Privacy-Enhanced Event Scheduling

Benjamin Kellermann and Rainer Böhme

Faculty of Computer Science, Technische Universität Dresden, 01062 Dresden, Germany
Email: {Benjamin.Kellermann|Rainer.Boehme}@tu-dresden.de

***Abstract*—Event schedulers, well-known from groupware and social software, typically share the problem that they disclose detailed availability patterns of their users. This paper distinguishes event scheduling from electronic voting and proposes a privacy-enhanced event scheduling scheme. Based on superposed sending and Diffie–Hellman key agreement, it is designed to be efficient enough for practical implementations while requiring minimal trust in a central entity. Protocols to enable dynamic joining and leaving of participants are given.**

***Keywords*-event scheduling; electronic voting; superposed sending; anonymity; privacy-enhanced application design**

I. Introduction

So-called 'Web 2.0' [1] applications became popular over the past years. Not only wikis, forums and social networking sites boomed massively, also some special purpose Web 2.0 applications like twitter or flickr got much attention.

One of these applications is doodle [2]. It lets users create polls to "schedule an event such as a board meeting, business lunch, conference call, family reunion, movie night or any other group event" [2]. As in most Web 2.0 applications, privacy is only a secondary goal. To keep doodle as simple as possible, everybody may create polls, cast votes to existing polls, see results of other polls, and even revise casted votes of running polls. When participating in a doodle poll, one has to share personal information with the server, the other participants, and even with the whole world. This information includes when exactly a particular user is available for the event. The so-called *availability patterns* often contain sensitive information in at least two respects. First, direct inference from the availability at a particular date may reveal information about one's private life ('Will my husband vote for the date of our wedding anniversary?'). Second, indirect inference arises from the fact that availability patterns contain much entropy and thus allow to (re-)identify individuals who would otherwise remain pseudonymous ('The availability pattern of user *bunny23* looks suspiciously like the one of my employee John Doe!').

Recently even the doodle developers have recognized that users demand some privacy. The current versions of their application includes the possibility to create "hidden polls" in which only the administrator can see all votes. However, one still has to trust the administrator and the doodle service.

This paper proposes a privacy-friendly solution for scheduling a single event. We define the problem and distinguish it from electronic voting in Section II. An overview of related work can be found in Section III. We present the basic scheme in Section IV and show extensions with regard to simplifying the key exchange and considering dynamic joining and leaving in Section V. Section VI explains why the proposed scheme is secure and efficient enough to be suitable for practical implementations.

II. Problem Definition and Notation

A. Single Event Scheduling Problem

Privacy-enhanced event scheduling can be understood as distributed constraint satisfaction/optimization problem (DCSP/DCOP) or as an instance of an electronic voting scheme. As privacy is merely a secondary aspect in DCOP (and many schemes in fact leak information [3], [4] or are not verifiable [5]; despite high complexity), we frame our contribution in the context of e-voting, where exact security analyzes with regard to voter anonymity have a longer tradition.

The view that existing e-voting schemes can be directly applied to the special case of event scheduling is not fully correct, either: the main difference between existing schemes and the event scheduling problem is the parameter in which the system has to scale efficiently. Typical governmental or committee voting schemes in the literature scale in the number of *voters*, each of whom has one (or a few) vote(s). By contrast, event scheduling requires a limited number of voters to make many binary choices, one for every possible point in time. Hence it has to scale in the number of *independent choices*. As opposed to typical governmental election schemes, which have to cope with millions of voters, it is not so crucial for event scheduling to scale gently in the number of participants. Typically we schedule events for closed groups of a few dozens.

For example it may be desirable to agree on a 1 hour meeting within the next two weeks. As a typical working-week has 40 hours, assuming that meetings are aligned to full hours, we would end up with 40 votes per week. Assuming possible start dates at every quarter of the hour, we would end up with $40 \cdot 4 = 160$ starting times per week. This totals to 320 different starting times which all require a binary vote. Figure 1 on the following page illustrates the difference between governmental elections in the literature and our special case.

Current implementations like doodle certainly run into usability problems when offering polls with 320 options. But it is easy to conceive interfaces to personal electronic calendars, similar to common office groupware. Obviously, such a system designed as transparent as current tools, i. e., without considering privacy, would end up in publishing large and detailed availability patterns of its users. So protecting

	Vote
Candidate #1	
Candidate #2	✘
⋮	
Candidate #10	

	Date #1	Date #2	…	Date #320
Yes	✘			✘
No		✘		

Figure 1. Two example voting sheets: left for a governmental election; right for event scheduling. In both sheets, each voter has one choice per column.

this personal data can be formulated as main motivation and requirement for our scheme.

B. Requirements

Requirements for e-voting were already discussed in the literature [6]–[9] and obviously there are legal requirements for governmental e-voting [10, e. g.]. Event scheduling is subject to fewer legal restrictions. However, as the problem is very similar to e-voting, one may take these requirements as a starting point.

It is often required that governmental elections are general, free, equal, and secret [11]. Especially for e-voting, two partially conflicting requirements emerge: verifiability and receipt-freeness. The latter is desirable to discourage vote-selling. Transparency is another requirement of elections. It means that everybody should understand what happens and how his or her vote has been counted. This is particularly valid when cryptography comes into play.

Derived from typical requirements in e-voting, the following requirements apply to privacy-enhanced event scheduling:

1) Verifiability: Every voter should be able to verify that no other voter has cheated and that his vote has been counted.

2) Privacy: Nobody should learn more than absolutely necessary about the availability of other voters and thus should not be able to infer on their identity, i. e., every participant should only learn that the other participants are available at the one specific date which was chosen.

3) Untrusted server: As little trust as possible should be placed in any central entity (e. g., vote server).

4) Usability: The scheme should not require many more steps than existing event schedulers, i. e., it should not require much more user interaction and message exchanges.

5) Efficiency: The scheme should be more efficient than existing e-voting or DCOP schemes. More precisely, it should be efficient for large scheduling problems with many possible event dates.

C. Notation

We will use the following terminology and notation in this paper.

1) Initiator: A person who sets up a poll. This person may also participate in the poll as a voter.

2) Voter (p): A participant in the poll. P is the ordered set of all voters so that $|P|$ is the number of voters in the poll.[1]

[1]We used the term 'voter' here instead of 'participant' to be more conform to the existing literature on e-voting.

3) Time slot (t): A specific date and time at which an event can take place. The ordered set of time slots is denoted by T.

4) Anonymity set: A set of voters, in which a specific voter is not identifiable [12].

III. Related Work

Literature on electronic voting is abundant. For our purpose, only pure e-voting schemes are relevant [13]–[26], which have to be distinguished from paper-based elections with computer-assisted counting [27]–[31]. We can broadly classify the literature into three approaches to meet the basic requirements: mixes, homomorphic encryption, and blind signatures. We will discuss each of them briefly.

A. Mixes

Chaum invented mixes as building blocks for anonymous communication channels and he first proposed an election scheme based on them [13]. Many extensions have followed thereafter [17], [19]–[21]. The common idea is to build an anonymous blackboard with mixes. In a first phase, every voter has to generate an asymmetric key pair and publish the public key on that blackboard. In the voting phase, every voter encrypts his or her vote with the secret key and publishes the encrypted vote on the anonymous blackboard. Everybody can decrypt the votes and count the result.

Assuming ℓ mixes, to cast a vote, every voter has to encrypt his or her key and vote ℓ times asymmetrically. A naive adaption to event scheduling would imply one poll per time slot. Every voter would have to do $2 \cdot \ell \cdot |T|$ asymmetric encryptions. Further, one must trust that the mixes do not collude to compromise one's privacy, and the mixes have to perform additional decryption operations, which add to the overall complexity.

B. Homomorphic Encryption

Voting schemes based on a homomorphic encryption function use the property that one can add all the votes and decrypt the result without decrypting individual votes (i. e., one can find two operations $\oplus$ and $\otimes$ so that the encryption function E is homomorph to these operations $\mathrm{E}(x_1) \oplus \mathrm{E}(x_2) = \mathrm{E}(x_1 \otimes x_2)$). A problem of plain homomorphic encryption is that cheating voters stay undetected as their votes are not decrypted separately. So practical schemes require extra effort to prevent this.

A voting scheme based on a homomorphic encryption function was first described by Cohen and Fischer [14] and later extended by Benaloh and Yung [15]. The scheme consists of different parts where a central entity and the voters have to commit to values and prove their correctness without revealing them. Proving a "primary" ballot is done by committing to several "auxiliary" ones, decrypting half of them and showing that the others are type-equivalent to the primary ballot. This proof is done twice in the whole scheme.

A direct application of this method for event scheduling appears to be inefficient: considering only the vote encryptions, with a security parameter ℓ, every voter would have to do $\ell \cdot |T|$ asymmetric cryptographic operations.

$$\boldsymbol{k}_{p_i} = \begin{pmatrix} k_{p_i,p_1,t_1}, \mathrm{sig}_{p_1}(k_{p_i,p_1,t_1}) & \dots & k_{p_i,p_1,t_{|T|}}, \mathrm{sig}_{p_1}(k_{p_i,p_1,t_{|T|}}) \\ \vdots & \ddots & \vdots \\ k_{p_i,p_{i-1},t_1}, \mathrm{sig}_{p_{i-1}}(k_{p_i,p_{i-1},t_1}) & \dots & k_{p_i,p_{i-1},t_{|T|}}, \mathrm{sig}_{p_{i-1}}(k_{p_i,p_{i-1},t_{|T|}}) \\ k_{p_i,p_{i+1},t_1}, \mathrm{sig}_{p_{i+1}}(k_{p_i,p_{i+1},t_1}) & \dots & k_{p_i,p_{i+1},t_{|T|}}, \mathrm{sig}_{p_{i+1}}(k_{p_i,p_{i+1},t_{|T|}}) \\ \vdots & \ddots & \vdots \\ k_{p_i,p_{|P|},t_1}, \mathrm{sig}_{p_{|P|}}(k_{p_i,p_{|P|},t_1}) & \dots & k_{p_i,p_{|P|},t_{|T|}}, \mathrm{sig}_{p_{|P|}}(k_{p_i,p_{|P|},t_{|T|}}) \end{pmatrix} \quad (1)$$

Inspired by Benaloh, Sako and Kilian introduced a voting scheme that uses partially compatible homomorphisms [18]. Baudron et al. enhanced this scheme for multi-votes [22]. However, also in this scheme, a voter has to encrypt every single vote multiple times. Baudron's extension even targets multi-candidate elections, but this is not equivalent to the event scheduling problem, where repeated elections would be needed.

C. Blind Signatures

Shortly after his mix-approach, Chaum came up with another voting scheme [26] which uses blind signatures [32]. Fujioka et al. reduced the complexity of Chaums idea to adapt it to large scale elections [16] and many subsequent schemes were derived from the Fujioka–Okamoto–Ohta scheme [23]–[25]. Some of them led to implementations [33], [34].

The main idea is to split the protocol into two independent phases: (1) administration, which handles access control, and (2) counting of anonymously casted votes. The vote is blindly signed during administration, unblinded by the voter and then sent anonymously to the counter. As the casted votes contain no personal information, they can be published afterwards.

If one applies this scheme to the event scheduling problem, a voter would have to blind and unblind $|T|$ messages and verify the administrator's signature of every message. The administrator would have to verify $|P|$ signatures and has to sign $|T|$ messages. The counter would have to verify $|P| \cdot |T|$ signatures. The overall effort is $(|P|+2)\cdot|T|+|P|$ asymmetric cryptographic operations.

D. Specific Prior Work

We are aware about only one specific publication on secure and verifiable event scheduling [35], which in fact covers three different approaches to the problem. One is a solution based on a trusted third party, which is efficient, but stands in contrast to our requirement of limited trust in a central entity. The second is a straight application of general secure distributed computing. Consequently, it suffers from high computational and communication complexity. The third approach, a "custom-made negotiation protocol" is most interesting and promising. It can be best described as a hybrid between the techniques reviewed in Sects. III-A to III-C above. The protocol is designed to schedule single events through a combination of *homomorphic encryption* with respect to the equality operation (in fact, addition modulo two) to *blind* individual availability pattern, and an anonymous channel, which is established by letting voters act as re-encrypting *mixes*. While the cryptographic operations are comparably efficient, the scheme requires way more communication phases than our proposal. (The authors acknowledge this and discuss ways to trade off communication complexity against trust assumptions.) Moreover, dynamic joining and leaving of participants is not considered.

IV. Basic Scheme

Our proposed scheme consists of three mandatory phases and one optional verification phase that is run when inconsistencies occur:

1) poll initialization,
2) casting of the votes,
3) publication of the result,
4) (optional) verification of the result.

In the following we describe the phases in more detail.

A. Poll Initialization

In the poll initialization phase, the initiator has to define the set T of all possible time slots at which the event can take place. For now we consider a closed group, so the initiator has to define the set of voters P. Dynamic inclusion and exclusion of voters will be discussed in Sects. V-C and V-D.

To ensure the anonymity of each voter, we propose to use superposed sending, generalized to other abelian groups than $\mathrm{GF}(2)$ [36]. This anonymity scheme has an implied homomorphism, in which we can integrate the voting protocol. Let n be the modulus that defines the group $\mathbb{Z}_n$ for superposed sending. To ensure that no overflow occurs while summing up all votes, the modulus n has to be large enough, i. e., $n > |P|$.

All voters have to exchange symmetric keys beforehand. To be precise, each voter has to do a key exchange with $|P|-1$ other voters. Therefore he exchanges with each other voter, $|T|$ independent and uniformly distributed random numbers $r_t \in \mathbb{Z}_n$. This means he ends up with $(|P|-1)\cdot|T|$ random numbers. To detect cheating by key modification later on, the participants have to sign their keys. A digital signature by participant p of message m will be denoted as $\mathrm{sig}_p(m)$. Keys shared by a pair of voters are denoted as $k_{p_i,p_j,t}$ where $p_i, p_j \in P$ and $t \in T$. To run the key exchange, each pair of voters p_i, p_j where $i < j$ does:

1) Exchange a random number $r_t \in \mathbb{Z}_n$ for every $t \in T$.
2) Voter p_i stores the signatures $\mathrm{sig}_{p_j}(k_{p_i,p_j,t})$ and the keys $k_{p_i,p_j,t} = \quad r_t$.
3) Voter p_j stores the signatures $\mathrm{sig}_{p_i}(k_{p_j,p_i,t})$ and the keys $k_{p_j,p_i,t} = -r_t \bmod n$.

Afterwards each voter p_i holds a key matrix as displayed in Equation 1 on the previous page. Rows in this matrix contain the keys and signatures exchanged between two voters p_i, p_j and are denoted by $\vec{k}_{p_i,p_j}$. Establishing the whole key graph requires $\frac{|P|\cdot(|P|-1)}{2}$ key exchanges. Since such a key exchange protocol does not scale well in $|P|$, we will discuss a reduction of communication complexity by using asymmetric cryptography in Section V-A.

B. Casting of Votes

In this phase, each voter has to state for every given time slot whether he or she can participate in the event or not. The superposed sending, as underlying anonymity mechanism, sums up all messages. This suits well with our goal, as we are only interested in the sum of the given votes.

Each voter p has to calculate an encrypted vote vector $\vec{d}_p$ which consists of $|T|$ encrypted votes $d_{p,t}$,

$$\vec{d}_p = (d_{p,t_1}, d_{p,t_2}, \ldots, d_{p,t_{|T|}})\,. \tag{2}$$

An encrypted vote $d_{p,t}$ is calculated by adding all keys the voter has exchanged with other voters to the actual vote $v_{p,t}$ modulo n,

$$d_{p,t} = v_{p,t} + \sum_{i=1,p_i\neq p}^{|P|} k_{p,p_i,t} \bmod n\,. \tag{3}$$

$v_{p,t} \in \{0,1\}$ is the specific vote with the semantic that value 0 means the voter p is unavailable at time slot t and value 1 signals availability.

Vector $\vec{d}_p$ is sent to the server and will be published there after all voters have casted their votes.

C. Result Publication

When all vote vectors are published on a central server, any party can calculate the sum of these vectors. Since all keys should be pairwise inverse, the result should be a vector $\vec{v} \in (\mathbb{Z}_{|P|})^{|T|}$ containing the sum of all votes at the specific time slots.

As *selection rule* for the agreed time slot t_a, the earliest time slot for which $t_a = |P|$ is chosen. It is not sufficient to take the time slot with the maximum votes, since a malicious voter could send values $v_{p,t}$ below 0. This is certainly a limitation compared to other practical implementations without privacy, but the priority in our scheme is to ensure verifiability and privacy (see Section VI-A on page 6). The cryptographic DCSP approach in [5] shares the same limitation while being not fully verifiable and substantially more complex.

The message exchange of the three mandatory phases is illustrated in Figure 2.

D. Result Verification

A malicious voter could send values $v_{p,t}$ different from 0 or 1. Since this would be completely invisible to the others, we have to verify the result after publication. The first step of this phase is that everybody checks, if he voted for this specific time slot. This is done in most cases anyway, when the voter inserts the event in his personal calendar. If a voter discovers an inconsistency here, he can request the decryption of the individual votes for the agreed time slot t_a. If everybody stuck to the protocol, this is no privacy problem because one can infer that everybody must have sent a 1 from the mere fact that the time slot has been selected. This means, after t_a is found, every voter p_i has to publish his signed shared secret keys $k_{p_i,p_j,t_a}, \mathrm{sig}_{p_j}(k_{p_i,p_j,t_a})$ for the selected time slot if at least one voter demands it. With these keys, the respective elements d_{p_i,t_a} of the encrypted vote vectors $\vec{d}_{p_i}$ can be decrypted and it can be verified that every voter has casted a 1. However, a malicious voter can send values higher than 1 in an attempt to compromise the privacy for one specific time slot ('Has my husband voted for the date of our wedding anniversary?'). This attack will be discussed in Section VI-B on page 6.

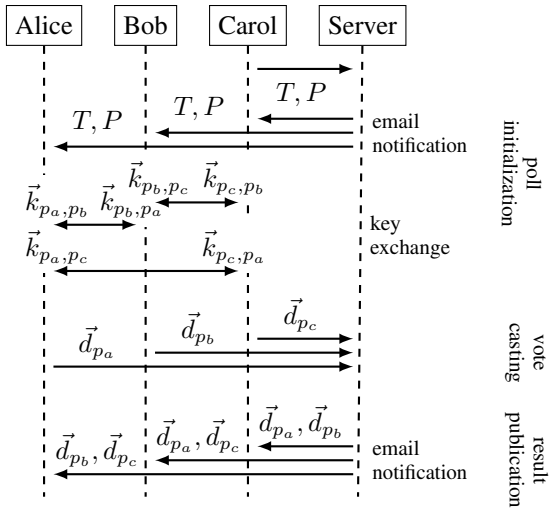

Figure 2. Basic scheme: message exchanges in successful protocol run

V. Extensions

A. Simplifying the Key Exchange

Securely exchanging a truly random number with every participant can be a problem for practical implementations. To avoid this, one can resort to Diffie–Hellman key agreement [37].

Instead of exchanging a random number with every other participant, a voter may register himself at a key exchange server. This enrollment can be used afterwards for several polls with different sets of voters.

Let q be the modulus and g the generator of the Diffie–Hellman key agreement protocol, both are constant for all potential voters and polls. Each voter p registers in three steps:

1) Fetch the modulus q and the generator g from the server.
2) Choose a random number and store it as Diffie-Hellman secret key sec_p.
3) Calculate the public key $\mathsf{pub}_p = g^{\mathsf{sec}_p} \bmod q$ and publish it on the server.

Instead of the key exchange described in Section IV-A on the preceding page, a key $k_{p_i,p_j,t}$ shared between voters p_i

and p_j in matrix $\boldsymbol{k}_{p_i}$ can be calculated for a specific poll with a universal unique poll identifier $\mathsf{uuid} \in \mathsf{UUID} \subseteq \mathbb{Z}$ as,

$$k_{p_i,p_j,t} = \begin{cases} \mathcal{G}(\mathsf{dh}_{p_i,p_j}, \mathsf{uuid}, t) \bmod n & \text{if } p_i < p_j \\ -\mathcal{G}(\mathsf{dh}_{p_i,p_j}, \mathsf{uuid}, t) \bmod n & \text{otherwise,} \end{cases} \tag{4}$$

where $\mathsf{dh}_{p_i,p_j} = g^{\mathsf{sec}_{p_i} \cdot \mathsf{sec}_{p_j}} \bmod q$ is the Diffie–Hellman secret calculated by p_i and p_j, and $\mathcal{G} : \mathbb{Z}_q \times \mathsf{UUID} \times T \to \mathbb{Z}$ is a deterministic function with the following properties:

1) Given a set of triples $(t_a, \mathsf{uuid}, k_{p_i,p_j,t_a})$ for two voters p_i, p_j with a time slot $t_a \in T$, a poll identifier $\mathsf{uuid} \in \mathsf{UUID}$ and the key k_{p_i,p_j,t_a}, it should be impossible to distinguish whether another triple $(t_b, \mathsf{uuid}', k_{p_i,p_j,t_b})$, for any other time slot $t_b \in T, a \neq b$ or poll id $\mathsf{uuid}' \in \mathsf{UUID}, \mathsf{uuid} \neq \mathsf{uuid}'$, is valid according to Equation 4.
2) For input triples $(\mathsf{dh}, \mathsf{uuid}, t)$ with any two parameters fixed and the third parameter drawn from a uniform distribution over its domain, $\mathcal{G}(\mathsf{dh}, \mathsf{uuid}, t) \bmod n$ is indistinguishable from uniform random numbers over $\mathbb{Z}_n$.

The first property ensures that nobody is able to calculate keys for arbitrary time slots even after the release of the key for the agreed time slot t_a in the verification phase (compare Sect. IV-D on the previous page) and even if keys from more than one poll are known. The uniform distribution ensures the secrecy of the votes: every key is equally probable.

A possible instance of $\mathcal{G}$ is a decryption function $\mathrm{decr}_{\mathsf{key}}(\text{ciphertext})$ of a symmetric cipher, which resists known-plaintext attacks. The construction is, $\mathcal{G}(\mathsf{dh}_{p_i,p_j}, \mathsf{uuid}, t) = \mathrm{decr}_{\mathsf{dh}_{p_i,p_j}}(\mathsf{uuid}||t)$, where $||$ is the concatenation operator.

B. Omitting the Key Signatures

To completely avoid initial communication between voters, the key signatures $\mathrm{sig}_{p_i}(k_{p_j,p_i,t})$ have to be omitted as well. For this, one can use a function $\mathcal{F} : \mathbb{Z} \to \mathbb{Z}$, such that

1) Property 2 mentioned above for $\mathcal{G}(\cdot) \bmod n$ should also apply to $\mathcal{F}(\mathcal{G}(\cdot)) \bmod n$, and
2) $\mathcal{F}(\cdot) \bmod n$ is preimage resistant, i. e., for any $y \in \mathbb{Z}_n$, it is hard to find an x such that $\mathcal{F}(x) \bmod n = y$.

Then Equation 4 can be replaced with:

$$k_{p_i,p_j,t} = \begin{cases} \mathcal{F}(\mathcal{G}(\mathsf{dh}_{p_i,p_j}, \mathsf{uuid}, t)) \bmod n & \text{if } p_i < p_j \\ -\mathcal{F}(\mathcal{G}(\mathsf{dh}_{p_i,p_j}, \mathsf{uuid}, t)) \bmod n & \text{otherwise.} \end{cases} \tag{5}$$

Instead of revealing the key $k_{p_i,p_j,t}$ in the result verification phase (compare Sect. IV-D), every voter p_i discloses $\mathcal{G}(\mathsf{dh}_{p_i,p_j}, \mathsf{uuid}, t)$, from which the keys can be calculated. The preimage resistancy of $\mathcal{F}(\cdot) \bmod n$ ensures that a malicious voter cannot cheat, as explained in Equation 8 on the following page. A hash function h can be used to implement $\mathcal{F}$.

With simplification of the key exchange and omitting the signatures, no mutual communication between the voters is needed, as illustrated in the simplified protocol of Figure 3. Putting everything together, the key calculation is,

$$k_{p_i,p_j,t} = \begin{cases} \mathrm{h}(\mathrm{decr}_{\mathsf{dh}_{p_i,p_j}}(\mathsf{uuid}||t)) \bmod n & \text{if } p_i < p_j \\ -\mathrm{h}(\mathrm{decr}_{\mathsf{dh}_{p_i,p_j}}(\mathsf{uuid}||t)) \bmod n & \text{otherwise.} \end{cases} \tag{6}$$

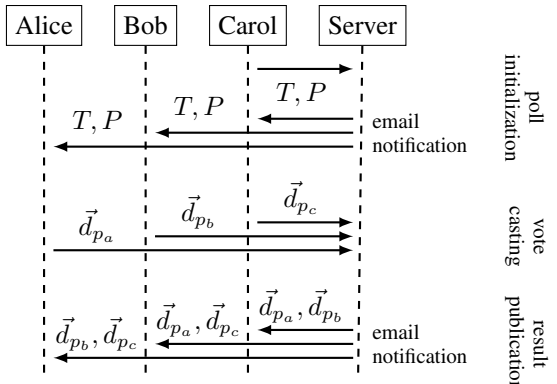

Figure 3. Extended scheme: message exchanges in successful protocol run

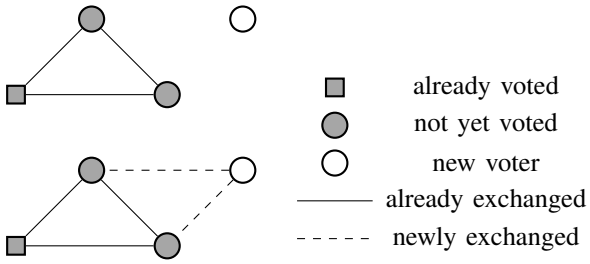

Figure 4. Key graphs before (top) and after (bottom) dynamic joining

C. Dynamic Joining

It may be desirable to add a voter after a poll has started. With some restrictions, this can be done. The new voter may now exchange keys with all voters who have not submitted a vote up to this moment.[2] Than, the anonymity set for the pending voters has increased and the anonymity set for all voters who have already participated stays the same. Figure 4 illustrates how the key graph changes after dynamic joining.

In theory it is possible for the new voter to exchange keys with voters who have already participated as well. These voters would resubmit a new encrypted vote vector including the newly exchanged key. Unfortunately, this does not increase the anonymity because the new key can be calculated by simply subtracting the old vote from the new one. To increase the anonymity of voters who already submitted, they have to exchange new keys with all others. This is equivalent to restarting the entire poll.

D. Dynamic Leaving

Analogously, it may occur that one or more voters should leave a running poll (e. g., one may want to schedule the event before all invited voters have actually casted their votes). This is possible too, as the remaining voters know all the shared keys of a leaving voter. Every voter p_i who shares a key with the leaving voter p has to agree to the reduction of the anonymity set. He does so by disclosing the $|T|$ keys $k_{p_i,p,t}$ that he shares

[2] If executing the extended scheme, they can be calculated newly instead of exchanging.

with the leaving voter.[3] With these keys, a surrogate encrypted vote vector $\vec{d}_p$ of the leaving voter p can be composed with it's elements:

$$d_{p,t} = 1 - \sum_{i=1, p_i \neq p}^{|P|} k_{p_i,p,t} \bmod n\,. \tag{7}$$

The protocol proceeds as described above including the surrogate vote(s).

Note that there is no need that voters who have not exchanged a key with the leaving voter agree to the anonymity set reduction. Their anonymity set is not affected anyway. Further there is no need to seek a leaving voter's cooperation.

VI. Analysis

Now we explain how our scheme fulfills the requirements stated in Section II-B. When necessary, we point out differences between the basic scheme (Sect. IV) and its extensions (Sect. V).

A. Verifiability

An attacker could try to manipulate the vote values $v_{p,t}$ to values different from 0 or 1. To prevent outside attackers from modifying the communication channel (e. g., adding or subtracting 1 to an encrypted vote $d_{p,t}$), the voters can verify if their own votes are listed correctly at the server. Another defense is signing the encrypted vote vectors $\vec{d}_p$.

The selection rule (cf. Sect. IV-C on page 4) and the result verification phase (cf. Sect. IV-D on page 4) prevent malicious voters from casting invalid votes.

Due to the selection rule, there is no reason to cheat by sending a logical value below 0. Assuming all others are honest, this would rule out the chance for the specific time slot to be chosen. But as we only accept time slots where all voters are available, it would make no difference for an attacker sending 0 or a lower value. Of course, a voter can always attack the availability of the system by casting 0 for every time slot and remain anonymous. Anonymity could also stimulate 'legal-but-selfish' votes, i. e., setting all time slots to 0 except the preferred one. Such strategies might be discouraged by the transparency in current privacy-unfriendly applications. However, this is ultimately a behavioral question and as such largely outside the scope of the protocol.

In the result verification phase, every voter can verify that no other voter sent a value higher than 1. To hide this attack, a malicious voter could tamper with the key in the verification phase to frame another (innocent) voter. For example, voter Mallory p_m has submitted $v_{p_m,t_a} = 2$ and tries to hide her behavior in the verification phase. This can be done by first decrementing one random key she added to the vote and then publishing this modified key. When decrypting her vote, it would now look like she has published a 1 and another voter (e. g., Bob p_b) has published the vote $v_{p_b,t_a} + 1$. As Bob does not cooperate with Mallory, he will publish the right key, which decrypts the correct votes and there is a situation of Bob's word against Mallory's. Due to the fact that all participants have signed the keys they share with the others, Bob can prove that Mallory presented a wrong key.

In the simplified key exchange (cp. Sect. V-A on page 4), Mallory cannot cheat, either. Assuming that $p_m < p_b$, she would at least have to find an x so that

$$k_{p_m,p_b,t} - 1 = \mathcal{F}(x) \bmod n. \tag{8}$$

Solving this problem is equivalent to breaking the preimage resistancy of $\mathcal{F}(\cdot) \bmod n$.

To further improve her strength, Mallory can collude within a set of voters $p_1, \ldots, p_c$ to:

1) enforce a time slot t by ensuring that the sum of their votes is higher than the number of cheating voters ($v_{p_1,t} + \ldots + v_{p_c,t} > c$); and to
2) hide this attack by consistently lying about their keys so that the result verification leads to votes in the allowed range ($0 \leq v_{p_1,t} + \ldots + v_{p_c,t} \leq c$).

The colluding voters' problem is to ensure that their key modifications sum up to 0. This property stems from the pairwise inverse key exchange. Since the colluding voters have to keep the sum of their keys constant, the sum of their votes remains constant, too. This renders the attack impossible.

B. Privacy

To attack Bob's privacy, Mallory has to cryptanalyze Bob's encrypted vote vector or parts of it. Since in the basic scheme, each element of this vector is encrypted with an information-theoretically secure one-time pad, the only possibility to obtain the availability pattern is in getting hold of all the other keys. Finding the right keys is hard in the basic scheme and assumed to be hard with the Diffie–Hellman extension. As already discussed in Section V-A on page 4, the properties of the two functions $\mathcal{F}$ and $\mathcal{G}$ guarantee the secrecy of the vote in case of the extended key exchange.

The apparent feature dynamic leaving (cp. Section V-D on the previous page) can in fact be abused to obtain the keys from other voters. If Eve is able to hold Bob's message back, she probably can convince the others to kick Bob from the poll. If she convinces all other voters, she gets Bob's keys and can decrypt his entire vote vector. Thus she does not only prevent Bob from contributing his preferences, but also compromises his privacy regarding his availability pattern. A reliable broadcast channel would prevent Eve from successfully launching this attack.

As discussed in the previous section, a malicious voter can send a value higher than 1 to increase the chance of winning for a specific time slot. Consider for instance an attacker who guesses that three other voters cannot participate at his favorite time, so she could cast a value of 4 to compensate for the three missing votes. Because of result verification, the protocol violation will be detected and the agreed time slot will be declared as invalid. Nevertheless, this way an attacker still can compromise the privacy of voters. She thus could discover the availability of voters at a specific time slot. However, the

[3] Note that the key graph is not necessarily complete if dynamic joining has been used.

attacker will be discovered afterwards. If this goes along with some cost (penalty, reputation loss, etc.), it makes such attacks unattractive.

C. Untrusted Server

Only very little trust is required in the server. Its mere functionality is to provide a (not necessarily anonymous) blackboard which collects and jointly publishes the encrypted vote vectors. These must not be published before the last voter has voted. Otherwise one voter may wait until all but him have voted and can calculate the preliminary result vector before casting his own vote. This is not a serious restriction and it can be relaxed with one additional communication phase in which all voter have to commit to their votes.

D. Usability

Both the sanity check that the correct vote vector has been published and the result verification can be done by a client program, the usability of which is beyond the scope of the protocol. To discuss the usability of the scheme, the main criterion is the number of message exchanges, or more precisely the number of user interactions required to successfully schedule an event.

In the initialization phase, every voter has to exchange a vector of keys with every other voter. As this key exchange would require many message exchanges, we proposed to simplify it using Diffie–Hellman key agreement (cf. Sect. V-A). With this extension, every voter must register once at the voting server. As this initially exchanged key can be used for all future polls, this step can be neglected. It is not uncommon for practical applications to require a similarly complex enrollment, e. g., to register a user account.

For successful schedulings (i. e., no inconsistencies due to attacks occur), the remainder of our protocol needs no more message exchange compared to existing (privacy-unfriendly) event schedulers. In case of an attack, there is one more communication phase: the result verification. This is a trade-off made in our scheme to reduce the amount of trust to be placed in the server. Compared to the most efficient election schemes, the same number of message exchanges is needed to ensure multi-lateral security [38].

Note that the size of the messages is certainly somewhat larger than in naive implementations (approximately by factor $\log n$), but since bandwidth is not such a scarce resource in most cases, we deem the usability impact small compared to the gain in voters' privacy.

E. Computational Complexity

Finally we evaluate the overall efficiency of our extended scheme and show that it is more efficient than a repeated application of the election schemes in the literature.

Looking from the server's perspective, the scheme is absolutely efficient as no computation needs to be done there. The only purpose of the server is to provide a blackboard where the vote vectors are published, and to keep them secret until the last one has been received.

Regarding the voters, the initialization requires a random number sec_p and deriving a public key $\mathsf{pub}_p = g^{\mathsf{sec}_p} \bmod q$. As mentioned in the previous section, this initial key exchange is reusable for all future polls. So this step can be neglected. Remember that for the poll initialization of our basic scheme, the user would have to exchange a key for every time slot and calculate a signature for each. This would be rather inefficient. Although the dimension of the key matrix is the same for the extended scheme, it can be computed more efficiently. The most time-consuming operation in Equation 6 on page 5 is the exponentiation $g^{\mathsf{sec}_{p_i} \cdot \mathsf{sec}_{p_j}} \bmod q$. As we can reuse the result for each time slot, every voter p_i has to calculate it only once for every other voter $p_j, j \neq i$. Furthermore, every voter has to calculate $|T| \cdot (|P| - 1)$ symmetric decryptions and if the result verification phase is needed, every voter has to calculate $|T| \cdot (|P| - 1)$ hashes to retrieve the keys for the agreed time slot t_a. Another time-consuming operation is the asymmetric signature introduced in Section VI-A on the previous page. Every voter has to calculate 1 and verify $|P| - 1$ signatures.

To sum it up, in a successful run of the extended scheme, every voter has to calculate $|P| - 1$ discrete exponentiations, $|T| \cdot (|P| - 1)$ hashes and symmetric decryptions, and one digital signature. All other computations are additions modulo n, which are cheap enough to be neglected. Efficient and sufficiently secure hash as well as symmetric ciphers are abundant, so the reuse of expensive operations lets our scheme scale well in the number of time slots.

Remark that even without the extensions, our basic scheme is more efficient than existing election schemes repeated $|T|$ times.

VII. Conclusion

We have proposed a privacy-enhanced event scheduling scheme that is suitable to be implemented in practical Web 2.0 sites, groupware applications, or as function in future privacy-enhanced identity management systems [39].

The scheme is efficient and scales, in terms of hash and symmetric encryption functions, linearly in the number of possible points in time. The effort per voter in terms of asymmetric cryptographic operations scales linearly in the number of voters. Moreover, no central trusted entity is required. Practical aspects are considered, such as dynamic joining and leaving of participants in running polls.

Future extensions could complement these features, e. g., by integrating a threshold scheme, dynamic insertion and deletion of time slots, updating and revoking votes, or allowing other decision rules than unanimous agreement to schedule an event (e. g., maximum vote, or more than two options). Even with unanimous agreement, the risk of denial-of-service or legal-but-selfish votes (cf. Sect. VI-A on the preceding page) could be reduced by letting voters prove that they signaled availability for more than a certain minimum number of time slots. Other directions of research could be approaches to event scheduling based on mixes or blind signatures.

In the near future, we plan to actually implement the proposed scheme and make it available to the public as Web 2.0

application. Even in this scenario, we want to realize the gentle trust assumptions with respect to a web server and execute as much as possible in the clients' web browsers. We are very confident that we will reach a better performance for verification than found in the literature at other places (e. g., Helios by Adida needs 4 hours for result-verification [40]).

ACKNOWLEDGMENTS

The authors want to thank Immanuel Scholz for ideas and discussion. Helpful comments from Matthias Kirchner, Stefan Köpsell, Andreas Pfitzmann, Sandra Steinbrecher, and the anonymous reviewers have been incorporated.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement No. 216483.

REFERENCES

[1] T. O'Reilly, "What is Web 2.0," http://www.oreilly.de/artikel/web20.html, Sep. 2005.

[2] M. Näf, "Doodle homepage," http://www.doodle.com, Jan. 2009.

[3] M. S. Franzin, E. C. Freuder, F. Rossi, and R. Wallace, "Multi-agent meeting scheduling with preferences: Efficiency, privacy loss, and solution quality," AAAI Technical Report WS-02-13, 2002.

[4] R. Greenstadt, J. P. Pearce, E. Bowring, and M. Tambe, "Experimental analysis of privacy loss in DCOP algorithms," in *Proc. of ACM AAMAS*. New York: ACM Press, 2006, pp. 1424–1426.

[5] M. Yokoo, K. Suzuki, and K. Hirayama, "Secure distributed constraint satisfaction: Reaching agreement without revealing private information," *Artificial Intelligence*, vol. 161, pp. 229–245, 2005.

[6] R. Grimm, R. Krimmer, N. Meißner, K. Reinhard, M. Volkamer, and M. Weinand, "Security requirements for non-political internet voting," in *Electronic Voting*, ser. LNI, R. Krimmer, Ed., vol. 86. GI, 2006, pp. 203–212.

[7] O. Cetinkaya, "Analysis of security requirements for cryptographic voting protocols (extended abstract)," in *ARES*. IEEE Computer Society, 2008, pp. 1451–1456.

[8] L. Mitrou, D. Gritzalis, and S. K. Katsikas, "Revisiting legal and regulatory requirements for secure e-voting," in *SEC*, ser. IFIP Conference Proceedings, A. Ghonaimy, M. T. El-Hadidi, and H. K. Aslan, Eds., vol. 214. Kluwer, 2002, pp. 469–480.

[9] E. Gerck, "Voting system requirements," *The Bell*, vol. 2, no. 2, pp. 3–15, Feb. 2001.

[10] Deutscher Bundestag, "Bundeswahlgesetz für die Bundesrepublik Deutschland," Mar. 2008, § 35.

[11] Parlamentarischer Rat, "Grundgesetz für die Bundesrepublik Deutschland," May 1949, art. 38.

[12] A. Pfitzmann and M. Hansen, "Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology," http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, Feb. 2008, v0.31.

[13] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[14] J. D. Cohen and M. J. Fischer, "A robust and verifiable cryptographically secure election scheme," in *SFCS '85: Proceedings of the 26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. Washington, DC, USA: IEEE Computer Society, 1985, pp. 372–382.

[15] J. C. Benaloh and M. Yung, "Distributing the power of a government to enhance the privacy of voters," in *PODC '86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 1986, pp. 52–62.

[16] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *AUSCRYPT*, ser. Lecture Notes in Computer Science, J. Seberry and Y. Zheng, Eds., vol. 718. Springer, 1992, pp. 244–251.

[17] C. Park, K. Itoh, and K. Kurosawa, "Efficient anonymous channel and all/nothing election scheme," in *EUROCRYPT*, 1993, pp. 248–259.

[18] K. Sako and J. Kilian, "Secure voting using partially compatible homomorphisms," in *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1994, pp. 411–424.

[19] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani, "Fault tolerant anonymous channel," in *ICICS*, ser. Lecture Notes in Computer Science, Y. Han, T. Okamoto, and S. Qing, Eds., vol. 1334. Springer, 1997, pp. 440–444.

[20] M. Abe, "Universally verifiable mix-net with verification work indendent of the number of mix-servers," in *EUROCRYPT*, 1998, pp. 437–447.

[21] M. Jakobsson, "A practical mix," in *EUROCRYPT*, 1998, pp. 448–461.

[22] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, "Practical multi-candidate election system," in *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2001, pp. 274–283.

[23] K. Sako, "Electronic voting scheme allowing open objection to the tally," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 77, no. 1, pp. 24–30, 1994.

[24] M. Ohkubo, F. Miura, M. Abe, A. Fujioka, and T. Okamoto, "An improvement on a practical secret voting scheme," in *ISW*, ser. Lecture Notes in Computer Science, M. Mambo and Y. Zheng, Eds., vol. 1729. Springer, 1999, pp. 225–234.

[25] B. W. DuRette, "Multiple administrators for electronic voting. Bachelor's thesis, Massachusetts Institute of Technology," May 1999. [Online]. Available: http://theory.lcs.mit.edu/~cis/theses/DuRette-bachelors.pdf

[26] D. Chaum, "Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA," in *EUROCRYPT*, 1988, pp. 177–182.

[27] ——, "Secret-ballot receipts: True voter-verifiable elections," *Security & Privacy, IEEE*, vol. 2, no. 1, pp. 38–47, Jan.–Feb. 2004.

[28] D. Chaum, P. Y. Ryan, and S. Schneider, "A practical voter-verifiable election scheme," in *Computer Security – ESORICS 2005*, ser. Lecture Notes in Computer Science, vol. 3679/2005. Springer Berlin / Heidelberg, 2005, pp. 118–139. [Online]. Available: http://www.springerlink.com/content/ebrbl9kc81bhx98j/

[29] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora, "Scantegrity: End-to-end voter-verifiable optical- scan voting," *Security & Privacy, IEEE*, vol. 6, no. 3, pp. 40–46, May–June 2008.

[30] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman, "Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes." in *EVT*, D. L. Dill and T. Kohno, Eds. USENIX Association, 2008. [Online]. Available: http://dblp.uni-trier.de/db/conf/uss/evt2008.html#ChaumCCEPRRSS08

[31] J.-M. Bohli, J. Müller-Quade, and S. Röhrich, "Bingo voting: Secure and coercion-free voting using a trusted random number generator," in *VOTE-ID*, ser. Lecture Notes in Computer Science, A. Alkassar and M. Volkamer, Eds., vol. 4896. Springer, 2007, pp. 111–124.

[32] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Commun. ACM*, vol. 28, no. 10, pp. 1030–1044, 1985.

[33] L. Cranor and R. Cytron, "Sensus: A security-conscious electronic polling system for the internet," 1997. [Online]. Available: citeseer.ist.psu.edu/cranor97sensus.html

[34] M. A. Herschberg, "Secure electronic voting over the world wide web," Master's thesis, Massachusetts Institute of Technology, May 1997.

[35] T. Herlea et al., "On securely scheduling a meeting," in *Trusted Information — The New Decade Challenge (Proc. of IFIP SEC)*, M. Dupuy and P. Paradinas, Eds., 2001, pp. 183–198.

[36] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, Jan. 1988. [Online]. Available: http://www.springerlink.com/content/m74414x28822u525/

[37] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.

[38] A. Pfitzmann, "Multilateral security: Enabling technologies and their evaluation," in *ETRICS*, ser. Lecture Notes in Computer Science, G. Müller, Ed., vol. 3995. Springer, 2006, pp. 1–13.

[39] M. Hansen, P. Berlich, J. Camenisch, S. Clauß, A. Pfitzmann, and M. Waidner, "Privacy-enhancing identity management," *Information Security Technical Report*, vol. 9, no. 1, pp. 35–44, 2004.

[40] B. Adida, "Helios: Web-based open-audit voting," in *USENIX Security Symposium*, P. C. van Oorschot, Ed. USENIX Association, 2008, pp. 335–348.