Web Application for Privacy-preserving Scheduling using Secure Computation

Ágnes Kiss, Oliver Schick and Thomas Schneider
TU Darmstadt, Darmstadt, Germany
{kiss, schneider}@encrypto.cs.tu-darmstadt.de, oliver.schick92@gmail.com

Keywords: Event Scheduling, Secure Two-party Computation, Web Application, Efficient Implementation.

Abstract: Event scheduling applications such as Doodle allow for very limited privacy protection. Even if the participants are anonymous, their inputs are revealed to the poll administrator and the application server. There exist privacy-enhanced scheduling services (e.g., Kellermann and Böhme, CSE'09), but they require heavy computation and communication on the client's side, leak information to the participants or poll administrator, and allow only for a restricted scheduling functionality. In this work, we present a privacy-preserving scheduling system based on secure two-party computation, that allows to schedule meetings between a large number of participants efficiently, without requiring any participant to reveal its availability pattern or other sensitive information to any other participant, server, or even the poll administrator. The protocol allows for various functional extensions and requires the client to perform very little work when securely submitting its inputs. Our protocol is secure against semi-honest non-colluding servers and malicious participants.

1 INTRODUCTION

Arranging a meeting between multiple persons is a highly recurring task that can take a lot of time, especially when many people and organisations are involved. While most bigger companies already provide internal solutions to this problem, these solutions only apply when scheduling internal meetings. There are multiple online solutions such as Doodle (Doodle, 2018) or DFN (DFN, 2018), but several concerns regarding privacy arise when using these.

In some cases, all participants can see the selections made by other participants, which might leak information of the availability patterns. Leaking such availability patterns might result for example in burglars breaking into the user's apartment. Moreover, the last users submitting their votes can lie about their availability in order to make their preferred date more likely to be chosen.

Problems can arise even when the (potentially anonymized) selections of the participants can only be seen by the poll administrators. Firstly, a poll administrator cannot always be found, especially when arranging a meeting among several organizations. Secondly, the administrator has the opportunity to cheat by selecting a preferred date instead of the winning time slot based on the number of participants.

Our Contributions. Our main contribution is a privacy-preserving web-based system that allows to schedule a meeting between any number of participants using a selection function f that is used to select the winning time slot based on the availability of the participants. We show that using generic solutions for secure two-party computation, our practical implementation is more efficient and more private than previous protocols specifically designed for the poll functionality. The protocol guarantees that even malicious participants gain neither advantage in the selection of the scheduled time nor information about the selections made by other participants. Moreover, the poll administrator learns no information except the output of the computation. Our presented system requires that all three participating servers are semi-honest and the two backend servers do not collude in order to guarantee correctness of the execution and nondisclosure of the selections of the participants.

2 RELATED WORK

In this section, we review already existing scheduling applications based on privacy-enhancing technologies. We summarize and compare their most important features in Tab. 1.

2.1 Doodle and DFN

The most widely used web-based scheduling application, Doodle (Doodle, 2018), and the scheduling service provided by the German National Research and Education Network (Deutsches Forschungsnetz) (DFN, 2018), do not provide any protection against the service provider or poll administrator.

Doodle is the currently most popular web-based solution to scheduling between multiple parties. A poll is initiated by a poll administrator, who defines the available time slots from which the participants of the poll select. By default, the input of every participant is visible to everyone and no access control is provided, hence anyone in possession of the link can see the selections made by all participants. Doodle also offers hidden polls, where only the poll administrator can see the inputs of every other participant.

The DFN provides an alternative scheduling application but promises to use the data sent to them only for the scheduling. They guarantee to delete all data they receive and thus require the poll administrator to input a termination date, when all data related to the poll is deleted. However, one has no means to check if DFN deletes the data and does not use it elsewhere. Therefore, DFN also provides security by trust only.

2.2 Dudle

Kellermann and Böhme presented a protocol in (Kellermann and Böhme, 2009), (Kellermann, 2010), (Kellermann, 2011), that allows participants to schedule a meeting without revealing their availabilities to the server or other participants. Their protocol is used in the scheduling system Dudle (Dresden, 2018).

The protocol consists of three phases. In the first phase (*poll generation*), a poll administrator initiates the poll and every participant communicates with all other participants to obtain keys for the later phases. In the next phase (*voting*), the participants encrypt their votes for all time slots and send these ciphertexts to the server. In the final phase (*evaluation*), each participant can calculate the sum of the available participants for each time slot and define the winning time slot. The authors argue that these sums do not reveal significant information about the individual votes. However, in some cases (e.g., with few participants) this might provide limited privacy guarantees.

Kellermann provided an extension against malicious participants providing invalid inputs (Kellermann, 2011). The extension allows to detect malicious behavior and identify the malicious participant.

The protocol proposed by Kellermann and Böhme allows to securely schedule a date, but provides the minimum in terms of usability. Extensions to dynamically insert and remove participants, to allow additional (e.g., yes-no-maybe) options, and to change votes retrospectively have been proposed, but these extensions have serious limitations (Kellermann and Böhme, 2009). For instance, removing a participant and changing votes requires additional rounds of interaction with the participants, which is unrealistic, since users may go offline after submitting their votes.

2.3 Other Solutions for Privacy-preserving Scheduling

Solutions to a different type of privacy-preserving scheduling task exist, where the problem is to find a common available time slot given the availability pattern of all participants. This problem can be solved using private set intersection or homomorphic encryption as in (Bilogrevic et al., 2011), (Demmler et al., 2014), (Huang et al., 2011). In our work, we seek a solution for a different functionality: we aim to find the schedule where *most participants* are available but do not require that *all* participants are available.

Secure computation of surveys has been implemented in (Feigenbaum et al., 2004) where different statistics of the input can be calculated. Though this solution can be adapted to scheduling, and its design is similar to ours, the computing servers are required to stay online on the Internet during voting which yields a larger attack surface. In the next section, we introduce our design for privacy-preserving scheduling that uses public-key encryption and where the computing servers only participate for computing the final result of the scheduling.

3 PRELIMINARIES

We introduce the primitives used in this paper.

3.1 Oblivious Transfer

A 1-out-of-n oblivious transfer (OT) protocol allows a sender to transfer one out of n messages $m_1, \ldots, m_n$ selected by the receiver using a selection $s \in \{1, \ldots, n\}$, without the receiver learning any information about the other messages besides m_s and the sender learning selection s. OT protocols rely on public key cryptography, but can be sped up using OT extension (Ishai et al., 2003), (Asharov et al., 2013), i.e., by calculating a symmetric security parameter number of so-called base OTs and then computing a large amount of OTs using faster symmetric key cryptography. We utilize the OT protocol of (Naor and Pinkas,

Table 1: Comparison between the security and usability features of the different solutions for privacy-preserving scheduling. The parenthesized checkmarks (✓) indicate that a feature is not fully available, or introduces security or usability concerns.

Property		Doodle (Doodle, 2018)/ DFN (DFN, 2018)	Dudle (Dresden, 2018), (Kellermann and Böhme, 2009)	Our Work
nondisclosure in presence of	semi-honest participants	×	✓	✓
	malicious participants	×	✓	✓
	semi-honest server(s)	×	✓	✓
correctness in presence of	semi-honest participants	✓	✓	✓
	malicious participants	×	✓	✓
	semi-honest server(s)	✓	✓	✓
visibility of participants' identity		public	not disclosed	frontend server
possible selection functions		any	only highest sum	any
dynamically add participants		✓	(✓)	✓
dynamically remove participants		✓	(✓)	✓
multiple options (e.g., yes-no-maybe)		✓	(✓)	✓
change votes retrospectively		✓	(✓)	✓

2001) for base OTs and the semi-honest secure OT extension of (Asharov et al., 2013).

3.2 Secure Two-party Computation

Secure two-party computation allows two parties to jointly evaluate a function without revealing any information about their inputs except for the output of the function. For secure computation, we consider the *semi-honest* (passive) adversary model, where the adversary is assumed to follow the protocol and cannot learn any information about the secret inputs.

Yao's Garbled Circuits Protocol. Yao's garbled circuits protocol (Yao, 1986) is a protocol that allows two parties to jointly compute a function while none of the inputs to the function is revealed to the other party, where the function to be computed is described as a Boolean circuit. There are several optimizations to the original protocol such as free-XOR (Kolesnikov and Schneider, 2008), fixed-key AES garbling (Bellare et al., 2013), and half-gates (Zahur et al., 2015).

The GMW Protocol. The GMW protocol (Goldreich et al., 1987) is a protocol for secure two-party computation which also requires the function to be represented as a Boolean circuit. In the GMW protocol every value v of each wire is shared among the two parties using a secret sharing scheme such that $v = v_1 \oplus v_2$ for two random-looking shares v_1, v_2 where one party holds v_1 and the other holds v_2. XOR gates can be evaluated locally, while AND-gates require using 1-out-of-4 OT or so-called multiplication triples (Beaver, 1991).

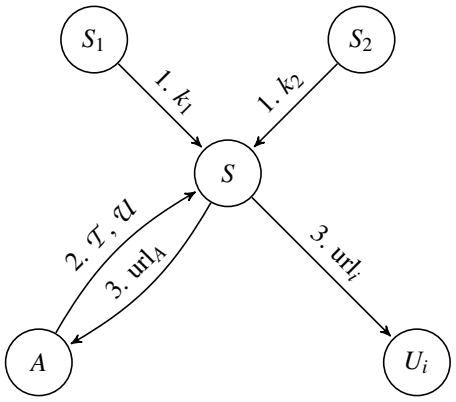
4 WEB APPLICATION FOR PRIVACY-PRESERVING SCHEDULING

We present our privacy-preserving scheduling system that is based on secure two-party computation and allows for calculating any scheduling function without compromising the privacy of the participants' inputs. Our aim is to reveal no additional information besides the output of the poll, which can be specified: e.g., in some cases, it might be enough to learn the winning time slot, while organizing an event might require additionally the number of expected participants (i.e., number of participants that voted for the time slot). Moreover, we support yes-no-maybe votes and also weighted sums, which enables the poll administrator to weight certain participants over others. For example, if a company wants to securely schedule a press conference with different media representatives, one could give higher weights to the representatives of media companies with higher reach.

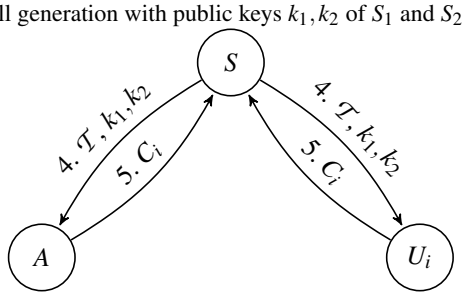
We discuss a too simple alternative in §4.1. In §4.2, we present our privacy-preserving scheduling system, while in §4.3 and §4.4, we describe its security guarantees and functional extensions.

4.1 Too Simple Solutions

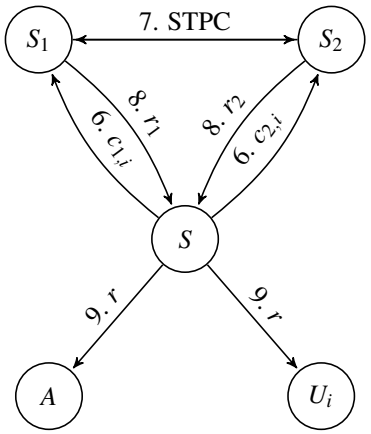
We note that in a straightforward solution, each participant can additively secret share their vote and the backend servers can add these shares together and send the result to the frontend server, who learns the number of votes for each option. Additively homomorphic encryption could also be used with a single



(a) Poll generation with public keys k_1, k_2 of S_1 and S_2, resp.



(b) Voting with $C_i = \{c_{1,i}, c_{2,i}\}$, where $c_{1,i} = Enc_{k_1}(\overrightarrow{\phi_i} \oplus \overrightarrow{\rho_i})$, and $c_{2,i} = Enc_{k_2}(\overrightarrow{\rho_i})$ with $\overrightarrow{\phi_i}$ denoting the selections and $\overrightarrow{\rho_i}$ the random vectors used for secret sharing.



(c) Evaluation with secure two-party computation (STPC) with resulting secret shares of the output, i.e., $r = r_1 \oplus r_2$.

Figure 1: Privacy-preserving scheduling with time slots $\mathcal{T}$, participants $U_i \in \mathcal{U}$, $i \in \{1, \ldots, |\mathcal{U}|\}$, poll administrator A, frontend server S, and computing backend servers S_1 and S_2.

computation server who sums up the votes under encryption and sends this to the frontend server who decrypts the result. Using these techniques does not hide the sum of the votes for each time slot, i.e., this protocol provides the same privacy guarantees as the protocol of Kellermann and Böhme (Kellermann and Böhme, 2009). Our aim is to design an application that reveals *only* the final output of the computation.

4.2 Our System

In our system, we use secure two-party computation (STPC) that allows two parties to jointly evaluate a function on their private inputs without revealing anything except for the (potentially secret shared) output of the functionality. We use Yao's garbled circuit protocol (Yao, 1986) and the GMW protocol (Goldreich et al., 1987) with security against semi-honest adversaries and state-of-the-art optimizations as implemented in the ABY framework (Demmler et al., 2015). We make use of two non-colluding *backend servers* S_1 and S_2, who perform the STPC and an additional third *frontend server* S, who is responsible for transferring messages between the clients and the backend servers. We note that it is possible to modify our design to work with only two non-colluding servers by letting either $S = S_1$ or $S = S_2$, since the information known by S and one of the backend servers does not reveal anything about the inputs of the users besides the output of the computation.

To prevent the frontend server S from gaining any information about the selections made by the participants, S receives public keys k_1 and k_2 from backend server S_1 and S_2, respectively, which S then forwards to the participants of the poll. The keys k_1 and k_2 are public keys of any semantically secure public-key encryption scheme, e.g., based on RSA or ElGamal.

We present our protocol which consists, similarly to the protocol of (Kellermann and Böhme, 2009), of three phases *poll generation*, *voting* and *evaluation* as depicted in Fig. 1, with the respective steps numbered both in the figure and in the description below.

Poll Generation (Fig. 1a): The data needed for the poll (i.e., identity of the participants, available time slots) are initialized.

1. S receives public keys k_1 and k_2 from backend servers S_1 and S_2, respectively.
2. The poll is initiated by the *poll administrator* A, who defines the sets of time slots $\mathcal{T}$ and partici-

pants $\mathcal{U}$ and sends them to the frontend server S.

3. S generates and sends a unique URL to each participant $U_i \in \mathcal{U}$.

Voting (Fig. 1b): The participants submit their availability.

4. S sends the available time slots $t_1, \ldots, t_{|\mathcal{T}|} \in \mathcal{T}$ and the two public keys k_1 and k_2 to the participants, who request the previously sent URL. The participants $U_i \in \mathcal{U}$, who followed the URL proceed by selecting one of the options "yes", "no" (or "maybe") for every time slot $t_j \in \mathcal{T}$, but instead of sending their selections $\overrightarrow{\phi_i} = (\phi_{1,i}, \ldots, \phi_{|\mathcal{T}|,i})$ directly back to the frontend server S, they generate a vector of random numbers $\overrightarrow{\rho_i} = (\rho_{1,i}, \ldots, \rho_{|\mathcal{T}|,i})$.
5. The participants $U_i \in \mathcal{U}$ calculate the share of their selection vector $\overrightarrow{d_i} = (\phi_{1,i} \oplus \rho_{1,i}, \ldots, \phi_{|\mathcal{T}|,i} \oplus \rho_{|\mathcal{T}|,i})$ and send ciphertexts $Enc_{k_1}(\overrightarrow{d_i})$ and $Enc_{k_2}(\overrightarrow{\rho_i})$ to S, where they are stored. Due to the semantic security of the encryption scheme, S learns no information about $\overrightarrow{d_i}$ or $\overrightarrow{\rho_i}$.

Evaluation (Fig. 1c): A winning time slot is securely calculated and revealed, without revealing the selections of the participants to any party.

6. When every participant submitted its input or the deadline the poll administrator set is reached, S sends encryptions $\{Enc_{k_1}(\overrightarrow{d})\}_{u \in U}$ to S_1 and encryptions $\{Enc_{k_2}(\overrightarrow{\rho})\}_{u \in U}$ to S_2.
7. S_1 (resp. S_2) possesses the private key corresponding to the public key k_1 (resp. k_2), and decrypts $\{Enc_{k_1}(\overrightarrow{d})\}_{u \in U}$ (resp. $\{Enc_{k_2}(\overrightarrow{\rho})\}_{u \in U}$). S_1 and S_2 then securely evaluate the previously defined selection function f, using secure two-party computation (STPC).
8. As result of the STPC, S_1 and S_2 obtain shares r_1 and r_2 of $f(\overrightarrow{\rho})$, i.e., the winning time slot r, which are sent to S.
9. S recombines and sends the result $r = r_1 \oplus r_2$ to each participant and the poll administrator A.

4.3 Security Guarantees

Our protocol provides security against malicious participants and semi-honest servers with the assumption that the backend servers do not collude. We describe the security guarantees against each party below.

Frontend Server S. A semi-honest frontend server S is not able to decrypt the encrypted vectors obtained by the participants, due to the semantic security of the public-key encryption scheme. Assuming a semi-honest frontend server is realistic, since a malicious frontend server S could always, instead of sending Javascript code to encrypt the participant's selections before submitting them, send malicious code (e.g., to not encrypt the selections at all or exfiltrate the votes to another machine).

Backend Servers S_1 and S_2. The non-colluding backend servers can only decrypt the shares encrypted using their public keys and therefore can only learn their own shares in clear. Security follows from the underlying secure two-party computation protocol (STPC), and therefore semi-honest servers learn no information about the selections.

Participants $U_i \in \mathcal{U}$. Participants send only a single message, which (given that it can be correctly decrypted) corresponds to a valid input: e.g., their input is regarded as a single bit, even if they provide larger numbers, and for polls with yes-no-maybe options a simple circuit has to be added to check that the non-allowed combination of the four possible 2-bit values does not occur. Since a malicious participant can thus only send valid inputs to the frontend server, our protocol is secure against malicious participants.

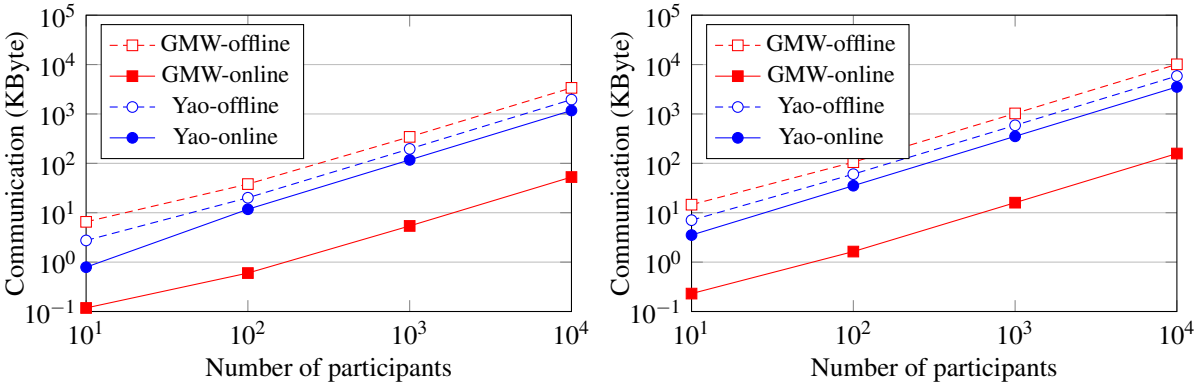
Theorem 1. *The security of our outsourcing scheme follows from the proof of (Kamara et al., 2011) that turns an N-party secure multi-party computation (SMPC) scheme into an outsourcing scheme to N non-colluding servers. We can thus turn the ABY secure two-party computation framework into an outsourcing scheme with $N = 2$ non-colluding servers and the security follows from that of the protocols implemented in ABY (Demmler et al., 2015).*

4.4 Functional Extensions

Our protocol maintains almost all the flexibility of insecure solutions such as Doodle or DFN, does not reveal the number of participants who voted for each time slot, and allows any selection function that can be represented by a Boolean circuit.

Dynamic insertion and removal of participants is trivial, and requires no cryptographic operations. Allowing the participants to change their votes retrospectively is also straightforward: the participant can send a new encrypted selection vector and an encrypted random vector to S, who replaces the old vectors with the new ones. However, as the former selection is encrypted with the public keys of S_1 and S_2, the participant cannot obtain it from S.

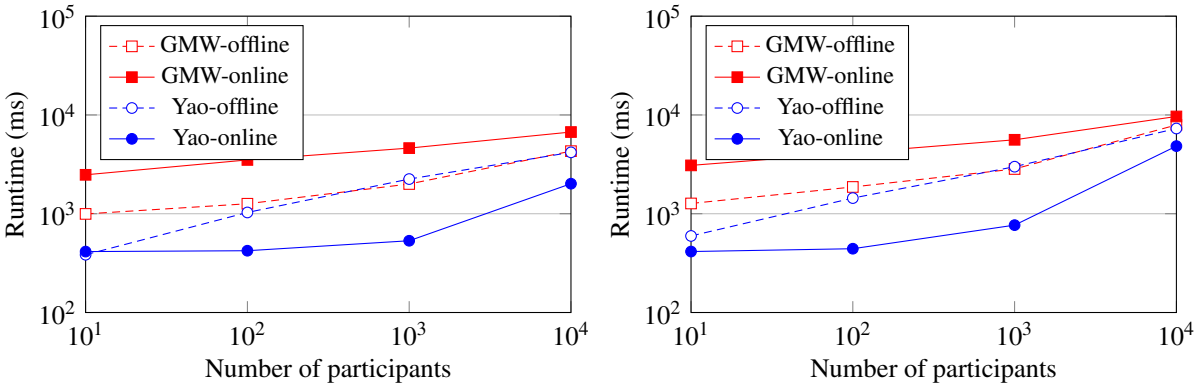
Another extension is to assign weights to the participants, in order to make the votes of more important participants count more. This extension is important for scenarios where there is an obvious hierarchy between the participants: it can be an internal meeting



(a) 10 time slots

(b) 30 time slots

Figure 2: Backend communication in KBytes. Both axes are in logscale.



(a) 10 time slots

(b) 30 time slots

Figure 3: Backend runtime in milliseconds. Both axes are in logscale.

of a large company or a press conference with journalists of newspapers with varying popularity. The extension is easy to include since it only requires some more computation on the backend servers: the poll administrator specifies the weights in the poll generation phase and sends them to S along with $\mathcal{U}$. In the beginning of the evaluation, S forwards these weights to S_1 and S_2, who include these in the computation.

5 IMPLEMENTATION AND EVALUATION

We implemented our event scheduling system as a web application and use the ABY framework for secure two-party computation. Our open-source implementation can be found at https://encrypto.de/code/scheduling. We present the empirical evaluation of our system with yes-no-maybe scheduling. All communication is secured by Transport Layer Security (TLS) (Tim Dierks, 2008), to ensure the integrity of the messages sent between the entities.

5.1 Implementation Details

Frontend. When the poll administrator creates the poll, the frontend server S generates and sends a random and unique link for every participant. When a participant follows the link, the server sends the two different 2048-bit RSA public keys of S_1 and S_2, and the participant can submit its secret shared and encrypted availability. With one RSA encryption we can encrypt multiple messages, e.g., for 2-bit inputs (i.e., yes-no-maybe options), 3 selections can be expressed by one base64 digit, enabling 1 byte to hold 3 selections, which means that the number of possible time slots we can encrypt using one RSA encryption with the padding described in PKCS #1 Version 1.5 (Kaliski, 1998), is $3 \cdot (2048/8 - 11) = 735$, which is enough for (almost) all real-world applications. Our implementation supports yes-no-maybe options (2-bit

inputs), but can also be used with only yes-no options (1-bit inputs). When all participants made their selections, the server sends the encrypted inputs of the participants to the two backend servers.

Backend. In order to achieve the security guarantees described in §4, the two backend servers are required to be executed on two different machines that are connected via a WAN network. As soon as they receive data from the frontend server, they decrypt the encrypted data using their private keys and generate and securely evaluate the Boolean circuit finding the optimal time slot with the fewest unavailable participants using secure two-party computation.

We use the ABY (Demmler et al., 2015) framework for secure two-party computation (STPC) that allows to create and evaluate Boolean circuits securely. The decrypted selections of the participants are known as XOR shares by S_1 and S_2. We can directly use these as inputs to the STPC protocol using a Boolean circuit that is minimized for the number of AND gates in order to minimize the communication and computation time. A summation is made for each time slot t, yielding the number of unavailable participants n^t, of which the minimal value is selected and the index of the column t_r is returned as XOR shares to the backend servers, who forward them to the frontend server. The frontend server recombines the result and publishes t_r or sends it to the poll administrator.

We support weighted sum as a selection function, i.e., the poll administrator is able to assign pre-defined weights ($1 \leq w \leq 2^8 - 1$) to all participants such that the vote of more important participants count more.

5.2 Experimental Evaluation

We show that our implementation is efficient, even when scheduling meetings between thousands of participants. The only cryptographic operation besides the backend servers' computation step (and efficient RSA decryptions) is the client's RSA encryption in Javascript. As mentioned before, encrypting all the shares costs one RSA encryption in realistic scenarios (with up to 735 available time slots). However, the time taken for the encryption depends on the browser and operating system. For instance, it takes around 450 ms to perform an RSA encryption in Google Chrome 62.0.3202 in Windows 10 on a standard PC.

Backend Communication. Secure two-party computation requires communication between the computing parties. In Figs. 2a and 2b, we depict the communication required by the protocols in the setup and online phases for 10 and 30 time slots, respectively.

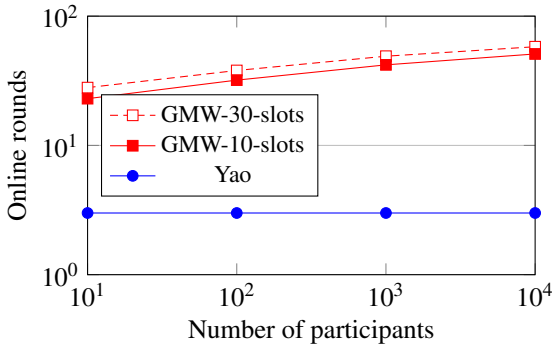


Figure 4: Backend online rounds. Both axes are in logscale.

In the setup phase, any computation independent of the private inputs (i.e., the two shares possessed by the backend servers) is performed. We observe that the GMW protocol requires less online communication, while Yao's protocol requires an oblivious transfer per input wire of the underlying Boolean circuit.

Backend Runtime (WAN). For our runtime measurements we use two machines with an Intel Core i7-4790 CPU at 3.6 GHz and 32 GB of RAM that support Intel's AES-NI for fast AES encryption and decryption. Our benchmarks are run in a simulated WAN setting with 100 Mbit/s throughput and 100 ms latency. Reported runtimes are averages of 10 executions. In Figs. 3a and 3b, we depict the performance of our protocols for 10 and 30 time slots, respectively. Yao's protocol is more efficient than the GMW protocol due to the lower number of communication rounds, though its offline phase becomes less efficient with increasing circuit sizes due to the large amount of communication. The total runtime of our privacy-preserving scheduling system with realistic parameters is at most 10 seconds. We note that the participants do not expect immediate response – everyone has to participate before the result is computed.

Backend Online Rounds. As opposed to Yao's protocol, which has a constant number of communication rounds, the GMW protocol requires d rounds of communication, where d denotes the depth of the underlying Boolean circuit. We depict the online round complexities of the yes-no-maybe protocol in Figs 4.

6 CONCLUSION

In this paper, we presented a privacy-preserving, web-based scheduling application based on secure two-party computation that provides the same flexibility

as existing web-based scheduling applications such as Doodle or DFN and additionally preserves privacy. Our system guarantees security against malicious participants and semi-honest non-colluding servers, and we have shown that it is truly practical even for a large number of participants and time slots.

ACKNOWLEDGEMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within the National Research Center for Applied Cybersecurity CRISP, and by the DFG as part of project E4 within the CRC 1119 CROSSING and project A.1 within the RTG 2050 "Privacy and Trust for Mobile Users".

REFERENCES

Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. (2013). More efficient oblivious transfer and extensions for faster secure computation. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*, pages 535–548. ACM.

Beaver, D. (1991). Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer.

Bellare, M., Hoang, V. T., Keelveedhi, S., and Rogaway, P. (2013). Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P'13)*, pages 478–492. IEEE Computer Society.

Bilogrevic, I., Jadliwala, M., Kumar, P., Walia, S. S., Hubaux, J.-P., Aad, I., and Niemi, V. (2011). Meetings through the cloud: Privacy-preserving scheduling on mobile devices. *Journal of Systems and Software*, 84(11):1910–1927.

Demmler, D., Schneider, T., and Zohner, M. (2014). Adhoc secure two-party computation on mobile devices using hardware tokens. In *USENIX Security Symposium*, pages 893–908. USENIX Association.

Demmler, D., Schneider, T., and Zohner, M. (2015). ABY - a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society.

DFN (2018). Scheduler. https://terminplaner.dfn.de/.

Doodle (2018). Get together with doodle. https://www.doodle.com.

Dresden, T. U. (2018). Dudle. https://dudle.inf.tu-dresden.de/.

Feigenbaum, J., Pinkas, B., Ryger, R., and Saint-Jean, F. (2004). Secure computation of surveys. In *EU Workshop on Secure Multiparty Protocols*.

Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM.

Huang, Y., Chapman, P., and Evans, D. (2011). Privacy-preserving applications on smartphones. In *USENIX Workshop on Hot Topics in Security (HotSec'11)*. USENIX Association.

Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. (2003). Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer.

Kaliski, B. (1998). RFC 2313: PKCS #1: RSA encryption version 1.5. https://tools.ietf.org/html/rfc2313.

Kamara, S., Mohassel, P., and Raykova, M. (2011). Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011:272. https://ia.cr/2011/272.

Kellermann, B. (2010). Open research questions of privacy-enhanced event scheduling. In *Open Research Problems in Network Security (iNetSec'10)*, LNCS, pages 9–19. Springer.

Kellermann, B. (2011). Privacy-enhanced web-based event scheduling with majority agreement. In *Information Security Conference (SEC'11)*, volume 354 of *IFIP Advances in Information and Communication Technology*, pages 235–246. Springer.

Kellermann, B. and Böhme, R. (2009). Privacy-enhanced event scheduling. In *Conference on Computational Science and Engineering (CSE'09)*, pages 52–59. IEEE Computer Society.

Kolesnikov, V. and Schneider, T. (2008). Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer.

Naor, M. and Pinkas, B. (2001). Efficient oblivious transfer protocols. In *Symposium on Discrete Algorithms (SODA'01)*, pages 448–457. ACM/SIAM.

Tim Dierks, E. R. (2008). RFC 5246: The transport layer security (TLS) protocol version 1.2. https://tools.ietf.org/html/rfc5246.

Yao, A. C.-C. (1986). How to generate and exchange secrets. In *Annual Symposium on Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE Computer Society.

Zahur, S., Rosulek, M., and Evans, D. (2015). Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology – EUROCRYPT'15*, volume 9057 of *LNCS*, pages 220–250. Springer.