Privacy-Enhanced Web-Based Event Scheduling with Majority Agreement

Benjamin Kellermann

Technische Universität Dresden, Faculty of Computer Science,
D-01062 Dresden, Germany
`Benjamin.Kellermann@tu-dresden.de`

Abstract. Applications which help users to schedule events are becoming more and more important. A drawback of most existing applications is, that the preferences of all participants are revealed to the others. Previously proposed privacy-friendly solutions could only schedule meetings if all participants were available at the same time slot.
We propose a new scheme, which overcomes this limitation, i.e., the meeting can be scheduled at the time slot, where just the majority of participants is available. Dudle (`http://dudle.inf.tu-dresden.de`), a web-application which implements the protocol is presented. We measured its performance in order to show that the protocol is practical and feasible.

Keywords: event scheduling, electronic voting, superposed sending, anonymity, privacy-enhanced application design

1 Introduction

There are numerous Web 2.0 applications (e.g., `doodle.com`, `moreganize.ch`, . . .), which allow users to create polls. The most important use case of these applications is to schedule events. They all have in common, that they disclose detailed *availability patterns* of their users. These patterns contain sensible information in at least two respects. First, one is able to read information *directly* out of such a pattern ("Does my boss work after 3pm?"). Secondly, due to the fact, that these patterns contain much entropy, one is able to connect them with other information sources easily to read *indirect* information and re-identify participants who would otherwise remain pseudonymous ("This availability pattern looks like Peters who goes to lunch everyday at 11:30."). All existing applications for event scheduling allow some privacy settings, but none of them tries to overcome the need of complete trust in the application server and the poll initiator.

A privacy-friendly and verifiable solution for scheduling a single event, which reveals only the sum of available participants at every time slot was proposed [1]. Despite being efficient enough for a Web 2.0-implementation, it has the drawback that unanimous agreement is required for resisting internal attacks.

In this paper we present, a better solution for the problem of internal attackers. Therefore we discuss two extensions to the original protocol described in [1]. These extensions prevent internal attacks and allow majority agreement simultaneously.

2 Related Work

There are several approaches dealing with event scheduling. It can be seen as distributed constraint satisfaction / optimization problem (DCSP/DCOP) or as an instance of electronic voting.

Many algorithms for DCSP [2–4] and DCOP [5–7] exist and measurements of the information leakage were done [8, 9]. With the help of these algorithms, complex scheduling problems may be solved (e. g., scheduling of many events, where different subsets of the participants participate in each event with constraints about place, travel time etc.). However, all DCOP algorithms share the problem that they are complex in terms of message exchanges even for basic scenarios. To solve the problem of message exchanges, agents are used, which send and receive the messages. As users do not want to setup such an agent at some server, they have to run it locally and have to be online at the same time. Therefore, the DCOP approach is too complex in terms of message exchanges to be implemented in a web application and a simpler solution for the simpler problem of scheduling a *single* meeting would be appropriate.

There is a lot of literature about electronic voting [10–16], some of them lead to implementations [17–19]. 4 observations occur, when event scheduling is done with e-voting: (1) Only the sum of available participants, not the single availabilities, can be used to schedule the time slot. (2) The short ballot assumption [20] does not hold as the availability pattern contains much entropy and there are few voters (participants). (3) All participants have to be voting officials to minimize trust in other entities. (4) Coercion resistance is not necessarily required.

To overcome the short ballot assumption, one has to apply an e-voting scheme for every time slot, which is scheduled. However, if the computational complexity of the scheme depends on the number of time slots and the number of participants (voting officials, assumption 3), an application will exceed the possibilities of current browsers. To illustrate this, we implemented a performance measurement for a discrete exponentiation modulo a 786 bit long integer using different libraries and browsers. We measured 3 of the 5 investigated libraries. The remaining two were too slow to be mentioned here. If, e.g., a scheme would need only one asymmetric operation per time slot and participant and a concrete poll would ask for 20 time slots for a meeting with 5 participants, the scheme would need *only* for the asymmetric operations about $5 \cdot 20 \cdot 0.78\,\mathrm{s} = 78\,\mathrm{s}$, using the BigInteger Library of Wu on a Firefox 3.6, IE 8 would be blocked for at least 4 minutes. Having a low computation complexity is even more important considering small mobile devices like smartphones (cp. the last column of Table 1).

Besides e-voting, there are two specific schemes, which try to solve event scheduling in a privacy-friendly way [1, 24], both offering unanimous agreement only. The main idea of [1] is to use DC-Net [25] to calculate the sum of available participants to the time slots. Therefore, a dedicated DC-Net round is executed for every nominated time slot. Each participant sends an encrypted 1 in the specific round if he is available at the time slot, and 0 otherwise. Through the homomorphism in the DC-Net, the sum of the votes is calculated. The result of one DC-Net round is the number of available participants at this time slot.

Table 1. Execution time for an exponentiation modulo a 786 bit long integer in JavaScript with different libraries. The first 5 columns were measured on an Intel Pentium 4 Duo with 2.8 GHz, 2 GB RAM running Windows XP SP3. The last one was measured on a Motorola Milestone running Android.

	IE 8.0.6001	Firefox 3.6.12	Safari 5.0.3	Opera 10.63	Chrome 8.0.552	Android Firefox 4.0b2
Wu [21]	2.80 s	0.78 s	0.91 s	0.31 s	0.10 s	7.87 s
Baird [22]	5.05 s	0.43 s	0.15 s	0.18 s	0.16 s	5.10 s
Shapiro [23]	21.47 s	2.12 s	1.18 s	0.99 s	0.61 s	(crashes)

	t_0	t_1	t_2	t_3
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	-1	-1	-1	1
$\sum$	0	1	-1	2

(a) attacking with -1

	t_0	t_1	t_2	t_3
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	0	0	0	2
$\sum$	1	2	0	3

(b) attacking with $+2$

Fig. 1. Different ways to attack a poll if Mallory wants t_3 to win. The plain text votes ($v_{u,t}$) are displayed.

The main problem of using the DC-Net is that a participant may send values different from 0 or 1. I.e., a participant may send values below 0 to lower the chance for a specific time slot of being chosen, and values above 1 to increase it. An example of this is illustrated in Fig. 1. The tables show two polls with 3 participants and 4 time slots ($t_0, \ldots, t_3$). The unencrypted votes are shown inside each table ($v_{u,t} \in \{0, 1\}$). Mallory manipulates the poll in a way that time slot t_3 results in the largest sum. Because of the anonymization of all messages through the DC-Net, Mallory's attack is hidden to Alice and Bob.

In the following, we will discuss how to extend the original protocol [1] in a way that sending values different from 0 or 1 can be prevented without the need of unanimous agreement (which was used there to overcome the problem).

3 Preventing (−1)-Attacks

The main idea for preventing (-1)-attacks is the fact that the results of a DC-Net at some time slot must not be lower than 0. I.e., in Fig. 1a the attack cannot be detected at time slot t_0 and t_1, but it is visible at time slot t_2, where the result is -1. If all participants are honest, the result of every time slot should be a value between 0 and the number of all participants.

Instead of using one dedicated DC-Net round for every time slot, we use several simultaneously running DC-Net rounds. Let I be the number of simultaneously running DC-Net rounds (we will see later in this section, that I should be chosen in some dependence of the number of participants). Every participant u splits each vote $v_{u,t} \in \{0,1\}$ into I partial votes $\bar{v}_{u,t,0}, \ldots, \bar{v}_{u,t,I-1}$ such that:

1. An index $j \in \mathbb{Z}_I$ for one partial vote is chosen randomly and kept secret.
2. The partial vote with index j ($\bar{v}_{u,t,j}$) is equal to the actual vote $v_{u,t}$.
3. The remaining $I-1$ partial votes are equal to 0.

Let T be the set of time slots, U the set of participants of the poll, σ_t the number of available participants at time slot t, $\bar{k}_{u,u',t,i}$ the DC-Key between two voters, and $\bar{d}_{u,t,i}$ the encrypted vote within a DC-Net round ($\bar{d}_{u,t,i} = \bar{v}_{u,t,i} + \sum_{u' \in U, u' \neq u} \bar{k}_{u,u',t,i}$). If all participants are honest, the following properties result from the construction:

1. For all time slots $t \in T$ and partial vote indices $i \in \mathbb{Z}_I$, the sum of all partial votes of all participants is an element between 0 and the number of participants ($\forall t, i : \sum_{u \in U} \bar{v}_{u,t,i} = \sum_{u \in U} \bar{d}_{u,t,i} \in \{0, \ldots, |U|\}$).
2. At one time slot, the sum of all partial votes of all participants is the sum of all available participants at this time slot ($\sigma_t = \sum_{u \in U, i \in \mathbb{Z}_I} \bar{v}_{u,t,i}$).

In Fig. 1, we have seen that an attacker has to guess at which time slot a honest participant will send a 1. With the proposed extension it is not sufficient for the attacker Mallory to guess the availability of another participant, she further has to guess at which partial vote the actual vote was sent. This is difficult as long as the chosen partial vote index is random, kept secret, and the number of partial votes per time slot $\mathbb{Z}_I$ is sufficiently high.

Fig. 2 shows an example of the vote vector splitting with $I = 3$, where Mallory tries to send a -1 at t_1 ($v_{u_m,t_1} = -1$). The left table shows the original protocol. There the attack would remain undetected as all elements of the result vector are in the allowed range. The right table shows the same vote vectors split into several vectors. There, for time slot t_1 Alice has chosen the first table ($i = 0$) for her vote ($\bar{v}_{u_a,t_1,0} = v_{u_a,t_1} = 1$) and Bob has chosen the second one. As Mallory has chosen the third table ($i = 2$) her attack can be detected.

Note, that the enrypted vote vectors must not be published until the last participant has sent his vector. Otherwise, if the server would cooperate with Mallory, she may wait until all other participants sent their vote and then calculate the preliminary result before casting her own vote. This allows her to state -1s where other participants sent a 1. As already stated in the original protocol, this restriction can be relaxed with one additional communication phase in which all voters have to commit to their votes [1].

3.1 Verifiability

When verifying that no attack occurred, we can distinguish two cases: (1) In the simpler case, one DC-Net round results -1 ($\sum_{u \in U} \bar{d}_{u,t,i} = -1$). This case occurred in the example in Fig. 2. (2) In a more complex scenario, some participant sent a 1 in some DC-Net round, but the result equals 0 due to a (-1)-attack.

	t_0	t_1	t_2	t_3
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	0	−1	0	1
$\sum$	1	1	0	2

	t_0	t_1	t_2	t_3	
Alice	0	1	0	0	$i = 0$
Bob	0	0	0	0	
Mallory	0	0	0	0	
$\sum$	0	1	0	0	
Alice	0	0	0	0	$i = 1$
Bob	0	1	0	1	
Mallory	0	0	0	1	
$\sum$	0	1	0	2	
Alice	0	0	0	0	$i = 2$
Bob	1	0	0	0	
Mallory	0	−1	0	0	
$\sum$	1	−1	0	0	
$\sum$	1	1	0	2	

Fig. 2. Split the votes into several partial votes. While Mallory's attack remains undetected in the left table, Alice and Bob are able to detect it in the right one.

This may also occur in the original protocol. Bob for example can detect that Mallory has cheated at t_0 in Fig. 1a. However, to prove that Mallory has cheated at t_0, Bob has to give up his privacy. In such a case, Bob can decide for himself what is worth more, his privacy or to unmask Mallory.

We first discuss the case where privacy is the most valuable good, and we discuss situations later where participants are willing to give up their privacy for unmasking attackers. For reasons of simplicity, we write all calculations which are done in the DC-Net without the modulo operations. We consider only 1 attacker.

Without Privacy-Loss Checking the correctness of a poll can be expressed by a function. It takes all messages sent within the poll and returns true if the poll was correct, and false otherwise $\mathcal{V} : \mathbb{Z}^{|U|\times|T|\times I} \to \{\text{true}, \text{false}\}$. Let $\bar{\boldsymbol{d}} \in \mathbb{Z}^{|U|\times|T|\times I}$ be the 3-dimensional array of all DC-Net messages containing elements of $\bar{d}_{u,t,i}$ for a DC-Net message from participant u at time slot t and partial vote index i. The function which checks the correctness of the poll is defined as:

$$\mathcal{V}(\bar{\boldsymbol{d}}) = \begin{cases} \text{true} & \text{if } \forall t \in T, i \in \mathbb{Z}_I : \sum_{u \in U} \bar{d}_{u,t,i} \in \{0, \dots, |U|\} \\ \text{false} & \text{otherwise.} \end{cases} \tag{1}$$

We assume one attacker Mallory (u_m) who tries to send a −1 at time slot t. With an increasing amount of participants, voting for t, the probability of detection would decrease. In a worst case scenario w.r.t. detecting attackers all honest participants ($|U| - 1$) send a 1 for all time slots and therefore the lower bound of the probability to detect the attack is

$$P(\mathcal{V}(\bar{\boldsymbol{d}}) = \text{false} \,|\, v_{u_m,t} = -1) \geq \left(\frac{I-1}{I}\right)^{|U|-1} . \tag{2}$$

The probability of successfully performing an attack would increase with an increasing number of participants. Therefore, one should choose the number of DC-Net rounds I dependent on the number of participants. If Mallory tries to send a -2 the chance of detection increases, but this is out of scope of this paper.

With Privacy-Loss We already discussed that a person who sent a 1 in a DC-Net which results in 0 is in the position to unmask the attacker with the drawback of giving up his privacy.[1] For such a case, we can define another function checking the correctness of the poll. This function will return false if there exists a participant u_p, who can prove that he sent a 1 at a time slot t and DC-Net round with partial vote index i which resulted in 0 ($\sum_{u\in U} \bar{d}_{u,t,i} = 0$).[2] Let $\bar{\boldsymbol{k}} \in \mathbb{Z}^{|U|\times(|U|-1)\times|T|\times I}$ be the 4-dimensional array of all keys used in all DC-Nets of the poll. The function which checks the correctness of the poll under the assumption that all participants are willing to disclose their availability at one time slot to unmask an attacker is defined as

$$\mathcal{B}\left(\bar{\boldsymbol{d}},\bar{\boldsymbol{k}}\right) = \begin{cases} \text{true} & \text{if } \neg\exists u_p \in U, t \in T, i \in \mathbb{Z}_I : \\ & \quad \left(\sum_{u\in U} \bar{d}_{u,t,i} = 0\right) \wedge \left(\bar{d}_{u_p,t,i} + \sum_{u\in U, u\neq u_p} \bar{k}_{u_p,u,t,i} = 1\right) \\ \text{false} & \text{otherwise.} \end{cases} \tag{3}$$

Now, Mallory needs two honest participants sending a 1 in the same DC-Net round to hide her (-1)-attack under these assumptions.[3] The probability that this attack is detected is calculated by adding the probabilities of the two cases (1) nobody choses Mallory's DC-Net, and (2) one participant choses Mallory's DC-Net. For case 1, the sum of the attacked DC-Net will be -1 and therefore $\mathcal{V}(\bar{\boldsymbol{d}})$ will fail (see Equation 2). The lower bound[4] for the probability of case 2 is the probability where the output of $\mathcal{B}\left(\bar{\boldsymbol{d}},\bar{\boldsymbol{k}}\right)$ is false and can be calculated with

$$P(\mathcal{B}\left(\bar{\boldsymbol{d}},\bar{\boldsymbol{k}}\right) \;=\; \text{false} \,|\, v_{u,t} \;=\; -1) \;\geq\; (|U|-1)\cdot\frac{1}{I}\cdot\left(\frac{I-1}{I}\right)^{|U|-2}. \tag{4}$$

Note that it is not possible, that $\mathcal{V}(\bar{\boldsymbol{d}}) = \text{false}$ and $\mathcal{B}\left(\bar{\boldsymbol{d}},\bar{\boldsymbol{k}}\right) = \text{false}$ (cp. Footnote 2) if we stick to only one attack at one time and therefore we can add the probabilities of Equations 2 and 4 to get the overall probability of detecting a (-1)-attack if users are willing to disclose their availability to unmask attackers.

Summary Table 2 illustrates these formulas with some example values. One can see that splitting the vote vector into 20 partial vote vectors makes it rather

[1] E. g., Bob could detect that Mallory cheated at t_0 in Fig. 1a.

[2] If this sum is lower than 0, an attack occurred as well. However, as this attack would be discovered by function $\mathcal{V}(\bar{\boldsymbol{d}})$ (Equation 1), we want to neglect this case here.

[3] This is like trying to perform a (-2)-attack without privacy-loss, but choosing one partial DC-Net to send the -2.

[4] All $|U|-1$ honest participants voted for the attacked time slot

Table 2. Lower bounds for the probability of successfully detecting an attack

I	$\|U\|$	(a)	(b)	(c)
20	15	48.8 %	84.7 %	(35.9 %)
20	5	81.5 %	98.6 %	(17.1 %)
50	15	75.4 %	96.9 %	(21.5 %)
50	5	92.2 %	99.8 %	(7.5 %)
100	15	86.9 %	99.2 %	(12.3 %)
100	5	96.1 %	99.9 %	(3.9 %)

(a) without giving up privacy $P(\mathcal{V}(\bar{\boldsymbol{d}}) = \text{false} \mid v_{u,t} = -1)$
(b) with privacy-loss $P(\mathcal{V}(\bar{\boldsymbol{d}}) = \text{false} \vee \mathcal{B}\left(\bar{\boldsymbol{d}}, \bar{\boldsymbol{k}}\right) = \text{false} \mid v_{u,t} = -1)$
(c) probability of privacy-loss $P(\mathcal{B}\left(\bar{\boldsymbol{d}}, \bar{\boldsymbol{k}}\right) = \text{false} \mid v_{u,t} = -1)$

unlikely to perform an undetected attack against small polls with 5 participants. The chance to detect a (-1)-vote at each time slot is at least[5] 81.5 %. Additionally to these 81.5 %, the attack can be discovered with a probability of 17.1 % by one of the participants. If all participants are willing to disclose their availability at the attacked time slot, the detection probability is at least 98.6 %.

In case of $\mathcal{V}(\bar{\boldsymbol{d}}) = \text{false}$, the decryption of the DC-Net round can be requested where the invalid value occured. Therefore every participant has to reveal his key for the DC-Net round. The single votes can be decrypted with the keys which identifies the attacker. The attacker may modify her key to hide her attack in this phase. However, this can be prevented in the same way as it was proposed for the verification phase of the original protocol.

However, if availabilities should not be disclosed under any circumstances, the attacker identification phase may be skipped. One has to accept in this case that attackers are able to perform denial of service attacks anonymously. Then one may decide with function $\mathcal{V}(\bar{\boldsymbol{d}})$ (Equation 1) that some attack occurred, but skip revealing keys to avoid possible decryption of votes. Note that the decision to perform a poll with identification phase or without has to be accepted by all participants. If every participant may decide on his own, an attacker will always refuse to reveal her keys, stating she has to cover some vote.

If a participant discovers an attack with the function $\mathcal{B}\left(\bar{\boldsymbol{d}}, \bar{\boldsymbol{k}}\right)$, he may decide on his own if he gives up his privacy to unmask the attacker. Therefore, the lower bound for the probability of detecting an attack is between both lower bounds.

3.2 Privacy

The possible decryption in the identification phase to detect the attacker may be a privacy problem. The original protocol had the same decryption in the verification phase. However, unlike in the original protocol, the probability that this really is a privacy problem is very low, as the attacker has to guess the index for the DC-Net round where the victim sent his vote. The probability of guessing the index of a specific victim is $\frac{1}{I}$.

Attacking more than one DC-Net round with negative values to increase the probability of hitting the victim's DC-Net does not help the attacker, because

[5] if all 4 honest participants vote for a time slot

the goal of the honest participants is to find only one DC-Net which was attacked. Therefore, it is enough to disclose the keys for one attacked round. The algorithm to choose the DC-Net round to be disclosed should choose one of the rounds with the lowest sum. The function $\mathcal{D} : \mathbb{Z}_n^{|U|\times|T|\times I} \to \mathcal{P}(T \times \mathbb{Z}_I)$ which takes all DC-Net messages as input, and results a set of time slot-partial vote index-pairs (t, i) which should be disclosed, can be defined as

$$\mathcal{D}(\boldsymbol{d}) = \left\{ (t,i) : \sum_{u\in U} \bar{d}_{u,t,i} = \min_{t'\in T, i'\in\mathbb{Z}_I} \left\{ \sum_{u\in U} \bar{d}_{u,t',i'} \right\} \right\}. \tag{5}$$

However, as already discussed in Section 3.1, one may decide to skip attacker identification, with the drawback, that denial of service attacks are possible then.

The privacy of a message in the DC-Net depends only on the secrecy of the keys. This is the same for the original as well as the new protocol but splitting the vote vector into several parts introduces a new point of attack. Now, the anonymity of the message also depends on the randomness and secrecy of the partial vote index. If an attacker can predict at which DC-Net rounds messages from some participants occur, she can separate the other messages into smaller anonymity sets. If all participants distribute their votes randomly over all rounds, Mallory needs the cooperation of the other participants to deanonymize her victim. However, deanonymization can be done without the help of the different partial DC-Nets, if participants disclose their shared DC-Net keys.

In a successful schedule with no attacker, one sum for every time slot with the number of all available participants is disclosed.

4 Preventing (+2)-Attacks

In the example of Fig. 1b, Alice can detect a (+2)-attack of Mallory as she knows that the sum is an element of $\{0, \ldots, |U| - 1\}$. However, it may be the case that nobody voted for a time slot (cp. e.g., time slot t_2 of Fig. 1b) where a 2 is sent. In such a case neither Alice nor Bob can detect the attack on their own.

A simple solution to this attack would be to request the verification phase in any case for the agreed time slot. This would neither be privacy-friendly nor efficient in terms of message exchanges.

In the following, another solution is proposed. The main idea is to reduce the problem of preventing (+2)-attacks to the already solved problem of preventing -1-attacks. Therefore, in addition to the normal poll, every participant sends his votes for the same time slots in a check poll. Every participant u calculates for every time slot t a check vote $v'_{u,t}$ which depends on his vote $v_{u,t}$ such that

$$v_{u,t} + v'_{u,t} = 1. \tag{6}$$

With the check votes $v'_{u,t}$, a check poll is done like for the normal poll.

The sum of both result vectors, the one from the normal poll and the one from the check poll, should be a vector where all elements are equal to the normal poll

	t_0	t_1	t_2	t_3
Alice	0	1	0	0
Bob	1	1	0	1
Mallory	0	0	0	2
$\sum$	1	2	0	3

$\sum$	3	3	3	3

check poll

	t_0	t_1	t_2	t_3
Alice	1	0	1	1
Bob	0	0	1	0
Mallory	1	1	1	-1
$\sum$	2	1	3	0

Fig. 3. By the use of a check poll, the (+2)-attack can be reduced to the (-1)-attack. Mallory has to send a -1 at t_3 in the right table, because the sum regarding t_3 of both tables would not be equal to the number of participants otherwise.

number of participants $|U|$. By checking this property, it is ensured that every participant calculated the check vote vector according to Equation 6. If Mallory wants to send a 2 for some time slot, she then has to send a -1 in the check poll. However, splitting the check vote vector into several ones, this attack can be prevented in the same way (-1)-attacks were prevented within the vote vector. Fig. 3 illustrates the whole process.

Let $\bar{\boldsymbol{d}}$ be the 3-dimensional array of all DC-Net messages sent to the normal poll and $\bar{\boldsymbol{d}}'$ be the 3-dimensional array of all DC-Net messages sent to the check DC-Nets. The verification function of correctness is defined as

$$\mathcal{C}(\bar{\boldsymbol{d}}, \bar{\boldsymbol{d}}') = \begin{cases} \text{true} & \text{if } \forall t \in T : |U| = \sum_{u \in U, i \in \mathbb{Z}_I} \left(\bar{d}_{u,t,i} + \bar{d}'_{u,t,i}\right) \\ \text{false} & \text{otherwise.} \end{cases} \tag{7}$$

4.1 Verifiability

When evaluating the result, two kinds of inconsistencies may occur: the sum of both polls may be lower or higher than the number of participants. As we split the votes into several ones (cp. Section 3), we do not consider (-1)-attacks at this point. Therefore, a value lower than the number of participants may occur only if one or more participants sent $v_{u,t} = v'_{u,t} = 0$. Having such a case would mean that the number of available participants σ_t would be the result of the normal poll at least. The difference of the sum of both polls and the total number of participants are wrongly cast votes.

The second kind of inconsistency is if the sum of both polls is higher than the number of participants. As we prevented (-1)-votes, the result of the normal poll at a time slot t should be greater or equal than the number of available participants at this time slot. In addition, the number of available participants is greater or equal than the total number of participants minus the result of the check poll at a certain time slot. Putting both inequations together, one obtains a range which expresses the possible number of available participants:

$$\sum_{i \in I, u \in U} \bar{d}_{u,t,i} \geq \sigma_t \geq |U| - \sum_{i \in I, u \in U} \bar{d}'_{u,t,i}. \tag{8}$$

Table 3. Comparison of the computational complexity of the original protocol with unanimous agreement and the scheme with our extension (assuming no attack).

	discrete exp.	symmetric decr.	hash values
original protocol	$\|U\| - 1$	$\|T\| \cdot (\|U\| - 1)$	$\|T\| \cdot (\|U\| - 1)$
new scheme	$\|U\| - 1$	$2 \cdot I \cdot \|T\| \cdot (\|U\| - 1)$	$2 \cdot I \cdot \|T\| \cdot (\|U\| - 1)$

However, as an attacker may manipulate his vote in a way that this range results in $\{0, \ldots, |U|\}$, he may attack the availability of the poll in such a way. To unmask the attacker, all DC-Net rounds for the inconsistent time slot can be decrypted as shown before. If the attack discovery goes along with some cost (penalty, reputation loss, etc.), it makes such attacks unattractive.

4.2 Privacy

During the verification phase of an inconsistent check poll, the availability of all participants at the inconsistent time slot are disclosed. To avoid disclosure of all availabilities, one may disclose the DC-Net rounds step by step and stop when the attacker is found. The sequence of disclosure should therefore be a fixed order, which is not known before every participant stated his vote.[6]

In a successful run, no more information is disclosed than in the original protocol, the check poll contains only redundant information.

4.3 Computational Complexity

The extension presented in this paper affects the computational complexity of the original protocol only in terms of symmetric cryptographic operations, i.e., the amount of asymmetric cryptographic operations is not affected. The number symmetric operations of the original scheme increases with the number of DC-Net rounds I and is doubled due to the check poll. Table 3 compares the complexity of our extension with the complexity of the original protocol.

5 Implementation

An implementation of the protocol is available at `dudle.inf.tu-dresden.de`. The cryptographic operations are implemented in JavaScript; no installation on the client side is needed.

The implementation has been done using the JavaScript BigInteger library from Tom Wu [21]. Like in the original protocol, a symmetric cipher and a hash function is used for key generation. AES-128 and SHA-256 from the JavaScript libraries of B. Poettering are used here [26].

[6] E.g., every participant may commit himself to a random number together with his vote vector. In case of verification all commitments are revealed and the random numbers are added to one single seed which is used to bootstrap a sequence.

Table 4. Performance measurement of the key calculation in a poll with $|U| = 5$, $|T| = 20$ and $I = 20$. The first 5 columns were measured on an Intel Pentium 4 Duo with 2.8 GHz, 2 GB RAM running Windows XP SP3. The last one was measured on a Motorola Milestone running Android.

	IE 8.0.6001	Firefox 3.6.12	Safari 5.0.3	Opera 10.63	Chrome 8.0.552	Android Firefox 4.0b2
AES-128+SHA-256	18.4 s	7.7 s	2.9 s	1.4 s	2.8 s	31.7 s
DH	15.0 s	11.7 s	8.2 s	2.0 s	2.5 s	40.5 s
total	37.6 s	22.5 s	13.5 s	4.3 s	6.3 s	84.2 s

Table 4 shows a performance measurement of the key calculation for an example poll. One can see, that the calculation needs about 23 s, using Firefox 3.6.12. If browsers run a script which needs longer calculation time, it is usual that the browser asks the user if he wants to stop the script. To avoid these pop-ups, the BigInteger library was modified in a way that it calculates exponentiations asynchronously with a callback function. This enables the calculation to be forked in the background and the browser remains responsive. The user can enter his availabilities, while the browser calculates the keys. The submit button is enabled after the calculation has been done. Assuming, that a user needs some time to enter his preferences (look up the time slots in his personal calendar, click the buttons etc.), there is no extra waiting time.

6 Conclusion

We proposed a scheme, which is able to schedule events in a privacy-enhanced way. Our scheme prevents attacks to neglect or promote certain time slots, has no negative influence on the privacy and affect the computational complexity only in terms of symmetric cryptographic operations. To demonstrate that the scheme performs in practice, we presented Dudle, an implementation of the scheme in JavaScript. Due to the use of JavaScript for all client side operations, no installation is needed for the user. We therefore showed, that complex cryptography is possible in zero footprint applications.

Acknowledgments The author wants to thank the whole DuD-Group in Dresden for the fruitful coffee breaks. Special thanks goes to Rainer Böhme, Sebastian Clauss, Stefan Köpsell and Sandra Steinbrecher. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement № 216483.

References

1. Kellermann, B., Böhme, R.: Privacy-enhanced event scheduling. In: CSE (3). IEEE Computer Society (2009) 52–59

2. Silaghi, M.C., Sam-Haroud, D., Faltings, B.: Asynchronous search with aggregations. In: AAAI/IAAI. AAAI Press / The MIT Press (2000) 917–922
3. Yokoo, M., Hirayama, K.: Algorithms for distributed constraint satisfaction: A review. Autonomous Agents and Multi-Agent Systems **3**(2) (2000) 185–207
4. Léauté, T., Faltings, B.: Privacy-preserving multi-agent constraint satisfaction. In: CSE (3). IEEE Computer Society (2009) 17–25
5. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artif. Intell. **161** (2005) 149–180
6. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: AAMAS. IEEE Computer Society (2004) 310–317
7. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS, Washington, DC, IEEE (2004) 438–445
8. Franzin, Freuder, Rossi, Wallace: Multi-agent meeting scheduling with preferences: Efficiency, privacy loss, and solution quality. AAAI Technical Report (2002)
9. Greenstadt, R., Pearce, J.P., Bowring, E., Tambe, M.: Experimental analysis of privacy loss in DCOP algorithms. In: AAMAS. ACM Press (2006) 1424–1426
10. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2) (1981) 84–90
11. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: AUSCRYPT. vol. 718. Springer (1992) 244–251
12. Benaloh, J., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: STOC, New York, NY, USA, ACM (1994) 544–553
13. Sako, K., Kilian, J.: Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. In: EUROCRYPT. Springer (1995) 393–403
14. Cramer, Gennaro, Schoenmakers: A secure and optimally efficient multi-authority election scheme. In: EUROCRYPT. vol. 1233 of LNCS. Springer (1997) 103–118
15. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: EUROCRYPT, Berlin, Heidelberg, Springer (2000) 539–556
16. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: WPES, New York, NY, USA, ACM (2005) 61–70
17. Herschberg, M.A.: Secure electronic voting over the world wide web. Master's thesis, Massachusetts Institute of Technology (May 1997)
18. Adida, B.: Helios: Web-based open-audit voting. In: USENIX. (2008) 335–348
19. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: Security and Privacy. IEEE (2008) 354–368
20. Rivest, R.L., Smith, W.D.: Three voting protocols: Threeballot, vav, and twin. In: USENIX, Berkeley, CA, USA, USENIX Association (2007) 16–16
21. Wu, T.: BigIntegers and RSA in JavaScript. `http://www-cs-students.stanford.edu/~tjw/jsbn/` (July 2010)
22. Baird, L.: BigIntegers in JavaScript. `http://www.leemon.com/crypto/BigInt.html` (July 2010) Version 5.4.
23. Shapiro, D.: BigInt, a suite of routines for performing multiple-precision arithmetic in JavaScript. `http://ohdave.com/rsa/BigInt.js` (July 2010) Version 5.4.
24. Herlea, Claessens, Preneel, Neven, Piessens, De Decker: On securely scheduling a meeting. In: SEC. vol. 193 of IFIP Conference Proceedings. Kluwer (2001) 183–198
25. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology **1**(1) (January 1988) 65–75
26. Poettering, B.: The AES block cipher and the SHA256 message digest in JavaScript. `http://point-at-infinity.org/` (July 2010) Version 0.1.