TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bachelor Thesis

Web application for privacy-preserving scheduling

Oliver Schick

December 12, 2017

CRISP
Center for Research
in Security and Privacy

Technische Universität Darmstadt
Center for Research in Security and Privacy
Engineering Cryptographic Protocols

Supervisors: Dr. Thomas Schneider
M.Sc. Ágnes Kiss

Thesis Statement
pursuant to §23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Oliver Schick, have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Darmstadt, December 12, 2017

Oliver Schick

Abstract

Arranging a meeting between multiple parties is a highly recurring task, that is becoming more and more time consuming, the more people and organisations get involved. Without any means of automatization, the arranging of a meeting between a larger amount of parties is very difficult to handle and thus, the need for automatic tool support arises. However, using these tools also raises some concerns regarding the privacy of the participants, as private information may be inferred from the availability patterns the participants publish when partaking in the poll.

In this work we will introduce a protocol that allows to schedule meetings even between large amounts of participants, without requiring any participant to reveal his or her availability pattern to any other party. The protocol requires an evaluation function to be defined in order to find the scheduled time and allows for a large set of possible evaluation functions to be used, i.e. every function that can be represented by a Boolean circuit can be used as evaluation function in our protocol.

Our protocol needs two servers to compute the evaluation function using a secure two-party computation protocol in order to keep the availability patterns of the participants secret. Therefore, we have to assume that the servers do not collude in order to gain knowledge about the inputs of the participants. Furthermore, if the secure two-party computation protocol used when implementing our protocol does not provide security in the presence of malicious adversaries, we have to assume that the servers are semi-honest.

Contents

1 Introduction

Arranging a meeting between multiple persons or organisations is a highly recurring task that can take a lot of time, the more people and organisations get involved. While most bigger companies already provide internal solutions to this problem, these solutions only apply when scheduling internal meetings. As soon as the need arises to schedule a meeting between multiple organisations, the internal solutions do not work anymore. There are multitudes of available online solutions to this problem, but several concerns regarding the privacy arise when using these solutions.

The first privacy issue is that all participants can see the selections made by other participants, which raises some privacy concerns as information might be leaked through the availability patterns of the participants. For example a participant P knows that another organisation A will hold an important board meeting and knows that all members of the executive committee will attend, but since the date is being held secret, P does not know when. If all members of the executive committee participate in a poll for arranging another, unrelated meeting in which P also participates, then P can gain information about the date of the board meeting, simply by checking when none of the members of the executive comittee are available. Submitting votes anonymously does not solve this problem as availability patterns are individual and thus leak information about the person. Another concern, which cannot be solved by simply anonymizing the votes, is that the last users submitting their votes can lie about their availability in order to make their preferred date more likely to be chosen, for example by saying they are not available at a given date d, although they are, because they see, that d is likely to be chosen if they say they are available that day.

The second security issue arises when the selections made by the participants can only be seen by a subset of participants (or non-participants) called administrators. Unfortunately a set of administrators cannot always be found, especially when arranging a meeting among several organizations, as an organization might trust one of their employees to see all selections made by their staff but none of the employees of another organization and vice versa. Also, since only the administrators can see the selections made by the participants, they have the opportunity to cheat by selecting the date they prefer and not the best date in terms of, for example, number of available participants. A simple solution would be to let the server be the only one able to see the inputs of the participants and let the server decide based on a readily implemented algorithm, like for instance, selecting the date where most participants are available. This, however, requires trust in the server, which cannot always be assumed.

1.1 Contributions

There are two main contributions we made: The first being the presentation of a web-based protocol that allows to arrange a meeting between any number of participants using a predefined selection function f. The protocol guarantees that no participant gain neither advantage in the selection of the scheduled time nor information about the selections made by other participants, by deviating from the protocol. However, the presented protocol does neither guarantee correctness of the execution, nor nondisclosure of the selections made by the participants in the presence of a server deviating from the protocol. Furthermore, the nondisclosure is not guaranteed if the servers participating in the evaluation of the selection function f collude in order to gain knowledge about the selections made by the participants.

The second contribution we made is the implementation of two predefined selection functions f_1 and f_2. The first selection function f_1 allows to select a time, where the number of available participants is maximized, while the selection function f_2 allows to additionally assign different importance to different participants, thus selecting a time, where the weighted sum of attendence is maximized. Both functions reveal furthermore the identity of the participants that are not available for the selected time. Besides the output generated by the function no additional information is leaked, to neither the participants nor the servers executing the function.

1.2 Outline

We will introduce in Chapter 2 the concepts used in the subsequent chapters, such as DC-Nets (introduced in Section 2.1) and Commitment Schemes (introduced in Section 2.2), which are both used in Chapter 3. We further introduce the concepts of Boolean circuits (Section 2.3), oblivious transfer (Section 2.4) and secure two-party computation (Section 2.5), all of which are concepts used in our protocol presented in Chapters 4 and 5.

In Chapter 3 we will describe an already existing web-application for secure scheduling, that guarantees correct execution of the protocol and nondisclosure of the selections made by the honest participants in the presence of malicious participants and server. However, that protocol only allows for a limited set of predefined selection functions f and furthermore reveals the sum of each selection made by the participants in each time slot.

We then introduce our protocol in Section 4.1, later discussing its security guarantees and weakpoints in Section 4.2 and finally suggesting possible extensions in order to increase the usability in Section 4.3. We then proceed by presenting the design choices taken during the implementation of our protocol in Chapter 5 and continue by explaining the thought processes made, so that our protocol minimizes the overhead in computation and communication incurred. We then present the runtime measures of our implementation in Chapter 6. Finally, we suggest some potential improvements of our protocol for the future

2 Preliminaries

In this chapter we introduce the concepts used in the subsequent chapters. We first give a short explanation of Dining Cryptographer Networks in Section 2.1 and Commitment Schemes in Section 2.2, both of which are concepts used in Chapter 3. In Section 2.3 we illustrate the concept of Boolean circuits and in Section 2.4 oblivious transfer, both of which are necessary building blocks of the secure two-party computation protocols presented in Section 2.5. We further introduce two widely known protocols for secure two-party computation, namely Yao's protocol (Section 2.5.1) and the GMW protocol (Section 2.5.2).

2.1 Dining Cryptographer Network

A *Dining Cryptographer Network* or *DC-Net* for short, is a network introduced by David Chaum as a solution to his proposed Dining Cryptographer Problem in [Cha88]. DC-Nets allow messages to be broadcasted within a network, without allowing neither internal nor external observers to identify the sender.

Consider n participants in a ring (overlay-) network, which additionally allows for broadcasting messages. Each participant generates a random 1-bit key and shares it with his right neighbor. Now each participant u_i, with u_h being his left neighbor and u_j being his right neighbor, has two 1-bit keys $k_{h,i}$ and $k_{i,j}$, the former being shared with u_h, the latter with u_j. If u_i wants to send a 1-bit message m, assuming he is the only one in the network sending, he broadcasts $b_i = k_{h,i} \oplus k_{i,j} \oplus m$. Every other user broadcasts $b_q = k_{p,q} \oplus k_{q,r}$, where $p, q, r \neq i$ and u_p, u_q, u_r being neighbors. One can easily see that $b_1 \oplus ... \oplus b_n = m$, since every $k_{p,q}$ is contained in both, b_p and b_q for any neighboring participants u_p and u_q.

DC-Nets are not limited to only transmit binary numbers. In fact one can, for example, transmit any element of a group $(\mathbb{Z}_m, +)$, where m is called the *modulus* of the DC-Net and $+$ is the modular addition within Z_m. The protocol described above is thus a DC-Net over the group $(\mathbb{Z}_2, \oplus)$.

2.2 Commitment Scheme

A *Commitment Scheme* is a cryptographic primitive that allows one party P, that does not want to reveal a secret value v yet, to instead send a commitment message μ to a recipient R, which allows R to verify if v was changed after receipt of μ or not.

Commitment schemes can be implemented using cryptographic hash functions. On the one hand it is, for a cryptographic hash function h, computationally hard to find a v' such that $h(v) = h(v')$, which prevents P from changing v retrospectively. On the other hand, given a hash value μ, it is computationally hard to find any v such that $\mu = h(v)$, thus preventing R from inferring the secret value v.

2.3 Boolean Circuits

Let B be a finite set of Boolean functions $g : \{0,1\}^p \to \{0,1\}$. A finite directed acyclic graph containing n input nodes, m output nodes and k gate nodes is called a Boolean circuit over B. The input nodes represent a binary input $i \in \{0,1\}$, the output nodes a binary output $o \in \{0,1\}$ and the gate nodes a Boolean function $g \in B$. Every gate node has at least one outgoing edge and exactly p incoming edges, where p is the arity of the Boolean function represented by the gate node. Incoming edges of a gate node model an input to the Boolean function, while outgoing edges model the output of the previously mentioned function. Output nodes have at least one incoming edge and no outgoing edge. Input nodes have no incoming edges and at least one outgoing edge. For every input node there is at least one path to at least one output node.

Boolean functions can be defined by simply listing an output value for every possible input value in a table, which is then called a truth table. If for every Boolean function $b : \{0,1\}^q \to \{0,1\}$ there exists a Boolean circuit over B, then B is called functionally complete. An example of such a set is $\{AND, XOR\}$, where $AND : \{0,1\}^2 \to \{0,1\}$ is defined as the logical conjunction and $XOR : \{0,1\}^2 \to \{0,1\}$ is defined as the logical exclusive disjunction. A Boolean circuit over a functionally complete set B can represent any mathematical function $f : \{0,1\}^n \to \{0,1\}^m$, which in turn represents any mathematical function $h : A \to B$, over finite sets A, B, by simply defining an injective function $p_i : A \to \{0,1\}^n$ and a surjective function $p_o : \{0,1\}^m \to B$ and then setting $h := p_o \circ f \circ p_i$. If not otherwise stated, Boolean circuits are always assumed to be over $\{AND, XOR\}$.

The number of gates in a Boolean circuit C over B is referred to as *size* of C and the path from an input to an output node containing the most gates is referred to as the *depth* of C. If $B = \{AND, XOR\}$ then the number of AND gates is referred to as the *multiplicative size* of C and the path from an input to an output node containing the most AND gates is referred to as the *multiplicative depth* of C. Edges in a Boolean circuit are considered as *wires*. More information about Boolean circuits can be read in [Vol99, p. 8].

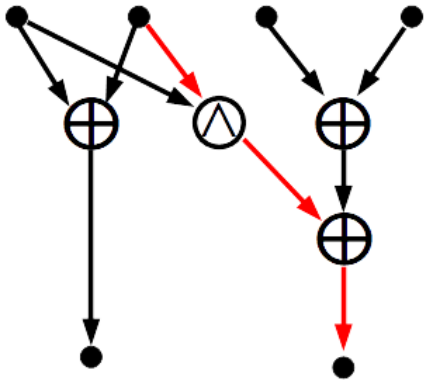

Figure 2.1: A Boolean circuit over $\{AND, XOR\}$ with 4 input nodes and 2 output nodes. The $\oplus$-nodes represent XOR gates and the $\wedge$-nodes represent AND gates. The circuit size is 4, the multiplicative size is 1 and the multiplicative depth is 1 (the path shown in red being the path containing the most AND gates).

2.4 Oblivious Transfer Protocol

A 1-out-of-n oblivious transfer (OT) protocol is a protocol where the a sender wants to transfer 1 out of n distinct messages $m_1, ...m_n$, while the receiver holds a selection number $s \in \{1, ..., n\}$. The sender wants the receiver to only gain information about one out of the n messages and no information about the others, while the receiver does not want the sender to gain any information about s. After executing the OT protocol, only the selected message m_s is learned by the receiver and no information is gained about s by the sender. OT protocols use rather slow public key operations, but can considerably be sped up by using a small amount of so called base-OTs to compute a large amount of OTs using faster symmetric key operations, which is then called *OT extension* [IKNP03; ALSZ13].

2.5 Secure Two-Party Computation

Secure two-party computation is a cryptographic primitive allowing two parties to jointly evaluate a function without any party revealing any information about their input, not accounting for any information leaked by the output of the function. The protocol is required to provide the same secrecy about the input as would a perfectly trustworthy and incorruptible third party evaluating the function. Secure two-party computation can be generalized to include more parties.

Respective to the security models, there are two important adversary models: the *semi-honest* and the *malicious* adversary model. In the semi-honest model, the adversary is assumed to follow the protocol but using every intermediate value of the protocol execution to gain information about the secrets. In the malicious model, the adversary is allowed to arbitrarily

deviate from the protocol to gather information about the secrets. The protocols presented in Section 2.5.1 and Section 2.5.2 assume a semi-honest adversary.

2.5.1 Yao's Garbled Circuit Protocol

Yao's garbeled circuit protocol [Yao86] or Yao's protocol for short, is an asymmetric protocol that allows two parties to jointly compute a function while none of the inputs to the function is revealed to the other party. The function to be computed is described as a Boolean circuit (Section 2.3). One party, usually referred as *garbler* or Alice first generates for each wire two random strings of length k (usually 128) called *labels*, where one label represents the boolean value 0 and the other represents 1. The truth table of each gate is replaced by the corresponding labels and the output is set to $Enc_{k_x^a}(Enc_{k_y^b}(k_z^c))$, where $Enc_k(\bullet)$ is a symmetric encryption scheme and k_x^a, k_y^b, k_z^c are labels corresponding to the values $a, b, c \in 0, 1$ of the wires x, y, z. The encrypted table is called *garbled table*. Alice then sends the labels corresponding to her input value and for each gate a random permutation of the encrypted output value to the other party, called the *evaluator* (Bob). Bob gets his input labels from Alice by using a 1-out-of-2 OT (Section 2.4) protocol and evaluates the circuit by decrypting the encrypted output value using the obtained input labels from Alice. For every gate, Bob can only decrypt one output value correctly, which is then used as input label for the next gate, or sent back to Alice if and only if the label corresponds to a wire going to an output node. Alice then reveals the corresponding plain text value of the output label to Bob.

There are several optimizations to the original protocols, one important being the Free-XOR optimization [KS08], which allows for garbling and evaluating the XOR gates basically for free, since neither communication nor cryptographic operations are needed to evaluate it. Circuits are therefore optimized in minimizing the number of AND gates instead of minimizing the size of the circuit. Other optimizations to Yao's protocol include the *Point-and-Permute*-optimization, which allows Bob to immediately find the correct row to decrypt, without trying decrypting every row until the right one is decrypted, the *Fixed-key-Blockcipher*-optimization, speeding up the garbling and evaluation of the AND-gates by using fixed-key AES (see [BHKR13]) and the *Half-And*-optimization, which allows to transmit 2 rows of the garbled table instead of 4 when evaluating an AND-gate (see [NPS99]).

When considering the communication cost there are two relevant factors to take into account: the number of communication rounds and the communication itself. While Yao's protocol is cheap regarding the number communication rounds, which is independent of the circuit and therefore constant, the communication itself is rather expensive, since a garbled table has to be transferred for every AND gate in the circuit.

2.5.2 GMW

The GMW protocol is a symmetric protocol used for secure two-party function evaluation. Like Yao's protocol, the GMW protocol needs a Boolean circuit representation of the function to be evaluated. In the GMW protocol every value v of each wire is shared among the two parties using a secret sharing scheme such that $v = v_0 \oplus v_1$ for two random-looking shares v_0, v_1 and one party holding v_0 and the other holding v_1. *XOR* gates can be evaluated locally by simply XORing the shares, due to the associativity of the XOR operation. There are two ways to evaluate an AND gate. One way of evaluating $x \wedge y$ with $x = x_0 \oplus x_1, y = y_0 \oplus y_1$ is using a 1-out-of-4 Oblivious transfer. The sender chooses a random share s_1 and provides four data points $s_1 \oplus ((x_0 \oplus x_1) \wedge (y_0 \oplus y_1))$ for every possible combination of x_0, y_0. The receiver can then obliviously get the right data point using his two shares x_0, y_0. Another way of evaluating $x \wedge y$ are *multiplication triples*. Multiplication triples are random shares a_i, b_i, c_i satifying $c_0 \oplus c_1 = (a_0 \oplus a_1) \wedge (b_0 \oplus b_1)$ and which can be generated before evaluating the circuit, during the *setup phase*. The parties exchange $d_i = x_i \oplus a_i$ and $e_i = y_i \oplus b_i$ to obtain $d = d_0 \oplus d_1$ and $e = e_0 \oplus e_1$. The output shares can then be computed as $z_0 = (d \wedge e) \oplus (b_0 \wedge d) \oplus (a_0 \wedge e) \oplus c$ and $z_1 = (b_1 \wedge d) \oplus (a_1 \wedge e) \oplus c_1$. Using the aforementioned property of the multiplication triples one can easily see that $z_0 \oplus z_1 = x \wedge y$.

Unlike Yao's protocol, which has a constant number of rounds, the number of rounds of the GMW protocol are linear in the multiplicative depth of the circuit. However, the communication itself is very low compared to Yao's protocol: GMW only transfers 4 bits per AND gate during the online phase, when using multiplication triples, while Yao's protocol needs to transfer a garbled table per AND gate. The GMW protocol is therefore usually more efficient when working on a circuit with low multiplicative depth and/or executing it on a low latency, low bandwith network, while Yao's protocol is much better when the multiplicative depth of the circuit is high and/or the network's latency and bandwith are high.

3 Related Work

In this chapter we will present already existing web-based scheduling applications. We will first introduce the currently most popular and used scheduling application in Doodle[1] and mention the scheduling application provided by the "Deutsches Forschungsnetzwerk", or DFN for short[2], which is an already existing alternative to doodle that provides security by trust. We finally present the protocol proposed by Benjamin Kellermann.

3.1 Doodle and DFN

Doodle is the currently most popular web-based solution to scheduling between multiple parties. A poll can easily be initiated by a participant, who defines the title and the available time slots the participants of the poll may select from. The inputs of every participant is visible to everyone and no access control is provided by default, hence whoever is in possession of the link leading to Doodle, can see the selections made by all other participants. Moreover, Doodle also mentions, that all data sent to Doodle may be used for advertising purposes.

The DFN proposed an alternative scheduling application, which essentially works like Doodle, but promises to use the data sent to them only for the scheduling. They also guarantee to delete all data they received and thus require the poll initiator to input a termination date, where all data related to the poll will be deleted. However, one has no means to check, if they are indeed deleting the data and not using them elsewhere, hence the DFN provides only security by trust.

3.2 Kellermann's Protocol for Semi-Honest Participants

Benjamin Kellermann suggested a protocol in [Kel11] that allows participants to schedule a meeting without revealing their availability patterns to neither the server nor other participants. Kellermann first describes a simple protocol which requires the participants to only give valid inputs in order for the protocol to work correctly. He then shows how to extend the protocol to prevent malicious participants from gaining any advantages in the

[1] `http://www.doodle.com`
[2] `https://terminplaner.dfn.de/`

time selection, by sending invalid inputs. Finally, Kellermann proposes some extensions to the simple protocol, which increase the usability by introducing additional features like dynamic insertion and removal of participants, without requiring any additional trust into neither particiants nor the server.

Kellermann's protocol consists of the three steps *Poll Generation* (Section 3.2.1), *Voting Process* (Section 3.2.2) and *Evaluation Process* (Section 3.2.3). In the poll generation phase the basic data, like participants and available time slots, are initialized. Then the participants each select their preferred time slots in the voting process and finally, when all participants have cast their votes, the number of votes for each time slot is calculated in the evaluation process by each participant individually. Thus, the information about how many participants voted for each time slot is revealed, which allows the participants to select a winning time slot according to some selection function (like taking the time slots which gained the most votes for example).

This simple protocol offers only security against semi-honest participants, as cheating participants are able to control to a certain extent the time that will be chosen in the evaluation phase. However, even malicious participants will not gain any information about the input of the honest participants (aside the information leaked by the result of the protocol).

3.2.1 Poll Generation

One party called the *poll intitiator* starts by defining a set of time options T and a totally ordered set of participants U. To keep the anonymity of the participants, all communication is done using a DC-Net (Section 2.1) in the Group $(\mathbb{Z}_n, +)$ with $n > \left|U\right|$. Every participant $u \in U$ sends to every other participant $u' \in U \backslash \{u\}$ an equally distributed and independent random number $r^t_{u'} \in \mathbb{Z}_n$ for every time slot $t \in T$. Every participant then obtains a DC-Net-key by calculating:

$$k^t_{u,u'} := \begin{cases} r^t_{u'} & \text{if } u < u' \\ -r^t_{u'} \bmod n & \text{otherwise} \end{cases} \tag{3.1}$$

and finally obtain a key matrix:

$$k_u := \begin{pmatrix} k^{t_1}_{u,u_1} & \cdots & k^{t_{s_T}}_{u,u_1} \\ \vdots & \ddots & \vdots \\ k^{t_1}_{u,u_{s_U-1}} & \cdots & k^{t_{s_T}}_{u,u_{s_U-1}} \end{pmatrix}, \{u_1, ..., u_{s_U}\} = U \backslash \{u\}, s_U = \left|U\right|, s_T = \left|T\right|. \tag{3.2}$$

3.2.2 Voting Process

Every participant u encrypts his vote $\phi_u^t \in \{0,1\}$ for a given time slot t as:

$$d_u^t := \left(\phi_u^t + \sum_{u' \in U \setminus \{u\}} k_{u,u'}^t\right) \bmod n \tag{3.3}$$

and sends the encrypted availability vector $\overrightarrow{d_u} := (d_u^{t_1}, ..., d_u^{t_{s_T}})^T$ to the server.

3.2.3 Evaluation Process

As soon as all participants delivered their encrypted availability vector, all vectors are revealed to each participant. They can then individually compute for each time slot t the number of available participants:

$$\overrightarrow{o_t} = \sum_{u \in U} \overrightarrow{d_u} \bmod n. \tag{3.4}$$

Due to the fact that $k_{u,u'} \equiv -k_{u',u} \pmod n$, $\overrightarrow{o_t}$ is indeed the sum of the availability vectors $\overrightarrow{\phi_u} = (\phi_u^{t_1}, ..., \phi_u^{t_{s_T}})^T$.

The scheduled date can now be calculated using a selection function S which is only allowed to take the number of available persons as input. Thus, it is not possible to, for example, choose a meeting based on who is available and who is not.

3.3 Extension of the Protocol to Protect Against Malicious Participants

When assuming all participants to be semi-honest, one can assume $\phi_u^t \in \{0,1\}$ to be true. However if the participants are allowed to deviate from the protocol, only $\phi \in \mathbb{Z}_n$ can be assumed. Therefore two attacks are possible to manipulate the outcome of the poll:

The first possibility is to send a value $\phi_u^t < 0$ to reduce the indicated number of available participants o_t an thus the probability that time slot t is selected (assuming that a higher number of available participants yield a higher probability that t is selected). As sending a (-1) is at least as hard to detect as sending any number $v < 0$, this kind of attack is referred to as *(-1)-attack*.

The second possibility is to analogously send a value $\phi_u^t > 1$ to increase the probability that time slot t is selected. As sending a 2 is at least as hard to detect as sending any number $v > 1$, this kind of attack is referred to as *(+2)-attack*.

Kellermann [Kel11] proposes an extension to the aforementioned protocol in Section 3.2 to detect this kind of attacks.

3.3.1 Detection of (-1)-Attacks

Instead of executing one DC-Net round per time slot, l DC-Net rounds are executed in parallel per time slot. The vote ϕ_u^t is split among l partial votes $\varphi_u^{(t,0)}, ..., \varphi_u^{(t,l)}$ such that for a uniformly distributed, independent $r \in \mathbb{Z}_l$ yields $\varphi_u^{(t,r)} = \varphi_u^t$ and $\varphi_u^{(t,q)} = 0, q \in \mathbb{Z}_l, r \neq q$. The participants send each partial vote in a seperate DC-Net and thus exchange a key $k_{u,u'}^{(t,i)}$ for every time slot t and every DC-Net round index $i \in \mathbb{Z}_l$ instead of just one key $k_{u,u'}^t$ for every time slot t. The keys are then used to encrypt each partial vote $\varphi_u^{(t,i)}$ as described in Section 3.2.2 to obtain the encrypted vote $d_u^{(t,i)}$. If all participants are honest, then the following two assertions can be made:

$$\forall t \in T, i \in \mathbb{Z}_l : \Big(\sum_{u \in U} d_u^{(t,i)} \bmod n \Big) \in \big\{0, ..., \big|U\big|\big\}, \tag{3.5}$$

$$\forall t \in T : \Big(\sum_{u \in U, i \in \mathbb{Z}_l} d_u^{(t,i)} \bmod n \Big) \in \big\{0, ..., \big|U\big|\big\}. \tag{3.6}$$

If the DC-Net modulus $n \gg 2 \cdot \big|U\big|$ then a (-1)-attack can be detected, if, for example, the attacker sends a (-1) where every other participant sent a 0 for a given DC-Net round δ, since in that case at least one assertion is violated. The DC-Net modulus is required to be much higher than $2 \cdot \big|U\big|$, else it would be possible that one or multiple (-1)-attacks go undetected, because the sum is still within the allowed set $\big\{0, ..., \big|U\big|\big\}$, although all honest participants sent a 0, due to modulus overflow. Another way of detecting a (-1)-attack is, if only one participant u sent a 1 for a given DC-Net round δ and the result is 0. In that case u can publish his vote ϕ_u^δ and show that a (-1)-attack occured.

This protocol is only secure if the server does not cooperate with the malicious participant u_m, as in that case u_m would simply wait for every other participant u' to commit their votes, get the votes from the server, calculates the number of available participants for each time slot and round index and then setting his votes in a way that yields plausible results for every time slot and round index.

This attack can be prevented if all participants commit to a value using a *commitment scheme* (Section 2.2, share the commitment among each other and then send the votes to the server. This, however, requires an additional blocking protocol round, where the participants have to wait for every other participant to submit their commitment.

3.3.2 Detection of (+2)-Attacks

To detect a (+2)-attack every participant sends additionaly to his votes ϕ_u^t a test-vote $\phi_u^{t'}$ such that $\phi_u^t + \phi_u^{t'} = 1$. The test-vote is then processed the same way the normal vote is (see

Section 3.2 and Section 3.3.1). Since every participant has to send a vote of 1 in either of the two votes the following assertion can be made:

$$\forall t \in T : \sum_{u \in U, i \in \mathbb{Z}_l} d_u^{(t,i)} + \sum_{u \in U, i \in \mathbb{Z}_l} d_u^{(t,i)'} \equiv \left|U\right| \pmod{n} \tag{3.7}$$

where $d_u^{(t,i)'}$ is the encryption of the partial vote $\varphi_u^{(t,i)'}$ of $\phi_u^{t'}$. If a (+2)-attack was made, then the assertion is violated, except if the attacker makes a (−1)-attack on the test vote, which can also be detected as described in Section 3.3.1.

3.3.3 Identification of the Attacker

Detecting an attack on the integrity of the voting process is usually not enough, as an attacker can still prevent the vote from happening by always sending invalid inputs, also known as *Denial of Service Attack*. Therefore it is also necessary to find out the identity of the attacker to allow the honest participants to launch countermeasures against the attacker, such as preventing him to participte in subsequent voting processes.

If an attack on the vote has been detected through a violation either of the assertion described in (3.5), (3.6) or (3.7), then the DC-Net rounds where the violation occured are uncovered. Let δ be a DC-Net round where an attack has been detected. Every participant u reveals his $(\left|U\right|-1)$ keys $k_{u,u'}^{\delta}$, $u' \in U \backslash \{u\}$. These keys can then be used to decrypt the single votes of every participant and to check for which participant $u_m \in U$ the vote is not whithin the legal range $\{0, 1\}$.

However, u_m can still cheat at this point. Instead of sending the true key $k_{u_m,u'}^{\delta}$ for any participant $u' \in U \backslash \{u_m\}$, u_m sends a false key $\overline{k_{u_m,u'}^{\delta}} = (k_{u_m,u'}^{\delta} - \nu) \bmod n$, where ν is the illegal value which was originally sent by u_m. In that case it would look like u_m sent the value $\phi_{u_m}^{\delta} = 0$ and u' sent the value $\phi_{u'}^{\delta} = \nu$, but since $\overline{k_{u_m,u'}^{\delta}} \not\equiv k_{u',u_m}^{\delta} \pmod{n}$ one can not decide whether u_m sent the wrong key or u'. To prevent u_m from changing his key, the participants send a commitment for each key using a commitment scheme, when sending their votes to the server. This prevents u_m from retrospectively change his key.

The drawback of this detection is that each participant has to publish their partial vote for a DC-Net round δ where an attack has been detected. If all rounds where an attack occured are uncovered, then an attacker can make all other participants publish their availability patterns, by attacking all DC-Net rounds. It is therefore necessary that only a few of the attacked DC-Net rounds are uncovered. Moreover, if the potential publication of a vote is a concern, the participants may agree on skipping the identification of the attacker. This, however, has to be agreed upon before the protocol execution, as an attacker could simply refuse to publish his keys if his attack has been detected, thus preventing his own identification.

3.4 Extending the Protocol to Increase Usability

Kellermann's protocol so far allows to securely schedule a date, but provides the absolute minimum in terms of usability. To this point, it is not possible to dynamically remove or add any participants, the participants cannot change their vote retrospectively and they are required to choose between only two options. Kellermann [Kel11] proposes extensions to address these issues.

3.4.1 Dynamically Insert Participants

When adding participants to the poll after starting the poll, some participants might already have submitted their vote. In that case the new participant exchanges keys only with the participants, who did not submit a vote yet. The participants who did not submit a vote yet, then use the key of the new participant as well, when submitting their vote.

It has to be noted that this extension only works if at least one participant did not submit his vote yet. If all participants submitted their votes, then no single participant can be added, as there would be no one to exchange keys with.

3.4.2 Dynamically Remove Participants

Dynamically removing participants from the poll is more important than dynamically adding participants, as the sum of the votes can only be computed, if all participants submitted their vote. Therefore a participant might block the protocol by simply never submitting his vote. To remove a participant u_i from the poll, every participant u publishes the key k^{δ}_{u,u_i} he shares with u_i. Using those keys, a substitution vector $\overrightarrow{d^{\delta}_u}$ consisting of the following elements can be calculated:

$$d^{\delta}_u = 0 - \sum_{u \in U \setminus \{u\}} k^{\delta}_{u,u_i} \bmod n. \tag{3.8}$$

The substitution vector is basically an encrypted substitution vote $\phi^{\delta}_{u_i} = 0$ of the removed participant u_i.

If a malicious participant u_m is able to hold back the messages of another participant u', then u_m can convince the other participants to remove u' and thus obtaining all keys necessary to decrypt the inputs of u'. Unfortunately, no solutions to this attack have been proposed by Kellermann.

3.4.3 Additional Options

Most scheduling applications allow for more than two options to be selected. For instance, the most widely used scheduling platform Doodle[3] allows the participants to select one of the three options "yes", "no" or "maybe". In the current protocol a participant u has to vote either "yes" ($\phi_u^t = 1, \phi_u^{t'} = 0$) or "no" ($\phi_u^t = 0, \phi_u^{t'} = 1$). In order to allow for multiple options, instead of sending a vote and a test-vote, a participant simply sends η votes $\phi_u^{(t,0)}, ..., \phi_u^{(t,\eta)}$ where $\phi_u^{(t,\lambda)} = 1$ if u selects option λ and $\phi_u^{(t,\mu)} = 0$ for any $\mu \neq \lambda$. The following assertion holds for any given time slot:

$$\forall t \in T : \sum_{\lambda \in \mathbb{Z}_\eta} \phi_u^{(t,\lambda)} \equiv 1 \pmod{n} \tag{3.9}$$

And thus analogously to the assertion in Eq. (3.7):

$$\forall t \in T : \sum_{u \in U, i \in \mathbb{Z}_l, \lambda \in \mathbb{Z}_\eta} d_u^{(t,i,\lambda)} \equiv \left|U\right| \pmod{n} \tag{3.10}$$

This can be used to detect (+2)-attacks or illegal answers, like selecting two different options simultaneously.

3.4.4 Changing Votes Retrospectively

Often the participant's availability might change over time. Therefore it is often necessary that the participants are able to change their vote after they already submitted it. If a participant u simply submits a new encrypted vote $\overline{d_u^\delta}$, then his vote can simply be inferred by calculating:

$$\sum_{u' \in U \setminus \{u\}} d_{u'}^\delta + d_u^\delta - \left(sum_{u' \in U \setminus \{u\}} d_{u'}^\delta + \overline{d_u^\delta}\right) \bmod n. \tag{3.11}$$

If the server can be trusted and the votes have not already been published, then the server simply replaces the old vote with the new one. If the server cannot be trusted, then u can use the following scheme, without introducing an additional protocol round: Let U_1 be the set of participants who have already submitted their votes to the server excluding u and U_2 the set of participants who have not submitted their votes to the server yet. Before submitting his vote, u exchanges new keys $\overline{k_{u,u'}^\delta}$, $u' \in U_2$ with the participants of U_2 and then calculates the new encrypted vote:

$$\overline{d_u^\delta} = \left(\phi_u^\delta + \sum_{u' \in U_1} k_{u,u'}^\delta + \sum_{u' \in U_2} \overline{k_{u,u'}^\delta}\right) \bmod n. \tag{3.12}$$

[3] `www.doodle.com`

The participants $u' \in U_2$ then encrypt their vote using the following formula:

$$d^{\delta}_{u'} = \left(\phi^{\delta}_{u'} + k^{\delta}_{u',u} + \sum_{u^* \in U \setminus \{u,u'\}} k^{\delta}_{u',u^*}\right) \bmod n. \tag{3.13}$$

This scheme is secure as long as the server does not cooperate with all participants in U_2 to gain knowledge about u's vote.

3.5 Discussion

The protocol presented by Kellermann in [Kel11] provides security against malicious participants and to some extent malicious servers at the cost of increased communication and computation cost, compared to the insecure case. However, the number of blocking communication rounds, where the participants need to wait for the submission of every other participant, remains the same as in insecure protocols, if security against malicious participants cooperating with malicious servers is not a concern. If it is a concern, then the participants need to execute an additional commit round before executing the voting process.

As Kellermann's protocol is a web application, one needs to keep in mind that, at the time of this work, all web applications are vulnerable to getting malicious Javascript code from the server. As all cryptographic operations are done in Javascript, the only means a client has to protect himself from the Javascript leaking his secrets, is to check the Javascript code before executing it, which is an unrealistically high burden on the client. Kellermann suggested that the Javascript code could be checked by one or multiple parties and then appending a digitally signed confirmation to the Javascript code, forming a system based on reputation. The browser can then check those signatures and show to the participant how trustworthy the downloaded Javascript code is. However this is not implemented in any browser currently in use and thus would require a special browser extension.

4 Design

Kellermann's protocol [Kel11] prevents the server and the participants from gaining any information about the input of the other participants, but comes at the cost of increased computation and communication and reduced flexibility. While the increased computation and communication may not be a big concern, the reduced flexibility is. Extending the protocol to dynamically insert and remove participants (Sections 3.4.1 and 3.4.2) is difficult and in the case of removing a participant u, all other participants are required to publish the key they share with u and thus they must wait until everyone else has published the key. Additionally, this feature opens new means for a malicious participant for decrypting the inputs of another participant u' as described in Section 3.4.2.

Another drawback of Kellermann's protocol is that the sum of participants who made a selection for each individual timeslot is revealed, which can leak some information. For example if two organisations A and B want to schedule a meeting, then the members of organisation A could share the selections they made among each other, thus obtaining the sum of each possible selection and each time slot t made by participants of organisation B. This knowledge could leak some information about the internal workings of organisation B, like for example if all participants in organisation B selected "no" for the timeslot t, the members of organisation A could infer that B is holding an important meeting at t. Additionally, the selection function f is very limited, as it can only take the number of votes for each time slot as input. For example, if a company wants to securely schedule a press conference with different media representatives, one usually weights more importance to the attendance of the representatives of media companies with higher reach. However, Kellermann's protocol does not allow to make a weighted sum over the participant's votes.

The security guarantees provided by Kellermann's protocol are quite high. While one cannot always trust a server, assuming a server to be semi-honest is usually sufficient and in fact, as discussed in Section 3.5, it is very difficult to protect against a malicious server sending malicious javascript code, thus one usually has to assume a semi-honest server for web-based applications or at least a server unwilling to take the risk of getting caught sending malicious Javascript code.

In this work, we introduce a protocol based on secure two-party computation between two computing servers S_1 and S_2, also referred to as *backend servers* and an additional third server S, referred to as *frontend server*, which does not take part in the secure two-party computation.

The protocol provides security against malicious participants, a semi-honest frontend server and at least semi-honest backend server, which can be extended by the implementation to give protection against non-colluding backend servers. The protocol maintains almost all the flexibility of insecure protocols, does not reveal the number of participants who voted for each timeslot and allows any selection function that can be represented by a Boolean circuit. As we will show in Section 4.2, all three servers are required to be at least semi-honest in order for the protocol to be secure.

Table 4.1: Comparison between the security and usability features of the different already existing solutions to privacy-preserving scheduling. The parenthesized checkmarks (✓) indicate that a feature is either not fully available, or introduces security concerns.

		Doodle/DFN	Kellermann [Kel11]	Our Work
nondisclosure in presence of	semi-honest participants	×	✓	✓
	malicious participants	×	✓	✓
	semi-honest server(s)	×	✓	✓
	malicious server(s)	×	✓	×
correctness in presence of	semi-honest participants	✓	✓	✓
	malicious participants	×	✓	✓
	semi-honest server(s)	×	✓	✓
	malicious server(s)	×	✓	×
visibility of participants' identity		public	not disclosed	frontend server + initiator
possible selection functions		any	only sum	any predefined
dynamically add participants		✓	(✓)	✓
dynamically remove participants		✓	(✓)	✓
multiple options (e.g. yes, no, maybe)		✓	✓	✓
change votes retrospectively		✓	✓	✓

4.1 The Protocol

In this section we present our protocol, which consists, in a similar fashion to the protocol proposed by Benjamin Kellerman [Kel11] (see also Section 3.2), of the three phases *Poll Generation*, *Voting Process* and *Evaluation Process*. In the poll generation, the data needed for the poll, like the identity of the participants and the selectable time slots, are initialized.

Then, in the voting process, the participants each select their preferred time slots and finally, when all participant have cast their votes, the votes are evaluated to calculate a winning time slot, without revealing the selections made by the participants to any party and finally revealed to the participants.

The participants never communicate directly with the backend servers and always send the selections they made to the frontend server. In order to prevent the frontend server S from gaining any information about the selections made by the participants, S receives a public key k_1 from the first backend server S_1 and a public key k_2 from the second backend server S_2, which S will then forward to the participants participating in any poll. The keys k_1 and k_2 are public keys of any semantically secure asymmetric encryption scheme (2048-bit RSA [RSA78] with the padding described in PKCS #1 Version 1.5[1] in our case).

4.1.1 Poll Generation

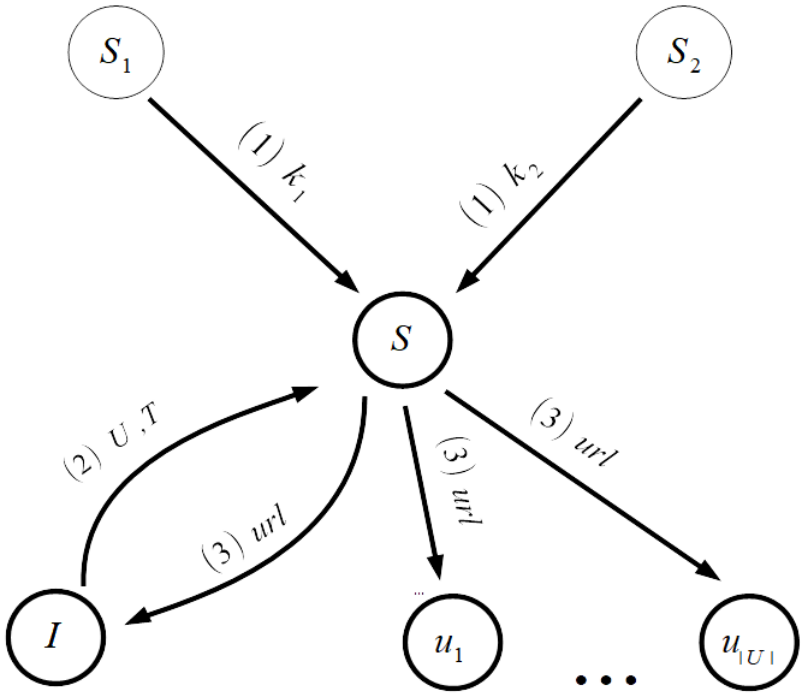

Figure 4.1: (1) S gets the two public keys k_1 and k_2 from the backend servers S_1 and S_2.
(2) The poll Initiator defines the set of participants U and the set of available time slots T and sends them to the frontend server S.
(3) S generates unique urls for each participant and sends them to the corresponding participant.

[1] See RFC 2313: `https://tools.ietf.org/html/rfc2313`

The poll is initiated by a party called the *poll initiator*, who defines a set of time options T and a set of participants U and sends this information to a server S. S then generates $|U|$ partially random and unique urls and sends one of them to each participant.

4.1.2 Voting Process

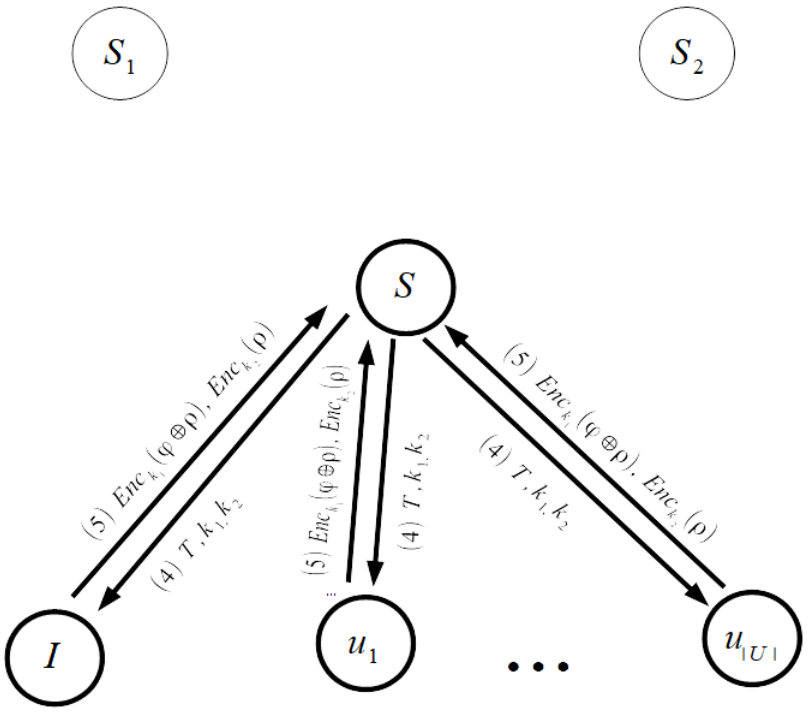

Figure 4.2: (4) S sends the available time slots T and the two public keys k_1 and k_2 to the participants, who request the previously sent url.
(5) The participants send back their encrypted selections $Enc_{k_1}(\varphi \oplus \rho), Enc_{k_2}(\rho)$ back to S

.

The participants who followed the url obtained by S, proceed by selecting one of the options "yes", "no" or "maybe" for every time slot t, but instead of sending their selections $\overrightarrow{\phi} = (\phi_{t_1}, ..., \phi_{t_{|T|}})^T, t_1, ..., t_{|T|} \in T$ directly back to the frontend server S, the participants generate a random vector $\overrightarrow{\rho} = (\rho_{t_1}, ..., \rho_{|T|})^T$, where $\rho_1, ..., \rho_{|T|}$ are random numbers. They then calculate the selection vector $\overrightarrow{d} = (\phi_{t_1} \oplus \rho_{t_1}, ..., \phi_{t_{|T|}} \oplus \rho_{t_{|T|}})^T$ and send the encryptions $Enc_{k_1}(\overrightarrow{d})$ and $Enc_{k_2}(\overrightarrow{\rho})$ back to S, where they are stored. S is a server, that only sends the Javascript code necessary for executing the protocol to the participants and that collects the selections of the participants to send them back to the servers S_1 and S_2 when the poll finishes. As a consequence, S does not possess the corresponding private keys to the public

keys k_1 and k_2 and is therefore unable to learn either $\overrightarrow{d}$ or $\overrightarrow{\rho}$. The semantic security of the asymmetric encryption scheme also prevents the server from gaining information about the input of the participants by perfoming a guessing attack.

4.1.3 Evaluation Process

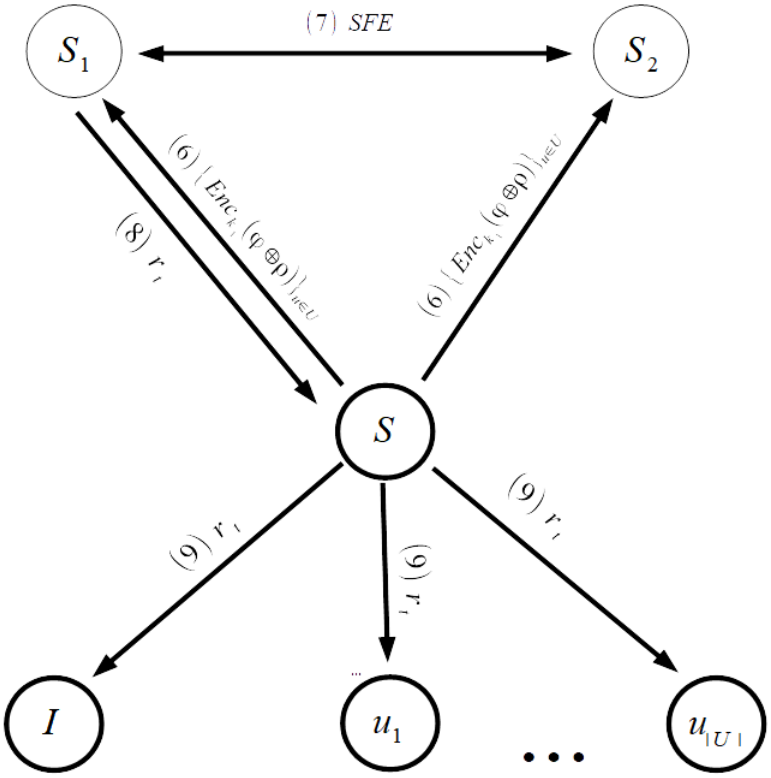

Figure 4.3: (6) S sends the encryptions $\{Enc_{k_1}(\varphi \oplus \rho)\}_{u \in U}$ to S_1 and the encryptions $\{Enc_{k_2}(\rho)\}_{u \in U}$ to S_2
(7) The two backend servers securely calculate the evaluation function.
(8) S_1 sends the result to S_1.
(9) S forwards the result to each participant

When every participant has submitted their input or the poll initiator triggers the end of the poll, S sends the encrypted selection vectors $\overrightarrow{d_u}$ he obtained from every participant $u \in U$ to the server S_1 and the random vectors $\overrightarrow{\rho_u}$ to the server S_2. S_1 possesses the private key to the public key k_1 and S_2 possesses the private key to the public key k_2. S_1 then decrypts the selection vectors $\overrightarrow{d}$ and S_2 decrypts the random vectors $\overrightarrow{\rho}$. S_1 and S_2 then securely evaluate any previously defined selection function f, using either Yao's garbled circuit protocol described in Section 2.5.1 or the GMW protocol described in Section 2.5.2. The result of f,

which is the winning time slot t and the participants who selected "no" for t is then sent back to S, who then reveals it to the participants and the initiator.

4.2 Security

As already mentioned, our protocol provides security against malicious participants and semi-honest servers. S is not able to decrypt the encrypted vectors obtained by the participants, as S does not possess the private keys necessary for decryption. However, if any of the servers are allowed to deviate from the protocol, neither the nondisclosure of the participants' inputs, nor fairness in the selection of the scheduled time can be guaranteed. We will show in the subsequent sections, which are the potential attacks each party can execute on the protocol.

4.2.1 Potential Attacks of the Frontend Server

If the frontend server S was allowed to deviate from the protocol, he could send his own public keys instead in order to be able to decrypt the inputs. While this kind of attack might be prevented by requiring ssl certificates[2] that are issued and verified by a trusted third party, usually called *certificate authority*, this protection is very limited, as the asymmetric encryption is done within the Javascript code sent by the server S. As such, S might, instead of sending Javascript code to encrypt the participant's selections before submitting them, send Javascript code to not encrypt the selections at all.

Another potential attack S can execute on the protocol is, instead of sending the encrypted selections of the participants to S_1 and S_2, send the encrypted inputs of only one participant u and generate fake selections, so that the result returned by S_1 or S_2 combined with the fake selections, reveals information about the selections of u. This attack also has the advantage, that S does not risk getting caught deviating from the protocol, as opposed to sending malicious Javascript code to the participants. Furthermore, it can often be assumed that a semi-honest S stores all data received during the protocol execution. If S becomes malicious at a later time point after the protocol execution - which is a realistic assumption - S could use the stored inputs to perform the aforementioned attack. Note that the case of sending malicious Javascript code, S has to be malicious before sending the Javascript code in order to get the participants' inputs, but in this case, S can be malicious at a later point and still get the participants' inputs.

This attack can be prevented by letting the clients send their encrypted selections directly to the servers S_1 and S_2 instead of sending them to S. However, this requires all three servers to be online at the same time, thus reducing the stability of the protocol, as if only one server is not available, the participants are not able to cast their votes.

[2] See `http://tldp.org/HOWTO/SSL-Certificates-HOWTO/x64.html`

4.2.2 Potential Attacks of the Backend Servers

S_1 does not gain any knowledge about the selections, as S_1 only sees the encrypted selections $\phi_t^u \oplus \rho_t^u$, with $t \in T, u \in U$, ϕ_t^u being a selection and ρ_t^u being a random number, therefore providing information-theoretic security [**Sha49**], as S_1 has no knowledge about ρ_t^u. S_2 conversely has knowledge about the random numbers ρ_t^u, but no knowledge about the selections ϕ_t^u. When executing a secure two-party computation protocol, the inputs of S_1 and S_2 can be securely combined, to obtain the selections ϕ_t^u and evaluate the selection function f. As the secure two-party computation does not reveal any information about the input of the other party, S_1 learns nothing about ρ_t^u and S_2 learns nothing about $\phi_t^u \oplus \rho_t^u$, thus gaining no information about the selections ϕ_t^u. However, if either S_1 or S_2 were to deviate from the protocol, he could gain information about the other party's input and thus gain information about ϕ_t^u, as neither Yao's garbled circuit protocol (Section 2.5.1), nor the GMW protocol (Section 2.5.2) guarantee the nondisclosure of the inputs in the presence of malicious adversaries.

This attack could be prevented by using a secure two-party computation protocol like the one proposed in [LP07], which achieves nondisclosure of the inputs, even in the presence of malicious adversaries. However, this approach does not provide any security if both servers, S_1 and S_2, are allowed to collude in order to gain information about the selections made by the participants.

4.2.3 Potential Attacks of the Participants

As the participants do not see the selections of the other participants, they gain no knowledge about them. As neither server S, S_1 nor S_2 are able see the inputs made by the participants, a malicious participant u_m might send invalid inputs without the servers noticing it. However, as the servers S_1 and S_2 reduce the inputs they get using the modulo operation, the inputs sent by u_m are always parsed as, in this case unspecified, but valid inputs. Therefore, u_m is not able to gain any advantage or break the system by sending invalid inputs. However, if u_m is able to guess the partially random part of the unique url sent to another participant u', then u_m could vote in place of u'. However, if the random part of the url is long enough, i.e. the number of possible unique urls is much larger than $|U|$, the probability of u_m successfully guessing the unique url of any other participant u' is negligible. As S generates only one random and unique url for every participant, the server is able to uniquely identify the participants through that url and thus prevent participants from voting multiple times.

4.2.4 Potential Attacks of the Initiator

It is necessary to assume that the poll initiator u_i initiates the poll in a fair way. If u_i was to define a set of participants U, where u_i is able to control multiple participants $U' \subseteq U$, then u_i would gain an advantage over other participants, by being able to vote $|U'|$ times. Moreover,

if the poll initiator is allowed to add new participants after evaluating the protocol once and execute it again, he could infer information about the other participants' selections, by adding participants he knows the selection of and combining it with the results of the different execution. While the first attack cannot easily be prevented without revealing the identity of each participant to every other participant, the second attack can easily be prevented by allowing only one evaluation per poll.

4.3 Extension of the Protocol

Extending the protocol to allow dynamic insertion and removal of participants is easy, as it requires no cryptographic operations. Dynamically inserting a participant only requires to create a new empty entry on the server and dynamically removing a participant only requires to delete the entry of the participant on the server. Allowing the participants to change their votes retrospectively is also easy, as the participant simply sends a new encrypted selection vector and a new encrypted random vector back to S, where S replaces the old vectors with the new ones. However, as the former selections are encrypted, the participants cannot obtain their previous selections from S. Another extension, which the protocol presented by Kellermann [Kel11] does not support, is to allow some sort of weight to the participants, in order to make it less likely that a time slot is selected if an important participant, i.e. a participant having a high weight, is unable to attend.

Although extending the protocol is easy to implement, it comes at the cost of assuming the poll initiator u_i to be semi-honest, as u_i could use the extensions to gain an advantage in the time selection. In the case of dynamic insertion of participants, u_i could insert additional participants he controls, right before ending the poll, thus allowing him to vote multiple times. In the same fashion u_i could set a high weight for himself, so that his preferred time slot is selected.

5 Implementation

In this chapter we present the implementation of the design introduced in Chapter 4. The implementation is seperated in two parts: the first being the *frontend part* in which the clients initialize a poll and make their selections. The second part, called the *backend part*, consists of securely evaluating the selections made by the clients, in order to find the best time to schedule. The frontend part is executed by one *frontend server* and the backend part is executed by two different *backend servers*. In order to achieve the security guarantees described in Section 4.2 the two backend servers are required to be executed on different machines. However, the frontend server is allowed to share one machine with either one of the two backend servers. Additionally, all communication is secured by Transport Layer Security[1] (TLS), to ensure the integrity of the messages sent between the different entities.

5.1 Frontend

In the frontend part, a client, also known as the *poll initiator* or *initiator* for short, starts a poll by defining a title of the poll and entering his email address. Then the initiator enters the email addresses of the clients who are allowed to participate in the poll (the initiator is implicitly a participant, thus additional input of his email address is not necessary). Note that the poll initiator might input one email address multiple times, but the server will simply remove duplicates in that case, thus preventing a participant to have multiple votes. Finally, the initiator selects the time slots, which the participants might choose from. The number of possible time slots that can be defined is actually limited (we will see later why this is the case), but since the limit is very high (it equals to 735), this is not an issue for (almost all) realistic scenarios.

When the initiator finished creating the poll, the frontend server generates a partially random and unique link for every participant and poll. It is important that the participants do not share the link with any third party, as whoever is in possession of the link, can make or change a selection.

When a participant follows the link he received from the server, the server sends the two different 2048-bit public RSA keys of $S1$ and $S2$ and polls the participant for every time slot to select whether he's either available (option "yes" encoded as integer 0), maybe available (option "maybe" encoded as integer 1) or not available at that time slot (option "no" encoded

[1] See RFC 5246: `https://tools.ietf.org/html/rfc5246`

as integer 3). If the participant does not make a selection for a given time slot, then it will be interpreted as a "no" for that time slot. As soon as the participant submits his inputs by clicking on the button "Done", random numbers are generated for each time slot. The selections made by the participant are then XORed with the generated random numbers and then encrypted with a semantically secure RSA encryption scheme, using the first key previously received from the server. The generated random numbers are also encrypted using the same scheme, but using the second key. Finally, the encryptions are sent back to the frontend server. Note that since the RSA encryption scheme allows to only encrypt messages not exceeding a certain length, the number of time slot that can be selected is affected by that limitation. However, since the selections made by the participants are only 2 bits long, 3 selections can be expressed by one base64 digit, enabling 1 byte to hold 3 selections. Therefore, the maximum number of possible time slots to select from, when using RSA with the padding described in PKCS #1 Version 1.5[2], is $3 \cdot (2048/8 - 11) = 735$, which is far enough for (almost) all possible real world scenarios.

When implementing the extensions suggested in Section 4.3 one can redirect the initiator to a poll admin page, where he can insert or remove participants from the poll, define a weight for every participant (which should be 1 for every participant when initiating the poll), end the poll or input his own selections by executing the procedure described above.

When every participant made their selections, the server will send the encrypted inputs of every participant, minus the participants who did not make a selection, to the two backend servers as described in Section 4.1. When implementing weight support for the participants, the frontend server could generate a random number r_w for each weight w and additionaly send $r_w \oplus w$ to one backend server and r_w to the other. The backend servers then evaluate the circuit as will be described in the following section and send back the result to the frontend server.

The result obtained by the backend servers consist of the winning time slot and an array of bits b_i, where $b_i = 1$ if the participant at index i selected "no" for the winning time slot and $b_i = 0$ in all other cases. The frontend server then publishes the winning time slot to every participant and additionaly the email of the participants who selected "no" for the winning time slot to the poll initiator.

5.2 Backend

The backend part consists of two backend servers that are connected to each other and listening for incoming data of the frontend server. As soon as they receive data from the frontend server, they decrypt the RSA encrypted data of the participants using their RSA private key and then generate a Boolean circuit that will be used to start a secure two-party computation finding the optimal time slot. The optimal time slot is calculated by minimizing

[2] See RFC 2313: `https://tools.ietf.org/html/rfc2313`

the number of participants who selected "no" and in case of a tie, select the time slot in which the fewest participants selected "maybe" among the tying time slots. If there are multiple optimal time slots, then one will be chosen deterministically. However, it is not specified which time slot will be chosen among the optimal ones, but executing the circuit multiple times with the same inputs will always yield the same result.

5.2.1 The ABY Framework

ABY [DSZ15] is a framework that allows to create and execute Boolean and arithmetic circuits [ADI+17] securely. ABY supports arithmetic sharing to execute the arithmetic circuits securely and either Yao's garbled circuit protocol (Section 2.5.1) or the GMW protocol (Section 2.5.2) to execute the Boolean circuit securely. It is also possible to combine the different protocols in one execution, as ABY allows for efficient conversion between the different protocols. We will only focus on the Boolean circuit support of the ABY framework, i.e. Yao's garbled circuit protocol and the GMW protocol in this work.

5.2.2 Boolean Circuit Generation

The selections made by the participants for a given time slot t are encoded as 2 bit values $b_1^{(t,u)} b_2^{(t,u)}$ with "yes" being encoded as 00_2, "maybe" being encoded as 01_2 and "no" being encoded as 11_2. The selections are first decrypted within the circuit using the XOR operation when executing the circuit with Yao's garbled circuit protocol. When executing the circuit with the GMW protocol, this step is ommitted as the encrypted values each server posseses already represent a share of in the GMW protocol. This omits the sharing of the values at the start of the GMW protocol, thus reducing the communication.

Two summations are then made for each time slot t: the first being over the $b_1^{(t,u)}$ values of each participant u and the second being over the $b_2^{(t,u)}$ values, obtaining the number of nos n^t and the number of maybes m^t for each time slot t. Both sums n^t and m^t are concatenated, yielding c^t. The concatenation is an operation equivalent to $c^t = n^t \cdot 2^{n^t} + m^t$, but without requiring any AND gate. For every c^t, an index i^t is created and put as a constant known to both parties into the Boolean circuit, thus obtaining the tuple (c^t, i^t). Then, the c^t values are minimized, obtaining the tuple $(c^{t_{min}}, i^{t_{min}})$, where $c^{t_{min}} \leq c^t$ for each time slot t. The value $c^{t_{min}}$ is finally discarded and only $i^{t_{min}}$ is set as the output node.

Since $c^t = n^t \cdot 2^{n^t} + m^t$ with n^t being the sum over the $b_1^{(t,u)}$ values, and m^t being the sum over the $b_2^{(t,u)}$ values, the described Boolean circuit minimizes n^t and in case of several minimal n^t, minimizes the m^t among them. As $b_1^{(t,u)} = 1$ if and only if the participant u selected "no" for time slot t, n^t is indeed the sum of participants who selected "no". However, m^t does not describe the number of "maybe" selections for time slot t, but the sum of "no" selections and "maybe" selections for t. Nonetheless, as m^t is only minimized when the number of "no" selections is equal for two time slots t_1 and t_2, $m^{t_1} < m^{t_2}$ if and only if the

number of "maybe" selections for t_1 is smaller than the number of "maybe" selections for t_2.

As the number of legal choices is 3, but the number of possible inputs is 4, the backend servers have to prevent a participant from gaining any advantage by sending an illegal input. This case has to be solved within the Boolean circuit, as neither server is able to see and thus check the inputs for validity. However, the only illegal value is 10_2 and sending this value instead of "no" or "yes" yields no advantage in the poll: if a malicious participant u_m wanted a time slot t to be selected, then his best bet would be to send 00_2, as sending 10_2 would increase the n^t value, thus decreasing the likelihood that t is selected. Conversely, if u_m wants t to not be selected, his best bet would be to send 11_2, as sending 10_2 would yield a smaller m^t value and thus increase the likelihood of t being selected in case of tying n^t values. As sending 10_2 yields no advantage, the Boolean circuit treats the value 10_2 as if it was a legal, thus requiring no additional gates. Note that if the "no" selection was encoded as 10_2, u_m would be able to gain an advantage by sending the illegal value 11_2, thus requiring additional gates to prevent this from happening.

When generating the Boolean circuit that will be evaluated by either Yao's garbled circuit protocol (Section 2.5.1) or the GMW protocol (Section 2.5.2) it is necessary to minimize the number of AND gates in order to minimize the communication and computation time. Since the circuit presented above consists mostly of binary adder and the number of AND gates within these adders is linear in the number of bits of the summands, keeping the bit length of the summands to a minimum is mandatory. However, the size of the summands rapidly increase when adding linearly: adding 2 1-bit values yield a 2-bit value, then adding a 2-bit value and a 1-bit value yield a 3-bit value and so on. Therefore, the summations are made in a tree-like fashion, which additionally helps decreasing the depth of the circuit. Consequently, the minimization of the c^t values is also done in a tree-like fashion.

The size of the binary adders can be further decreased by keeping track of the maximum possible value after each operation. For example adding a 2-bit value a and a 1-bit value b yields a 3-bit value c. However, if it is known that a can only hold at most the integer 2, then $a + b$ cannot be greater than 3, thus yielding a 2-bit value c. Since many summations over 1-bit values are made, where the maximum possible value is 1, this track-keeping allows to remove even more unnecessary AND-gates.

When allowing different weights to the participants, the selections made by a participant u with weight w_u count as if w_u participants made the selections. The circuit described above can simply be extended to support this use case. Instead of adding up the values $b_i^{(t,u)}, i \in \{1,2\}$ of each participant u, the values $v_i^{(t,u)}$ are add up, where the $v_i^{(t,u)}$ are calculated as follows: Let ℓ be the bit length of the highest weight w_{max}. Now, the 1-bit value $b_i^{(t,u)}$ is expanded to the bit length ℓ, i.e. the bit $b_i^{(t,u)}$ is copied ℓ times to obtain the ℓ-bit value $\beta_i^{(t,u)}$, which is then used to calculate the biwise AND operation between the $\beta_i^{(t,u)}$ value of participant u and his weight w_u, yielding $v_i^{(t,u)}$.

As supporting a weighted sum requires some overhead that increases with the bit length of the highest weight w_{max}, it is important that the frontend server sends whether the participants are weighted and, if it is the case, the value of w_{max}, as the backend servers only see the encrypted weights.

5.2.3 Executing the Boolean Circuit and Revealing the Result

The Boolean circuit built as described in the previous section is then executed and the result t_r, which is the winning time slot, published to both backend servers. When requested by the protocol, a second Boolean circuit is then built, taking the cleartext result t_r as input and returning a vector of bits $\overrightarrow{b_u}$, where the component $b_u = 1$ if the participant u selected "no" for the winning time slot t_r and $b_u = 0$ in all other cases. This circuit can be built using no AND gates, by simply retrieving the $b_2^{(t_r,u)}$ of every participant u. The backend servers finally send the results t_r and $\overrightarrow{b_u}$ back to the frontend server.

6 Evaluation

In this chapter, we will show that our implementation is efficient, even when scheduling meetings between thousands of participants. We will first calculate an estimate about the maximum number of AND gates that should be in the circuit, depending on the problem size $p_s = (n_u, n_t)$, where n_u denotes the number of participants and n_t denotes the number of time slots. We will then show, that the number of AND gates our circuits have for different p_s is smaller that our estimate, when the size-optimized building blocks of the ABY framework are used. We finally show the runtimes and communication sizes when executing the circuits on different p_s.

6.1 Estimate on the Number of AND Gates in the Circuit Constructions

Let n_u be the number of participants and n_t the number of time slots. When constructing a circuit as described in Section 5.2.2 for Yao's garbled circuit protocol (Yao circuit), one can easily see that the number of adders is exactly $2 \cdot (n_u - 1)$ for each time slot t, as we make a summation over each bit value $b_1^{(t,u)}$ of the selections made by the participants u at timeslot t and another summation over all $b_2^{(t,u)}$ values of timeslot t. As we are making these summations for each time slot t and no other summation elsewhere, the total number of adders in the circuit is $n_{add} = 2 \cdot (n_u - 1) \cdot n_t$.

In the ABY framework [DSZ15], each adder a has exactly ℓ_r AND gates [KSS09], where ℓ_r is the minimum number of bits needed to store the result $r = x + y$ and x and y being the two inputs of the adder a. ℓ_r can be calculated as $\ell_r = \lceil \log_2(x) \rceil + \lceil \log_2(y) \rceil + 1)$. Since we are only making summations over 1-bit values, each sum s_i^t over the participants selections $b_i^{(t,u)}, i \in \{1, 2\}$ is $s_i^t \leq n_u$ and thus, the bitlength $\ell_{s_i^t}$ of s_i^t is $l_{s_i^t} \leq \lceil \log_2(n_u) \rceil$. Therefore, the number of AND gates g_{add} in each adder is $g_{add} \leq \lceil \log_2(n_u) \rceil$. More precisely, as the summation is made in a tree-like fashion and every input at the leaf is exactly 1 bit long, the add gates at level 1 have 1 AND gate, at level 2, 2 AND gates and so on. Lets denote an adder at level j of the tree as ADD^j and the number of AND gates within ADD^j as $|ADD^j|$. We can easily see that $j \in \{1, ..., \lceil \log_2(n_u) \rceil\}$ and that there are exactly $\left\lceil \frac{n_u}{2^j} \right\rceil$ adder at level j of the tree. The number of AND gates in a summation over all $b_i^{(t,u)}$ values of a time slot t can then

be calculated as:

$$g_i^t = \sum_{j=1}^{\lceil \log_2(n_u) \rceil} \left\lceil \frac{n_u}{2^j} \right\rceil \cdot |ADD^j| = n_u \cdot \sum_{j=1}^{\lceil \log_2(n_u) \rceil} \left\lceil \frac{j}{2^j} \right\rceil < 2 \cdot n_u \cdot \left(1 - \frac{1}{n_u}\right) = 2 \cdot (n_u - 1). \quad (6.1)$$

Finally, the total number of AND gates within the totality of adders in the circuit can be calculated as:

$$g_{SUM} = n_t \cdot (g_1^t + g_2^t) < 4 \cdot n_t \cdot (n_u - 1). \quad (6.2)$$

When the sums s_i^t for each time slot t are calculated, the s_1^t and s_2^t sums are concatenated, which requires no AND gate and yields a value c^t of bitlength $\ell_{c^t} = \ell_{s_1^t} + \ell s_2^t = 2 \cdot \lceil \log_2(n_u) \rceil$, where $\ell_{s_1^t}$ and $\ell_{s_2^t}$ are the bitlength of s_1^t and s_2^t. The c^t values are minimized to find the minimal index i, which consists of n_t minimizing with index operations, all of which in turn consist of one comparison operation and two multiplex operations (one for the value and one for the index). This is an equivalent construction to the efficient minimum circuit presented in [KSS09]. Now let i_{MAX} be the maximum index value (i.e. $\forall i : i \leq i_{MAX}$) and c_{MAX} be the biggest number among all c^t values ($\forall c^t : c^t \leq c_{MAX}$). We define $\ell_{i_{MAX}} = \lceil \log_2(i_{MAX}) \rceil$ as being the bitlength of i_{MAX} and $\ell_{c_{MAX}} = \lceil \log_2(c_{MAX}) \rceil$ as being the bitlength of c_{MAX}. Note that $\ell_{c^t} = \ell_{c_{MAX}}$ for every c^t value as the c^t are not known during circuit evaluation and therefore their bitlength is required to be able to hold every possible value, notably c_{MAX}. The number of AND gates in the efficient minimum circuit can be deduced from [KSS09] as

$$g_{MIN} < 2 \cdot \ell_{c_{MAX}} \cdot (n_t - 1) + n_t + 1. \quad (6.3)$$

The estimate in the number of AND gates within the circuit can thus be calculated, using Equations (6.2) and (6.3), as follows:

$$\begin{aligned} g_{AND} = g_{SUM} + g_{MIN} &< 4 \cdot n_t \cdot (n_u - 1) + 2 \cdot \ell_{c_{MAX}} \cdot (n_t - 1) + n_t + 1 \\ &= 4 \cdot (n_t \cdot (n_u - 1) + \lceil \log_2(n_u) \rceil \cdot (n_t - 1)) + n_t + 1. \end{aligned} \quad (6.4)$$

When estimating the number of AND gates in our circuit that supports weighted sums (weighted Yao circuit), we can simply use the fact that the number of AND gates in a circuit containing one ℓ-bit adder is equal to the number of AND gates in a circuit containing ℓ 1-bit adders. In a similar fashion, we can see that the number of AND gates in a minimizing with index circuit over ℓ-bit values, is smaller or equal to the number of AND gates in ℓ minimizing with index circuits over 1-bit values. Let w_max be the maximum weight in the weighted sum circuit. We can therefore estimate the number of AND gates in the weighted summations part as $g_{wsum} = \lceil \log_2(w_{MAX}) \rceil \cdot g_{SUM}$ and in the minimizing with index part as $g_{wmin} \leq \lceil \log_2(w_{MAX}) \rceil \cdot g_{MIN}$. As mentioned in Section 5.2.2 we also need $\ell_{w_{MAX}}$ additional AND gates for every input node, thus obtaining the following estimate in the number of AND gates:

$$g_{WAND} = g_{wsum} + g_{wmin} + n_u \cdot n_t \cdot \lceil \log_2(w_{MAX}) \rceil = \lceil \log_2(w_{MAX}] \cdot (g_{AND} + n_u \cdot n_t) \quad (6.5)$$

6.2 Circuit Size Comparision

In this section we show the circuit sizes for different problem sizes p_s, when building the Yao circuit and the weighted Yao circuit. We can see in the Tables 6.1 and 6.2, that both circuit constructions undercut the estimates we made in the Equations (6.4) and (6.5), when constructing for Yao. However, as we can see in Tables **??** and **??**, when building for the GMW protocol, our construction exceeds our estimation by a factor of about 2. This is due to the fact, that the ABY building blocks optimize for depth instead of size when building for the GMW protocol, thus obtainig larger, but flatter circuits.

Table 6.1: Comparison between our estimate of AND gates for different promblem sizes, and the real cicuit size when building the Yao circuit. T denotes the number of time slots and U denotes the number of participants

	Yao			Estimates from Equation (6.4)		
	$T = 10$	$T = 20$	$T = 30$	$T = 10$	$T = 20$	$T = 30$
$U = 10$	480	999	1509	515	1045	1575
$U = 50$	2112	4271	6421	2187	4397	6607
$U = 100$	4128	8307	12477	4223	8473	12723
$U = 500$	20160	40379	60589	20295	40625	60955
$U = 1000$	40176	80415	120645	40331	80701	121071
$U = 5000$	200224	400523	600813	200439	400929	601419
$U = 10000$	400240	800559	1200869	400475	801005	1201535

Table 6.2: Comparison between our estimate of AND gates for different promblem sizes and $w_{MAX} = 65535$, with the real cicuit size when building the weighted Yao circuit. t denotes the number of time slots and U denotes the number of participants

	Weighted Yao			Estimates from Equation (6.5)		
	$T = 10$	$T = 20$	$T = 30$	$T = 10$	$T = 20$	$T = 30$
$U = 10$	7016	14135	21245	9840	19920	30000
$U = 50$	33648	67407	101157	42992	86352	129712
$U = 100$	66944	134003	201053	83568	167568	251568
$U = 500$	330656	661435	992205	404720	810000	1215280
$U = 1000$	660912	1321951	1982981	805296	1611216	2417136

Table 6.3: Comparison between our estimate of AND gates for different promblem sizes, and the real cicuit size when building the circuit for the GMW protocol. T denotes the number of time slots and U denotes the number of participants

	GMW			Estimates from Equation (6.4)		
	$T = 10$	$T = 20$	$T = 30$	$T = 10$	$T = 20$	$T = 30$
$U = 10$	711	1441	2171	515	1045	1575
$U = 50$	4010	8050	12090	2187	4397	6607
$U = 100$	8204	16444	24684	4223	8473	12723
$U = 500$	43023	86093	129163	20295	40625	60955
$U = 1000$	86377	172807	259237	40331	80701	121071
$U = 5000$	430839	861749	1292659	200439	400929	601419
$U = 10000$	862233	1724543	2586853	400475	801005	1201535

Table 6.4: Comparison between our estimate of AND gates for different promblem sizes, and the real cicuit size when building the circuit, supporting weighted summations for the GMW protocol. T denotes the number of time slots and U denotes the number of participants. The maximum wight supported by the circuits is always $w_{MAX} = 65535$.

	Weighted GMW			Estimates from Equation (6.5)		
	$T = 10$	$T = 20$	$T = 30$	$T = 10$	$T = 20$	$T = 30$
$U = 10$	19308	38728	58148	9840	19920	30000
$U = 50$	100096	200316	300536	42992	86352	129712
$U = 100$	201370	402870	604370	83568	167568	251568
$U = 500$	1001398	2002938	3004478	404720	810000	1215280
$U = 1000$	2004232	4008612	6012992	805296	1611216	2417136

6.3 Runtime Measurements

In this section we present the runtimes and communications when evaluating the circuits built as described in Section 5.2.2. We show the runtimes when executing the circuits generated for Yao's Garbled Circuit Protocol and the GMW protocol plus the runtimes when executing the weighted sum circuits for both protocols.

Table 6.5: Runtimes in milliseconds for Yao's protocol and the GMW protocol. U denotes the number of participants and T the number of available time slots. Measured on a simulated 1 Gbit LAN.

	Yao		GMW	
	setup	online	setup	online
$T = 10$				
$U = 10$	1.1	0.6	1.2	0.8
$U = 50$	3.1	1.9	2.2	1.7
$U = 100$	4.2	3.0	3.6	3.3
$U = 500$	18.9	15.3	11.2	18.3
$U = 1000$	37.9	30.3	20.9	41.9
$U = 5000$	184.2	152.8	74.7	247.9
$U = 10000$	349.7	303.0	141.6	511.8
$T = 20$				
$U = 10$	2.1	0.9	1.5	1.4
$U = 50$	5.0	3.5	3.4	5.6
$U = 100$	8.1	7.1	5.7	8.1
$U = 500$	37.5	30.3	20.3	43.8
$U = 1000$	72.3	60.9	33.3	88.8
$U = 5000$	347.2	304.1	138.1	507.8
$U = 10000$	697.5	625.7	271.7	1057.7
$T = 30$				
$U = 10$	2.2	1.2	1.6	1.7
$U = 50$	7.0	5.2	4.8	8.6
$U = 100$	13.3	10.9	7.5	14.4
$U = 500$	54.7	46.5	27.5	65.8
$U = 1000$	108.6	90.8	50.0	135.8
$U = 5000$	634.5	469.4	200.4	768.1
$U = 10000$	1369.7	1022.6	410.6	1627.4

Table 6.6: Runtimes in milliseconds for Yao's protocol and the GMW protocol on a circuit supporting weighted sums up until a weight of $w_{MAX} = 65535$. U denotes the number of participants and T the number of available time slots. Measured on a simulated 1 Gbit LAN

	Yao		GMW	
	setup	online	setup	online
$T = 10$				
$U = 10$	3.7	2.6	8.2	6.6
$U = 50$	15.2	11.5	24.0	37.3
$U = 100$	29.7	22.9	40.5	89.4
$U = 500$	147.5	121.5	156.7	427.6
$U = 1000$	282.9	250.9	300.2	898.4
$T = 20$				
$U = 10$	6.4	5.1	11.6	12.0
$U = 50$	29.5	23.3	42.4	75.5
$U = 100$	56.0	47.5	71.3	149.1
$U = 500$	278.2	234.8	307.8	881.7
$U = 1000$	578.1	459.4	610.2	1873.4
$T = 30$				
$U = 10$	9.4	7.5	13.7	17.1
$U = 50$	43.5	36.3	58.9	122.1
$U = 100$	83.2	68.9	96.0	242.2
$U = 500$	417.2	335.8	450.6	1347.8
$U = 1000$	930.6	696.1	885.1	2814.7

We can see that Yao's protocol is much slower than the GMW protocol in the setup phase, with increasing differences in increasing circuit sizes. However, Yao is faster than the GMW protocol in the online phase and the proportional difference increases with the circuit size. For the weighted circuits, the GMW protocol and Yao's protocol are about equal in the setup phase. For the online phase, Yao's protocol is much faster than the GMW protocol, with a much bigger proportional difference than in the non-weighted circuit.

Table 6.7: Communication size in KB for Yao's protocol and the GMW protocol. For the GMW protocol the number of protocol rounds is additionally given, as for Yao's protocol the number of protocol rounds is constant and equals to 3

	Yao		GMW		
	setup	online	setup	online	#rounds
$T = 10$					
$U = 10$	19.7	14.0	44.3	0.9	23
$U = 50$	82.1	47.1	152.3	2.7	30
$U = 100$	161.1	94.0	292.3	4.8	32
$U = 500$	788.1	470.0	1380.3	22.0	41
$U = 1000$	1569.6	940.0	2736.5	43.2	42
$U = 5000$	7821.3	4699.8	13509.5	211.6	49
$U = 10000$	15634.1	9399.5	26994.8	422.3	51
$T = 20$					
$U = 10$	39.9	28.2	72.3	1.4	28
$U = 50$	165.5	94.0	288.3	4.8	36
$U = 100$	323.7	188.0	548.3	9.0	38
$U = 500$	1575.9	940.0	2736.5	43.2	48
$U = 1000$	3139.1	1879.9	5448.7	85.6	49
$U = 5000$	15642.9	9399.5	26978.8	422.1	56
$U = 10000$	31268.5	18798.9	53945.4	843.5	58
$T = 30$					
$U = 10$	57.2	42.4	92.3	1.8	28
$U = 50$	248.7	141.0	412.3	6.8	36
$U = 100$	484.0	282.0	808.3	13.0	38
$U = 500$	2363.5	1410.0	4084.6	64.3	48
$U = 1000$	4708.4	2819.9	8149.0	127.8	49
$U = 5000$	23464.2	14099.2	40448.1	632.6	56
$U = 10000$	46904.7	28198.3	80895.9	1264.6	58

Table 6.8: Communication size in KB for Yao's protocol and the GMW protocol on a Boolean circuit construction allowing for weighted sums up until $w_{MAX} = 65535$. For the GMW protocol the number of protocol rounds is additionally given. For Yao's protocol the number is a constant 3

	Weighted Yao		Wighted GMW		
	setup	online	setup	online	#rounds
$T = 10$					
$U = 10$	227.0	17.0	652.3	10.8	51
$U = 50$	1081.6	84.6	3176.5	50.4	58
$U = 100$	2150.1	169.2	6344.8	99.9	61
$U = 500$	10615.3	846.0	31355.2	490.7	69
$U = 1000$	21218.0	1691.9	62698.2	980.5	73
$T = 20$					
$U = 10$	457.9	114.7	1260.3	20.5	59
$U = 50$	2150.6	131.6	6320.8	99.5	66
$U = 100$	4284.5	263.2	12649.4	198.5	69
$U = 500$	21239.5	1316.0	62658.2	980.0	77
$U = 1000$	42537.8	2631.9	125352.2	1959.4	81
$T = 30$					
$U = 10$	677.2	66.2	1864.4	30.1	59
$U = 50$	3221.3	178.6	9453.1	148.6	66
$U = 100$	6438.7	357.2	18946.0	297.0	69
$U = 500$	31897.9	1785.9	93969.2	1469.2	77
$U = 1000$	64226.1	3571.8	187994.2	2938.2	81

We can see that in the setup phase, Yao's protcol fares better than the GMW protocol. However, In the online phase, the GMW protocol requires almost no communation in comparison to Yao's protocol. However, the communication overall, i.e. the sum of the communication in the setup phase and the online phase is about equal between Yao's protocol and the GMW protocol. This is due to the fact, that the circuits built for the GMW protocol contain more AND gates (about twice as much), than the circuits built for Yao's protocol.

7 Conclusion

We provided a web-based solution to privacy-preserving scheduling that allows a wider range of possible privacy-preserving selection functions than comparable solutions presented so far, while maintaining efficiency in communication and computation for almost all real world scenarios, taking only a few milliseconds in the most extreme cases of thousands of participants. Furthermore, our solution is easily extensible to mimic all the features provided by non-privacy-preserving scheduling applications. The security guarantees provided by our protocol are high for almost all common use cases.

Future work include on the one hand further optimizing the protocol. The participants could for example send two arithmetic shares, thus allowing the summations to be made through efficient additive homomorphic operations. Then the conversion functions in the ABY framework can be used to convert into a Boolean circuit, which allows for efficient evaluation of the minimizing function. On the other hand, the security guarantees provided by our protocol could be further improved. One improvement includes using a secure two-party protocol that guarantees nondisclosure in the presence of malicious opponents, thus lifting the semi-honest assumption on the backend servers. Another improvement would be to combine our protocol with the protocol proposed by Kellermann [Kel11]. This combination would allow the participants to additionally encrypt their inputs in a similar fashion done in Kellermanns protocol, ebfore proceeding with our protocol. In this case the frontend server would generate the keys and send them to the participants instead of letting the participants generate their own keys and exchange them with every other participant. This would provide security, even in the presence of colluding backend servers, as in this case, they would only be able to gain knowledge about the sum of the participants' selections in each time slot. However, the resulting protocol would not be able to allow every evaluation function to be defined, notably the weighted sums evaluation would not be possible in this protocol.

List of Figures

List of Tables

List of Abbreviations

Bibliography

[ADI+17] B. APPLEBAUM, I. DAMGÅRD, Y. ISHAI, M. NIELSEN, L. ZICHRON. **"Secure Arithmetic Computation with Constant Computational Overhead"**. In: *Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Cham: Springer International Publishing, 2017, pp. 223–254. URL: https://doi.org/10.1007/978-3-319-63688-7_8 (cit. on p. 26).

[ALSZ13] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. **"More Efficient Oblivious Transfer and Extensions for Faster Secure Computation"**. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. Berlin, Germany: ACM, 2013, pp. 535–548. URL: http://doi.acm.org/10.1145/2508859.2516738 (cit. on p. 5).

[BHKR13] M. BELLARE, V. T. HOANG, S. KEELVEEDHI, P. ROGAWAY. **"Efficient Garbling from a Fixed-Key Blockcipher"**. In: *Proceedings of the 2013 IEEE Symposium on Security and Privacy*. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 478–492. URL: http://dx.doi.org/10.1109/SP.2013.39 (cit. on p. 6).

[Cha88] D. CHAUM. **"The dining cryptographers problem: Unconditional sender and recipient untraceability"**. In: *Journal of Cryptology* 1.1 (01/1988), pp. 65–75. URL: https://doi.org/10.1007/BF00206326 (cit. on p. 3).

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *Network and Distributed System Security Symposium (NDSS'15)*. 02/2015 (cit. on pp. 26, 29).

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **"Extending Oblivious Transfers Efficiently"**. In: *Advances in Cryptology (CRYPTO'03)*. Vol. 2729. LNCS. Springer, 2003, pp. 145–161 (cit. on p. 5).

[Kel11] B. KELLERMANN. "Mehrseitig sichere Web 2.0-Terminabstimmung". PhD thesis. Technische Universität Dresden, 2011 (cit. on pp. 8, 10, 13, 15 sqq., 23, 38).

[KS08] V. KOLESNIKOV, T. SCHNEIDER. **"Improved Garbled Circuit: Free XOR Gates and Application"**. In: *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II*. Springer Berlin Heidelberg, 2008, pp. 486–498. URL: https://doi.org/10.1007/978-3-540-70583-3_40 (cit. on p. 6).

[KSS09] V. KOLESNIKOV, A.-R. SADEGHI, T. SCHNEIDER. **"Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima"**. In: *Proceedings of the 8th International Conference on Cryptology and Network Security*. CANS '09. Kanazawa, Japan: Springer-Verlag, 2009, pp. 1–20. URL: https://doi.org/10.1007/978-3-642-10433-6_1 (cit. on pp. 29 sq.).

[LP07] Y. LINDELL, B. PINKAS. **"An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries"**. In: *Advances in Cryptology - EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings*. Ed. by Moni Naor. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 52–78. URL: https://doi.org/10.1007/978-3-540-72540-4_4 (cit. on p. 22).

[NPS99] M. NAOR, B. PINKAS, R. SUMNER. **"Privacy Preserving Auctions and Mechanism Design"**. In: *Proceedings of the 1st ACM Conference on Electronic Commerce*. EC '99. Denver, Colorado, USA: ACM, 1999, pp. 129–139. URL: http://doi.acm.org/10.1145/336992.337028 (cit. on p. 6).

[RSA78] R. L. RIVEST, A. SHAMIR, L. ADLEMAN. **"A Method for Obtaining Digital Signatures and Public-key Cryptosystems"**. In: *Commun. ACM* 21.2 (02/1978), pp. 120–126. URL: http://doi.acm.org/10.1145/359340.359342 (cit. on p. 18).

[Vol99] H. VOLLMER. *Introduction to Circuit Complexity: A Uniform Approach*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999 (cit. on p. 4).

[Yao86] A. C.-C. YAO. **"How to Generate and Exchange Secrets"**. In: *Foundations of Computer Science (FOCS'86)*. IEEE, 1986, pp. 162–167 (cit. on p. 6).

A Appendix

A.1 Here might be an appendix section

And some text.