 4.1.2. The free time slots of each party are intersected by performing \( n - 1 \) bitwise AND operations, arranged as a balanced tree with the \( m \) time slots processed in parallel. The tree’s depth of \( \log n \) is reflected by the first term. The second term results from the final filtering operation, which clears all set bits except the first one. For this purpose, the \( m \) bits are traversed in a linear fashion, where the state of bit \( j \) depends on the combined state of all previous bits and therefore cannot make use of concurrent evaluation.

The *best match* scheme, on the other hand, has a multiplicative depth complexity of \( \mathcal{O}(\log n + m \log n + m) \). Although this behavior resembles the *first match* case, the explanation for the logarithmic term is completely different. As a first step, integral counters are set up to accumulate the number of votes per time slot. The bit length \( \ell \) of these integers is chosen in a way that the maximal possible number of votes \( n \) can be stored, which is most efficiently achieved by allocating \( \ell = \lceil \log(n + 1) \rceil \) bits. Counting is performed sequentially over the integer length, starting from the least significant bit,
since the carryover at index $i$ is incorporated into the computation for position $i+1$. The scores for all $m$ time slots can be computed in parallel, contributing the term $\mathcal{O}(\log n)$ to the overall depth complexity. The second step involves comparing the scores in order to find the largest value. As detailed in section 4.1.1, comparison operations are implemented similar to addition circuits and therefore share the same depth complexity $\mathcal{O}(\ell) = \mathcal{O}(\log n)$. Because it is necessary to compare the $m$ scores in sequence to keep track of the maximum at any time, this stage introduces the additional $\mathcal{O}(m \log n)$ term. The remaining component linear in $m$ again originates from the filtering step at the end.

Figure 6.2: Computation time per depth layer in a LAN setting with negligible latency.

Figure 6.2 correlates the overall protocol running time with the circuit’s multiplicative depth. For a fixed number of time slots $m$, the average time required per depth layer increases linearly with the number of parties $n$. This behavior is in accordance with the theoretical understanding of the GMW protocol: For each depth layer, the communication overhead of an individual party is determined by the time it takes to serve $n-1$ connections to other parties. However, it is remarkable that the amortized effort per depth layer significantly decreases with a higher number of time slots $m$. This observation can be explained by the fact that SCAPI makes use of OT extensions. Instead of carrying out all required OTs independent of each other, only a small number of base OTs is prepared initially. On this basis, a large number of additional OTs can be derived at considerably lower computational costs. With higher $m$, the number of AND gates in the circuit and thus the number of required OTs increases, which in turn makes the use of OT extensions much
more efficient. As a result, the average time per depth layer diminishes with larger circuit sizes since the cost of an individual layer is directly determined by the rate at which OTs can be carried into effect.

![Graphs showing running time vs. number of parties for different values of m.](image)

(a) First match  
(b) Best match

**Figure 6.3:** Overall protocol running time with a simulated network round-trip time of 100ms.

Secure multi-party computation protocols such as GMW regularly require interaction between all parties, which is realized by communication over the network. Therefore, it is clear that network latency plays an important role in regard to the expected performance. Running time measurements with significant simulated latency are presented in figure 6.3. The round-trip time, i.e. the time it takes to send a message to a remote host and successfully receive a response, was set to 100 milliseconds by introducing an artificial network delay. This parameter choice is considered a meaningful reference for cable-based communication between parties located in different countries, which conforms to the corporate use case. Besides the WAN setting, a round-trip time of 100 milliseconds is also exemplary for medium-ranged connections from a mobile device, using wireless telecommunication technology. The diagram shows that the scheduling computation can still be completed in the order of minutes with up to 20 participants, even when suffering from high latency. However, a large number of time slots substantially increases the overall running time, narrowing down the time frame to which the appointment is bound.

Figure 6.4 visualizes the average computation time with respect to a single depth layer. The linear trend confirms that the prior assumptions regarding the protocol’s asymptotic complexity still remain valid in a WAN environment. It shows that the OT extension technique again reduces the effort
per layer more significantly the larger the circuit gets, where the circuit size in turn is most notably affected by the choice of $m$. The time it takes for OTs to succeed continues to be the determining factor, although the absolute time per instance is higher compared to the LAN setting due to the increased round-trip time.

### 6.1.2 Network Utilization

Another performance metric is the total volume of exchanged messages over the network, measured in bytes. The lower the required bandwidth, the better the protocol is applicable in constrained network environments, for instance when using a public access point with limited resources shared between a large number of users. Another example is the usage of mobile devices to securely schedule appointments over cellular data networks. These services are typically billed on a per volume basis, highly favoring protocols with low bandwidth requirements.

The total volume of data transferred during the computation phase is shown in figure 6.5 as a function of $n$ and $m$. The listed measurements incorporate messages sent by any party, excluding packet headers and thus focusing only on application layer information. Therefore, these values serve as a meaningful lower bound, where additional overhead is likely to be introduced dependent on the underlying transport protocol. In order to collect the presented samples, the diagnostics tool *Process Monitor*\(^2\) was configured to

\(^2\)https://docs.microsoft.com/sysinternals/downloads/procmon
filter all TCP Send events issued by the scheduling application [SRI12]. The reference value for the overall network utilization is then obtained by computing the sum over the payload sizes of all captured segments. The same result can also be achieved with the packet analyzer Wireshark\(^3\), but this approach would have involved more manual steps and was thus abandoned during the course of this evaluation [San17].

As illustrated by the above diagram, the volume of messages that need to be exchanged when securely evaluating a given scheduling circuit lies in the general order of several megabytes. It becomes evident that network utilization increases considerably with the number of parties \(n\), which is consistent with the fact that data has to be transferred between each pair of hosts. Because of the protocol’s peer-to-peer nature, the communication volume per AND gate evaluation inherently scales in \(O(n^2)\). It turns out that the number of time slots \(m\) also severely impacts the measured data volume due to the parameter’s effect on the overall circuit size.

It proves beneficial to study to what extent scheduling circuits grow as a function of \(n\) and \(m\) in order to fully understand the measured message volume. The amounts of AND gates that are required to construct scheduling circuits for different parameter combinations are listed in table 6.2. In contrast to the multiplicative depth values presented in table 6.1, the shown quantities do not carry any information on how the AND gates interrelate. Whereas AND gates independent of each other can be evaluated in parallel and thus save execution time, this property does not play any role when it

\(^3\)https://www.wireshark.org
6.1. Performance Measurements

Table 6.2: Total number of AND gates in the evaluated circuit for different values of $n$ and $m$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m = 10$</th>
<th>$m = 100$</th>
<th>$m = 1000$</th>
<th>$n$</th>
<th>$m = 10$</th>
<th>$m = 100$</th>
<th>$m = 1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>27</td>
<td>297</td>
<td>2997</td>
<td>2</td>
<td>72</td>
<td>792</td>
<td>7992</td>
</tr>
<tr>
<td>4</td>
<td>47</td>
<td>497</td>
<td>4997</td>
<td>4</td>
<td>160</td>
<td>1690</td>
<td>16990</td>
</tr>
<tr>
<td>6</td>
<td>67</td>
<td>697</td>
<td>6997</td>
<td>6</td>
<td>220</td>
<td>2290</td>
<td>22990</td>
</tr>
<tr>
<td>8</td>
<td>87</td>
<td>897</td>
<td>8997</td>
<td>8</td>
<td>368</td>
<td>3788</td>
<td>37988</td>
</tr>
<tr>
<td>10</td>
<td>107</td>
<td>1097</td>
<td>10997</td>
<td>10</td>
<td>448</td>
<td>4588</td>
<td>45988</td>
</tr>
<tr>
<td>12</td>
<td>127</td>
<td>1297</td>
<td>12997</td>
<td>12</td>
<td>528</td>
<td>5388</td>
<td>53988</td>
</tr>
<tr>
<td>14</td>
<td>147</td>
<td>1497</td>
<td>14997</td>
<td>14</td>
<td>608</td>
<td>6188</td>
<td>61988</td>
</tr>
<tr>
<td>16</td>
<td>167</td>
<td>1697</td>
<td>16997</td>
<td>16</td>
<td>856</td>
<td>8686</td>
<td>86986</td>
</tr>
<tr>
<td>18</td>
<td>187</td>
<td>1897</td>
<td>18997</td>
<td>18</td>
<td>956</td>
<td>9686</td>
<td>96986</td>
</tr>
<tr>
<td>20</td>
<td>207</td>
<td>2097</td>
<td>20997</td>
<td>20</td>
<td>1056</td>
<td>10686</td>
<td>106986</td>
</tr>
</tbody>
</table>

(a) First match  
(b) Best match

comes to network utilization. In the first match case, the number of AND gates increases in the order $O(m \cdot n)$ since $n - 1$ bitwise AND operations need to be performed to combine the $m$-bit inputs provided by each party. The best match circuit, on the other hand, theoretically grows in $O(m \cdot n \log n)$ due to the use of counter registers: For each of the $m \cdot n$ input bits, an addition of $\ell$-bit integers is required to increment the score for the respective time slot, where the bit length $\ell$ is logarithmic in $n$. However, it is worth to note that the addend can only attain the values 00...0 or 10...0 and therefore $\ell - 1$ digits actually hold constant bits. As described in section 5.1.1, the static optimizer can make use of this knowledge to remove a significant number of redundant AND gates. The impact of this optimization is difficult to quantify though. Therefore, an increase in the order of $O(m \cdot n \log n)$ is considered a reasonable upper bound when it comes to the number of AND gates in the circuit.

By understanding how the parameters $n$ and $m$ affect the circuit size, one can finally draw conclusions on the asymptotic complexity in terms of network utilization. As mentioned earlier in this section, the overall number of messages that need to be exchanged in order to evaluate a single AND gate scales quadratically with $n$, due to the fact that communication is necessary between each pair of parties. Combined with the circuits’ increase in the number of AND gates, it turns out that data in the order of $O(m \cdot n^3)$ needs to be transferred in the first match case and $O(m \cdot n^3 \log n)$ when performing best match scheduling. These theoretical findings are consistent with the practical measurements presented in figure 6.5. It is important that the cubic
growth rate is taken into account when scheduling an appointment with a large number of parties because the protocol’s high bandwidth utilization can easily monopolize the available network resources, especially when relying on mobile communication.

6.2 Threat Assessment

The measurements discussed in this chapter confirm that calendar scheduling based on MPC protocols is indeed applicable for large numbers of participants as it is common in corporate environments. It can therefore be stated with confidence that the goal of scalability has been achieved. However, additional requirements were formulated in section 3.3 that are important for the practical use of the developed software. The most fundamental goal was to ensure privacy regarding the users’ personal timetables, which is in fact the sole reason why this work strives towards a decentralized alternative to common scheduling software. By relying on the GMW protocol that makes use of cryptographic mechanisms such as secret sharing, it is guaranteed that the parties’ inputs remain private even in the presence of an arbitrary number of passive attackers.

As motivated in section 3.1, security against semi-honest participants is considered sufficient in a corporate environment. It is nevertheless insightful to discuss on a theoretical basis to which extent active attackers can take advantage of interfering with the scheduling protocol.

Recall that, following the GMW protocol, a wire value $v$ is randomly split into the binary shares $v_1, \ldots, v_n$ in a way that $v = v_1 \oplus \ldots \oplus v_n$ is satisfied. Each party $P_i$ holds a single share $v_i$, meaning that all parties need to cooperate to reveal the wire’s true value $v$. Let party $P_x$ be corrupted by an active adversary. Although the knowledge of share $v_x$ does not allow to deduce any information on $v$, local changes to $v_x$ still propagate to $v$. More precisely, by pretending to possess the negated bit $\neg v_x$ instead of $v_x$, an active attacker is capable of blindly changing the combined wire value to $\neg v = v_1 \oplus \ldots \oplus \neg v_x \oplus \ldots \oplus v_n$ and can thus purposefully flip any bit in the circuit. This attack relies on the assumption that party $P_x$ is the only one deviating from the protocol, i.e. that every other party $P_i$ correctly submits their unaltered share $v_i$. In terms of boolean logic, the ability to deliberately negate expressions implies that the attacker can turn any AND gate into an OR gate following De Morgan’s laws. By negating the AND gate’s inputs $u$ and $v$ as well as the output, the involved parties effectively compute $\neg(\neg u \cdot \neg v) = u \lor v$. In addition to the ability to arbitrarily flip wire values during the execution, the adversary can also freely choose his own
input bits in order to change the protocol’s course in his favor.

There are two different objectives an active attacker might try to accomplish. First, he might interfere with the protocol to change the outcome of the computation, fixing an appointment date of his choice. This attack compromises the computation’s correctness. Another strategy targets the privacy of personal information: The attacker tries to manipulate the protocol in a way that the result leaks parts of the users’ secret inputs. In the following, a few exemplary attack paths are presented. However, there certainly exist many more attack vectors an active adversary can make use of. Instead of providing an exhaustive security analysis, this section is intended to give a brief overview that serves as a starting point when reasoning about potential threats.

6.2.1 Attacks on Correctness

The result of the first match scheduling circuit can be altered by a malicious attacker in a straightforward manner. By choosing input values of 0 for every time slot, the bitwise conjunction $\text{And}(b_1^n, b_1^n, \ldots, b_m^n, b_1^n, \ldots, b_m^n)$ is also guaranteed to be all-zero. As a last step, the attacker can then flip his share of the $j$-th bit in order to make time slot $j$ the ostensible outcome of the computation. This strategy, however, comes with a risk. All those honest participants who marked time slot $j$ as unavailable immediately know that the protocol has been tampered with, due to the fact that the observed output can impossibly be produced by the correctly working algorithm. Hence it is likely that there are consequences on the human level in reaction to the attack.

Manipulating an instance of the best match scheme is more difficult since there is no way to predict the output without knowing the other parties’ inputs. Passing an all-zero input to the computation is not an option anymore because it does not prevent the remaining participants from contributing their inputs to the overall result. The structure of the circuit can nevertheless be exploited to force a certain outcome: As detailed in section 4.1.2, the scores for each time slot are compared to each other to derive the bits $c_1, \ldots, c_m$, where $c_j = 1$ holds only if the scores of all subsequent time slots $k > j$ are less than or equal to the score of slot $j$. In particular, this construction means that $c_1$ is zero whenever there exists a time slot with higher score than the first one. The attack now relies on the fact that it is unlikely that the first slot indeed gets the most votes. Therefore, the attacker has to provide the inputs $01\ldots1$ to the computation to promote any but the first time slot. Under the additional simplifying assumption that the votes of all other parties follow a random distribution, the probability that the first slot accumulates the
Chapter 6. Evaluation

The highest score is less than $\frac{1}{m}$. Therefore, the attacker can presume that $c_1 = 0$ holds. By flipping the first bit $c_1$ to the expected value of 1, the result of the computation after the final filtering step is $10\ldots0$ with a probability higher than $\frac{m-1}{m}$. Analogous to the first match case, the attacker can finally negate both the first and a $j$-th bit of his choice to accomplish his purpose.

Theoretically, the described strategy has a high probability of succeeding, but fails in practice as soon as the first time slot receives the highest number of votes. In this case, more than one bit is set in the output, which is easily detected by other participants. Since it is impossible that a correct circuit evaluation produces a malformed result, this observation is considered proof for the presence of an active attacker. Although there exist no protocol mechanisms to identify the corrupt party, the attacker still has to fear external investigations as soon as suspicious actions are detected. It is worth emphasizing that the chance of success directly depends on $m$, meaning that a fraudulent act is much less likely to be uncovered for appointments with a large number of time slot candidates to choose from.

6.2.2 Attacks on Privacy

Instead of changing the computation’s result to his own advantage, the attacker might also be interested in uncovering the private inputs of other participants. These pieces of information directly provide insights into the users’ personal timetables and therefore represent a valuable target. Since the actual input values are split into shares that are kept secret, the only option for the attacker is to manipulate the protocol in a way that input bits are encoded in the output of the computation and thus disclosed eventually. As a consequence, it becomes clear that the attacker can leak at most one private bit per protocol execution without being detected. As soon as more than one set bit is observed in the output, the other parties immediately infer that the computation has been compromised.

The most simple attack on privacy requires $n - 1$ semi-honest parties to collude in order to obtain information on the remaining party’s input. Let Alice be the unsuspecting victim, denoted as the first party without loss of generality. The other $n - 1$ participants now work together with the goal of finding out her input bits $b_1^1 \ldots b_m^1$. By providing $n - 1$ times the input $11\ldots1$, the bitwise conjunction in the first match scheduling scheme actually computes $\text{And}(b_1^1 \ldots b_m^1, 11\ldots1, \ldots, 11\ldots1) = b_1^1 \ldots b_m^1$ and thereby passes Alice’s inputs through unchanged. Analogously, when using the best match scheme, the same procedure causes the scores per time slot to exactly reflect Alice’s votes. After the final filtering step, the output coincides with her input in the first non-zero bit. If the computation is repeated multiple times,
6.2. Threat Assessment

The attackers can mask certain time slots by clearing the respective bits in their input, revealing Alice’s private input bits one at a time. Although it is not very likely that only one user is fully honest out of a large number of participants, this attack is nevertheless relevant for small scheduling sessions. In particular, the two-party case becomes inherently insecure because one party can trivially obtain input bits of the other party. It is already sufficient to choose the own inputs accordingly, without manipulating the protocol and thus the risk of being detected.

More research needs to be done in order to assess to which extent a single active attacker can extract private input bits by manipulating the protocol. With respect to the first match scheduling scheme, a conceivable strategy might rely on purposefully turning certain AND gates into OR gates, therefore computing either the intersection or the union of two sets containing the users’ available time slots. Carefully arranging set operations in the predefined, tree-like structure together with the ability to freely choose the own input bits appears to be a promising approach. The best match scheme, on the other hand, is expected to be more resilient against attacks because the inputs do not immediately affect the outcome. Instead, there are additional layers of score accumulation and maximization in between which significantly complicate the correlation of input and output bits. A methodical analysis of active attack paths in this context certainly yields interesting results, but is out of scope of this work and thus postponed for future evaluation.

Conclusively, it can be claimed that the prototype developed during the course of this thesis indeed brings scheduling applications an important step further towards the goal of privacy. Security against semi-honest participants is already a valuable guarantee, especially combined with external measures such as software restriction policies. Even under the presence of malicious adversaries, the protocol does by no means break completely. Instead, it is difficult to extract private information at all and interfering with the collaborative computation exposes the attacker to a substantial risk of being detected. In case that the provided level of security is not considered sufficient anymore, the presented scheduling system is also flexible enough to exchange the MPC framework with one that explicitly includes mechanisms against active attacks. However, one must be aware that higher privacy guarantees usually result in increased computational complexity, which conflicts with the scalability requirement. Choosing a suitable MPC framework typically comes with balancing the tradeoff between running time and security, which must therefore be undertaken carefully in consideration of the given use case.
6.3 Practical Use

After evaluating the prototype both in terms of scalability and privacy, it remains to discuss the implications of a decentralized scheduling approach on the usability requirement. Since the application is intended for practical, everyday use, it is of interest to compare the workflow to established centralized scheduling software. The web-based service Doodle is used as an example in the following.

![Comparison of primary user interfaces](image)

**Figure 6.6:** Comparison of primary user interfaces that provide an overview of the appointment in question as well as mechanisms for reviewing time slots. Besides the prototype developed during the course of this work (a), a screenshot of Doodle is shown as a reference (b).

Conceptually, the steps necessary to eventually agree on an appointment date are very similar in both cases. An initiator first needs to set up a session by entering a description of the event and selecting a number of time slot candidates. Afterwards, invites can be sent to potential participants. For this purpose, the prototype implements network broadcasts, whereas Doodle utilizes confidential URLs to the website as the primary exchange mechanism. Both applications expose a graphical user interface for managing the active scheduling session, illustrated in figure 6.6. Aside from the fact that Doodle, as a customer-oriented service, puts more emphasis on a visually appealing presentation, the included components are nearly the same. The upper portion of the screen shows the title of the event together with a short description of the subject. Below, one finds the list of participants and a tool that allows to view the time slot candidates while marking them as either...
free or occupied. Whereas Doodle shows the votes of all parties by default, the implemented prototype only has knowledge of the local inputs due to the underlying understanding of privacy. As a consequence, Doodle is capable of showing intermediate results even if not all participants have provided their inputs yet.

Since the prototype’s GUI is designed to generally allow the same workflow as established scheduling services, it is expected that users quickly learn to cope with the new application. There exists full support for integrating universal calendar software based on iCalendar files. Thus, users can continue to rely on their favorite tool for managing their timetables, such as Outlook or Apple Calendar. Due to the fact that Doodle provides a similar workflow, this feature should be straightforward to use. However, the decentralized nature of the networking protocol presented in this work has one major implication as far as usability is concerned: All collaborating parties need to be online at the same time in order to successfully carry out the scheduling operation. In contrast, services such as Doodle are backed by a central server and users can therefore submit their inputs at any time, independent of the other participants. In a corporate environment, this issue can probably be considered insignificant since most employees have permanent access to their computer anyway. Nevertheless, users need to get accustomed to this restriction, which is necessary for ensuring privacy of their personal information.

Finally, it remains to be emphasized that the goal of developing a scheduling application for practical use has indeed been achieved. The prototype comprises all necessary components that allow deployment in a corporate context, ranging from interoperability with common calendar software to an intuitive user interface. Although there are still some limitations, for instance the fact that communication is limited to the local network, none of these prevent successful practical operation. Furthermore, it is likely that most limitations can be removed in the future by further improving the existing system.

6.4 Summary

The subject of this chapter was to evaluate to which extent the implemented scheduling application satisfies the previously formulated requirements. Whereas running time and network utilization measurements provided the basis for evaluating scalability, additional unquantifiable goals such as privacy and usability needed to be assessed by other means.

Running time was considered the primary metric since the delay until
an appointment is successfully scheduled is directly perceived by the user. Generally, the measured time values did not significantly exceed one minute even for party numbers up to 30. It was pointed out that the running time is most notably affected by the multiplicative depth of the evaluated circuit, which is the number of interdependent layers of AND gates. For a fixed number of time slots $m$, the circuit depth increases logarithmic in $n$. Since a party needs to interact with all $n - 1$ other parties to evaluate a layer of AND gates, the overall runtime complexity can be characterized as $O(n \log n)$ as long as $m$ is considered constant. The impact of an increasing number of time slots is difficult to quantify, though. Since SCAPI makes use of OT extensions, the average costs per AND gate diminish for larger circuits as attributable to the effect of $m$, mitigating the additional overhead. In network settings with high latency, the asymptotic complexity remains unchanged, although the higher communication delay per depth layer results in a slightly increased overall running time.

The network utilization, on the other hand, lies in the order of several megabytes for common real-world parameter choices. However, it turns out that the total message volume grows at least in $O(m \cdot n^3)$ and therefore quickly becomes the determining factor for high party numbers, which is particularly relevant in constrained network environments.

Although semi-honest security was already considered sufficient in a corporate use case, this chapter briefly discussed the capabilities of malicious adversaries with regard to calendar scheduling. Due to the inner workings of the GMW protocol, active attackers are generally able to blindly flip any bit during circuit evaluation without knowing its actual value. This technique allows the attacker to freely change the scheduling result to any desired outcome, albeit being at risk that the manipulation is detected by other parties. A different category of attacks aims at revealing private inputs provided by honest users. It became clear that an adversary can safely leak at most one private bit per protocol execution, otherwise a malformed output serves as proof of manipulation. Although a theoretical semi-honest attack on privacy was presented that requires $n - 1$ corrupted parties, it appears that there exist no straightforward active attacks achieving the same goal.

Lastly, the implemented prototype was examined in terms of usability. Since its workflow closely resembles that of established scheduling software, the secure application is expected to be similarly intuitive for users. However, there are differences that require to change one’s mindset: First, users can only see their own votes during the process, which directly originates from the privacy requirement. Second, it is necessary that all participants are online at the same time in order to complete the computation. It is likely that users accept these minor inconveniences in favor of protecting their personal data.
To conclude this chapter, Table 6.3 briefly summarizes the evaluation results by listing the requirements formulated in Section 3.3 as well as the taken measures to fulfill them. It shows that the approach presented in this work indeed achieved all predefined goals.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Fulfilled</th>
<th>General Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privacy</td>
<td>✓</td>
<td>GMW protocol for secure computation in a semi-honest setting</td>
</tr>
<tr>
<td>Scalability</td>
<td>✓</td>
<td>Use of SCAPI as a low-level framework and optimization of boolean circuits</td>
</tr>
<tr>
<td>Workflow integration/usability</td>
<td>✓</td>
<td>Intuitive graphical user interface and iCalendar interoperability</td>
</tr>
<tr>
<td>Flexibility</td>
<td>✓</td>
<td>Easy-to-use API providing general-purpose building blocks for modeling arbitrary functionality as boolean circuits</td>
</tr>
<tr>
<td>Extensibility</td>
<td>✓</td>
<td>Well-defined interfaces for circuit construction, secure evaluation and networking to allow substitution of individual components</td>
</tr>
</tbody>
</table>

Table 6.3: Overview of the requirements that guided the development of the secure scheduling application during this work, outlining how the individual problems were solved.
Chapter 7

Discussion

Throughout the preceding chapters, it was detailed how multi-party computation protocols can be utilized in order to achieve secure appointment scheduling. However, there exist many more applications that potentially profit from protecting the users’ privacy by means of cryptography, for example voting and auctioning. The goal of this chapter is to examine in which ways the implemented mechanisms can be generalized or extended to prove beneficial in other application areas, thereby discussing the findings of this work in a greater context.

The conceptual approach outlined in chapter 4 involved the separation between a setup phase and a computation phase, the former assigning a meaning to the abstract time slots addressed in the scheduling circuit. All collaborating parties agree on a mapping that links each index $j$ with a time interval from the real-world domain, thereby transforming the inputs and outputs of the computation from mere numbers into relevant information. In fact, the $m$ different indices do not have to represent time slots at all: A simple poll, for instance, can be easily realized by assigning a textual description to each index. Instead of marking the indices that correspond to free time slots, users then select exactly those options they are comfortable with. Figure 7.1 exemplarily illustrates the necessary adjustments to the prototype’s user interface so that voting is possible. Since the presented scheduling circuits rely on strictly mathematical concepts such as indices and bit arrays and are therefore independent of the imposed real-world mapping, the MPC back end does not need to be changed at all. The first match and best match schemes prove to be equally useful in the context of voting to find the option that pleases either all or most of the participants, respectively.

As far as auctions are concerned, more in-depth customization is necessary. There is no need to distinguish between $m$ binary options anymore, instead users must be able to enter arbitrary quantities in order to place their
bids. Suitable boolean circuits can be easily constructed with the help of the circuit generation API presented in this work, which already comes with building blocks for various integer operations. The GUI has to be reworked as well in order to accept numerical inputs from the user and correctly display the result on the screen. However, the underlying communication protocol that allows users to initiate and join sessions, followed by a collaborative computation, again remains unchanged.

The two presented examples, voting and auctions, show that there exist many practical use cases for secure multi-party computation that follow a similar pattern as covered in this work. It is generally necessary to establish a session first, which allows to introduce the participants to each other and to agree on the circuit to evaluate, together with a common interpretation of inputs and outputs. In a second step, as soon as every party confirms to be ready, a self-contained computation is carried out in a secure manner. Abstracted from the context of appointment scheduling, the contribution of this work is to provide an extensible protocol for this two-phase usage pattern, designed around an MPC framework as its core component. The tools and building blocks necessary to generate boolean circuits for various application areas are made available via an easy-to-use API. Hopefully, these accomplishments serve as a basis to enhance privacy in a number of other use cases in the future.
Chapter 8

Conclusion

Scheduling an appointment with a number of friends or colleagues is a task well covered by software specifically designed for this purpose. However, users are generally not aware that by using these services, private information in the form of personal timetables is shared not only with other participants, but also with an intermediary third party. Subject of this work was to show that privacy can indeed be preserved in the context of calendar scheduling without sacrificing functionality and usability. A practical approach was presented that makes use of secure multi-party computation, a cryptographic technique for jointly computing a common function while keeping the inputs private. Thereby, this work evaluated the applicability of MPC protocols for real-world problems, providing valuable insights to the question of scalability as well as how privacy enhancing building blocks can be integrated into complex software systems.

The approach proposed in this thesis consists of two phases with different privacy policies and demands in terms of the underlying network topology. The invitation phase can be considered an openly accessible session setup that allows potential participants to form a group around an initiator as well as to agree on common protocol parameters. Subsequently, the computation phase encapsulates a secure evaluation of the scheduling algorithm, modeled as a boolean circuit, based on MPC. Several scheduling schemes are provided to allow flexible usage in different scenarios. In order to achieve interoperability, iCalendar files serve as a generalized interface towards existing calendar software. As a proof of concept, this work is accompanied by a practical prototype implementing the described functionality.

Relying on SCAPI as a suitable MPC framework, the prototype was evaluated in terms of privacy, scalability and usability. Privacy in a semi-honest setting directly follows from the cryptographic guarantees the GMW protocol provides. In addition, it was motivated that the specific structure of the con-
structured scheduling circuits also proves resilient against active attacks. By analyzing the running time of the computation, it was shown that scheduling can be performed in a secure fashion while remaining interactive even with a large number of participants. Measurements for values of \( n \) as high as 30 indicate that the result can still be obtained in a reasonable time frame of few minutes, meaning that the perceived delay most likely does not disrupt a user’s daily routine. It was noticed however that network utilization rapidly increases the more parties take part in the computation, which can easily become the determining factor dependent on the available network resources. Finally, it was stated that the prototype indeed provides an intuitive workflow due to its straightforward user interface and adherence to the iCalendar standard for compatibility. Therefore, the implemented secure application is considered to come close to established appointment scheduling software in regard to usability.

Aside from a fully functional implementation of private calendar scheduling, another contribution of this work is to offer a powerful API for constructing boolean circuits from basic building blocks for secure evaluation, which is expected to also prove valuable outside the context of appointment agreements. Furthermore, the presented two-phase protocol is designed general enough to be beneficial in various other application areas where MPC techniques are planned to be used to preserve privacy.

8.1 Future Work

The primary limitation of the developed prototype is its lack of secure communication channels for session setup as well as for carrying out the MPC protocol. Hence, the most important task in the future is to rework the network layer to rely on TLS instead of plain TCP connections, providing data confidentiality and authentication. These measures are sufficient to exclude eavesdroppers in the local network and to prevent fraudulent impersonation of other participants. The ability to unequivocally identify each party allows to safely make use of asymmetric scheduling schemes like those assigning differing priorities to users. Altogether, these enhancements are essential to obtain a scheduling application that is truly secure in every aspect.

Despite the proposed implementation being fast enough to allow multi-party scheduling for large groups of participants, there is nevertheless an upper bound in terms of scalability. Eventually, the running time of the protocol exceeds the limit that is acceptable for users. Even if interactivity is not considered the primary concern, there exists a point at which the increasing volume of exchanged messages fully monopolizes the provided net-
work resources. Thus, it makes sense to evaluate the following strategies for improving the overall performance:

- **Circuit optimization**: The most direct way of reducing the computation’s running time is to minimize the number of gates from which the scheduling circuit is assembled. XOR gates should be preferred over AND gates wherever possible since they can be computed locally at basically no costs. Additional effort needs to be put into constructing depth-efficient circuits to profit from parallelized evaluation of gates.

- **Protocol optimization**: Another approach is to tweak the underlying MPC implementation in order to make it faster in the given use case. Frequent use of multithreading is one potential mechanism to examine, another being the introduction of an offline phase to which certain tasks can be moved to speed up the actual computation. It might even prove beneficial to exchange the MPC framework as a whole as soon as more efficient implementations emerge.

- **Hierarchical computation**: The previously mentioned optimization strategies cannot solve the inherent problem that MPC protocols scale unfavorably with an increasing number of parties. To bypass this fact, a possible solution may be to split up the scheduling procedure into multiple MPC instances with a lower number of parties each. This way, one hundred participants can agree on an appointment date by performing five independent secure computations of practical size, combining the results afterwards. It is subject for discussion how the intermediate results need to be formed to reveal only a minimum of private information, as well as the question which parties are granted the responsibility of the final combination step in a way that cheating is prevented.

With these options at hand, it is interesting to assess whether privacy-preserving techniques can indeed be made practical on a massive scale.

The circuit generation API designed during the course of this work is intended to facilitate the specification of boolean circuits by providing high-level constructs to express algorithms in a natural way. Even though there already exist many general purpose building blocks such as integer operations, it is certainly helpful to extend the toolbox with additional primitives, for example fixed-point arithmetic. Thereby, it hopefully becomes much easier to develop new applications based on MPC in the future.

Lastly, more research needs to be conducted to precisely determine the capabilities of active attackers when relying on MPC protocols with semi-honest security. The findings in section 6.2 can serve as a starting point for
an in-depth threat analysis. Discussing different attack paths certainly yields interesting results on the optimal structure of scheduling circuits in terms of resilience against malicious adversaries. On this basis, an informed decision can be made whether or not privacy in a semi-honest setting is considered sufficient even if active attacks cannot be prevented by external measures.
Bibliography


[Gam+95] Erich Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.


